

GELAB – The Cutting Edge of Grammatical Evolution

KRISHN KUMAR GUPTA^{1,2}, MUHAMMAD ADIL RAJA^{3,7}, AIDAN MURPHY^{2,4,5,7},
AYMAN YOUSSEF^{2,4,6,7}, AND CONOR RYAN^{1,2,4,7}, (Senior Member, IEEE)

¹Department of Electrical and Electronic Engineering, Technological University of the Shannon: Midlands Midwest, Limerick, V94 ECST Ireland

²Bio-Computing and Developmental Systems (BDS) Research Group, University of Limerick, Limerick, V94 T9PX Ireland

³Regulated Software Research Center (RSRC), Dundalk Institute of Technology (DkIT), Dundalk, A91 K584 Ireland

⁴Department of Computer Science and Information Systems, University of Limerick, Limerick, V94 T9PX Ireland

⁵Complex Software Laboratory, University College Dublin, Belfield, Dublin 4, D04V1W8 Ireland

⁶Department of Computers and Systems, Electronics Research Institute, Cairo 12622, Egypt

⁷The Irish Software Research Centre, Lero, University of Limerick, Limerick, V94 T9PX Ireland

Corresponding author: Ayman Youssef (aiman.mahgoub@ul.ie)


This work was supported by the Science Foundation Ireland (SFI) under Grant #16/IA/4605 and Grant #13/RC/2094.

ABSTRACT The advent of cloud-based super-computing platforms has given rise to a Data Science (DS) boom. Many types of technological problems that were once considered prohibitively expensive to tackle are now candidates for exploration. Machine Learning (ML) tools that were valued only in academic environments are quickly being embraced by industrial giants and tiny startups alike. Coupled with modern-day computing power, ML tools can be looked at as hammers that can deal with even the most stubborn nails. ML tools have become so ubiquitous that the current industrial expectation is that they should not only deliver accurate and intelligent solutions but also do so rapidly. In order to keep pace with these requirements, a new enterprise, referred to as MLOps has blossomed in recent years. MLOps combines the process of ML and DS with an agile software engineering technique to develop and deliver solutions in a fast and iterative way. One of the key challenges to this is that ML and DS tools should be efficient and have better usability characteristics than were traditionally offered. In this paper, we present a novel software for Grammatical Evolution (GE) that meets both of these expectations. Our tool, GELAB, is a toolbox for GE in Matlab which has numerous features that distinguish it from existing contemporary GE software. Firstly, it is user-friendly and its development was aimed for use by non-specialists. Secondly, it is capable of hybrid optimization, in which standard numerical optimization techniques can be added to GE. We have shown experimentally that when hybridized with meta-heuristics GELAB has an overall better performance as compared with standard GE.

INDEX TERMS Grammatical evolution, diversity, hybrid optimization.

I. INTRODUCTION

AIOps and MLOps are buzzwords that are popular in the technological and innovative arenas. Leveraging from DevOps practices, they modulate the typical activities of ML engineers with a certain agile Software Engineering (SE) technique, such as Scrum, continuous development, or lean development. Additionally, just as in DevOps, ML practitioners are expected to work in conjunction with the operational teams. The goal here is to expedite the developmental cycle of software or ML. As a result, continuous delivery of software or ML products ensues. Not only do the total developmental

The associate editor coordinating the review of this manuscript and approving it for publication was Shaoyong Zheng .

costs decrease, but the process of development also becomes fast and flexible [1].

A typical ML practitioner, nonetheless, requires tools nowadays with which they can work quickly and easily so as to keep pace with the rest of the developmental and operational teams. Apart from that, they should have access to a wide variety of ML algorithms in their toolbox.

A major field in ML is Evolutionary Computation (EC). These algorithms mimic the natural evolutionary process where the Darwinian paradigm of the survival of the fittest applies. They start with a population of individuals created, usually, at random, each of which is tested by a *fitness function* to assess its performance on the task at hand. Better performing individuals survive to the next generation and are

recombined with other good performing individuals using the processes of crossover and mutation.

GE is an evolutionary algorithm invented at the University of Limerick. The algorithm is widely used for different applications such as software testing [2], digital circuit design [3], symbolic regression problems [4], and stock market rules prediction [5] to name a few.

GE differs from most EC algorithms in that it can produce compilable as well as interpretable computer code; because of this, its early implementations were in C++. LibGE is the first, and still widely used, implementation of GE in C++ [6], although versions now exist in languages such as Python. Although quite efficient to run and known for producing accurate results, it could be prohibitive to use in an MLOps context, primarily because it requires considerable pre-configuration and set-up. Furthermore, C++ has not enjoyed the same support for ML algorithms that other languages have, making it difficult to integrate other techniques with it.

We have implemented GE in Matlab. Referred to as a Numerical Swiss Army Knife, Matlab is sometimes considered to be good at everything while best at nothing [7]. The reason it is believed to be good at almost everything is the wide variety of proprietary as well as open-source third-party toolboxes that are available in and for Matlab in a large number of scientific domains and technical disciplines. Along with this, Matlab maintains its own implementations of various ML tools and optimization algorithms. This makes Matlab a great framework ready for MLOps, rapid prototyping and development of ML algorithms.

Our main goal was to implement a system that has better usability characteristics: a user-friendly system that would save the ML practitioner having to go through the tedium of installation, esoteric configurations, and daunting coding. GELAB is plug-and-play and can be freely downloaded from the Internet.¹ Moreover, the implementation was based on the simplest software engineering practices and a software design that is easy to follow. While the built-in features are many, a new scheme can be readily implemented and augmented to work with the existing code. Gluing GELAB to existing Matlab toolboxes is straightforward because most of the toolboxes are highly inter-operable. As a matter of fact, we have already joined it with the global optimization toolbox of Matlab to achieve the hybrid optimization capability reported in this paper.

This paper describes GELAB, its various features and thoroughly reports its results. The rest of this paper is organized as follows. In section II, we describe GE system. GELAB is presented in section III. Section IV discusses hybrid optimization, which is one of the core features of GELAB, which is made possible because of our use of Matlab. In Section V, we explain how we perform hybrid optimization using GELAB. Section VI presents a thorough analysis of the computational costs associated with running

GELAB on a multi-core architecture. We also report on aspects of population diversity in an Evolutionary Algorithm (EA) and how it is computed and reported in GELAB in Section VII. Section VIII presents an analysis of various crossover operations that are implemented in GELAB. Similarly, section IX shows various mutation operators that are implemented in GELAB, followed by an analysis. Finally, section X concludes the paper.

II. GRAMMATICAL EVOLUTION

GE is a variant of Genetic Programming (GP). The main goal of the algorithm is to search a program space to find the best program or function that can efficiently solve a user-specified problem.

GE employs a grammar, usually specified in Backus Naur Form (BNF), to map a binary or an integer coded string to a program or a function. In GE, the string is referred to as the *genotype* and the resulting computer program as the *phenotype*. This idea was inspired by genetics where a particular genotype (that encodes the genetic code of an organism) leads to a certain phenotype that characterizes the typical behavior or appearance of an organism. A pictorial representation of the analogy between genetics and GE is shown in Figure 1a.

In GE, this genotype to phenotype mapping is achieved by utilizing the BNF grammar. Hence, grammar is at the heart of any GE system and constitutes the major component of the genotype/phenotype mapper. The mapping of an integer-coded genome to a corresponding phenotype is shown in Figure 2.

Genetic operators (selection, mutation, and crossover) are applied on the binary/integer coded genotype string to accomplish an evolutionary search process. The algorithm is largely agnostic of the underlying programming language. The induction of grammar also allows the introduction of domain knowledge into program production rules. Hence, GE has been proven to be useful in addressing a wide variety of computational problems from a myriad of problem domains [8].

III. GELAB

GELAB is a Matlab version of GE that is based on 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. GELAB's mapper, the piece of software that is responsible for genotype to phenotype mapping, is implemented in Java using an almost exact port of libGE.

LibGE, however, requires some background knowledge in Computer Science, specifically computer programming, to be able to work with it. By implementing a Matlab version of GE, we hope it will be much easier for researchers from different backgrounds to engage in scientific activities using GE. One other advantage of GELAB is the ability to integrate the toolbox with other toolboxes in Matlab, such as the image processing, signal processing, or circuit design toolboxes, to implement different GE applications.

¹Gelab can be downloaded from: <https://github.com/adilraja/GELAB>

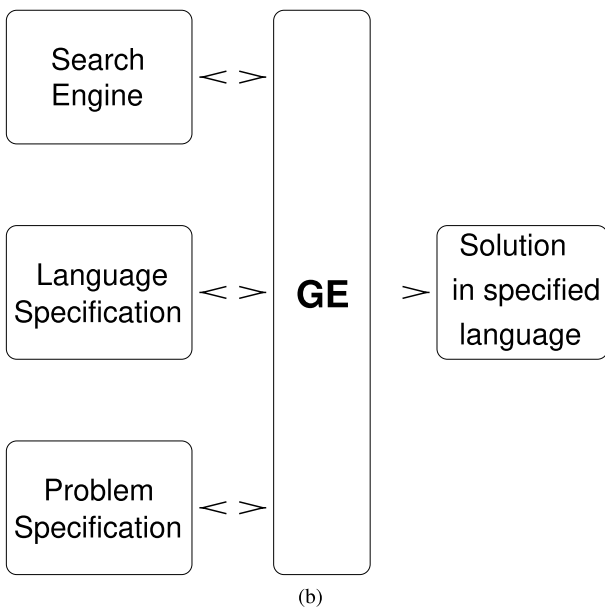
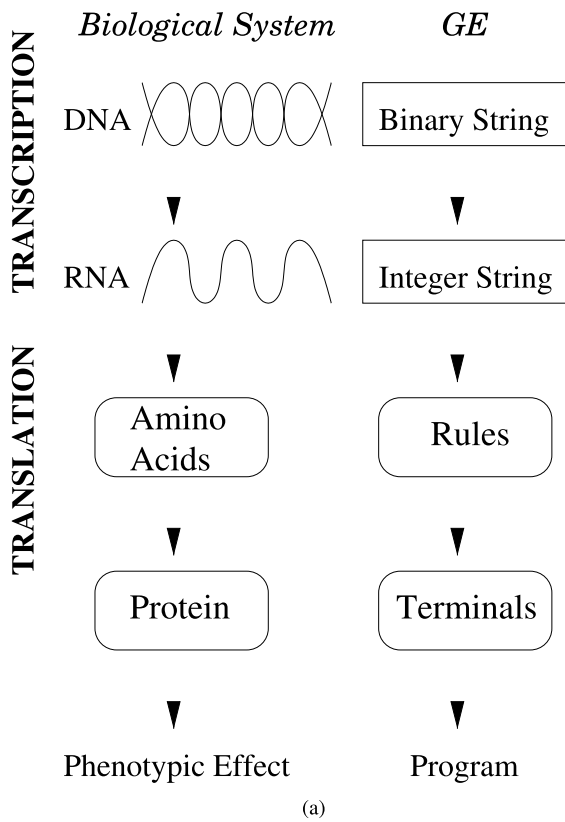


FIGURE 1. (a) Genotype to phenotype mapping in biological systems and in GE (b) Conceptual diagram of GE's mapping process.

GELAB has four salient features. This is to reduce computational and memory requirements and to make it easy for the toolbox to solve a wider range of problems. The first feature is the ability of GELAB to utilize the parallel computing toolbox of Matlab to execute different runs of a GE experiment in parallel on multi-core hardware architecture. GELAB can also benefit from Matlab's proprietary distributed computing

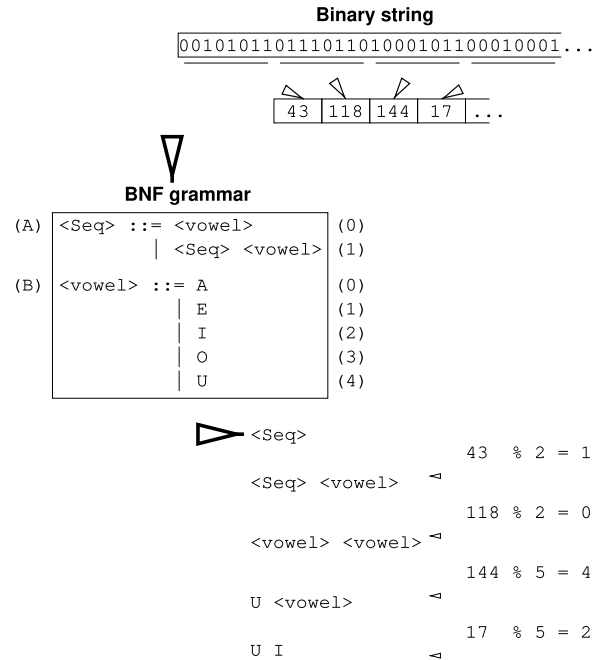


FIGURE 2. Mapping of the integer-coded genome to a corresponding computer program using grammar.

center to run evolutionary experiments in a cluster or cloud computing environment.

The second implemented feature is an integer-valued version of the Compact Genetic Algorithm (cGA) [9]. One of the main benefits of the cGA is that it emulates the evolutionary dynamics with reduced computational complexity and reduces the requirement to have a large memory, enabling it to scale to more complex problems. Being population-based meta-heuristics, EAs are naturally resource hungry. The amount of memory required to run an EA normally scales with the population size of any experiment. A small memory footprint, as in cGA, allows for the execution of experiments with larger population sizes without ever scaling the required memory requirements in proportion to the population size. This feature of a cGA has been deemed particularly useful in literature to run experiments that require a large population sizes as well as to run experiments for memory-constrained problems such as in the case of evolvable hardware [10].

The third feature is a program or individual cache, which allows the toolbox to maintain a pool of frequently occurring individuals which have the same genotype in a cache along with their phenotypes as well as evaluations. This feature is added to the GELAB toolbox to reduce the computational time required to perform computationally expensive genotype to phenotype mapping, as well as evaluation of an individual on the problem data.

The fourth feature is the ability to evolve multi-tree programs. This allows GELAB to evolve Multiple Input Multiple Output (MIMO) systems. These systems accept multiple inputs and are expected to generate multiple outputs simultaneously. Notable examples are controllers for driverless cars and Unmanned Aerial Vehicles (UAVs) [11].

GELAB has been thoroughly tested, initially with a small number of data sets, for more than two years. During this period, the toolbox was observed and all software errors were dealt with either in Java or Matlab. GELAB was initially tested against data related to speech quality estimation [12], [13]. Apart from that, GELAB was also tested against various well-known benchmark problems related to symbolic regression [14]–[16].

IV. HYBRID OPTIMIZATION

Algorithms such as GP and GE are well known for their ability to perform a structural search for a target program. At the time they were invented this was almost a paradigm shift as, traditionally, optimization generally meant to search for the optimal set of values of a target computer program or a mathematical model. This implies that either the right structure of the target program or model is already known or the burden of finding it rests with the ML scientist. The advent of GP and GE considerably lessened this problem. However, this ability to innovate computer programs came at a price. The job of finding the right coefficients was left to these algorithms. The most usual way of finding the coefficients while using these algorithms was to use random numbers as possible values and they generally do not have the ability to perform focused searches on specific coefficients, rather sub-expressions must be generated to combine the various initial random values into more useful numbers.

Clearly, this was a problem. One solution is to augment each of these algorithms with a numerical optimization algorithm. In such a scheme, GP or GE would search for the right structure of the target program and the job of finding the optimal values of the coefficients would now be outsourced to some numerical optimization algorithm. Such a scheme has occasionally been used with GP and it is termed hybrid optimization. With GE, this scheme has rarely been employed, although a few examples exist [17]–[19]. GE, nonetheless, is also an ideal candidate for hybrid optimization. We readily exploited this capability by integrating it with Matlab's built-in meta-heuristic algorithms for numerical optimization.

V. HYBRID OPTIMIZATION WITH GELAB

As described in the previous section, in this approach GE is used for searching for the structure of the target mathematical model and meta-heuristic or numerical optimization algorithms are used to tune the constants (or coefficients) of the mathematical model. In this work, GELAB was used for evolving the structure of the target model and was integrated with three meta-heuristic algorithms, namely Simulated Annealing (SA) [20], Genetic Algorithm (GA) [21], and Particle Swarm Optimization (PSO) [22]. Rather conveniently, all of these meta-heuristics belong to the proprietary toolboxes of Matlab and GELAB has been integrated with these algorithms.

Consider the following target expression:

$$y = -4.7 + (.52 * \cos(x)) \quad (1)$$

In this equation, y is a function of the independent variable x . In this example, GELAB is responsible for generating or evolving the overall skeleton of the target expression including any functions having a sinusoidal behavior. The optimum values for the coefficients of the target expression are derived by further tuning the coefficients of the derived models either with SA, GA, or PSO. The best values discovered are stored in an object that has both the genotypic and phenotypic details of an individual. Every time a phenotype is used, these values are used for the respective constants. It must be noted that the genotype of an individual remains unaffected because of this process, only the phenotype is affected.

A. A SCHEME TO DECREASE THE RUNNING TIME OF GELAB

It is well known that hybrid optimization can be quite resource-hungry. Optimizing every individual created with GE, in every generation, with a meta-heuristic can require substantial computing resources. As a result, this can require tremendous computation time. The research community has been cognizant of this in the past. As a solution, researchers have often reverted to schemes in which a handful of individuals of the whole population are optimized with a meta-heuristic. The decision about which individuals to optimize can be based on certain practical considerations [21]. For instance, one of the criteria could be to figure out first as to which individuals could benefit most from hybrid optimization. To this end, a small number of the fitness-wise best individuals can be chosen for hybrid optimization at every generation.

In our work, we introduced an eligibility scheme for hybrid optimization that is based on the age of an individual. In our scheme, as an individual is created, its fitness is evaluated using Mean Squared Error (MSE). At this point, it is marked as having an age of zero and keeps using MSE to measure its fitness until it is one generation old. The age of an individual is incremented by one at the point of fitness evaluation in every subsequent generation.

As the age of an individual reaches two generations, it is marked as eligible for hybrid optimization. At this stage, it is treated with the meta-heuristic with a certain small probability. The individual remains eligible for hybrid optimization until the time it is five generations old.

After this, the fitness function is changed to linear scaling, (MSE_s), for all the subsequent generations. Linear scaling is defined according to equation 2 and it has been found to be quite beneficial for symbolic regression in the past,

$$MSE_s(y, t) = 1/n \sum_i^n (t_i - (a + by_i))^2 \quad (2)$$

where y is a function of the input parameters (a mathematical expression), y_i represents the value produced by a GE individual, and t_i represents the target value. a and b adjust the slope and y -intercept of the evolved expression to minimize the

```

⟨expr⟩ ::= ⟨lexpr⟩⟨op⟩⟨expr⟩
        | ⟨b-pre-op⟩ (⟨expr⟩, ⟨expr⟩)
        | ⟨var⟩
        | ⟨const⟩
⟨lexpr⟩ ::= ⟨expr⟩
⟨op⟩ ::= + | - | .*
⟨b-pre-op⟩ ::= ge_divide
⟨var⟩ ::= X(:, 1)
⟨const⟩ ::= w(1)
⟨pop⟩ ::= ""

```

FIGURE 3. BNF grammar.

squared error. They are computed according to equation (3).

$$a = \bar{t} - b\bar{y}, \quad b = \frac{\text{cov}(t, y)}{\text{var}(y)} \quad (3)$$

Moreover, an individual is chosen next time for further optimization with a meta-heuristic only if it showed improvement the last time it was optimized. The reason for this is to save on computation time. It is customary to run a meta-heuristic on a model multiple times due to the stochastic nature of the algorithm. However, given the practical concern of reducing computation time, we implemented this trade-off. Only those individuals that showed an improvement in fitness the last time they were treated with it will be further optimized by the meta-heuristic.

B. PERFORMANCE ANALYSIS OF GELAB

We ran GELAB on a number of well-known benchmark polynomial problems from the domain of symbolic regression. These polynomials are shown in Table 1. These have been used in past works such as those reported on in [14]–[16], [23]. We also compared the results of GELAB with GP. In order to conduct the GP experiments, we used GPLab [24], a well-known GP toolbox for Matlab implemented by Sara Silva.

In order to obtain data, we generated uniformly distributed random variables in the range $[-1.5, 1.5]$. 100 data instances were generated. The benchmark polynomials were executed using this data to obtain the corresponding target values. Half of the data instances were used for training and the remaining half were set aside for testing.

The grammar shown in Figure 3 was used in all the experiments: We have already used this grammar in [16]. The initial population was made of 500 randomly generated individuals. Tournament-based selection, along with Lexicographic Parsimony Pressure (LPP) [25], was implemented. With LPP, individuals are picked for mating using the traditional tournament selection used in EC. However, if there is a tie in fitness between two individuals, the one with a smaller tree size is picked for competition. An elitist replacement criterion was practiced. Adaptive crossover and mutation probabilities were implemented with a selection of different operators.

Sensible initialization [26] was used to generate the initial population of GE. Ramped half-and-half initialization was used for GP. Each experiment was composed of 40 runs and each run was 50 generations long. These configuration parameters are shown in Table 2.

C. RESULTS

For each of the polynomials mentioned in Table 1, we performed five experiments. Three of the experiments involved either SA, GA, or PSO for hybrid optimization. The fourth experiment involved GE and used MSE only as the fitness function. The fifth experiment involved GP. This resulted in a total of thirty experiments.

Figure 4 shows the results of the experiments. The x-axis represents 50 generations throughout. The y-axis represents normalized scores with respect to MSE. The scores are computed according to equation 4.

$$\text{score} = \frac{1}{1 + \text{MSE}} \quad (4)$$

Each sampled point in the plots shows an average of over 40 independently conducted runs. The following equation is used to compute the 95% confidence limits of the error bars:

$$\bar{X} \pm 1.96 \frac{\sigma}{\sqrt{n}} \quad (5)$$

where \bar{X} and σ are the mean and standard deviation of the fitness scores respectively of n runs ($n = 40$ in our case). It means that one can be 95% confident that the fitness scores from all the runs lie within these error bars. Moreover, a lack of overlap between any two of the modeled schemes means that the corresponding populations are statistically different.

Figure 4 shows that various flavors of GE performed at least as well as GP on 5 out of the 6 chosen benchmark problems on test data. Only on one problem, Figure 4c, the results of GE were significantly inferior to GP. Moreover, on two problems, Figures 4a and 4f, all variants of GE performed significantly better than GP. To this end, our results are similar to those reported in [23].

VI. COMPUTATIONAL COST OF RUNNING GELAB

This section reflects on the resource consumption of GELAB. Comparisons are made for various flavors of GE, with and without hybridization with meta-heuristic algorithms.

The computational time of an EAs is a well-known practical concern. In this section, we discuss experiments we ran on GELAB comparing the resource consumption of GELAB with a C++ implementation of GE. We also have some insights into the effects of hybridization on resource consumption.

In our experimental setup, we use four different configurations i.e., GELAB hybridized with SA, GA, or PSO, and GELAB running without hybridization. Moreover, we also compared the results of running GE using the legacy libGE system. The hardware on which we ran GELAB had an Intel Xeon Gold 6138 CPU @ 2.00GHz with 40 cores and 128GB of RAM.

TABLE 1. Benchmark polynomials.

Sr. No.	No.	Polynomial
1	Keijzer1	$f(x) = 0.3x\sin(2\pi x)$
2	Keijzer2	$f(x) = 1 + 3x + 3x^2 + x^3$
3	Keijzer3	$f(x, y) = 8/(2 + x^2 + y^2)$
4	Keijzer4	$f(x, y) = x^4 - x^3 + y^2/2 - y$
5	Keijzer5	$f(x, y) = x^3/5 + y^3/2 - y - x$
6	Keijzer6	$f(x_1, \dots, x_{10}) = 10.59x_1x_2 + 100.5967x_3x_4 - 50.59x_5x_6 + 20x_1x_7x_9 + 5x_3x_6x_{10}$

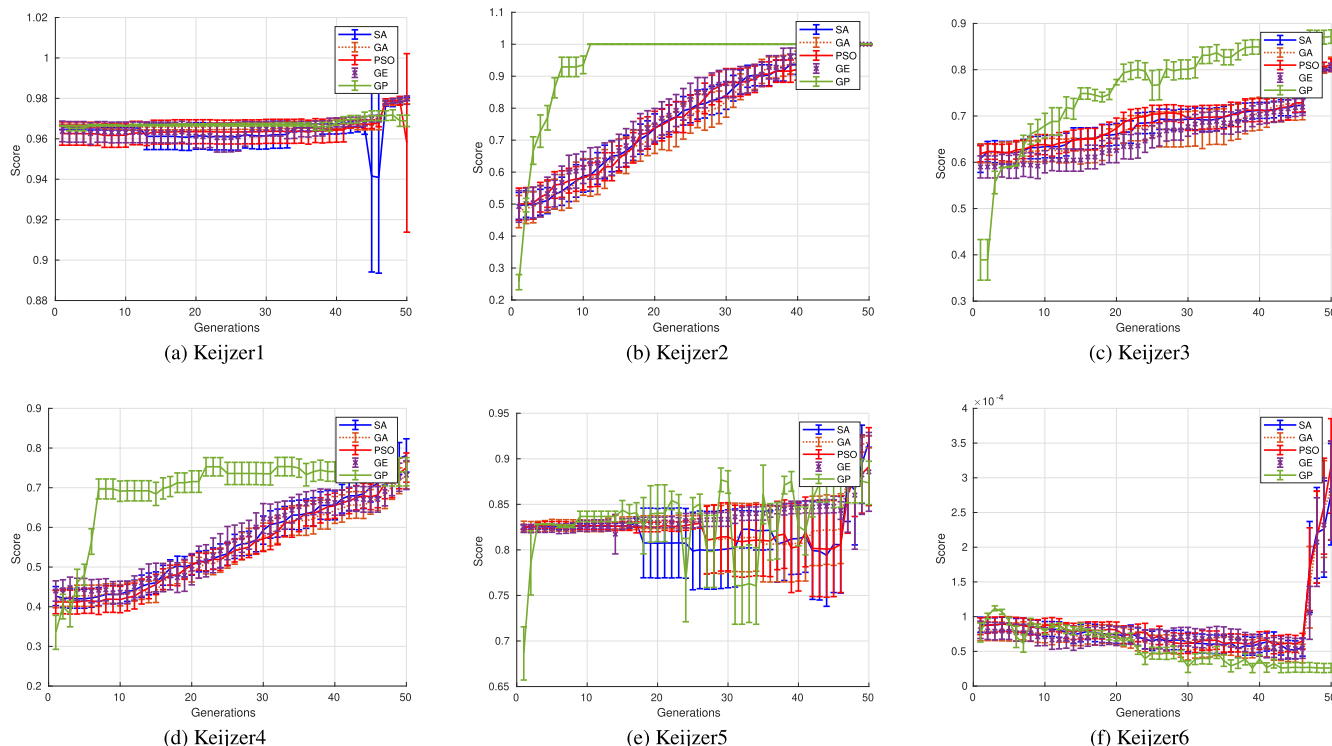


FIGURE 4. Mean of test fitness score along with error bars for each generation.

TABLE 2. GE experimental setup.

Parameter	Value
Runs	40
Total Generations	50
Population	500
Selection	Tournament
Crossover	0.9 (Effective)
Mutation	0.01
Elitism	Yes
Initialisation	Sensible

Figure 5 shows the average of the cumulative time it takes the algorithm to evolve a certain number of generations. The first two polynomials in Table 1 take a similar order of time to complete 50 generations of evolution as shown in Figures 5a and 5b. However, the remaining polynomials in Table 1 took a longer duration as shown in Figure 5c-5f. The time increase is due to the increase in complexity of

respective polynomials in terms of an increasing number of variables, constants or arithmetic operators. This experiment reflects the computational efficiency of GE hybridized with several meta-heuristics. We observe in Figure 5 that on average, libGE consumes less time to complete one run than any variant of GELAB shown. This is mainly due to libGE running as a sequential algorithm and the way it uses the core architecture of the computer used. A nice discussion and experimental details are presented in [14]. GELAB is extremely easy to parallelize using the parallel computing toolbox of Matlab. As a matter of fact, the way it has been implemented already, it runs in the parallel computing mode. So the results achieved in terms of computation time simply add to the cornucopia of features GELAB offers.

VII. POPULATION DIVERSITY IN GELAB

Population diversity is a very important aspect of any evolutionary system. A population with a diverse set of individuals

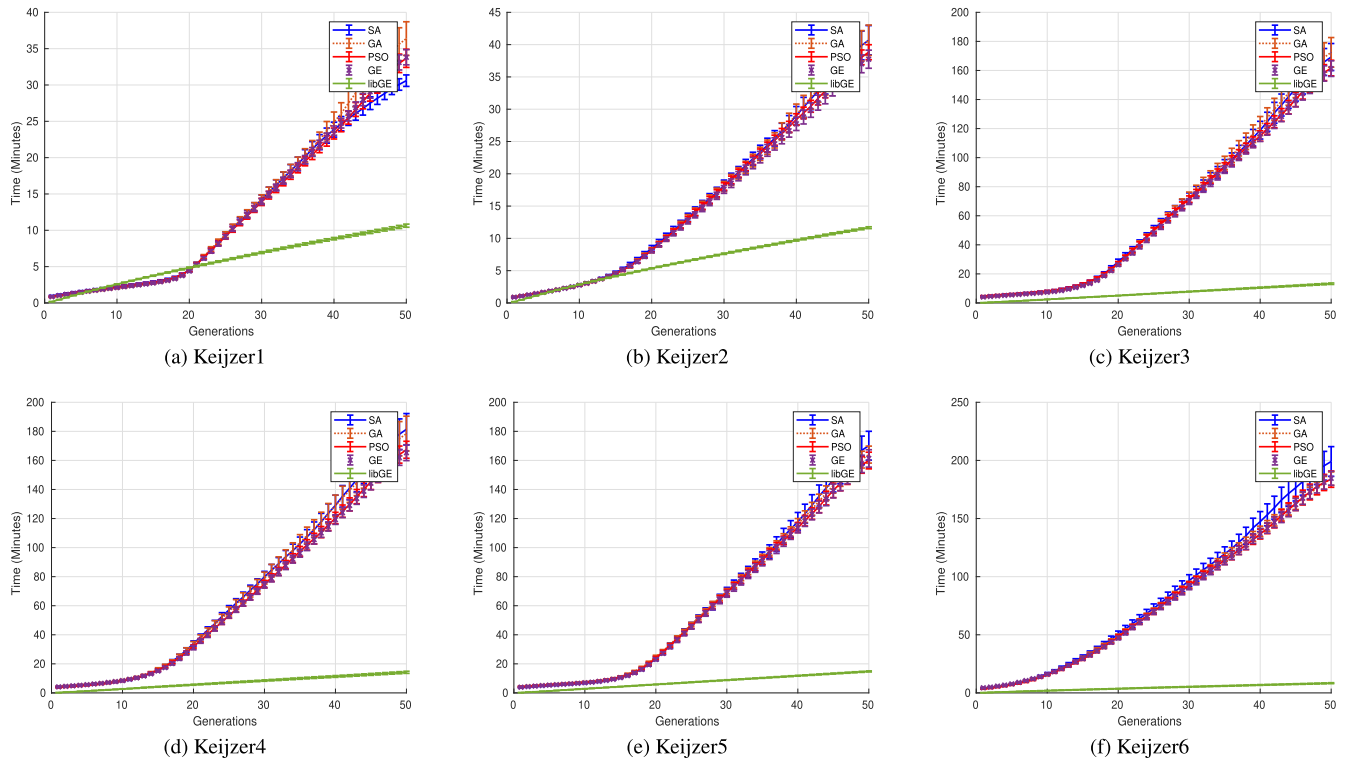


FIGURE 5. Mean cumulative time for the completion of each evolutionary run.

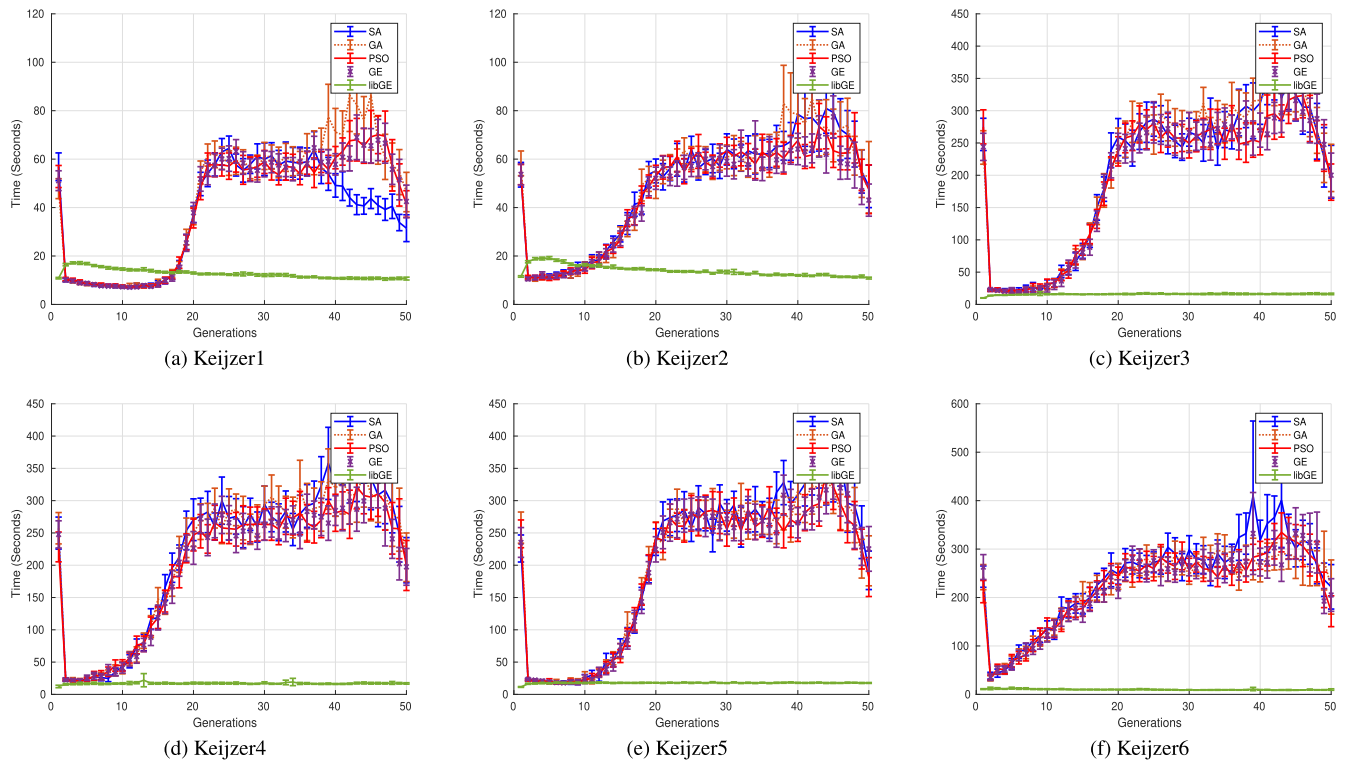


FIGURE 6. The average time taken by the algorithm to evolve a generation of GE programs.

has a better capability of generating a variety of offspring. In EA, a higher population diversity means the algorithm has a better ability to explore the search space. A diverse

population increases the possibility of finding the globally optimal solution without the population getting stuck at some locally optimal point. This section gives some analysis of

diversity using GELAB. We also observe the effect of hybrid optimization on diversity.

Our experimental setup for this analysis was similar to the ones reported above. In particular, we tried three flavors of GE hybridized with either SA, GA, or PSO. Moreover, there was one experiment per polynomial in which bare GE was used.

A. RESULTS

Figure 7 shows the error bars associated with diversity in terms of the standard deviation of the fitness of the best individuals over the whole population. The figure is a retrospective viewpoint as the variation in diversity is shown across all the generations of a run. It can be noted that although the system maintains high diversity in the beginning, diversity abruptly decreases as evolution proceeds. Diversity becomes steady after the first few generations' rapid fall. This same pattern is observed in all experiments, so rather than showing individual graphs for all of them, we instead show just a single graph for benchmark Keijzer1 in Figure 7a. Diversity in Figure 7b is reported in terms of the proportion of dissimilar individuals present in the population during evolution at any given evolutionary step. Dissimilarity is computed in terms of the uniqueness of fitness. This is to say that two individuals are considered dissimilar if their fitness values differ from each other. To this end, to compute dissimilarity the number of individuals that have unique fitness values is counted. This number is divided by the size of the whole population to give the measure. Again, the obtained results on all *six* benchmarks are identical. Hence, we omit to present all of them. However, we present the result of Keijzer1 for reference purposes.

VIII. Crossover Operators

Much of the beauty and emergent behavior of an EA lies in the way chromosomes of pairs of individuals are recombined and crossed-over to form new offspring. It is mainly due to crossover that the offspring retain certain similarities with their parents. At the same time, crossover is also responsible for the difference in the genotype of the children with respect to their parents. GE being an EA cannot remain oblivious from crossover and its effects. Considerable literature has been written about crossover in GE. However, not much has been written about employing multiple crossover operators at the same time in a single GE system. In this section, we propose a novel scheme in which a GE system is provided with numerous crossover and recombination operators. The system can choose a suitable operator for recombining a pair of parent individuals to form new offspring. The choice is made by the system in a stochastic sense that depends on the probability of a certain operator to be chosen. The probability of each operator is updated at every generation during evolution. The update is performed by taking into account the utility of that operator in creating the child population. Moreover, the update also takes into account the fitness of the individuals. Our results are interesting in the sense that

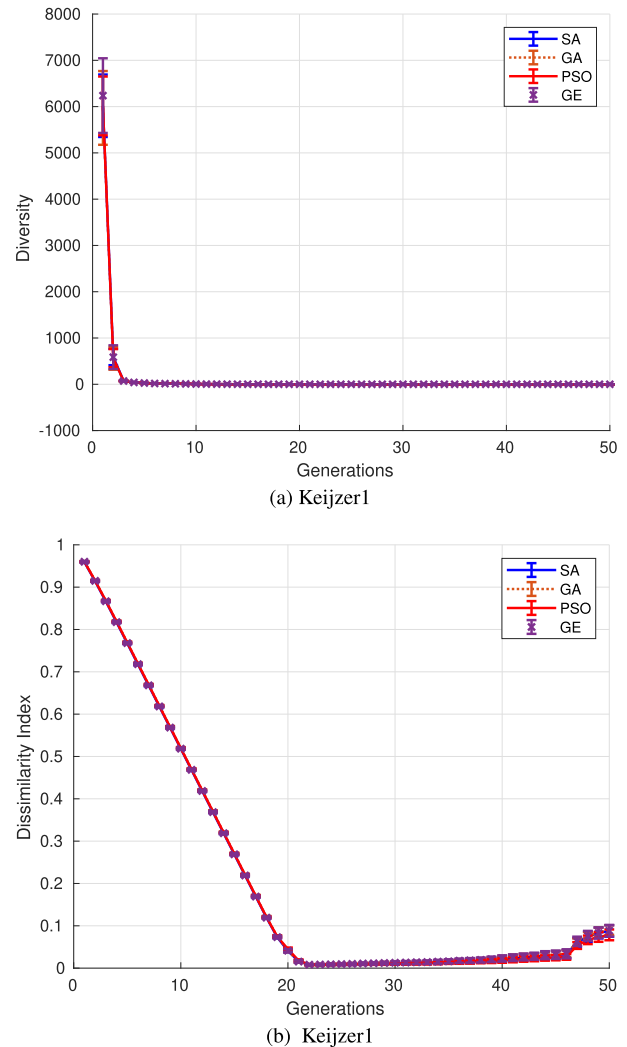


FIGURE 7. Diversity history, along with error bars in terms of the (a) proportion of dissimilar individuals in the whole population, (b) standard deviation of the best fitness over the whole population.

they tell us about which operators played a greater role in evolution, and were subsequently chosen by the GE system.

A. LITERATURE SURVEY OF CROSSOVER IN GE

Based on various theoretical and practical concepts, considerable work has been reported in the past to create newer techniques for adapting genetic operator probabilities. Schemes for adaptive crossover probabilities have been implemented in a wide range of EAs. For example, [27] proposed a method for adapting operator probabilities at run-time. A variant of their method is implemented in GPLAB, the tool we used to run our GP experiments.

Serpell [28] examined a method for adapting the choice of the operator during run-time along with various methods for adapting the rate at which the chosen operator is applied. Another work [29] describes a GA hybrid operator probability adaptation for minimum cost routing using an object-oriented representation of networks.

In a recent similar work [30], a multi-operator GA is used for finding a minimum cost spanning tree for a generalized Minimum Spanning Tree (MST) problem. In [31], the authors propose a novel scheme for adapting crossover and mutation probabilities in a GA using a clustering technique.

An adaptive probability of crossover and mutation in [32] is used to sustain the convergence capacity of a GA while maintaining the population. The proposed approach regulates the crossover and mutation probabilities in accordance with the fitness scores of the solutions. In [33], Harper *et al.* introduced a self-selecting crossover mechanism, while [34] introduced a statistical mechanism for updating crossover probabilities in GAs. Their system uses the statistical information in the alleles to use them in crossover. A comparison of an adaptive GA and a standard GA for the optimization of several multi-modal functions is presented in [32].

B. TYPES OF CROSSOVER OPERATORS IMPLEMENTED

In this section, we briefly describe the various crossover operators that we implemented in GELAB.

1) SINGLE POINT CROSSOVER

This is possibly the most primitive crossover operator that is used in EAs. In this crossover scheme, two genomes are chosen initially through an appropriate selection mechanism. After that, a single integral index that is lower than the lengths of both the genomes is randomly generated. The segments of both genomes beyond this index are swapped to create two new offspring individuals.

2) VARIABLE POINT CROSSOVER

In this scheme, two integer indices are randomly generated for each of the two genomes. The numbers are bound to be lower than the lengths of each of the respective genomes. Genome segments beyond these points are swapped to create two new offspring individuals.

3) LHS REPLACEMENT CROSSOVER

This is an implementation of the LHS Replacement Crossover, also known as subtree crossover, as described in [35]. In this, two subtrees of the same type are chosen randomly on both of the parents. Genome segments corresponding with these subtrees are eventually swapped. This is a structure-preserving crossover operator and it is known to perform crossover in GE without disruption.

4) WEAVE OPERATOR

We also introduce a novel recombination operator that uses a linear combination of the integral genome. In this, we initially take the genomes of two selected individuals. If the lengths of both the genomes are unequal, they are made equal first. This is done by piggybacking the smaller genome with genes containing *zeros*. After that, the first child genome is created by adding the two-parent genomes. Next, the second child genome is created by subtracting the values of genes of one genome from the respective genes of the other. The absolute

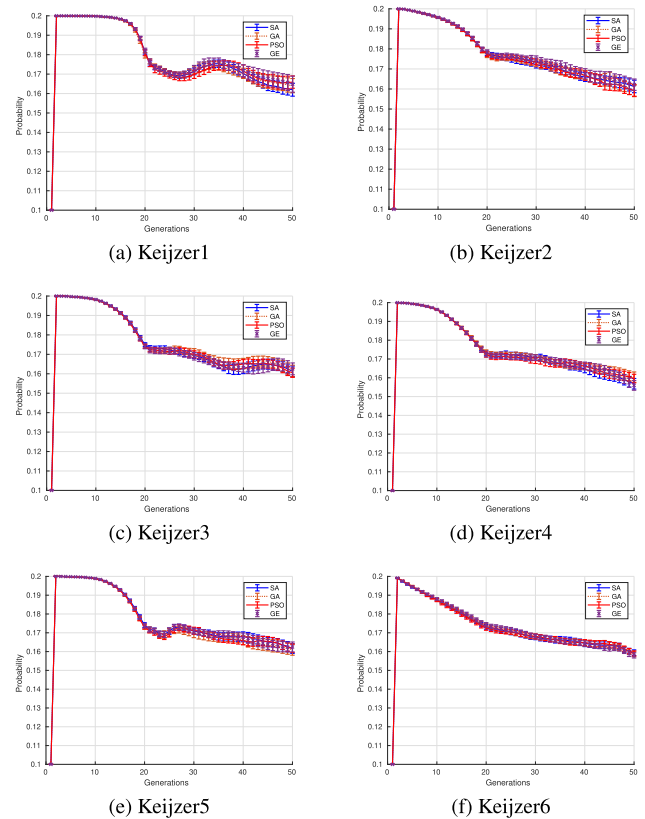


FIGURE 8. Generation-by-generation variation of single-point crossover.

values of the result are used as the genome for the second child.

5) TIGHT WEAVE OPERATOR

We also investigate a slight variant on the weave operator. This is another novel operator that is similar to the weave operator as described above. The only difference is that the first child individual is created by adding to the values of the genes of the first parent's genome randomly perturbed values of the genes of the second parent's genome. Similarly, the second individual is created by adding to the genes of the second parent randomly perturbed values of the genes of the first parent's genome. We call this the *tight weave operator*.

6) NO CROSSOVER

As the name suggests, no crossover is performed by this operator. To this end, two individuals are chosen through a selection mechanism and they are used to create two offspring individuals as such i.e. without any alteration.

C. RESULTS

The experimental setup for this study was the same as those reported in the previous sections. The experimental results are shown in Figures 8– 13. Figure 8 shows the variation of single point crossover probability with error bars across 50 generations of the experiment. The default value for

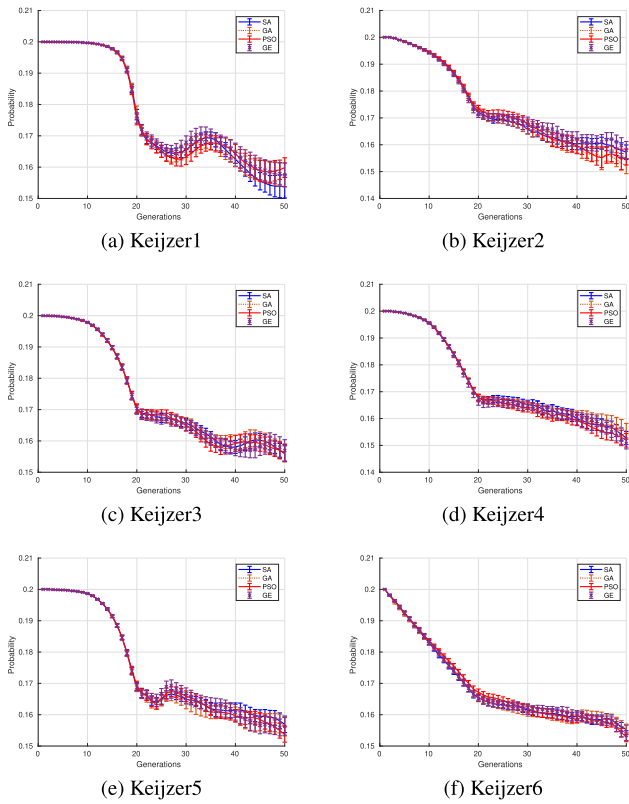


FIGURE 9. Generation-by-generation variation of variable-point crossover.

TABLE 3. Default probabilities of various crossover and recombination operators.

Operator	Probability
Single Point	0.1
Variable Point	0.2
LHS Replacement	0.2
Weave	0.2
Tight Weave	0.2
No Xover	0.1

single point crossover probability used in this experiment is 0.1, which is used while initiating the very first generation of the evolution. Followed by a steep rise, the crossover probability is steady across the first few generations, but varies with further evolution. It is worth mentioning that the default crossover probability used in this experiment differs for different crossover operators, as mentioned in Table 3.

The result in Figure 8f is significantly different than that of other benchmarks where the probability declines in a linear fashion at the beginning of the evolutionary runs, but follows a trend similar to the other experiments as evolution proceeds.

Figure 9 shows the error bar associated with the variable-point crossover parameters against 50 generations of evolution. Initially, the crossover probability is steady for the first few generations (Figure 9a-9e) of the evolution, and drops as the evolution proceeds, except in Figure 9f where

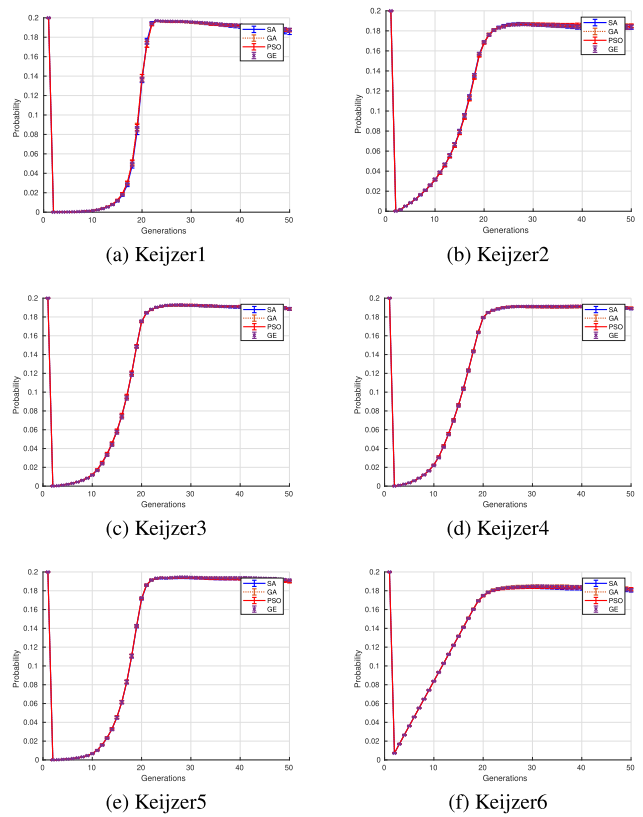


FIGURE 10. Generation-by-generation variation of subtree crossover.

the variable-point crossover probability observes a rapid fall in the first few generations. Note that the default crossover probability used in this experiment is 0.2.

Figure 10 shows the probability of the LHS Replacement crossover operator, with error bar, across all the generations of a run. The findings are different than other experimental results but not surprising as we believe this to be due to the structure-preserving nature of the LHS replacement crossover operator. Although the probability of the subtree crossover operator falls to 0 or ≈ 0 from the default probability (0.2) after the very first generation of the evolution, it adopts a steady value. The maximum probability of the LHS Replacement crossover is maintained in such a way that it does not exceed the default probability of crossover (i.e. always remains ≤ 0.2 .)

The experimental results of the proposed weave operator are presented in Figure 11. The axis representations are similar to what has been used in the previous results. Unlike the variation observed for crossover operators used in the previous experiments, the variation of the weave operator is surprisingly different. The drop in the crossover operator is not as much for the first few generations of the evolution and rises back to approximately default probability with further evolution.

A similar pattern can be observed for the first few generations of experiments using the tight weave operator in Figure 12. The rise in the crossover probability in the latter

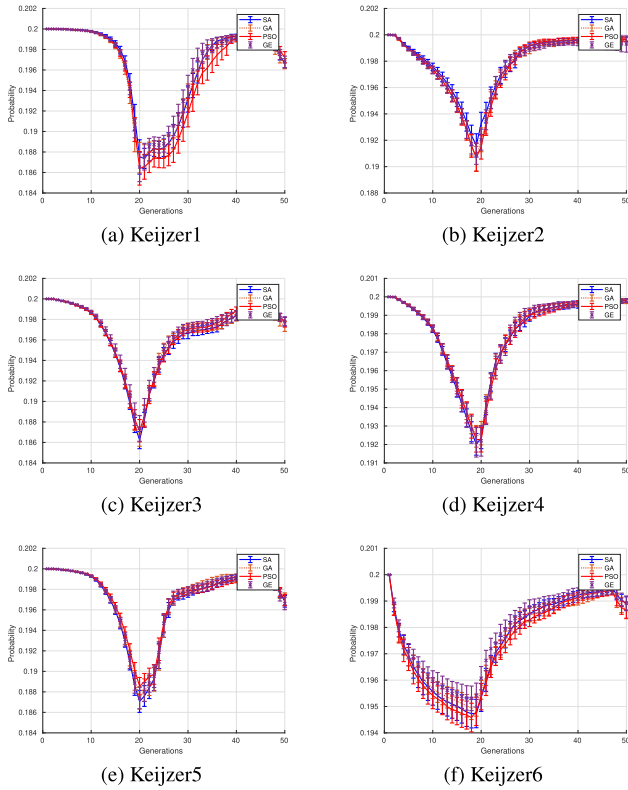


FIGURE 11. Generation-by-generation variation of weave operator.

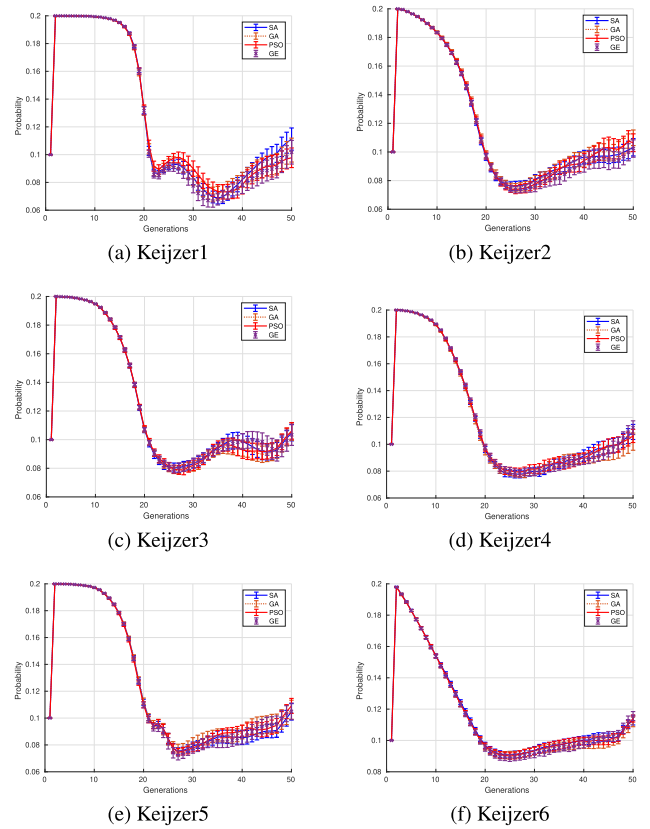


FIGURE 13. Generation-by-generation variation of no crossover.

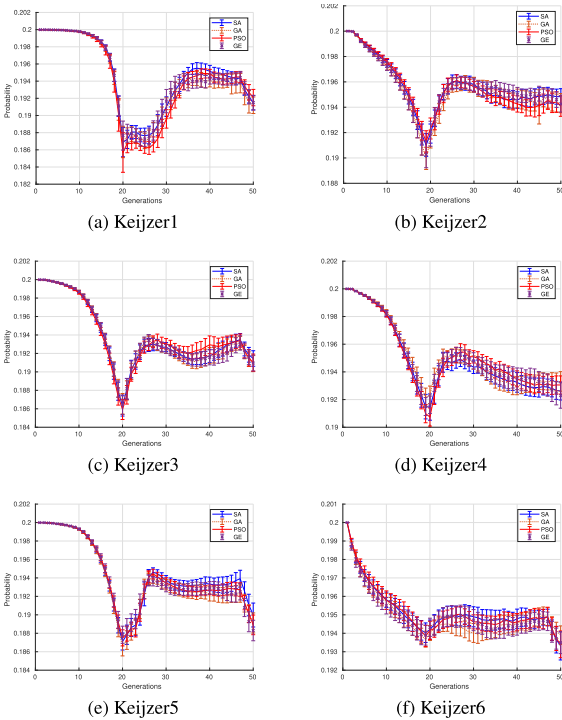


FIGURE 12. Generation-by-generation variation of tight weave operator.

phase is not as high as observed using the weave operator in Figure 11. A generation-wise variation of the No Crossover is shown in Figure 13.

D. ON ADAPTING OPERATOR PROBABILITIES

In the first generation, we ensure that all individuals in the initial population are unique. As an individual is created it is marked that it was created using no crossover and no mutation.

Before the creation of the initial population, the evolutionary system is passed a set of parameters that have initial probabilities of each of the aforementioned crossover and recombination operators. The default probabilities of each of the crossover operators are mentioned in Table 3, and for each, the mutation operator is 0.2. It is worth noting that all of the probability values sum up to “1” for each group of operators (i.e. they sum up to “1” for all the crossover operators and they sum up to “1” for all the mutation operators).

After this, the evolution proceeds, and the next generation is created using the default probabilities. From this point onward, once a new generation has been created, the probabilities for different operators are updated using either equation 6 or 7 depending on whether the fitness criterion is based on *lower is better* or *higher is better*.

$$P_{go_i} = \frac{1 - \frac{\sum_{j=1}^N fitness_{goij}}{F_p}}{\sum_{k=1}^M P_{k=1}^M} \quad (6)$$

$$P_{go_i} = \frac{\sum_{i=1}^N fitness_{goi}}{F_p} \quad (7)$$

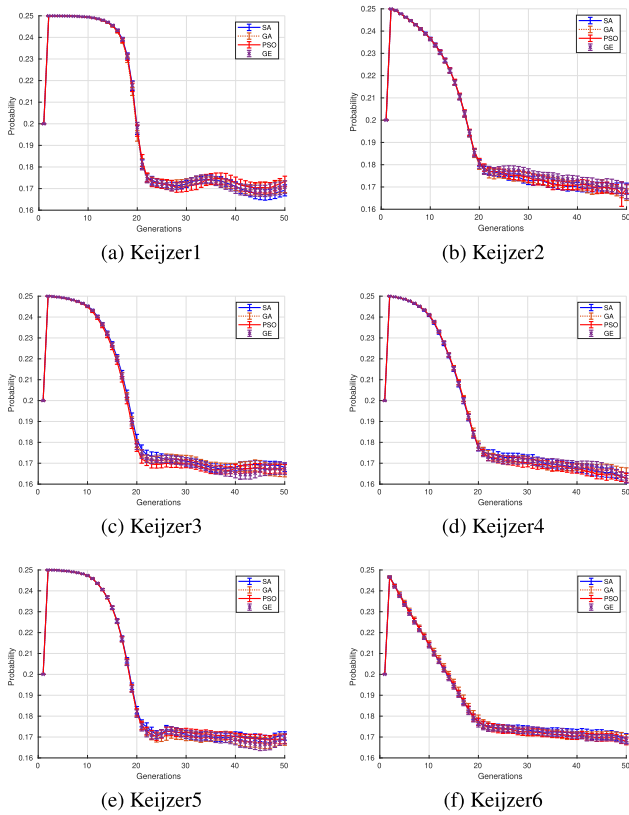


FIGURE 14. Generation by generation variation of point mutation.

where P_{go_i} is the probability of the i_{th} genetic operator (crossover or recombination). $fitness_{goij}$ is the fitness of the j_{th} individual of the population that was also created with the i_{th} genetic operator. F_p is the sum of fitness values of the whole population. P_k is the proportion given in equation 8 if *lower is better* or equation 9 otherwise.

$$P_k = 1 - \frac{\sum_{j=1}^N fitness_{goij}}{F_p} \quad (8)$$

$$P_k = \frac{\sum_{j=1}^N fitness_{goij}}{F_p} \quad (9)$$

IX. MUTATION OPERATORS

In this work, we chose four different mutation operators. Along with that, we also had the option of having no mutation once an individual has been chosen. In what follows, brief descriptions of the various mutation operators are given.

A. TYPES OF MUTATION OPERATORS

1) POINT MUTATION

This is a primitive mutation operator. In this, a random number is generated that lies within a certain range (ranging between 0 and 50 in our case). After this, a random index is chosen on the genome. The random number generated is added to the value of the gene at this index.

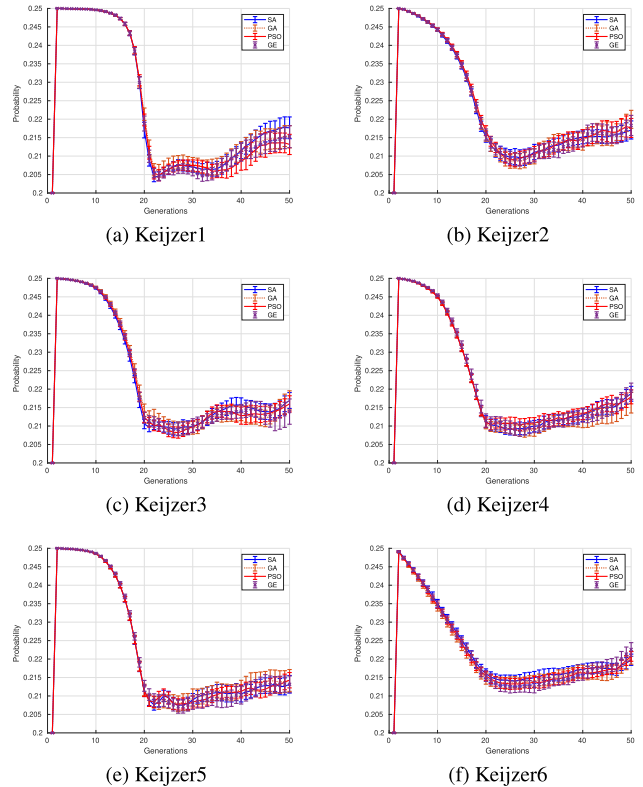


FIGURE 15. Generation by generation variation of few points mutation.

2) FEW POINTS MUTATION (FPM)

A few indices (points) of the genome are randomly chosen. The number of indices is also chosen randomly. The values of genes at these indices are mutated as follows.

Random numbers equal to the number of indices are generated within a certain range (between 0 and 50 in our case). These random numbers are added to the values of genes that were chosen for mutation. One random number is added per gene.

3) FIXED BOUNDS MUTATION (FBM)

In this scheme, random numbers equal to the length of the genome are generated within a certain range. The range here is again between 0 and 50. The newly generated perturbation vector is added to the respective values of the genes in the genome.

4) SUBTREE MUTATION

This involves an implementation of the LHS Replacement Crossover as described in [35]. In this scheme, a new individual is created using the initialization given in Table 2. After this, LHS replacement crossover is performed between the target individual to be mutated and the newly created individual. The child individual that receives the subtree from the newly created individual is chosen as the mutated individual. The other individual that is created as a result of LHS replacement crossover is littered.

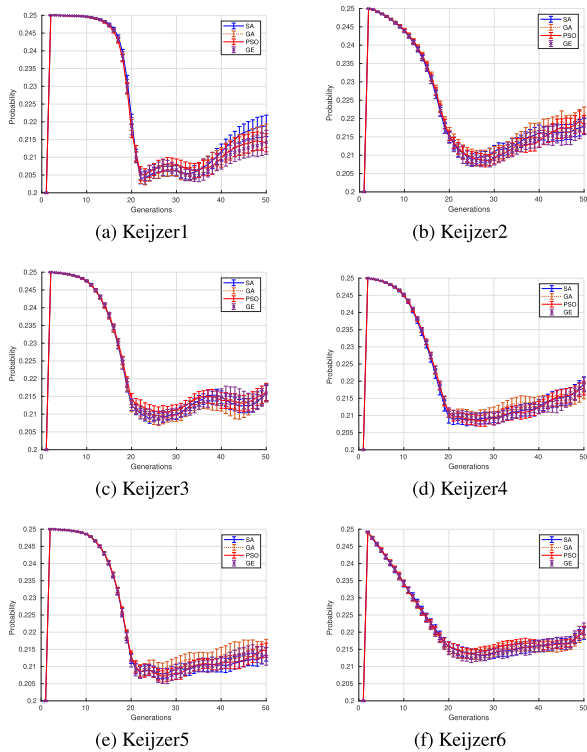


FIGURE 16. Generation by generation variation of fixed bounds mutation.

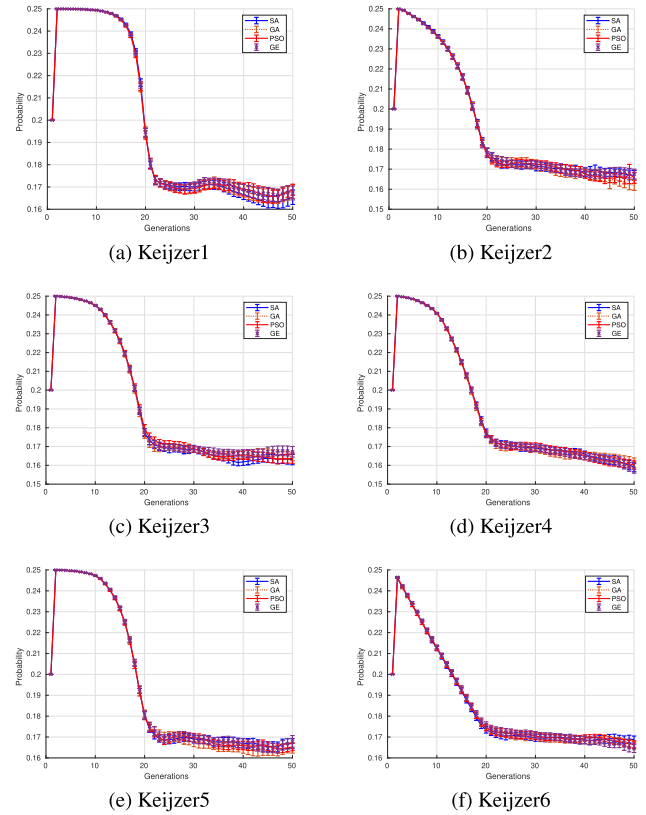


FIGURE 18. Generation by generation variation of no mutation.

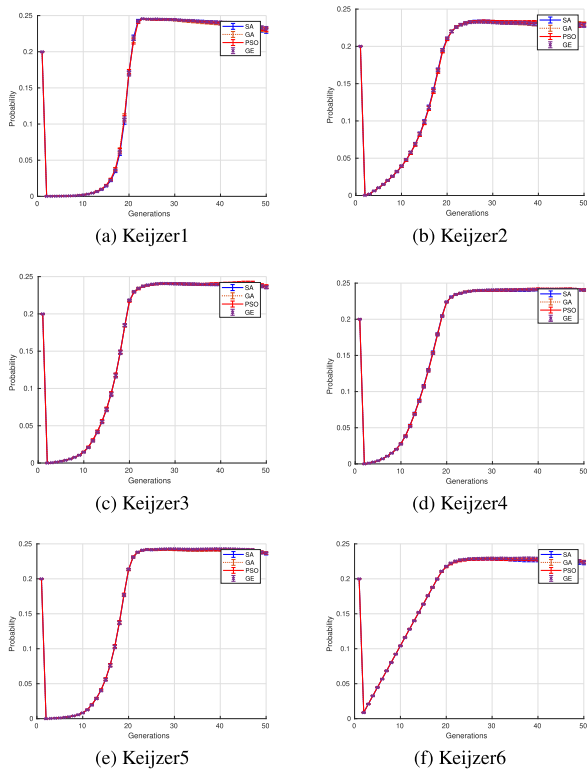


FIGURE 17. Generation by generation variation of subtree mutation.

5) NO MUTATION

As the name suggests, no mutation is performed by this operator. To this end, an individual is chosen first. It is passed as

such to the next generation. Each of these mutation operators had a default initial probability of 0.2.

B. RESULTS

The experimental setup in this study was the same as the one used in the previous experiments. Figure 14 shows the error bars associated with point mutation probability over 50 generations of evolution. A rapid increase in the operator’s probability is observed which gradually decreases for the first few generations of the evolution and becomes almost steady as evolution proceeds.

A similar pattern is observed in the results of experiments using FPM operator in Figure 15, and FBM operator in Figure 16. Unlike the case of *point mutation*, the mutation probability didn’t drop below the default value using *FPM* or *FBM* for all the *six* benchmark problems. Note that, in contrast to the experiments involving crossover operators where different probability was used for different crossover operators, an initial mutation probability of 0.2 is used for all of the five mutation operators.

Figure 17 shows the error bars associated with the subtree mutation probability throughout the 50 generations. Unlike the other mutation operators used in this paper, the subtree mutation operator is unique in terms of the variation of its probability. The mutation probability drops to 0 after the first generation and increases gradually as the evolution proceeds. Its variation has a different trend than the variation of the other

mutation operators used. Remember such a case from LHS replacement crossover experiment? We believe this is due to the crossover scheme involved in the subtree mutation.

A generation-wise probability of the *no mutation* operator is presented in Figure 18. Various flavors of GE performed similarly well with different mutation operators.

X. CONCLUSION

In this paper, we have proposed a novel toolbox for GE in MATLAB. We call it GELAB. It presents an easier way to use GE in research and development, and specifically in an MLOps context. It is far easier to use than libGE and numerous contemporary implementations of GE. The user only has the onus of specifying the data and setting a few parameters for which the algorithm should be run. The user is prompted at every stage before the algorithm starts running to configure various parameters or default values are used. Curious users can also alter the code and add their own modules to the code. Moreover, it is also easy to integrate it with third-party software. We have bench-marked GELAB for many months before committing the most recent stable release.

In this paper, we also presented a novel approach to hybridize GE with other meta-heuristic algorithms. In particular, SA, GA and PSO are used as different options for hybridizing. The effectiveness of the proposed toolbox and the hybrid optimization were tested on a set of benchmark problems and the results were encouraging. Moreover, it proved to be easy to understand and implement along with the flexibility of tweaking the code for effective executions of different problems. During the hybrid optimization, four well-known crossover operators were used individually, along with the two crossover operators newly proposed in this paper. In addition to that, we incorporated five different mutation operators that are popularly used in the literature to test the tool. Rigorous experiments involving various meta-heuristic algorithms, crossover, and mutation operators on a set of diverse benchmark problems performed similarly well; which indicates GELAB is a capable candidate to solve optimization problems.

In terms of the diverse variety of crossover and mutation operators and their probabilistic adaption, we would like to assert that we aim to take up this project quite aggressively in the future. In our view, this is an extremely important avenue in the design and improvement of EAs as it has been shown in the literature also. We aim to run GELAB on more challenging problems that test the limits of these evolutionary operators. We aspire to employ GELAB in a variety of problem domains and thoroughly benchmark the system.

REFERENCES

- [1] S. Alla and S. K. Adari, "What is MLOps?" in *Beginning MLOps With MLFlow: Deploy Models in AWS SageMaker, Google Cloud, and Microsoft Azure*. Berkeley, CA, USA: Apress, 2021, pp. 79–124, doi: [10.1007/978-1-4842-6549-9_3](https://doi.org/10.1007/978-1-4842-6549-9_3).
- [2] M. S. Anjum and C. Ryan, "Seeding grammars in grammatical evolution to improve search-based software testing," *Social Netw. Comput. Sci.*, vol. 2, no. 4, pp. 1–19, Jul. 2021.
- [3] M. K. Tetteh, D. M. Dias, and C. Ryan, "Evolution of complex combinational logic circuits using grammatical evolution with SystemVerilog," in *Genetic Programming*, T. Hu, N. Lourenço, and E. Medvet, Eds. Cham, Switzerland: Springer, 2021, pp. 146–161.
- [4] M. Ali, M. Kshirsagar, E. Naredo, and C. Ryan, "AutoGE: A tool for estimation of grammatical evolution models," in *Proc. 13th Int. Conf. Agents Artif. Intell.*, 2021, pp. 1274–1281.
- [5] M. O'Neill, A. Brabazon, C. Ryan, and J. J. Collins, "Evolving market index trading rules using grammatical evolution," in *Applications of Evolutionary Computing*, E. J. W. Boers, Ed. Berlin, Germany: Springer, 2001, pp. 343–352.
- [6] C. Ryan, J. J. Collins, and M. O. Neill, "Grammatical evolution: Evolving programs for an arbitrary language," in *Genetic Programming*, W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Fogarty, Eds. Berlin, Germany: Springer, 1998, pp. 83–96.
- [7] S. D. Bradley, *MATLAB*. Hoboken, NJ, USA: Wiley, Nov. 2017, pp. 1–3, doi: [10.1002/9781118901731.icrnm0136](https://doi.org/10.1002/9781118901731.icrnm0136).
- [8] R. Burbidge, J. H. Walker, and M. S. Wilson, "Grammatical evolution of a robot controller," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2009, pp. 357–362.
- [9] G. R. Harik, F. G. Lobo, and D. E. Goldberg, "The compact genetic algorithm," *IEEE Trans. Evol. Comput.*, vol. 3, no. 4, pp. 287–297, Nov. 1999.
- [10] E. Mininno, F. Cupertino, and D. Naso, "Real-valued compact genetic algorithms for embedded microcontroller optimization," *IEEE Trans. Evol. Comput.*, vol. 12, no. 2, pp. 203–219, Apr. 2008.
- [11] M. A. Raja and S. U. Rahman, "A tutorial on simulating unmanned aerial vehicles," in *Proc. Int. Multi-Topic Conf. (INMIC)*, Nov. 2017, pp. 1–6.
- [12] A. Raja and C. Flanagan, "Real-time, non-intrusive speech quality estimation: A signal-based model," in *Genetic Programming*, M. O'Neill, L. Vanneschi, S. Gustafson, A. I. E. Alcázar, I. De Falco, A. D. Cioppa, and E. Tarantino, Eds. Berlin, Germany: Springer, 2008, pp. 37–48.
- [13] M. A. Raja and C. Ryan, "GELAB—A MATLAB toolbox for grammatical evolution," in *Intelligent Data Engineering and Automated Learning—IDEAL 2018*, H. Yin, D. Camacho, P. Novais, and A. J. Tallón-Ballesteros, Eds. Cham, Switzerland: Springer, 2018, pp. 191–200.
- [14] A. Murphy, A. Youssef, K. K. Gupt, M. A. Raja, and C. Ryan, "Time is on the side of grammatical evolution," in *Proc. Int. Conf. Comput. Commun. Inform. (ICCCI)*, Jan. 2021, pp. 1–7.
- [15] A. Youssef, K. K. Gupt, M. A. Raja, A. Murphy, and C. Ryan, "Evolutionary computing based analysis of diversity in grammatical evolution," in *Proc. Int. Conf. Artif. Intell. Smart Syst. (ICAIS)*, Mar. 2021, pp. 1688–1693.
- [16] M. A. Raja, A. Murphy, and C. Ryan, "GELAB and hybrid optimization using grammatical evolution," in *Intelligent Data Engineering and Automated Learning—IDEAL 2020*, C. Analide, P. Novais, D. Camacho, and H. Yin, Eds. Cham, Switzerland: Springer, 2020, pp. 292–303.
- [17] J. M. Colmenar, J. I. Hidalgo, and S. Salcedo-Sanz, "Automatic generation of models for energy demand estimation using grammatical evolution," *Energy*, vol. 164, pp. 183–193, Dec. 2018.
- [18] I. Tsoulos, D. Gavrilis, and E. Glavas, "Neural network construction and training using grammatical evolution," *Neurocomputing*, vol. 72, nos. 1–3, pp. 269–277, Dec. 2008.
- [19] L. M. L. de Campos, R. C. L. de Oliveira, and M. Roisenberg, "Optimization of neural networks through grammatical evolution and a genetic algorithm," *Expert Syst. Appl.*, vol. 56, pp. 368–384, Sep. 2016.
- [20] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.
- [21] A. Topchy and W. F. Punch, "Faster genetic programming based on local gradient search of numeric leaf values," in *Proc. Genetic Evol. Comput. Conf. (GECCO)*, vol. 155162. San Mateo, CA, USA: Morgan Kaufmann, 2001, pp. 1–8.
- [22] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw. (ICNN)*, vol. 4, Nov./Dec. 1995, pp. 1942–1948.
- [23] R. M. A. Azad and C. Ryan, "Comparing methods to creating constants in grammatical evolution," in *Handbook of Grammatical Evolution*. Cham, Switzerland: Springer, 2018, pp. 245–262.
- [24] S. Silva and J. Almeida, "GPLAB—A genetic programming toolbox for MATLAB," in *Proc. Nordic MATLAB Conf.*, 2003, pp. 273–278.
- [25] S. Luke and L. Panait, "Lexicographic parsimony pressure," in *Proc. 4th Annu. Conf. Genetic Evol. Comput.* San Francisco, CA, USA: Morgan Kaufmann, 2002, pp. 829–836.
- [26] C. Ryan and R. M. A. Azad, "Sensible initialisation in grammatical evolution," in *Proc. Bird Feather Workshops, Genetic Evol. Comput. Conf.*, A. M. Barry, Ed. Chigaco, IL, USA: AAAI, Jul. 2003, pp. 142–145.

- [27] L. Davis, "Adapting operator probabilities in genetic algorithms," in *Proc. 3rd Int. Conf. Genetic Algorithms*, 1989, pp. 61–69.
- [28] M. Serpell and J. E. Smith, "Self-adaptation of mutation operator and probability for permutation representations in genetic algorithms," *Evol. Comput.*, vol. 18, no. 3, pp. 491–514, Sep. 2010.
- [29] M. C. Sinclair, "Operator-probability adaptation in a genetic-algorithm/heuristic hybrid for optical network wavelength allocation," in *Proc. IEEE Int. Conf. Evol. Comput., IEEE World Congr. Comput. Intell.*, May 1998, pp. 840–845.
- [30] C. Contreras-Bolton, G. Gatica, C. R. Barra, and V. Parada, "A multi-operator genetic algorithm for the generalized minimum spanning tree problem," *Expert Syst. Appl.*, vol. 50, pp. 1–8, May 2016.
- [31] J. Zhang, H. S.-H. Chung, and W.-L. Lo, "Clustering-based adaptive crossover and mutation probabilities for genetic algorithms," *IEEE Trans. Evol. Comput.*, vol. 11, no. 3, pp. 326–335, Jun. 2007.
- [32] M. Srinivas and L. M. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, no. 4, pp. 656–667, Apr. 1994.
- [33] R. Harper and A. Blair, "A self-selecting crossover operator," in *Proc. IEEE Int. Conf. Evol. Comput.*, Jul. 2006, pp. 1420–1427.
- [34] S. Yang and B. Abbass, "Adaptive crossover in genetic algorithms using statistics mechanism," *Artif. Life*, vol. 8, pp. 182–185, Aug. 2002.
- [35] R. Harper and A. Blair, "A structure preserving crossover in grammatical evolution," in *Proc. IEEE Congr. Evol. Comput.*, vol. 3, Sep. 2005, pp. 2537–2544.



KRISHN KUMAR GUPT received the B.Sc. degree (Hons.) in electronics and the M.Sc. degree in informatics from the University of Delhi, India, in 2017 and 2019, respectively. He is currently pursuing the Ph.D. degree with the Department of Electrical and Electronic Engineering, Technological University of the Shannon: Midlands Midwest (TUS), Limerick, Ireland.

He is also working under the Science Foundation Ireland-funded project 'Automatic Design of Digital Circuits' (ADDC), based in the University of Limerick (UL) and the TUS, Ireland. His research interests include artificial intelligence, grammatical evolution, FPGA, circuit design, and test case optimization.

Mr. Gupt was awarded the Prestigious Research Fellowship (SRF) by the Indian Academy of Sciences (IASc), Bengaluru, India, in 2016 and 2018.



MUHAMMAD ADIL RAJA received the B.Eng. degree in metallurgical engineering and materials science from the University of Engineering and Technology, Lahore, Pakistan, in 2000, the M.S. degree in computer sciences from the Lahore University of Management Sciences (LUMS), Pakistan, and the Ph.D. degree in speech quality estimation from the University of Limerick, Ireland, in 2008.

After that, he worked as a Postdoctoral Researcher with Orange Labs, in 2010. He is currently working as a Postdoctoral Research Fellow with the Regulated Software Research Center (RSRC), Dundalk Institute of Technology, Ireland, where his core research responsibilities are to inform the group about opportunities for medical device software innovation using artificial intelligence (AI). Lately, he worked as a Postdoctoral Research Fellow with the BDS Group, Computer Science and Information Systems (CSIS) Department, UL. His research interests include theory and applications of ML and he contributed heavily to the development and implementation of GELAB.



AIDAN MURPHY received the bachelor's degree in theoretical physics from the Trinity College Dublin, the H.Dip. degree in statistics from the University College Dublin, and the Ph.D. degree in explainable Artificial Intelligence (AI) (X-AI) from the BDS Laboratory, University of Limerick. He is currently a Postdoctoral Research Fellow with the Complex Software Laboratory, University College Dublin, researching software testing and mutation analysis. His research interests include grammatical evolution, transfer learning, fuzzy logic, and X-AI.



AYMAN YOUSSEF received the B.Sc. degree from the Department of Electronics and Communication, Cairo University, Egypt, the M.S. degree from the Department of Electronics and Engineering, Cairo University, and the Ph.D. degree in electronics and communication from AIN Shams University. He is currently a Postdoctoral Researcher with the BDS Group, University of Limerick, Ireland. He is also a Researcher/Assistant Professor with the Electronics Research Institute, Egypt. His research interests include ANNs, fuzzy logic, FPGA design, and photovoltaic systems.



CONOR RYAN (Senior Member, IEEE) was a Fulbright Scholar at the Massachusetts Institute of Technology, in 2013 and 2014, and was the CTO of NVMDurance, a company that optimized flash memory products, until it had a successful exit, in early 2018. He is currently a Professor of machine learning with the CSIS Department, UL, Ireland, a Funded Investigator within Lero (the Irish Software Research Centre) and a Science Foundation of Ireland Principal Investigator. He is also the Director of the Bio-Computing and Developmental Systems (BDS) Research Group, University of Limerick, Ireland. He is the inventor of grammatical evolution, one of the most widely used automatic programming systems.

...