FUDMA Journal of Sciences (FJS)
ISSN online: 2616-1370
ISSN print: 2645 - 2944
Vol. 8 No. 3, June, (Special Issue) 2024, pp 402 - 408
DOI: https://doi.org/10.33003/fjs-2024-0803-2463

DEFINING VARIATION OPERATOR FOR GRAMMAR REACHABILITY SEARCH BASED VULNERABILITIES DETECTION

Umar Kabir

Department of Software Engineering, Faculty of Computing, Bayero University Kano.

*Corresponding authors' email: ukabir.se@buk.edu.ng

ABSTRACT

In population-based search algorithm such as Evolutionary Programming (EP), the search process typically involves seeding population of first generation with randomly generated individuals, selecting parents through fitness evaluation, producing offsprings through variation of parents, and selecting parents and offsprings into next generation of candidate solutions. Obviously, the quality of the variation operator is important in leading the search process towards global optimal solution. In this paper, a high-quality variation operator is proposed. The proposed variation operator has the capacity to bias search towards optimal solutions while ensuring adequate balance between exploration and exploitation of the search space so as to facilitate discovery of optimal solutions in fewer number of generations. The proposed variation operator was used in our published work named EPSQLiFix. The proposed variation operator demonstrated high performance. Thus, it can as well be applicable in other related problem domains.

Keywords: Variation Operator, Search Process, Vulnerabilities Detection, Grammar Reachability, Evolutionary Programming

INTRODUCTION

Over the years, search algorithms have been widely used for finding optimal solutions to class of problems for which finding exact solution is not feasible (Hidalgo-Herrero, 2013). These kinds of problems are termed as NP-hard problems or simply called search problems.

There are local search algorithms which look for candidate solutions within the neighborhood of a given individual solution. Notable weakness of local search algorithms is that they can only find a solution which is the best among its neighborhood but not necessarily the best within the entire search space. Thus, a local search algorithm may be trapped within local optima (Alhijawi & Awajan, 2024). Local search algorithms are usually individual based algorithm such as Greedy Search, Tarbu Search, Ant Colony, and so on.

On the other-hand, there are global search algorithm which has the capacity of looking for solution within the entire search space. Thus, a global search algorithm has the potential for finding solution which is the best possible solution in the entire search space. Global search algorithms are mostly population-based algorithms such as Genetic Algorithm, Evolutionary Programming, Genetic Programming, Particle Swarm, etc. (Alhijawi & Awajan, 2024; Obunadike et al.,2018; Emmanuel et al., 2022).

For both local and global search algorithms, the search process involves variation of parents to produce offsprings. In some algorithms, beside variation operation, cross-over operation may also be performed in the process of producing offsprings from parents. Obviously, the success of search algorithm is directly influenced by the quality of the applicable variation operator that is applied to generate modified copy of an individual (parent) in order to produce its offspring.

In the literature, a number of variation operators were proposed, each for specified task and problem domain. However, a common quality factor of variation operator is its capacity for increasing elitism of search process by ensuring adequate balance between exploitation and exploration of the search space.

This paper proposed variation operator that performs four variation operations on individual so as to produce its offspring. In our research work, detection of SQL injection vulnerabilities for web application was modelled as Grammar Reachability search problem in which individual candidate solution is a sequence of grammar production rules that were derived from the web page being analyzed. Details of our problem formulation was reported in (Umar et al., 2018a).

The proposed variation operator ensures adequacy of search space exploitation and exploration through randomization and facilitates early discovery of optimal solutions through biasness. The proposed variation operator has been used in our earlier work EPSQLiFix, (Umar et al., 2018b), and it's performance was remarkable. Thus, the proposed variation operator has potential applicability in related problem domains.

The remaining of this paper is organized as follows. Review of related literature is presented in the next section, followed by presentation of Detection of SQL Injection Vulnerabilities as Grammar Reachability Search Problem. This is followed by presentation of the proposed variation operator, the Section present definition and algorithm of the proposed variation operator, as well as illustrative example of its applicability. Lastly, the paper conclusion is presented.

In the literature, the terms variation operation and mutation operation are used interchangeably. There is no single variation or mutation operator that fits all, and there is no defined procedure for formulating variation or mutation operators. This section presents some research work which proposed variation/mutation operators.

Mutation testing involves changing a program in minor ways by applying mutation operations, which results in modified versions of the program (Wang et al., 2022). Crossover operators and mutation operators play a very important role in the development of an efficient search algorithm (Kumar et al., 2021). Search base mutation testing was proposed by (Uzunbayir & Kurtel, 2024). They presented novel approach which combines genetic algorithms and ant colony optimization to reduce test cases and enhance the effectiveness of the test suit for mutation testing. Mutation

testing emerges as an invaluable method for evaluating a test suite's fault detection capability. Their approach uses mutation operators to introduce minor changes to the code, to optimize existing test suites, and to improve mutant detection with fewer test cases, thus improving the overall testing quality.

In the work of (Li et al., 2023) Many mutation operators have been designed to address a variety of challenging optimization issues. They proposed reinforcement learning based operator selection strategy for improving exploration and exploitation of the search space. They used the reinforcement learning to control state transition, as well as to identify the appropriate operator for each parent that maximizes its cumulative improvement. Their method divides the state of population search into four categories and selects the optimal operator that varies with the change in population state. Al-Tashi et al., (2023) proposed integrated mutation operator that adds more informative features which can assist in enhancing classification accuracy. In their work, the continuous search space was converted into binary space using the sigmoid function, and they used wrapper-based Artificial Neural Network (ANN) to evaluate the classification performance of the selected feature subset.

It is interesting to mention that empirical evidences (Arcuri, 2008, 2011; Ackling et al., 2011) have shown that biasing of variation operations is very effective in improving effectiveness of search process for software repairs. Moreover, it has been shown empirically that not all variations result in valid programs (Arcuri, 2008; Ackling et al., 2011). This is because variation operations may result in syntax violation, logical incorrectness, or functional degradation (Arcuri, 2008, 2011; Dominguez-Jimenez et al., 2011). Moreover, in the literature, different techniques have been used to bias variation to appropriate part of candidate solution. For instance, Arcuri (2008, 2011) proposed a novel variation operator that utilizes information extracted by Tarantula (Ackling et al., 2011; Jones & Harrold, 2005) fault localization tool to bias variation to most suspicious location among n randomly selected nodes of candidate solution. Ackling et al. (2011) proposed technique that combines the above Tarantula approach and lookup tables for biasing variation to most suspicious location as well as restricting variation to only eligible nodes. Their approaches facilitate evolvement of better fitness candidate solutions.

Detection of SQL Injection Vulnerabilities as Grammar Reachability Search Problem

This section highlights the reformulation of SQL injection vulnerabilities detection as grammar reachability search problem, details of which was reported in our earlier work titled Formulation of SQL Injection Vulnerability Detection as Grammar Reachability Problem (Umar, 2018a). The section begins by presenting source code of hypothetical webpage example shown in Fig. 1. The hypothetical webpage is used as running example in the remaining sections of this paper. The example webpage performs basic user authentication in a Java web application. As shown in the figure, there are two data input fields namely "username" and "password". The data input field "username" is validated at line 13 using the function toSQL(N), whereas the data input field "password" is not validated at all. These two data input fields are subsequently used in generation of dynamic query string at line 14, and eventual dynamic execution of the query at line 17. Consequently, lack of validation of "password" makes the hypothetical webpage vulnerable to SQL injection.

```
1   protected void doPost(HttpServletRequest request,
2   HttpServletResponse response) {
3
4   String N ="";
5
6   String Q = null;
7   String R = null;
8   java.sql.Statement stat = null;
9   stmt = conn.createStatement();
10  java.sql.ResultSet S = null;
11  N = request.getParameter("username");
12  String P = request.getParameter("userpass");
13  R = toSQL(N); // validation for N (i.e. Username)
14  Q = "select * from userstbl where uname='" + R + "' AND
15  passwd ='" + P + "'";
16
17  S = Stmt.executeQuery(Q);
18   // exec qry at sensitive sink
19  }
```

Figure 1: Source code of Example Vulnerable Webpage (source: Umar, 2018a)

In addition to the above running example, three important terminologies are repeatedly used in our presentation. The first terminology is ***Application's Entry Point (AEP)***, which is a program statement at which input data gets into web application. In a source code, an AEP is identified by presence of data input function such as request.getParameter("username") in line 11 of fig. 1. The second terminology is ***Sensitive Sink (SS),*** which is a program statement at which dynamic query is executed (Medeiros et al., 2014; Halfond & Orso, 2005; Yan et al., 2013). In a source code, an SS is identified by presence of query execution command such as Stmt.executeQuery(Q) in line 17 of fig. 1. Finally, the third terminology is ***Data Validation Statement,*** which is a program statement that performs validation of input data. For example, the statement R = toSQL(N) in line 13 is a data validation statement which uses the data validation function toSQL(N) to secure the username input data.

The key idea behind our formulation of SQL injection vulnerabilities detection as grammar reachability search problem is that, where data flow path can be established from a given AEP (such as "password" in line 12 of Fig. 1.) and ends in an SS (such as line 17 of Fig. 1.), if data validation is performed along the path, then the associated AEP (such as

"username" in lines 11 of Fig. 1.) is said to be validated and secure, otherwise, if data validation is NOT performed along the path, then the corresponding AEP (such as "password" in line 12 of Fig. 1.) is vulnerable to SQL injection.

The AEP-to-SS path can be represented as a sequence of source code line numbers that shows corresponding data flow across program statements for example L11, L13, L14, L17 depicts data flow from AEP parameter "username" to dynamic query execution at SS in Line 17

Our strategy for vulnerabilities detection is to extract grammar production rules from declaration and assignment statements of the webpage, such that the nonterminal symbol of the LHS (Left Hand Side) of each grammar production rule represents the variable of the LHS of the corresponding statement from which the grammar rule is extracted. The extracted grammar rules are converted to context free grammar (CFG) production rules. Then, the CFG rules are used to test reachability from nonterminal symbol representing AEP statement to nonterminal symbol representing SS statement. However, in the extracted grammar, the rule extracted from a SS statement is considered as the start rule for tracking reachability. The Format of Extracted Grammar Production Rules is shown in table 1. Furthermore, corresponding CFG production rules extracted from source code of Fig.1. are shown in Fig.2.

Table 1: Format of Extracted Grammar Production Rules

Example	Description
$X_p \rightarrow X_q$	Unit rule, nonterminal produce single nonterminal
$X_p \rightarrow \alpha X_q \beta \quad where \ \alpha, \beta \in (\Sigma \cup N)^*$	Nonterminal produce combinations of terminals and nonterminals
$Y_{pa} \rightarrow \gamma \ \ where \ \gamma \in \Sigma^*$	Nonterminal produce sequence of terminal symbols
$X_{pb} \rightarrow \alpha$ **fun_name**$(X_q)\beta$	Nonterminal symbol produce function with one argument, e.g. **executeQuery**(X_p), **toSQL**(X_p)

(source: Umar, 2018a)

Note that, in the table, X_{ab} and Y_{ac} denotes nonterminal symbols, the subscript $a =$ p, q, r are line numbers of statements from which grammar rule is extracted, subscript $b =$ letter S or A or V and subscript $c = letter\ a$. Moreover, **fun_name** represents function names, denoting sensitive sink functions, AEP functions, and data validation functions.

$X_{17S} \rightarrow$ **executeQuery**(X_{14})
$X_{14} \rightarrow Y_{14a}\ X_{13}\ Y_{14b}\ X_{12A}\ Y_{14c}$
$Y_{14a} \rightarrow$ select * from userstbl where uname=' +
$Y_{14b} \rightarrow +$ "' AND passwd = '" +
$Y_{14c} \rightarrow +$ "'"
$X_{13V} \rightarrow$ **toSQL** (X_{11A})
$X_{11A} \rightarrow$ **request.getParameter**("username")
$X_{12A} \rightarrow$ **request.getParameter**("userpass")

Figure 2: CFG rules extracted from the running example (source: Umar, 2018a)

The extracted CFG rules are analyzed for establishing possible flow paths from AEPs to SSs by testing for grammar reachability. Where such reachability is found, we analyze the associated "reachability productions sequence" for detection of SQLIV. If data validation is found along productions sequence, then the associated AEP parameter is secured and not vulnerable, otherwise, absence of data validation along such productions sequence means that the associated AEP parameter is not secured, and thus, SQLIV is found.

By reformulating detection of SQL injection vulnerabilities as grammar reachability search problem using evolutionary programming (EP) algorithm, individual candidate solutions are formed as sequence of CFG production rules. Figure 3 show some candidate solutions for the running example in Fig.1. The search process evolves candidate solutions by application of the proposed variation operator and fitness evaluation through generations until optimal solutions are found. Details of the proposed variation operator is presented in the next section.

i. X_{14}, X_{12A}
ii. X_{14}
iii. $X_{17S}, X_{14}, X_{13V}, X_{11A}$

Figure 3: Example of Candidates for the running example web application (source: Umar, 2018a)

Proposed Variation Operator

The grammar reachability search process using evolutionary programming (EP) algorithm begins with seeding population of first generation with candidates, each consisting of a single randomly selected CFG production rule. The candidates grow along the search process through application of variation operations and fitness evaluation. In this section, we present the proposed variation operator that performs four variation

operations to produce offsprings. For instance, the variation operation could be "to append" a grammar production rule to a candidate or "to replace" an existing grammar production rule within a candidate.

Given a set of variation operations, and a pool of all production rules extracted from webpage WP containing SS and some AEPs, it is important that there exists a sequence of variation operations that can select set of rules which establish SS-to-AEPs reachability. If such sequence of operations exists, then it would be possible to find all reachability derivations for detection of SQLIVs in a given web page. However, the fact that such reachability exist does not mean that it is easy to establish it.

For instance, the variation operations must be performed with caution so as to avoid destructive operations. Obviously, not all forms of variations are constructive or even feasible, for example, replacement of appropriate production rule with an inappropriate rule within a candidate would only lead to less fit candidate. Moreover, since the population of first generation is seeded with candidates formed from single randomly selected production rules, it is very obvious that cross-over between two of such candidates could only be very destructive. This is because cross-over would result in exchanging different portions of grammar rules from the two candidates involved. The result would naturally be a candidate containing distorted and/or meaningless production rule. Consequently, our grammar reachability search approach excludes cross-over from applicable variation operations. Considering the above caution and the kind of task at hand, i.e., "searching for productions sequences that establish SSs-to-AEPs reachability", the following four variation operations are feasible: -

i. Append Head: this operation appends a production rule to the head, i.e., begin of a candidate. The newly added rule becomes the start rule in productions sequence.
ii. Append Tail: this operation appends a production rule to the tail, i.e., end of a candidate. The newly added rule becomes the last rule in productions sequence.
iii. Replace Head: this operation replaces the current start rule in a candidate with another production rule. The newly added rule becomes the start rule in productions sequence.
iv. Replace Tail: this operation replaces the current last rule in a candidate with another production rule. The newly added rule becomes the last rule in productions sequence.

The above four variation operations are meant to lead the search process to optimal solutions. However, in the literature of search-based software testing other types of variation operations such as delete, and insert are commonly applicable during search process, nevertheless, they may not be helpful in our task.

It is necessary to mention that there are some scenarios where some of the above listed four operations could be very destructive to the search process. For example, the "Append head" operation should not append a grammar rule when the current head rule already contains sensitive sink function. Also, both "Replace head" and "Replace tail" operations should avoid replacing an appropriate production rule with an inappropriate one within a given candidate. Handling these challenges require systematic application of variations in a very novel ways that improves the performance of the search process. Consequently, the proposed novel variation operator takes into account the existing production rules in a candidate solution and the candidate's fitness information to systematically bias application of the variation operations. In order words, the proposed variation operator aims at improving elitism for the search process while maintaining balance between exploration and exploitation of the search space (Floudas & Pardalos, 2014). definition of the variation operator is presented in the next subsection

Definition

The variation operator chooses any of the four variation operations with equal probability of 0.25. Once an operation is randomly chosen, the novel operator bias application of the chosen operation in two ways. First, it uses the notion that "a program statement is more likely to pass data to (i.e. to interact with) other statements within its neighborhood" and consequently, bias application of variation operations to neighborhood of production rules. Second, the variation operator uses candidate's fitness information to bias application of eligible variation operation, so as to avoid destructive variations.

In the EP based grammar reachability search process, biasing search to neighborhood may not be as straight forward, because variation operations can affect only head rule or tail rule of a candidate. We call this rule as "anchor-rule". For append head and replace head operations, the anchor-rule is the current head rule, while for append tail and replace tail operations, the anchor-rule is the current tail rule.

Consequently, the proposed variation operator biases the search to the neighborhood of the anchor-rule of candidate as follows: First, q rules are randomly selected from the pool of production rules for possible consideration. Second, tournament is applied to the q rules for choosing the rule which is nearest to the anchor-rule. The chosen rule is considered as best neighbor. Finally, the operation is applied using the best neighbor rule. Thus, the proposed variation operator is defined as three tuples using Equ. 1.

$$\mathrm{S_{opr}} = \{\mathrm{Opr}, \mathrm{C}, \mathrm{tournamentOf(q)}\} \quad (1)$$

Where Opr is the randomly chosen variation operation, C is the candidate solution, and $tournamentOf(q)$ is a function that applies tournament selection on q rules and returns the best neighbor rule accordingly.

This technique allows the proposed variation operator to bias the search to the neighborhood. However, the application of the chosen operation using best neighbor rule needs to be further biased to eligible operations guided by candidate's fitness information so as to avoid destructive variations.

As discussed earlier, certain variation operations are not eligible in some scenarios. For example, "append head" is only eligible in a candidate whose current head rule is not sensitive sink rule X_{iS}. Also, the "append tail" is only eligible in a candidate whose current tail rule does not contain AEP function. Fortunately, the type of production rule in candidate's head and tail position can be determined from the candidate's "productions sequence completeness" fitness values given by Equ. 2 and 3 respectively. The two equations are shown below. Details on formulation of these fitness equations is reported in our work (Umar et al., 2018a)

$$f_{com}(X_1) = \begin{cases} 0 & X_1 \rightarrow ss\_fun(\gamma) \\ 5 & otherwise \end{cases} \quad (2)$$

$$f_{com}(X_k) = \begin{cases} 0 & X_k \rightarrow aep\_fun("aep\_name") \\ 5 & otherwise \end{cases} \quad (3)$$

Using the fitness equations shown above, if fitness $f_{com}(X_1)$ evaluates to 0, it means the current head rule is a sensitive sink rule S and thus "append head" is not eligible. Similarly, if fitness $f_{com}(X_k)$ evaluates to 0, it means the current tail rule contains AEP function and thus "append tail" is not eligible. Consequently, from these fitness values, the proposed variation operator can know the type of rule in current head or current tail positions in a candidate solution and bias the

search process towards application of eligible variation operations.

In addition, the proposed variation operator needs to avoid replacement of appropriate production rule in a candidate with an inappropriate one. The proposed variation operator considers the "production fitness" of the anchor-rule that is about to be replaced. This fitness is given by Eq. 4. Using the equation, if the production fitness has value of 0, it means the anchor-rule is in an appropriate position, and should be maintained, otherwise it can be replaced.

$$f_{prod}(X_i) = \begin{cases} 0 & X_{i-x} \overset{+}{\Rightarrow} \alpha X_i \beta \\ 1 & otherwise \end{cases}$$
$$where\ \ i > 1, i > x, and\ x > 0 \qquad (4)$$

During the search process, both production fitness and completeness fitness of a candidate are considered so as to achieve highly effective biasing to eligible variation operations. The way in which the proposed variation operator uses the fitness information to bias application of eligible variation operations is summarized in Table 2.

Table 2: Biasing of Variation Operations

Variation Operation	Eligibility Condition	Interpretation
Append Head	$f_{com}(X_1) = 5$ and $f_{prod}(X_2) = 0$ if $k > 1$ $f_{com}(X_1) = 5$, if $k = 0$	Append head if first rule is not X_{iS}, but it is connected to second rule
Replace Head	$f_{prod}(X_2) = 1$	Replace head if first rule is not connected to second rule
Append Tail	$f_{com}(X_k) = 5$ and $f_{prod}(X_k) = 0$, if $k > 1$ $f_{com}(X_k) = 5$, if $k = 0$	Append tail if last rule does not contain "AEP fun", but is appropriate
Replace Tail	$f_{prod}(X_k) = 1$	Append tail if last rule is not appropriate

In Table 2, the first column shows variation operations that can be applied to candidate. The second column shows the values of candidate's fitness for which the corresponding variation operation is eligible. The last column provides explanation of what is done based on fitness values in column two.

Algorithm

This subsection presents algorithm of the proposed variation operator for applying the four variation operations to produce offspring during the search process. The algorithm as shown in Fig. 4. below, begins by getting the parent candidate into variable C, and randomly chooses one of the four variation operations (variable Opr). The algorithm determines the anchor-rule, and then obtains the best neighbor rule through tournament selection over randomly chosen q rules. Thereafter, the algorithm initializes the child as exact copy of the parent, and then bias application of the chosen variation operation accordingly to produce a child that differs from the parent. Finally, the child is added to the offspring collection.

```
Algorithm searchOperator(parent candidate)

begin
    // parent candidate
    C ← parent candidate
    //chose variation Opr at random
    Opr ← randomly chosen variation operation
    RP ← ϕ    // ϕ contains all extracted grammar rules
    tmpRulesArray ← random q rules from RP
    If (Opr = Append head or Replace head) {
      anchor_rule =  headOf(C)
    } else {
      anchor_rule =  tailOf(C)
    }
    // apply tournament with respect to anchor-rule
    best_neighbour_rule = tournamentOf(tmpRulesArray)
    // initialize child to exact copy of parent
    child ← C
    // now bias variation
    // append head operation
    If (Opr = Append head) {
      If(sizeOf(C) = 1) {
            child ←append best_neighbour_rule to head of C
      }
       else if (  f_prod(second_ruleOf(C)) = 0 and f_com(first_ruleOf(C)) = 5 ){
            child ←append best_neighbour_rule to head of C
      }
    }
    // replace head operation
    If (Opr = replace head) {
      if (f_prod(second_ruleOf(C)) = 1 ){
            child ←replace head of C  with best_neighbour_rule
```

```
    }
  }
  // append tail operation
  If (Opr = Append tail) {
    If(sizeOf(C) = 1) {
        child ←append best_neighbour_rule to tail of C
    }
     else if ( f_prod(iast_ruleOf(C)) = 0 and f_com(last_ruleOf(C)) = 5 ){
        child ←append best_neighbour_rule to tail of C
    }
  }
  // replace tail operation
  If (Opr = replace tail) {
    if (f_prod(last_ruleOf(C)) = 1 ){
        child ←replace tail of C with best_neighbour_rule
    }
  }
  // add child to offspring
  off − springs ← +child
end
```

Figure 4: Algorithm of the Proposed Variation Operator

The following section illustrates how the proposed variation operator is applied.

Example Application

For illustration purpose, let us use the proposed variation operator to apply variation operations to the example candidates defined earlier in Fig. 3. This exercise, as shown in table 3, would produce offspring for the candidates.

The first column of the Table 3. shows values of parameters which are required by the proposed variation operator. Note, the value assign to Opr is randomly selected from the four variation operations. Also, tournamentOf(q) apply tournament selection over q randomly selected rules and returns best neighbor of the anchor-rule.

Table 3: Applying Variation Operations to Example Candidates

Variation Operator Parameters	Testing Eligibility Condition	Offspring Produced
$C = X_{14}, X_{12A}$ $\mathrm{Opr} =$ "Append head" $\mathrm{anchor-rule} = X_{14}$ $\mathrm{tournamentOf(q)} = X_{17S}$	$f_{com}(X_{14}) = 5$ and $f_{prod}(X_{12}) = 0$ (Condition Satisfied)	X_{17S}, X_{14}, X_{12A}
$C = X_{14}$ $\mathrm{Opr} =$ "Append tail" $\mathrm{anchor-rule} = X_{14}$ $\mathrm{tournamentOf(q)} = X_{11A}$	$f_{com}(X_{14}) = 5,\ k = 1$ (Condition satisfied)	X_{14}, X_{11A}
$C = X_{17S}, X_{14}, X_{13V}, X_{11A}$ $\mathrm{Opr} =$ "Replace tail" $\mathrm{anchor-rule} = X_{11A}$ $\mathrm{tournamentOf(q)} = X_{12}$	$f_{prod}(X_{11A}) = 0$ (Condition Not Satisfied)	$X_{17S}, X_{14}, X_{13V}, X_{11A}$ (No change, offspring same as parent)

The second column shows values of fitness components that are used to check eligibility of the chosen variation operation with respects to the candidate. In first and second row, the eligibility conditions are satisfied, thus the proposed variation operator performs the biased variation using the chosen operation in Opr, and produce offspring accordingly. However, in the last row, the eligibility condition is not satisfied, and consequently, the proposed variation operator ignores application of the chosen variation operation and returns an offspring that is exact copy of the parent. This strategy helps to bias operations to neighborhood as well as to eligible operations that avoids destructive variations.

The third column shows the resulting offspring. Observe that, in the first row, the variation has resulted in offspring that is an optimal solution, which reveals SQLIV with respect to AEP "password" of the running example. The offspring produced in second row contains two grammar rules, showing example of how candidates grow during the search process. In the last row, the offspring is exact copy of the parent, this happens when the randomly chosen variation operation is likely to results in a destructive variation, and is thus ignored. This is a good way of ensuring elitism of the search process.

CONCLUSION

The efficiency of search process can be greatly improved by the quality of the variation operations which are applied for producing offsprings as well as the quality of the fitness evaluation strategy that is used for selecting better candidate solutions. In this paper, we define novel variation operator that forms part of evolutionary programming method which employs grammar reachability search process for detection of vulnerabilities. The proposed variation operator is able to perform four variation operations on candidate solution in order to produce its offspring. The proposed variation operator is able to bias application of variation operation towards better fitness candidates as well as eligible and non-

destructive variations, thus, improving elitism of the search process by ensuring adequate balance between exploration and exploitation of the search space. The proposed variation operator was used in EPSQLiFix which is an evolutionary programming method for SQL injection vulnerabilities detection and removal in web application. The proposed variation operator demonstrated remarkable performance in EPSQLiFix. Thus, we are optimistic that the ideas presented in defining the proposed variation operator can as well benefit research communities in improving performance of search process across related problem domains.

REFERENCES

Ackling, T., Alexander, B. & Grunert, I. (2011, July 12–16). Evolving patches for software repair. In Proceedings of the 13th Annual Conference on GECCO. Dublin, Ireland: ACM.

Alhijawi, B., & Awajan, A. (2024). Genetic algorithms: Theory, genetic operators, solutions, and applications. Evolutionary Intelligence, 17(3), 1245-1256.

Al-Tashi, Q., Shami, T. M., Abdulkadir, S. J., Akhir, E. A. P., Alwadain, A., Alhussain, H., Alqushaibi, A., Rais, H. M. D., Muneer, A., Saad, M. B., Wu, J., & Mirjalili, S. (2023). Enhanced Multi-Objective Grey Wolf Optimizer with Lévy Flight and Mutation Operators for Feature Selection. Computer Systems Science and Engineering, 47(2), 1937–1966. https://doi.org/10.32604/csse.2023.039788

Arcuri, A. (2008, May 10 - 18). On the automation of fixing software bugs. In Proceedings of the 30th International Conference on Software Engineering. (pp. 1003-1006). Leipzig, Germany: ACM. DOI: 10.1145/1370175.1370223

Arcuri, A. (2011, June). Evolutionary repair of faulty software. Journal of Applied Soft Computing, 11 (4), 3494–3514. DOI: 10.1016/j.asoc.2011.01.023.

Dominguez-Jimenez, J. J., Estero-Botaro, A., Garcia-Domingueze, A. & Medina-Bulo, I. (2011, Octorber). Evolutionary mutation testing. Journal of Information and Software Technology, 53 (10), 1108–1123, doi: org/10.1016/j.infsof.2011.03.008.

Emmanuel, S., Okoye, I., Ezenweke, C., Shobanke, D., & Adeniyi, I. (2022). Estimating nonlinear regression parameters using particle swarm optimization and genetic algorithm. FUDMA Journal Of Sciences, 6(6), 202-213.

Floudas, C. A., & Pardalos, P. M. (2014). Recent advances in global optimization.

Halfond, W. G. J., Orso, A. & Manolios, P. (2006b, November). Using positive tainting and syntax-aware evaluation to counter sql injection attacks. In Proceedings of the of the Symposium on the Foundations of Software Engineering (FSE 2006).

Hidalgo-Herrero, M., Rabanal, P., Rodriguez, I., & Rubio, F. (2013). Comparing problem solving strategies for NP-hard optimization problems. Fundamenta Informaticae, 124(1-2), 1-25.

Jones, J. A, & Harrold, M. J. (2005, November 07-11). Empirical evaluation of the tarantula automatic fault localization technique. In Proceedings of the 20th International Conference on ASE '05. (pp. 273-282). Long Beach, CA: IEEE/ACM. doi: 10.1145/1101908.1101949.

Kumar, R., Memoria, M., Gupta, A., & Awasthi, M. (2021). Critical Analysis of Genetic Algorithm under Crossover and Mutation Rate. Proceedings - 2021 3rd International Conference on Advances in Computing, Communication Control and Networking, ICAC3N 2021, December, 976–980. https://doi.org/10.1109/ICAC3N53548.2021.9725640

Li, W., Liang, P., Sun, B., Sun, Y., & Huang, Y. (2023). Reinforcement learning-based particle swarm optimization with neighborhood differential mutation strategy. Swarm and Evolutionary Computation, 78(February), 101274. https://doi.org/10.1016/j.swevo.2023.101274

Medeiros, I., Neves, N. F., & Correia, M. (2014). Automatic detection and correction of web application vulnerabilities using data mining to predict false positives. In Proceedings of the 23rd International Conference on World Wide Web. (pp. 63-74) New York: IEEE. DOI: 10.1145/2566486.2568024.

Obunadike, G., John, A., & Ismaila, I. (2018). OPTIMIZATION OF K-MODE ALGORITHM FOR DATA MINING USING PARTICLE SWARM OPTIMIZATION. FUDMA JOURNAL OF SCIENCES, 2(3), 24-33.

Umar, K., Sultan, A. B., Zulzalil, H., Admodisastro, N., & Abdullah, M. T. (2018a, July). Formulation of SQL Injection Vulnerability Detection as Grammar Reachability Problem. In 2018 International Conference on Information and Communication Technology for the Muslim World (ICT4M) (pp. 179-184). IEEE Computer Society.

Umar, K., Sultan, A. B., Zulzalil, H., Admodisastro, N., & Abdullah, M. T. (2018b). Comparing Web Vulnerability Scanners with a New Method for SQL Injection Vulnerabilities Detection and Removal EPSQLiFix. International Journal of Engineering & Technology, 7(4.31), 40-45.

Uzunbayir, S., & Kurtel, K. (2024). EvoColony: A Hybrid Approach to Search-Based Mutation Test Suite Reduction Using Genetic Algorithm and Ant Colony Optimization. International Journal of Intelligent Systems and Applications in Engineering, 12(1), 437–449.

Wang, X., Yu, T., Arcaini, P., Yue, T., & Ali, S. (2022). Mutation-based test generation for quantum programs with multi-objective search. In GECCO 2022 - Proceedings of the 2022 Genetic and Evolutionary Computation Conference (Vol. 1, Issue 1). Association for Computing Machinery. https://doi.org/10.1145/3512290.3528869

Yan, L., Li, X., Feng, R., Feng, Z. & Hu, J. (2013, October 29th). Detection method of the second-order SQL injection in web applications. In Proceedings of the Third International Workshop on SOFL+MSVL. (pp. 154–165). Queenstown, New Zealand: Springer. DOI: 10.1007/978-3-319-04915-1_11.