International Journal of Software Innovation
Volume 11 • Issue 1

A Method for Feature Subset Selection in Software Product Lines

Nahid Hajizadeh, Shiraz University of Technology, Iran
Peyman Jahanbazi, Shiraz University of Technology, Iran
Reza Akbari, Shiraz University of Technology, Iran*

ABSTRACT

Software product line (SPL) represents methods, tools, and techniques for creating a group of related software systems. Each product is a combination of multiple features. So, the task of production can be mapped to a feature subset selection problem, which is an NP-hard problem. This issue is very significant when the number of features in a software product line is huge. This chapter is aimed to address the feature subset selection in software product lines. Furthermore, the authors aim at studying the performance of a proposed multi-objective method in solving this NP-hard problem. Here, a multi-objective method (MOBAFS) is presented for feature selection in SPLs. The MOBAFS is a an optimization algorithm, which is inspired by the foraging behavior of honeybees. This technique is evaluated on five large-scale real-world software product lines in the range of 1,244 to 6,888 features. The proposed method is compared with the SATIBEA. According to the results of three solution quality indicators and two diversity metrics, the proposed method, in most cases, surpasses the other algorithm.

KEYWORDS

Bee algorithm, feature selection, Multi-objective optimization, Search Based Software Engineering, Software Product Lines

1. INTRODUCTION

Nowadays few software products are produced individually. Most organizations tend to develop families of similar software. These products share some common elements, which is named software asset. A software asset is a description of a solution or knowledge that application engineers use to develop or modify products in a software product line (Withey, 1996). By applying the reusability concept, shared software assets can be reused, instead of developing from scratch. The process of software asset reusability is important in the SPL. SPLs are software systems that share a common regulated set of features, which include architecture, design, documents, test cases, and other assets. Also, a standard definition of a software feature has been established by IEEE[1]:" *a distinguishing*

DOI: 10.4018/ijsi.315654 *Corresponding Author

characteristic of a software item (for example, performance, portability, or functionality) " (ANSI/IEEE, 1983). Features are employed to develop products. In other words, each product is a combination of some features, and these features are selected based on the attributes that are assigned to each feature.

The SPL development process is divided into two phases: domain engineering and application engineering (Pohl et al., 2005). Domain engineering is the process of defining and realizing the commonality and variability of the SPL. It refers to the core assets and application engineering process for developing particular applications by employing the variability of the SPL (Pohl et al., 2005). Application engineering is the product generation using the core asset achieved by domain engineering.

A feature model or feature diagram is a compact display of all products in terms of features in a software product line. Indeed, a feature model shows the principal features of a product's family in the domain and the relationships between them (Kang et al., 1990). A feature model is a compressed demonstration of all the products of SPL in respect of "features". A feature model has a tree structure, which defines the relationships between features in a hierarchical pattern. In Figure 1, a feature model for a mobile phone is illustrated. In a feature model, there are two types of relationships between features: 1) the relationship between a parent feature and its children's features, and 2) cross-tree constraints.

A feature model can be expressed using a Boolean expression, as depicted in Figure 2. Software products are created by merging the selected features from a feature model and considering all the constraints. Here the issue is to select and extract a set of features from a feature model and the goal is selecting the features set in a way that not only covers restrictions but also be optimized in terms of the objective functions. Feature models consist of hundreds and thousands of features on an industrial scale so it is practically impossible to use exact algorithms to derive products in a reasonable time. In other words, how to combine a set of features to make a product, optimally, is an NP-hard problem, as White et al. indicated, and is known as a feature subset selection problem in SPLs (White et al., 2008). Therefore, the need for a metaheuristic algorithm for solving such problems is felt. Hence, in this paper, a new method based on the bee algorithm is presented. The method is used to get an optimal solution to the problem in an acceptable time.

Figure 1. A sample feature model for a mobile phone product line (Benavides et al., 2010)

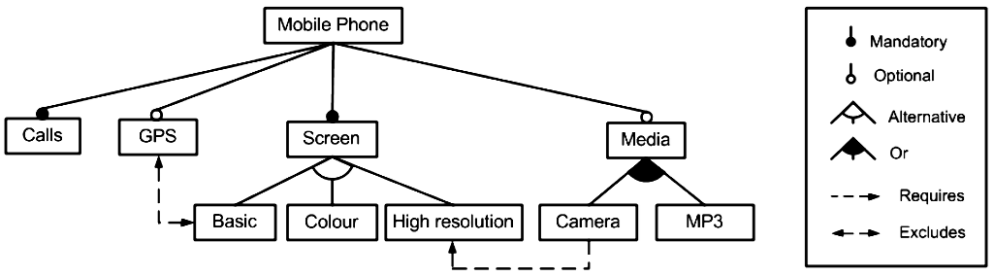


Figure 2. Feature model of mobile phone SPL in a Boolean expression format (Sayyad et al., 2013)

$$
\begin{aligned}
FM = \; & (\text{Mobile Phone} \leftrightarrow \text{Calls}) \\
& \wedge (\text{Mobile Phone} \leftrightarrow \text{Screen}) \\
& \wedge (\text{GPS} \rightarrow \text{Mobile Phone}) \\
& \wedge (\text{Media} \rightarrow \text{Mobile Phone}) \\
& \wedge (\text{Screen} \leftrightarrow \text{XOR (Basic, Color, High resolution)}) \\
& \wedge (\text{Media} \leftrightarrow \text{Camera} \vee \text{MP3}) \\
& \wedge (\text{Camera} \rightarrow \text{High resolution}) \\
& \wedge \neg(\text{GPS} \wedge \text{Basic})
\end{aligned}
$$

In recent years, artificial intelligence methods have been successfully applied to cope with software development problems (Nayyar et al., 2019, Kukkar et al., 2020, Sepahvand et al., 2022). Dealing with combinatorial optimization problems, in most cases, is time-consuming so this kind of problem has been an alive area of research for many decades. According to this point that optimization problems, in reality, become more complicated, and the need for more appropriate optimization algorithms is sensed regularly. In such problems, the goal is finding the optimum of the objective function (Gheisari et al., 2019).

Optimizing more than one conflicting objective simultaneously is known as Multi-Objective Optimization (MOO) (Hwang et al. 2012). The outputs of MOO are solutions that are optimal or near-optimal. Pareto Front is a set of non-dominated solutions, being chosen as optimal if no objective can be improved without sacrificing at least one other objective. On the other hand, a solution x* is referred to as dominated by another solution x if, and only if, x is equally good or better than x* with respect to all objectives and for at least one objective, x is strictly better than x* (Manne, 2016). The swarm intelligence-based methods try to find a near-optimal solution for NP-hard problems (Bonabeau et al., 1999). These algorithms are inspired by the natural behavior of social animals.

This work is aimed to propose a multi-objective bee algorithm for feature subset selection in SPLs. The proposed method uses different movement patterns to search the solution space efficiently. The results show that the proposed algorithm surpasses the previous most successful algorithm (i.e. SATIBEA) in the majority of performance metrics.

The rest of the paper is organized as follows: In Section 2, related work is introduced. Section 3 describes the MOBAFS algorithm. Sections 4 and 5 include experimental setup and experimental results, and finally, Section 6 presents some conclusions and future works.

2. RELATED WORK

In recent years, optimization methods have been used for the feature subset selection problem. These methods can be categorized as single and multi-objective methods. Benavides et al. introduced a method that mapped the feature selection problem in SPLs to a Constraint Satisfaction Problem (CSP) (Benavides et al., 2005). They examined their method on four problems (two synthesized and two real product lines). Their implementation showed an exponential behavior when the number of features in the feature models was increased. White et al. proposed a polynomial-time approximation technique which was called Filtered Cartesian Flattening (FCE) to achieve an approximately optimal solution by transforming the feature selection problem with resource constraints into an equivalent Multidimensional Multiple-choice Knapsack Problem (MMKP) and applying MMKP approximation algorithms (White et al. 2008). Their method showed 93% optimality on feature models with 5,000 features. White et al. proposed a formal model of multi-step SPL, and mapped this model to CSPs and called MUlti step Software Configuration probLEm (MUSCLE) (White et al., 2016). An artificial intelligence approach based on genetic algorithms called GAFES was applied by Guo et al. as a search-based technique to solve the optimized feature selection in SPLs. GAFES can produce solutions with 86-97% optimality (Guo et al., 2011). Soltani et al. introduced a framework that uses an artificial intelligence planning technique to select features that satisfy stakeholders' business concerns and resource constraints (Soltani et al., 2016).

Sayyad et al. used the Multi-objective Evolutionary Optimization Algorithm (MEOA) to solve the feature selection problem in SPLs (Sayyad, et al., 2016). MEOA can achieve an acceptable configuration for a large feature model (290 features) in 8 minutes while other algorithms found one acceptable configuration after 3 hours. Later, Sayyad et al. presented simple heuristics to solve the software product lines configuration problem, in the case of multi-objective, which led Indicator-Based Evolutionary Algorithm (IBEA) to find optimum configurations of large-scale models (Sayyad et al., 2017). They applied the "seed" technique to generate the initial population randomly and could find 30 sound solutions for configuring a set of 6000 features in 30 minutes.

Olaechea et al. compared an exact and an approximate algorithm in terms of accuracy, time consumption, scalability, and parameter setting requirements for solving the software product lines configuration problem in five case studies (Olaechea, 2014). Based on their experimental results, they claimed that exact techniques for the small multi-objective SPLs are possible, and also approximate methods can be applied for large-scale problems, however, we need considerable effort to find the best parameter setting for a satisfactory approximation.

Tan et al. presented a new approach by introducing a feedback-directed mechanism into various EAs (Tan et al. 2014). Their method is based on analyzing violated constraints and uses the analyzed results as feedback to guide the process of crossover and mutation operators. However, for the Linux repository, which contains 6888 features, their method could not find any correct solution.

Henard et al. proposed SATIBEA, a search-based feature subset selection algorithm for SPLs (Henard, et al., 2016). SATIBEA is a combination of the Indicator-Based Evolutionary Algorithm (IBEA) (Zitzler et al., 2016), and the satisfiability (SAT) solving technique. They considered 5 objects and evaluated SATIBEA on 5 huge real-world SPLs. Their significant results encouraged us to evaluate our approach with SATIBEA.

(Hierons et al., 2016) and (Xue et al., 2016) proposed new approaches based on IBEA. The point is that none of these articles have evaluated their works with considering SATIBEA. Despite the presence of SATIBEA and proof of being more powerful than IBEA, both articles compared their methods with IBEA. Another point is that Hierons et al. applied a real repository with a maximum of 290 features and a randomly generated feature model with 10,000 features. Both articles did not evaluate their results by common metrics specialized for multi-objective optimization algorithms. Due to the significant results of SATIBEA, in comparison with other algorithms such as IBEA, we would prefer to compare our method with that.

As the case of "test case selection" can be seen as a special situation of multi-objective product/configuration selection in feature models, so, some articles in the field of software testing are deserved to be introduced.

Parejo et al. applied the NSGA-II evolutionary algorithm to solve the multi-objective test case prioritization problem (Parejo et al., 2016). They proposed seven objective functions based on functional and non-functional data and found that multi-objective prioritization results in fault detection being faster than mono-objective prioritization.

Galindo et al. presented a variability-based testing approach to derive video sequence variants to test different input combinations when developing video processing software (Galindo et al., 2014). Combinational and multi-objective optimization testing techniques over feature models have been presented to generate a minimized number of configurations which is combinations of features to synthesize variants of video sequences.

SPL pairwise testing is what Lopez-Herrejon et al. have taken into consideration. They applied classical multi-objective evolutionary algorithms such as NSGA-II, MOCell, SPEA2, and PAES to select a set of products to test which maximizes the coverage and minimizes the test suite size (Lopez-Herrejon et al., 2014).

Pereira et al. introduced a new method to configure a product that considers both qualitative and quantitative feature properties (Pereira et al., 2017). They modeled the product configuration task as a combinatorial optimization problem. Their research was the first work in the literature that considered feature properties in both leaf and non-leaf features.

Abbas et al. proposed a multi-objective algorithm that consists of three independent paths. They applied heuristics to these paths and found that the first path is infeasible due to space and execution time complexity and the second path reduces the space complexity. They calculated the outcomes of all three paths and proved the significant improvement of optimum solution without constraint violation occurrence (Abbas et al., 2018).

Xue and Li exposed the mathematical nature of the optimal feature selection problem in the SPL and tried to implement three mathematical programming approaches to solve this problem at different

scales. The empirical results showed that their proposed method can find significantly more non-dominated solutions in similar or less execution time, in comparison with IBEA (Xue and Li, 2018).

Yu et al. proposed six hybrid algorithms that combine SAT solving with different MOEAs. Their case study was based on five large-scale, rich-constrained, and real-world SPLs. Empirical results demonstrated that the SATMOCell algorithm obtained a competitive optimization performance in comparison with the state-of-the-art that outperformed the SATIBEA in terms of quality Hypervolume metric for 2 out of 5 SPLs within the same time budget (Yu et al., 2018).

Shi et al. introduced a parallel portfolio algorithm, IBEAPORT, which designs three algorithm variants by incorporating constraint solving into the indicator-based evolutionary algorithm in different ways and performs these variants by utilizing parallelization techniques (Shi et al., 2018). Their approach utilized the exploration capabilities of different algorithms and improved optimality as far as possible within a limited time budget.

Khan et al. proposed a new feature selection method that supports multiple multi-level user-defined objectives (Khan et al., 2019). A new feature quantification method using twenty operators, capable of treating text-based and numeric values, and three selection algorithms called Falcon, Jaguar, and Snail are proposed. Falcon and Jaguar are based on a greedy algorithm while Snail is a variation of an exhaustive search algorithm. With an increase of 4% execution time, Jaguar performed 6% and 8% better than Falcon in terms of added value and the number of features selected.

Wägemann et al. introduced ADOOPLA, a tool-supported approach for the optimization of product line system architectures (Wägemann et al., 2019). In contrast to existing approaches where product-level approaches only support product-level criteria and product-line oriented approaches support product-line-wide criteria, their approach integrates criteria from both levels in the optimization of product line architectures. Also, the approach could handle multiple objectives at once, supporting the architect in exploring the multi-dimensional Pareto-front of a given problem.

Xue et al. introduced a new aggregation-based dominance (ADO) for Pareto-based evolutionary algorithms to direct the search for high-quality solutions (Xue et al., 2019). Their approach was tested on two widely used Pareto-based evolutionary algorithms: NSGA-II and SPEA2+SDE and validated on nine different SPLs with up to 10,000 features and two real-world SPLs with up to 7 objectives. Their experiments have shown the effectiveness and efficiency of both ADO-based NSGA-II and SPEA2+SDE.

Xiang et al. addressed the open research questions, of how different solvers affect the performance of a search algorithm, by performing a series of empirical studies on 21 feature models, where most of them are reverse-engineered from industrial SPLs (Xiang et al., 2020). They examined four conflict-driven clause learning solvers, two stochastic local search solvers, and two different ways to randomize solutions. Experimental results suggested that the performance could be affected by different SAT solvers, and by the ways to randomize solutions in the solvers. Their research served as a practical guideline for choosing and tuning SAT solvers for the many-objective optimal software product selection problem.

Saber et al. presented MILPIBEA, a novel hybrid algorithm that combines the scalability of a genetic algorithm (IBEA) with the accuracy of a mixed-integer linear programming solver (IBM ILOG CPLEX) (Saber et al., 2020). They also studied the behavior of their solution (MILPIBEA) in contrast with SATIBEA.

Lu et al. introduced a pattern-based, interactive configuration derivation methodology, called Pi-CD, to maximize opportunities of automatically deriving correct configurations of CPSs by benefiting from pre-defined constraints and configuration data of previous configuration steps (Lu et al., 2020). Pi-CD requires architectures of CPS product lines modeled with Unified Modeling Language extended with four types of variabilities, along with constraints specified in Object Constraint Language (OCL). Pi-CD is equipped with 324 configuration derivation patterns that they defined by systematically analyzing the OCL constructs and semantics.

Hierons et al. proposed a new technique, the grid-based evolution strategy (GrES), which considers several objective functions that assess a selection or prioritization and aims to optimize all of these (Hierons et al., 2020). The problem was thus a many-objective optimization problem. They used a new approach, in which all of the objective functions are considered but one (pairwise coverage) was seen as the most important. They also derived a novel evolution strategy based on domain knowledge.

Due to the NP-hardness of feature subset selection in SPLs, the exact algorithms are not applicable especially for large-sized problems. In such cases, metaheuristics can be used to find the near-optimal solution in a shorter time. Therefore, this work is aimed to develop a multi-objective method based on meta-heuristics to solve the problem in an acceptable time.

3. THE PROPOSED METHOD

The description of the proposed algorithm for feature subset selection in SPL is given in this section. A set of meta-heuristic algorithms are utilized to solve problems with exponential time complexity which are inspired by bee algorithms. The proposed algorithm is designed based on the multi-objective method presented by (Akbari & Ziarati, 2012). This algorithm is a population-based optimization technique that is inspired by the foraging behavior of honeybees.

The algorithm involves three types of bees; experienced forager, onlooker, and scout bees which fly in an D-dimensional search space $S \subset R^D$ to find the near-optimal solution. Each type of bee has a specific moving pattern which is used by the bees to adapt their flying direction. Experienced foragers use an adaptive windowing mechanism to select their leaders and regulate their next positions. In addition, for cutting the most crowded members of the archive, the adaptive windowing mechanism is applied too. In MOBAFS, scouts and adapting windowing mechanisms maintain diversity over the Pareto front. The structure of the algorithm is shown in Figure 3.

The MOBAFS input parameters are the population size, maximum iteration, and the maximum number of non-dominated bees. *Initialization* is the first phase of the algorithm where the number of bees is randomly generated and also non-dominated bees are determined. The second phase, *Update*, is a loop with max_iter iteration. In this phase, bees move according to their pattern in the search space. In each iteration, the type of bee is specified. Then the experienced forager bees, onlookers, and scout bees move in the search space with specific patterns of movement. Finally, if the number of solutions in the archive exceeds the archive size, some of the elements will be deleted. Further details about each step of the algorithm are given as follows.

3.1. Initialization

Figure 4 illustrates the initialization part of the MOBAFS algorithm. In this step, two sets of *bees* and *arch* are initialized which indicates the bees set and non-dominated bees set respectively. *Random()* function creates a random point in the D-dimensional space. In addition, the *add_non_dominated()* function receives two sets as the input (*arch* and *bees*' sets) and extracts the non-dominated set from inputs.

3.2. Update

This step is repeated by *max_iter*. At first, the type of bees is determined. Then each bee moves in the search space according to its type and finally, the *arch* set is updated. Figure 5 illustrates these steps.

Scout bees are determined by calling the *get_scout()* function. The *get_scout()* function chooses *ps* percent of bees randomly. In the next step, *select_non_dominated()* function determines experienced forager bees. This function intersects the *arch* and *bees'* set to calculate experienced forager bees. The result is subtracted from the Scout set. Lastly, onlooker bees are determined, the bees who are neither experienced forager nor scout.

Figure 3. The MOBAFS algorithm

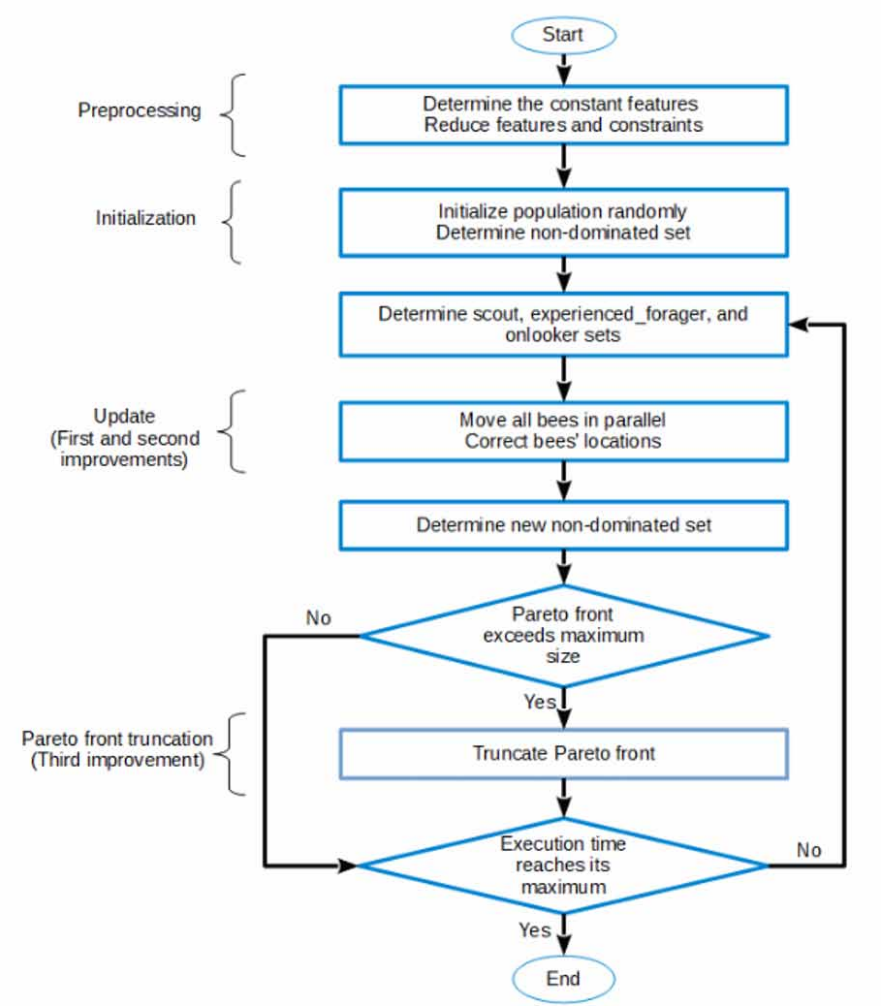


Figure 4. MOBAFS algorithm-Initialization step

bees = **random**(D);
arch = **add_non_dominated**(null, bees);

In the first step, a forager bee determines a leader bee. The leader bee is randomly selected from the *arch* set. The bees in the less crowded location in the search space have more probability to be selected as a leader. The details of this process are explained in (Akbari and Ziarati, 2012). In the next step, the new position of the bee is calculated, according to two parameters w_L and r_L which controls the importance of the information provided by the leader and a random variable to a uniform distribution in the range $[0.1]$, respectively.

The movement pattern of onlooker bees is very similar to forager bees' movement. The difference is that each bee randomly selects an elite bee from forager bees. The new position of an onlooker bee

Figure 5. MOBAFS algorithm-Update step

```
scout = get_scout(bees, ps);
experienced forager = select_non_dominated(arch, bees) - scout;
onlooker = bees - Experienced _forager – scout;
//Update foragers
For all bee in foragers
Begin
  leader = select_leader(arch);
  bee = bee + w_l + r_l * (leader – bee)
End
//Update onlookers
for all bee in onlookers
Begin
  elite = select_elite(experimced_forager);
  x = x + w_e * r_e * (elite – bee)
End
//Update scouts
for all bee in scouts
Begin
  b_1 = select_random(arch);
  b_2 = select_random(arch);
  bee = move_randomly(b_1, b_2);
End
//Update archive
arch = add_non_dominated(arch, bees);
if n(arch) > arch_size then
  truncate_archive(arch);
End
```

is determined based on two parameters w_e and r_e which controls the importance of the information provided by the elite and a random variable to a uniform distribution in the range $[0.1]$, respectively.

The Scout bees' movement begins with a random selection of two bees from the *arch* set as lower and upper bounds to represent the search space and then a Scout bee moves in this space.

The last part of the *Update* step is updating the archive. At first, from the *bees* and *arch* sets, non-dominated solutions are selected and stored in the *arch*. If the number of *arch*'s elements is more than the *arch_size,* the *truncate_archive()* function will eliminate the extra element. The bees in the most crowded locations have more probability to be removed.

To improve the efficiency of the MOBAFS algorithm and create more optimized solutions, two developments on MOBAFS have been done which will be described in the following.

The first development is related to generate solutions with fewer constraint violations. For this purpose, some changes have been done to movement functions. In all three movement functions (experienced forager, onlooker, and scout) according to constant features, the positions of bees are repaired. After that, the SAT (Satisfiability) solver checks whether any constraint has been violated due to the new positions. In case of constraint violation, some features are selected randomly and their statuses are changed to indeterminate, then the SAT solver tries to assign proper values to indeterminate features to eliminate constraint violation. This process continues until the SAT solver notifies that no constraints are violated. In this way, the appropriate value for features and subsequently the new

position of bees will be determined. Fig. 6 illustrates how the SAT solver achieves a solution without any constraint violation. In this work to generate valid configurations, Sat4j, (Berre and Parrain, 2010) a SAT solver which is one of the most popular ones, is applied.

The second development is related to the *truncate_archive()* function where its modified version is shown in Figure 6. This function is called, same as the previous version, at each iteration of MOBAFS. In the modified version of this function, two points are considered:

(1) Defining a process that can compare two non-dominated solutions to avoid accidental removal of high-quality solutions.
(2) Trying to keep solutions that are located in the non-crowded location of state space.

This function adds attributes GROUP, and RANK, to each node. The GROUP attribute helps to determine crowded positions and the RANK attribute orders non-dominated solutions based on the quality of those solutions. Finally, in each GROUP, in the number of RANK of the solution, adjacent solutions stay in the *arch* set. Figure 7 shows this function in more detail.

3.3. Speedup

In this section, the time complexity of the MOBAFS algorithm is investigated. If the total execution (in the case of serial implementation), initialization, loop iterations, and Pareto Front truncation time be T_{total}, T_{init}, T_{loop} and T_{trunc}, respectively, then the total processing time is calculated according to Eq.1:

Figure 6. An example of SAT solver role in repositioning bees

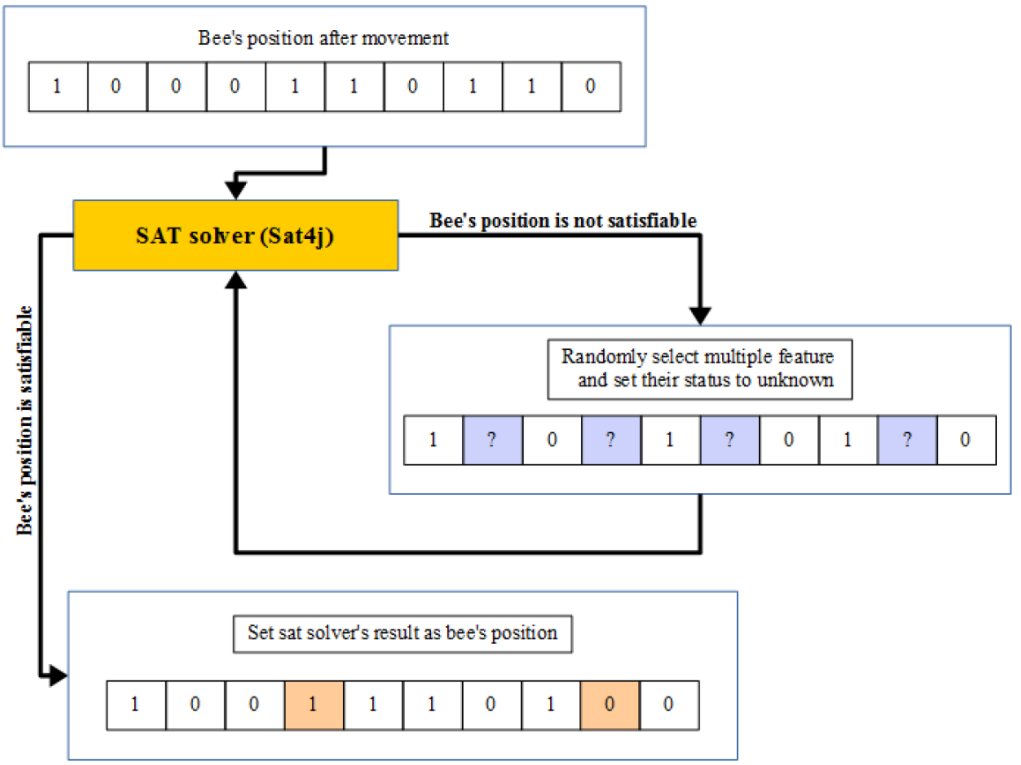


Figure 7. Truncate Pareto front

Algorithm: Truncate Pareto front

Input:
archive: list of non-dominated solutions
sp: number of section per each fitness function
Output:
newArchive: Minified list of non-dominated solutions

Function truncateArchive (archive, sp)
Begin
newArchive = {};
For all x **in** archive
Begin

$$a = \left\lfloor \frac{sp * (x.correctness - \min(correctness))}{\max(correctness) - \min(correctness)} \right\rfloor$$
$$b = \left\lfloor \frac{sp * (x.richness - \min(richness))}{\max(richness) - \min(richness)} \right\rfloor$$
$$c = \left\lfloor \frac{sp * (x.usedBefore - \min(usedBefore))}{\max(usedBefore) - \min(usedBefore)} \right\rfloor$$
$$d = \left\lfloor \frac{sp * (x.knowsDefects - \min(knowsDefects))}{\max(knowsDefects) - \min(knowsDefects)} \right\rfloor$$
$$e = \left\lfloor \frac{sp * (x.cost - \min(cost))}{\max(cost) - \min(cost)} \right\rfloor$$
$$x \cdot group = a + 255 * \left(b + 255 * \left(c + 255 * (d + 255 * e)\right)\right)$$
$$x \cdot rank = (sp + 1 - a)(sp + 1 - b)(sp + 1 - c)(sp + 1 - d)(sp + 1 - e)$$

End
$$T = \left\{x \in 2^{archive} \mid \forall m \,\&\, n \in x \quad m \cdot group = n \cdot group \,\&\, (\forall t \in T\ x \not\subset t \ or\ t = x)\right\}$$
For all t **in** T
Begin
R = rank(t)
Randomly select R member of t set and add to newArchive
End
Return newArchive
End

$$T_{total(serial)} = T_{init} + T_{loop} + T_{trunc} \tag{1}$$

All the times mentioned in Eq. (1) is measured through Eq. (2), Eq. (3), and Eq. (4) where I is the number of repetition of iterations loop, N is the population size, F is the number of features and C is the number of constraints (In these formulas, for simplicity, the constant factors are not mentioned):

$$T_{init} = N\left(F + C\right) + N^2 \tag{2}$$
$$T_{loop} = I * N\left(F + C + N\right) \tag{3}$$
$$T_{trunc} = N^2 \tag{4}$$

According to Eq. (2) to Eq. (4), the total time complexity of the MOBAFS algorithm that is shown in Eq. (1) can be rewritten as Eq. (5):

$$T_{total(serial)} = I * N\left(F + C + N\right) \tag{5}$$

With the aim of parallelism, the total time complexity is reduced to Eq.6 where P is the number of processors:

$$T_{total(parallel)} = I * N\left(\frac{F + C}{P} + N\right) \tag{6}$$

Consequently, Eq. (7) shows the speedup, based on Eq.5 and Eq.6:

$$Speedup = \frac{T_{total(serial)}}{T_{total(parallel)}} = \frac{I * N\left(F + C + N\right)}{I * N\left(\frac{F + C}{P} + N\right)} \tag{7}$$

The variables in Figure 7 are defined as: $x{\cdot}correctness$: The Correctness fitness function of Solution x $x{\cdot}richness$: The richness fitness function of Solution x $x{\cdot}usedBefore$: The used before fitness function of Solution x $x{\cdot}knowsDefects$: The known defect fitness function of Solution x $x{\cdot}cost$: The cost fitness function of Solution x

$$\max\left(correctness\right) = t{\cdot}correctness \mid t \in archive and \forall x \in archive t{\cdot}correctness \geq x{\cdot}correctness$$

$$\max\left(richness\right) = t{\cdot}richness \mid t \in archive and \forall x \in archive t{\cdot}richness s \geq x{\cdot}richness$$

$$\max\left(usedBefore\right) = t{\cdot}usedBefore \mid t \in archive and \forall x \in archive t{\cdot}usedBefore \geq x{\cdot}usedBefore$$

$$\max\left(knowsDefects\right) = t{\cdot}knowsDefects \mid t \in archive and \forall x \in archive t{\cdot}knowsDefects \geq x{\cdot}knowsDefects$$

$$\max\left(cost\right) = t{\cdot}cost \mid t \in archive and \forall x \in archive t{\cdot}cost \geq x{\cdot}cost$$

$$\min\left(correctness\right) = t{\cdot}correctness \mid t \in archive and \forall x \in archive t{\cdot}correctness \leq x{\cdot}correctness$$

$$\min\left(richness\right) = t{\cdot}richness \mid t \in archive and \forall x \in archive t{\cdot}richness s \leq x{\cdot}richness$$

$$\min\left(usedBefore\right) = t{\cdot}usedBefore \mid t \in archive and \forall x \in archive t{\cdot}usedBefore \leq x{\cdot}usedBefore$$

$$\min\left(knowsDefects\right) = t{\cdot}knowsDefects \mid t \in archive and \forall x \in archive t{\cdot}knowsDefects \leq x{\cdot}knowsDefects$$

$$\min\left(cost\right) = t{\cdot}cost \mid t \in archive and \forall x \in archive t{\cdot}cost \leq x{\cdot}cost$$

4. EXPERIMENTAL SETUP

In this section, the experiments that have been done to evaluate the performance of the MOBAFS algorithm on some data sets are represented. The performance of the MOBAFS algorithm is evaluated in comparison with SATIBEA (Henard et al., 2016). The comparisons are performed based on the famous metrics which have been broadly used to compare optimization algorithms. They are divided into two groups; Quality and Diversity metrics. These metrics and their definitions have been listed

in Table 1. Each algorithm was run 30 times per feature model and each run lasted 30 minutes for execution time.

4.1. Solution Modeling

The first step for applying the MOBAFS algorithm to any problem is to represent a solution as a point in a D-dimensional search space. Suppose a problem with F features is presented. To represent one solution, the state of F features must be determined. Each feature can have one of two situations; selected (1) or not (0). In this case, one solution will be shown with F bits. For transforming the F bits to integers (or real), every 32 features can be displayed with an integer number. Consequently displaying F features need $\frac{F}{32}$ integer. In other words, the domain of the problem will be $\mathbb{N}^{\frac{F}{32}}$.

4.2. Data Set

The feature models, which are used in this work, are taken from the LVAT (Linux Variability Analysis Tools) repository [2]. The 5 feature models and their characteristics (the version, the number of features, and constraints) are listed in Table 2.

4.3. Preprocessing

Feature selection is done based on the attributes of each feature. Due to the augmentation that has been done by (Sayyad et al., 2016), (Sayyad et al., 2013), and (Henard et al. 2015) three attributes are added to each feature: cost, used before, and defects. The values of these 3 attributes are set randomly. *Cost* is a real number in the range of 5.0 to 15.0, *Used before* is a Boolean variable and *Defects* is an integer in the range of 0 to 10. These 3 attributes have a dependency among them: if (not used before) then defects=0.

Another major preprocessing performed includes reducing in dimensions of the problem domain. In the previous section, it is mentioned that a problem with F features can be indicated by a point

Table 1. The metrics and their definitions

Quality	Metric	Definition
	Hypervolume (HV) (Brockhoff et al., 2008)	It evaluates how well a Pareto front fulfills the optimization objectives.
	Epsilon (ε) (Knowles et al., 2006)	It measures the shortest distance which is required to transform every solution in a Pareto front to dominate the reference front.
	Inverted Generational Distance (IGD) (Veldhuizen et al., 1998)	It is the average distance from the solutions owned by the reference front to the closest solution in a Pareto front.
Diversity	Pareto Front Size (PFS)	It is the number of solutions in a Pareto front.
	Spread (S) (Deb et al., 2002)	It determines the amount of spread in Pareto front's solutions.

Table 2. Feature Models (Henard et al., 2015)

Feature Model	Version	Features (mandatory)	Constraints
Linux	2.6.28.6	6,888 (58)	343,944
uClinux	20100825	1,850 (7)	2,468
Fiasco	2011081207	1,638 (49)	5,228
FreeBSD	8.0.0	1,396 (3)	62,183
eCos	3.0	1,244 (0)	3,146

in space with the size $\frac{F}{32}$. The smaller value for $\frac{F}{32}$ there is a smaller search space. Given that the value of F depends on the type of problem and its value cannot be changed, so only the value of $\frac{F}{32}$ has to be changed. To reduce the search space, by considering the constraints of any problem, the status of some features can be determined before the execution of the algorithm. The status of those features can be identified before running the algorithm as:

(1) Those features are mandatory
(2) Due to the status of mandatory features and constraints set, their values can be determined. These features are called constant features, although (Sayyad et al. 2017) called them "fixed features".
(3) To clarify these two modes, two examples are presented.
(4) The first mode: If p is a mandatory feature then before running the algorithm its value can be considered (1), i.e. *Enabled*.
(5) The second mode: if among constraints there is a constraint in form $\sim p \vee q$, according to the value of p (here p is a mandatory feature so its value is considered 1), to satisfy the constraint $\sim p \vee q$, the value of q has to be (1).
(6) Fig. 8 shows the function of constant and mandatory feature selection. In fact, by executing this function constant features are specified before executing MOBAFS. By executing the proposed function, the dimensions of the search space will be reduced from $\frac{F}{32}$ to the formula (8), where M is the number of mandatory features and t is the number of constant features.
(7) $\frac{F-M-t}{32}$
(8) Table 3 consists of feature models, the number of features, constant features, constraints, and declined constraints.

As it is evident, by using this technique, on average, 26.69 percent of feature status and 14.61 percent of constraints are determined before searching the search space. Determining the status of the features before searching the search space, in addition to reducing the search space has two more advantages:

1) The unfeasible solutions were not searched
2) The decline in the problem's constraints

The first advantage seems obvious because by determining the value of each feature before execution, that feature can't be assigned to any other value during execution. For the second advantage, it can be said that if the value of all the features of a proposition is determined in a way that the proposition has the right value, and also the value of features does not change during execution then that proposition will be right forever and it's not necessary to re-evaluate it.

4.4. Intended Optimization Objectives

According to the point that (Henard et al., 2015) applied 5 objects for this issue and this article tends to evaluate its performance in comparison with the mentioned paper, so in this work, 5 objects are considered too which are listed in Table 4.

Figure 8. Determining constant features

Algorithm: Determining constant features

Input:
Constraints: A CNF.

Output:
zero and one: Two arrays of constant features that zero is array of variables with false value and one is an array of variables with true value.

```
zero = {};
one = {};
defined = {};
Changed = true;
while (changed = true)
Begin
 changed = false;
 for (c in constraints)
 Begin
  for (x in c)
  Begin
   others = c - {x};
   if x ∉ defined and (∀ t ∈ others |t ∈ zero ∨ -t ∈ one) then
   Begin
    if x > 0 then
     one = one ∪ {x}
     defined = defined ∪ {x}
    else
     zero = zero ∪ {−x}
     defined = defined ∪ {−x}
    End
    changed = true;
   End
  End
 End
End
```

Table 3. Feature models with declined constraints

Feature Model	Features	Constant Features	Constraints	Declined Constraints
Linux	6,888	154	343,944	192
uClinux	1,850	1244	2,468	1256
Fiasco	1,638	1013	5,228	1059
FreeBSD	1,396	4	62,183	6
eCos	1,244	23	3,146	58

Table 4. The Objectives and their optimal situations

Objective	Optimal situation
Correctness	minimizing the violated constraints
Richness of features	minimizing the number of deselected features
Used before	minimizing the features that were not used before
Defects	minimizing the number of defects
Cost	minimizing the cost

4.5. MOBAFS Configuration

In this work, the MOBAFS algorithm is adjusted with adequate number of individuals and 30 independent runs have been done. Considering more individuals, contrary to what is common and less than this amount, is that by creating a larger initial population, the probability of finding more non-dominated solutions increases. The number of experienced forager and onlooker bees is 98 percent of the population which is divided between them at each iteration of the algorithm dynamically and consequently, scout bees are 2 percent of the population. The weighting coefficients w_l (the parameter which controls the importance of the knowledge provided by the leader bee; a randomly selected bee from the bees' archive) and w_e (the parameter which controls the importance of the knowledge provided by the Experienced forager bees) are adapted to 2.5 and 2.12 respectively. All experiments were performed on Ubuntu 16.04 LTS 64bit with Intel Core i7 4790K CPU 4GHz and 16GB RAM.

5. EXPERIMENTAL RESULTS

The MOBAFS and SATIBEA were run on the five feature models, listed in Table 3, to evaluate their performance. In Table 5, the results based on 5 metrics, indicated in Table 4, and 30 independent runs for each algorithm in 30 minutes are presented. The values of measured metrics are the average values in 30 runs.

As it is mentioned in Table 5, the Hypervolume (HV) metric shows the volume of the area which is dominated by a solution set. Therefore, the Pareto front of an algorithm with higher HV is selected more precisely.

Figure 9(a) presents Hypervolume of MOBAFS and SATIBEA. It is obvious that these two algorithms in two feature models, which are the densest in terms of the number of constraints, i.e., Linux and FreeBSD, have approximately the same values, though the HV values of these two algorithms for other feature models have a miner difference. Therefore, it can be said that the power of these two algorithms in the Hypervolume metric is roughly equal.

According to Epsilon metric definition in Table 5, a lower value for Epsilon shows better performance. Figure 9(b) illustrates the Epsilon values of MOBAFS and SATIBEA and for all feature models, excluding Linux, the Epsilon values related to MOBAFS are lower than SATIBEA's Epsilon values.

Based on the definition of the Inverted Generational Distance (IGD) metric in Table 5, the lower IGD, the better performance, so by considering the IGD values of MOBAFS and SATIBEA which is shown in Figure 9(c), MOBAFS has a marked superiority over SATIBEA in this stage due to having lower IGD values in all feature models.

By looking at the definition of the Pareto Front Size (PFS) metric, pointed out in Table 5, it is clear the higher value for this metric is more favorable. By paying attention to Figure 9(d), MOBAFS has competitive performance in comparison with SATIBEA.

The Spread (S) metric, according to its definition in Table 1 shows the spread in the Pareto front, so the higher Spread indicates the more distributed solutions. Based on the comparison illustrated in Figure 10, SATIBEA is more successful in generating of sporadic solutions.

In Fig. 11, the relationship between Hypervolume and violated constraints in the Linux feature model is illustrated. As it is obvious, with the increase in the number of violated constraints, the amount of Hypervolume will be decreased.

5.1. Statistical Test

In this work, transformed Vargha-Delaney effect size measurement is applied to assess the new algorithm. As indicated by (Neumann et al., 2015), the mentioned non-parametric effect size test returns a $\hat{A}_{12}$ statistic which is between 0 and 1. $\hat{A}_{12} = 0.5$ shows that the two algorithms are

Table 5. Experimental results in comparison with MOBAFS and SATIBEA

			MOBAFS	SATIBEA
Linux	Quality	HV	2.35E-01	2.38E-01
		Epsilon	1.51E+00	9.90E-01
		IGD	1.66E-03	5.52E-03
	Diversity	PFS	2.61E+03	2.91E+02
		S	6.43E-01	1.25E+00
uClinux	Quality	HV	2.41E-01	2.76E-01
		Epsilon	2.01E-01	1.31E+00
		IGD	1.49E-03	1.35E-02
	Diversity	PFS	7.96E+02	3.00E+02
		S	6.96E-01	1.37E+00
Fiasco	Quality	HV	2.08E-01	2.26E-01
		Epsilon	1.13E-01	1.75E+00
		IGD	1.28E-03	1.36E-02
	Diversity	PFS	1.40E+03	3.00E+02
		S	5.37E-01	1.37E+00
FreeBSD	Quality	HV	2.71E-01	2.71E-01
		Epsilon	1.34E-01	2.00E-01
		IGD	7.92E-04	3.91E-03
	Diversity	PFS	2.87E+03	2.85E+02
		S	6.40E-01	1.34E+00
eCos	Quality	HV	2.25E-01	2.83E-01
		Epsilon	1.08E-01	1.63E-01
		IGD	1.46E-03	8.33E-03
	Diversity	PFS	4.15E+03	3.00E+02
		S	7.23E-01	1.35E+00

completely equivalent; otherwise they have some difference. For instance, if $\hat{A}_{12} = 0.8$ then algorithm A overcomes algorithm B with higher values, 80% of the time.

Table 6 illustrates the $\hat{A}_{12}$ statistic to evaluate the results. Concerning HV, the most contrast is in uClinux and eCos (SATIBEA produces better results in 69.5% (1-0.305) and 77.7% (1-0.223) of the times, respectively) and for other feature models, there is no significant difference between the algorithms. Regarding Epsilon, for the Linux feature model, SATIBEA has better performance in 86.3% of the time. On the other hand, for other feature models, MOBAFS surpasses SATIBEA in 78.1% (1-0.219) to 100% of the time. Concerning IGD, for all feature models, MOBAFS gets better results in 87.4% (1-0.126) to 100% of the time. Regarding PFS, MOBAFS achieves more excellent results 100% of the time, for all feature models. Concerning the Spread metric, unlike PFS, SATIBEA gets the highest Spread for all feature models 100% of the time.

Figure 9. Comparison among MOBAFS and SATIBEA

Figure 10. Spread values for MOBAFS and SATIBEA

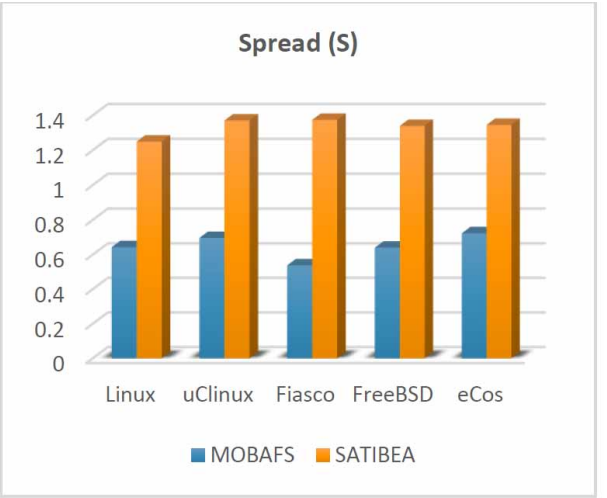
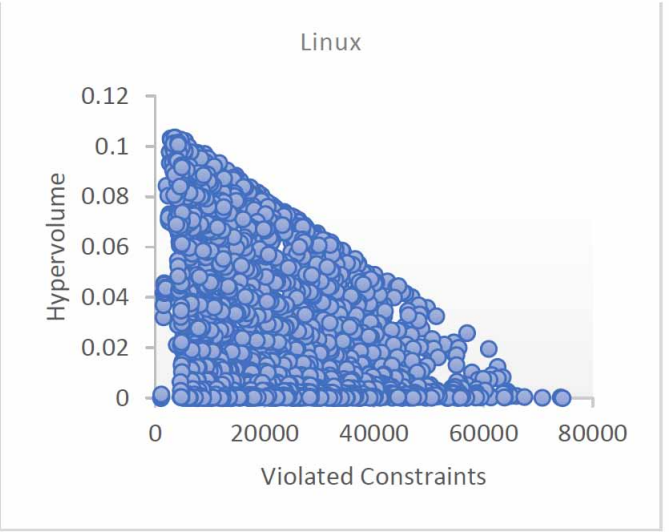


Figure 11. The relationship between Hypervolume and violated constraints



5.2. Threats to Validity

As indicated by Lopez-Herrejon et al. in (Lopez-Herrejon et al., 2014), a common internal validity threat is adequate parameter setting. Default parameter values, which were applied by their main authors, for the two algorithms under comparison are employed. The two external threats, as mentioned in (Lopez-Herrejon et al., 2014), are the selection of multi-objective algorithms to compare and the selection of feature models. According to this point that this is for the first time that the MOBAFS algorithm is used to solve an SBSE problem and SATIBEA is a strong contender in this field, so these two algorithms have been chosen. In terms of feature model, the most highly prestigious and the largest real feature models are considered which sometimes led to an increase in the execution time. Applying other algorithms and feature models could be a new research field.

5.3. Analysis of Overall Performance

In general, the proposed algorithm shows competitive performance in comparison with the SATIBEA method. However, MOBAFS in 3 metrics (i.e. Epsilon, IGD, and PFS) and SATIBEA in 2 metrics (i.e. HV and Spread) have better performance. It seems that, the proposed method generates solutions with better qualities, while the SATIBEA shows a better distribution of the solutions.

Table 6. Transformed $\hat{A}_{12}$ statistical test results for MOBAFS-SATIBEA

Feature Model	HV	ε	IGD	PFS	S
Linux	0.469	0.863	0.126	1.000	0.000
uClinux	0.305	0.000	0.000	1.000	0.000
Fiasco	0.430	0.000	0.000	1.000	0.000
FreeBSD	0.492	0.103	0.117	1.000	0.000
eCos	0.223	0.219	0.000	1.000	0.000

6. CONCLUSION AND FUTURE WORK

SPLs developers prefer to generate optimal products automatically. However, selecting an optimal subset of features, by considering constraints, is an NP-hard problem. This work, for automating product derivation in SPLs, proposed a new algorithm based on the intelligent behavior of honey bees as a feature selection method. Two improvements have been done to strengthen the power of MOBAFS; Parallelization and Generating solutions with higher quality. Our new MOBAFS algorithm was compared with SATIBEA, the recent most successful algorithm, based on the largest and most prestigious feature models. Experiments showed that the new method, in most cases, is competitive with SATIBEA. The truncation method which is used to control the size of the archive, the method for distribution of the solutions on the Pareto front, searching method over the search space have the main roles in the success of multi-objective methods. In future work, we aim to pay more attention to truncation methods along with new optimization methods for searching on the search space.

CONFLICTS OF INTEREST

The author declares no conflict of interest.

REFERENCES

Abbas, A., Siddiqui, I. F., Lee, S. U. J., Bashir, A. K., Ejaz, W., & Qureshi, N. M. F. (2018). Multi-objective optimum solutions for IoT-based feature models of software product line. *IEEE Access: Practical Innovations, Open Solutions*, *6*, 12228–12239. doi:10.1109/ACCESS.2018.2806944

Akbari, R., & Ziarati, K. (2012). Multi-objective bee swarm optimization. *International Journal of Innovative Computing, Information, & Control*, *8*(1B), 715–726.

Anne, P. (2010). The SAT4J library release 2.2 system description. Journal on Satisfiability [JSAT]. *Boolean Modeling and Computation*, *7*(2-3), 59–64. doi:10.3233/SAT190075

Anne, P. (2010). The SAT4J library release 2.2 system description. Journal on Satisfiability [JSAT]. *Boolean Modeling and Computation*, *7*(2-3), 59–64. doi:10.3233/SAT190075

ANSI/IEEE. (1990). IEEE Standard Glossary of Software Engineering Terminology, 729-1983. IEEE.

Benavides, D., Segura, S., & Ruiz-Cortés, A. (2010). Automated analysis of feature models 20 years later: A literature review. *Information Systems*, *35*(6), 615–636. doi:10.1016/j.is.2010.01.001

Benavides, D., Trinidad, P., & Ruiz-Cortés, A. (2005, June). Automated reasoning on feature models. In *International Conference on Advanced Information Systems Engineering* (pp. 491-503). Springer.

Bonabeau, E., Dorigo, M., Theraulaz, G., & Theraulaz, G. (1999). *Swarm intelligence: from natural to artificial systems (No. 1)*. Oxford university press. doi:10.1093/oso/9780195131581.001.0001

Brockhoff, D., Friedrich, T., & Neumann, F. (2008, September). Analyzing hypervolume indicator based algorithms. In *International Conference on Parallel Problem Solving from Nature* (pp. 651-660). Springer. doi:10.1007/978-3-540-87700-4_65

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. A. M. T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, *6*(2), 182–197. doi:10.1109/4235.996017

Galindo, J. A., Alférez, M., Acher, M., Baudry, B., & Benavides, D. (2014, July). A variability-based testing approach for synthesizing video sequences. In Proceedings of the 2014 International Symposium on Software Testing and Analysis (pp. 293-303). Association of Computing Machinery. doi:10.1145/2610384.2610411

Gheisari, M., Panwar, D., Tomar, P., Harsh, H., Zhang, X., Solanki, A., Nayyar, A., & Alzubi, J. A. (2019). An optimization model for software quality prediction with case study analysis using MATLAB. *IEEE Access: Practical Innovations, Open Solutions*, *7*, 85123–85138. doi:10.1109/ACCESS.2019.2920879

Guo, J., White, J., Wang, G., Li, J., & Wang, Y. (2011). A genetic algorithm for optimized feature selection with resource constraints in software product lines. *Journal of Systems and Software*, *84*(12), 2208–2221. doi:10.1016/j.jss.2011.06.026

Henard, C., Papadakis, M., Harman, M., & Le Traon, Y. (2015, May). Combining multi-objective search and constraint solving for configuring large software product lines. In *International Conference on Software Engineering* (*Vol. 1,* pp. 517-528). IEEE. doi:10.1109/ICSE.2015.69

Hierons, R. M., Li, M., Liu, X., Parejo, J. A., Segura, S., & Yao, X. (2020). Many-objective test suite generation for software product lines. [TOSEM]. *ACM Transactions on Software Engineering and Methodology*, *29*(1), 1–46. doi:10.1145/3361146

Hierons, R. M., Li, M., Liu, X., Segura, S., & Zheng, W. (2016). SIP: Optimal product selection from feature models using many-objective evolutionary optimization. [TOSEM]. *ACM Transactions on Software Engineering and Methodology*, *25*(2), 1–39. doi:10.1145/2897760

Hwang, C. L., & Masud, A. S. M. (2012). *Multiple objective decision making—methods and applications: a state-of-the-art survey* (Vol. 164). Springer Science & Business Media.

Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., & Peterson, A. S. (1990). *Feature-oriented domain analysis (FODA) feasibility study*. Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst. doi:10.21236/ADA235785

Khan, F. Q., Musa, S., Tsaramirsis, G., & Buhari, S. M. (2019). SPL features quantification and selection based on multiple multi-level objectives. *Applied Sciences*, *9*(11), 2212. doi:10.3390/app9112212

Knowles, J. D., Thiele, L., & Zitzler, E. (2006). A tutorial on the performance assessment of stochastic multiobjective optimizers. *TIK-report*, 214.

Kukkar, A., Mohana, R., Kumar, Y., Nayyar, A., Bilal, M., & Kwak, K. S. (2020). Duplicate bug report detection and classification system based on deep learning technique. *IEEE Access: Practical Innovations, Open Solutions*, *8*, 200749–200763. doi:10.1109/ACCESS.2020.3033045

Lopez-Herrejon, R. E., Ferrer, J., Chicano, F., Egyed, A., & Alba, E. (2014, July). Comparative analysis of classical multi-objective evolutionary algorithms and seeding strategies for pairwise testing of software product lines. In *congress on evolutionary computation (CEC)* (pp. 387-396). IEEE.

Lu, H., Yue, T., & Ali, S. (2020). Pattern-based Interactive Configuration Derivation for Cyber-physical System Product Lines. *ACM Transactions on Cyber-Physical Systems*, *4*(4), 1–24. doi:10.1145/3389397

Manne, J. R. (2016). Multiobjective optimization in water and environmental systems management-MODE approach. In *Handbook of research on advanced computational techniques for simulation-based engineering* (pp. 120–136). IGI Global. doi:10.4018/978-1-4666-9479-8.ch004

Nayyar, A. (2019). *Instant approach to software testing: Principles, applications, techniques, and practices*. BPB Publications.

Neumann, G., Harman, M., & Poulding, S. (2015, September). Transformed vargha-delaney effect size. In *International Symposium on Search Based Software Engineering* (pp. 318-324). Springer. doi:10.1007/978-3-319-22183-0_29

Niehaus, E., Pohl, K., & Böckle, G. (2005). *Software Product Line Engineering: Foundations*. Principles and Techniques, Kapitel Product Management.

Olaechea, R., Rayside, D., Guo, J., & Czarnecki, K. (2014, September). Comparison of exact and approximate multi-objective optimization for software product lines. In *Proceedings of the 18th International Software Product Line Conference, vol. 1,* (pp. 92-101). doi:10.1145/2648511.2648521

Parejo, J. A., Sánchez, A. B., Segura, S., Ruiz-Cortés, A., Lopez-Herrejon, R. E., & Egyed, A. (2016). Multi-objective test case prioritization in highly configurable systems: A case study. *Journal of Systems and Software*, *122*, 287–310. doi:10.1016/j.jss.2016.09.045

Pereira, J. A., Maciel, L., Noronha, T. F., & Figueiredo, E. (2017). Heuristic and exact algorithms for product configuration in software product lines. *International Transactions in Operational Research*, *24*(6), 1285–1306. doi:10.1111/itor.12414

Saber, T., Brevet, D., Botterweck, G., & Ventresque, A. (2020, April). Milpibea: Algorithm for multi-objective features selection in (evolving) software product lines. In *European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar),* (pp. 164-179). Springer.

Sayyad, A. S., Ingram, J., Menzies, T., & Ammar, H. (2013, November). Scalable product line configuration: A straw to break the camel's back. In *International Conference on Automated Software Engineering (ASE)* (pp. 465-474). IEEE. doi:10.1109/ASE.2013.6693104

Sayyad, A. S., Ingram, J., Menzies, T., & Ammar, H. (2013, November). Scalable product line configuration: A straw to break the camel's back. In *International Conference on Automated Software Engineering (ASE)* (pp. 465-474). IEEE. doi:10.1109/ASE.2013.6693104

Sayyad, A. S., Menzies, T., & Ammar, H. (2013, May). On the value of user preferences in search-based software engineering: A case study in software product lines. In *international conference on software engineering (ICSE)* (pp. 492-501). IEEE.

Sepahvand, R., Akbari, R., Hashemi, S., & Boushehrian, O. (2022). An Effective Model to Predict the Extension of Code Changes in Bug Fixing Process Using Text Classifiers. *Iranian Journal of Science and Technology. Transaction of Electrical Engineering*, *46*(1), 257–274. doi:10.1007/s40998-021-00458-1

Shi, K., Yu, H., Guo, J., Fan, G., & Yang, X. (2018). A parallel portfolio approach to configuration optimization for large software product lines. *Software, Practice & Experience*, *48*(9), 1588–1606. doi:10.1002/spe.2594

Soltani, S., Asadi, M., Hatala, M., Gašević, D., & Bagheri, E. (2011, November). Automated planning for feature model configuration based on stakeholders' business concerns. In *International Conference on Automated Software Engineering (ASE 2011)* (pp. 536-539). IEEE.

Tan, T. H., Xue, Y., Chen, M., Sun, J., Liu, Y., & Dong, J. S. (2015, July). Optimizing selection of competing features via feedback-directed evolutionary algorithms. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis* (pp. 246-256). Association for Computing Machinery. doi:10.1145/2771783.2771808

Van Veldhuizen, D. A., & Lamont, G. B. (1998). *Multiobjective evolutionary algorithm research: A history and analysis (*pp. 1-88). Technical Report TR-98-03, Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio, USA..

Wägemann, T., Tavakoli Kolagari, R., & Schmid, K. (2019, September). ADOOPLA-Combining Product-Line-and Product-Level Criteria in Multi-objective Optimization of Product Line Architectures. In *European Conference on Software Architecture* (pp. 126-142). Springer. doi:10.1007/978-3-030-29983-5_9

White, J., Dougherty, B., Schmidt, D. C., & Benavides Cuevas, D. F. (2009). Automated reasoning for multi-step feature model configuration problems. In *13th International Software Product Line Conference*, (pp. 11-20). ACM.

White, J., Doughtery, B., & Schmidt, D. C. (2008, September). Filtered Cartesian Flattening: An Approximation Technique for Optimally Selecting Features while Adhering to Resource Constraints. In SPLC, (2), (pp. 209-216).

Withey, J. (1996). *Investment Analysis of Software Assets for Product Lines*. DTIC Document.

Xiang, Y., Yang, X., Zhou, Y., Zheng, Z., Li, M., & Huang, H. (2020). Going deeper with optimal software products selection using many-objective optimization and satisfiability solvers. *Empirical Software Engineering*, *25*(1), 591–626. doi:10.1007/s10664-019-09761-2

Xue, Y., Li, M., Shepperd, M., Lauria, S., & Liu, X. (2019). A novel aggregation-based dominance for Pareto-based evolutionary algorithms to configure software product lines. *Neurocomputing*, *364*, 32–48. doi:10.1016/j.neucom.2019.06.075

Xue, Y., & Li, Y. F. (2018, May). Multi-objective integer programming approaches for solving optimal feature selection problem: a new perspective on multi-objective optimization problems in SBSE. In *Proceedings of the 40th International Conference on Software Engineering* (pp. 1231-1242). Association for Computing Machinery. doi:10.1145/3180155.3180257

Xue, Y., Zhong, J., Tan, T. H., Liu, Y., Cai, W., Chen, M., & Sun, J. (2016). IBED: Combining IBEA and DE for optimal feature selection in software product line engineering. *Applied Soft Computing*, *49*, 1215–1231. doi:10.1016/j.asoc.2016.07.040

Yu, H., Shi, K., Guo, J., Fan, G., Yang, X., & Chen, L. (2018, July). Combining constraint solving with different MOEAs for configuring large software product lines: a case study. In 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC) (Vol. 1, pp. 54-63). IEEE. doi:10.1109/COMPSAC.2018.00016

Zitzler, E., & Künzli, S. (2004, September). Indicator-based selection in multiobjective search. In *International conference on parallel problem solving from nature* (pp. 832-842). Springer.

ENDNOTES

1 The Institute of Electrical and Electronics Engineers
2 http://code.google.com/p/linux-variability-analysis-tools