Enhancing Search-Based Product Line Design with Crossover Operators

Diego Fernandes da Silva,
Luiz Fernando Okada
State University of Maringá
Maringá, Paraná, Brazil
pg400800@uem.br
ra107247@uem.br

Thelma Elita Colanzi
State University of Maringá
Maringá, Paraná, Brazil
thelma@din.uem.br

Wesley K. G. Assunção
Federal University of
Technology-Paraná
Toledo, Paraná, Brazil
wesleyk@utfpr.edu.br

ABSTRACT

The Product Line Architecture (PLA) is one of the most important artifacts of a Software Product Line. PLA designing has been formulated as a multi-objective optimization problem and successfully solved by a state-of-the-art search-based approach. However, the majority of empirical studies optimize PLA designs without applying one of the fundamental genetic operators: the crossover. An operator for PLA design, named Feature-driven Crossover, was proposed in a previous study. In spite of the promising results, this operator occasionally generated incomplete solutions. To overcome these limitations, this paper aims to enhance the search-based PLA design optimization by improving the Feature-driven Crossover and introducing a novel crossover operator specific for PLA design. The proposed operators were evaluated in two well-studied PLA designs, using three experimental configurations of NSGA-II in comparison with a baseline that uses only mutation operators. Empirical results show the usefulness and efficiency of the presented operators on reaching consistent solutions. We also observed that the two operators complement each other, leading to PLA design solutions with better feature modularization than the baseline experiment.

CCS CONCEPTS

• **Software and its engineering → Software product lines**; **Software architectures**; **Search-based software engineering**; • **Computing methodologies → Search methodologies**;

KEYWORDS

Multi-objective evolutionary algorithm, software product line, software architecture, recombination operators

ACM Reference Format:
Diego Fernandes da Silva, Luiz Fernando Okada, Thelma Elita Colanzi, and Wesley K. G. Assunção. 2020. Enhancing Search-Based Product Line Design with Crossover Operators. In *Genetic and Evolutionary Computation Conference (GECCO '20), July 8–12, 2020, Cancún, Mexico.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3377930.3390215

1 INTRODUCTION

The Product Line Architecture (PLA) is one of the main artifacts of a Software Product Line (SPL). A PLA comprises all available features of a product and presents the architectural variations in an SPL [18]. Designing a PLA is a complex and non-trivial task because the SPL architect should take into account important properties such as extensibility, reusability and modularity [2]. Particularly, feature modularization is an important property to be achieved in a PLA design as it directly impacts on SPL extensibility and reusability [12]. In this sense, a proper PLA design is important for the SPL life cycle since it has to be generic and flexible enough to support the configuration of many products [18].

The designing of PLA has been successfully optimized by multi-objective evolutionary algorithms [1, 2, 11]. However, the use of crossover operator has not been completely investigated in this context. The crossover operator is arguably one of the fundamental operators of evolutionary algorithms [6]. This operator combines parts of two different individuals (parents) to create offspring. Empirical results obtained in the software design context showed that the exclusive use of mutation resulted in very homogeneous populations and seemed to quickly land on a local optimum [14]. The combination of genes is an important way of maintaining the new qualities integrated by mutation operation inside the population. Hence, crossover operator is an essential step to the evolutionary process in order to generate improved solutions promoting diversity. This is the main motivation of our work.

Colanzi and Vergilio [1] proposed a crossover operator for PLA design, called Feature-driven Crossover, which is concerned with maintaining and improving feature modularization of a PLA design. A qualitative analysis showed that the Feature-driven Crossover contributes to reach good solutions in terms of fitness and diversity. In spite of this, it was observed the generation of some incomplete solutions, because some methods and attributes were missing in classes of the resulting PLAs [1]. To the best of our knowledge, this problem has not been addressed yet, remaining an open issue. In addition to this limitation, in literature we can find other operators for traditional software design that can be adapted for the context of PLA design. For example, there is a crossover operator, called Complementary Crossover, that operates trying to create a solution so that parents complement each other, finding solutions with higher quality than the asexual method (only mutation operation) [15]. This operator is not directly applicable to the PLA design as it uses a representation incompatible with that one adopted in [1]. However, the idea behind this operator can be adapted for PLA design.

Given the aforementioned context, this paper aims to enhance the search-based PLA design optimization by providing two effective crossover operators. Firstly, we present a new version of Feature-driven Crossover, correcting the problem of incomplete solutions (Section 4.2). Secondly, we propose a novel operator to the PLA design context (Section 4.3), named Complementary Crossover, which was inspired by a related work [15].

With the study herein presented we intend to answer the following research question: ***What is the benefit of using crossover operators in the optimization of PLA design in comparison to the state of the art?*** To do so, we carried out an empirical study (Section 5) involving two PLA designs and four experiments: (i) the baseline experiment with only mutation operators (state of the art), (ii) an experiment applying the improved Feature-driven Crossover and mutation operators, (iii) an experiment with the Complementary Crossover and mutation operators, and (iv) an experiment with the two crossover operators and the mutation ones.

The primary contributions of the paper can be summarized as: (i) we introduced a novel crossover operator for PLA design: Complementary Crossover; (ii) we improved the Feature-driven Crossover operator to prevent the generation of incomplete solutions; and, (iii) we reported an in-depth quantitative and qualitative evaluation of our crossover operators using two PLA designs that show the proposed operators are beneficial to PLA design (Section 6).

2 RELATED WORK

We performed a systematic mapping, following the guidelines of Petersen et al. [13], in order to discover crossover operators for software design, including PLA design. The crossover operators identified in this mapping are briefly described next.

In the context of optimizing traditional software architectures, Simons et al. [17] proposed an interactive evolutionary algorithm to optimize object-oriented class design. This algorithm uses a Trans-Positional crossover, in which attributes and methods are randomly chosen and swapped between two parents based on their class position. Positional swapping can only occur where swapping an attribute or method to another class would not leave the class empty of attributes or methods.

To preserve different quality attributes in software architectures, Räihä et al. [15] defined the Complementary Crossover operator. This operator chooses two optimized parents for given quality attributes in order to preserve and combine these good attributes from each parent in the child. For example, a parent with good modifiable fitness and another parent with good efficiency fitness are selected so that the children might inherit both desirable qualities. The authors designed two versions of Complementary Crossover, namely Simple Complementary Crossover and Complementary Gene-Selective Crossover. The difference between them is that the former applies a random crossover point to the parents, while the latter finds the best crossover point so that it uses the optimal portion of both parents in full. Experimental results showed that both versions of the Complementary Crossover improved versatility in the architectures produced, generating more elaborated solutions, being the solutions clearly superior in relation to fitness.

The operators presented by Simons et al. [17] and Räihä et al. [15] do not guarantee a proper modularization of architectural elements, sometimes leading to scattering of elements assigned to implementation of features. To deal with this problem, Harman and Hierons [7] have proposed a crossover operator based on a clustering algorithm that aims to preserve allocations of complete or partial modules from a parent to a child. Instead of selecting an arbitrary crossover point, the operator randomly choose similar modules in the two parents. The chosen module (and its components) is then exchanged between both parents.

Similarly to the crossover operator of Harman and Hierons [7], but taking into account feature modularization in a PLA design, Colanzi and Vergilio proposed the Feature-driven Crossover operator and studied its benefits to PLA design optimization [1]. This operator randomly selects a feature and then creates children by swapping the architectural elements (classes, interfaces, operations, etc.) that realize the selected feature. The goal is that children combine architectural elements that better modularize some SPL features that were inherited from their parents. Experimental results pointed out that the Feature-driven Crossover was beneficial for PLA design optimization since it improves the feature modularization of the obtained solutions. Despite the noticeable improvement in the exploration of the search space with this operator, it sometimes generated inconsistent solutions. The proponents of the Feature-driven Crossover acknowledged that improvements in the implementation of the operator would lead to better performance.

Despite the existence of different strategies of crossover to deal with architecture optimization, only few of them were applied in the context of PLAs. In addition, the state-of-the-art OPLA-Tool [5] for PLA optimization (see Section 3) only implements the Feature-driven Crossover [1], which sometimes generates inconsistent solutions. Based on that, we clearly see that there is a lack of more crossover operators to improve the optimization of PLA designs.

3 SEARCH-BASED DESIGN OF PLA

Colanzi et al. [2] introduced an approach named Multi-Objective Approach for Product-Line Architecture Design (MOA4PLA) to optimize PLA designs. In MOA4PLA, PLA designing is treated as a multi-objective optimization problem to be solved by search algorithms. The approach has an evaluation model [20] composed by several objective functions, which are formed by software metrics that provide indicators about architectural properties relevant to PLA design. Examples of objective functions are: feature modularization, SPL extensibility, variability, coupling and cohesion.

MOA4PLA receives as input a PLA design modeled in a UML class diagram. In such diagram, each architectural element is associated to the feature(s) that it realizes by using UML stereotypes. The PLA is encoded using a metamodel presented in [2]. The solution representation obtained through the metamodel contains all architectural elements such as components, classes, their attributes and methods, interfaces, operations, their inter-relationships, variabilities, variation points and variants. This representation allows handling the architectural elements during the search process.

The PLA design is optimized by genetic operators during the evolutionary process. MOA4PLA includes six mutation operators specific for PLA design [2]: Move Method, Move Attribute, Add Class, Move Operation, Add Package Manager, and Feature-driven Mutation. The last operator aims at modularizing interlaced and scattered features. Feature-driven Crossover is the single crossover operator included in MOA4PLA [1]. The mutation and crossover operators are responsible for moving architectural elements (methods, attributes, operations), adding elements (class, package) or modularizing features in a way that positively impacts some property of the solution. The output of the optimization process is a set of solutions (decoded to class diagrams) that are alternative designs for the PLA given as input. This set contains solutions with the optimized trade-off among the objectives that are optimized, being different modeling possibilities to the PLA design.

As previously mentioned, MOA4PLA uses an evaluation model proposed to PLA design [20]. Such a model provides a set of objective functions. The user must select the functions related to the properties that he/she wants to optimize. The three objective functions used in our experiments are named COE (Relational Cohesion), ACLASS (Class Coupling) and FM (Feature Modularization). COE evaluates the cohesion of PLA design in terms of internal relationship of the design classes. ACLASS measures class coupling by the number of architectural elements that depend on other design classes, added to the elements number in which each class depends. FM provides indicator about feature modularization in terms of feature scattering and feature interlacing over architectural elements and feature-driven cohesion. The equations of the objective functions and their metrics are presented in [20].

OPLA-Tool [5] is the open source tool that automates the application of MOA4PLA. It allows the usage of different multi-objective algorithms to optimize PLA designs, such as NSGA-II [4].

4 PROPOSAL OF CROSSOVER OPERATORS FOR PLA DESIGN

In this section, we present two crossover operators to enhance the PLA design optimization: Feature-driven Crossover and Complementary Crossover. We propose a new version of the former, originally proposed in [1], which was improved in order to guarantee completeness and consistency. The latter was inspired in the Simple Complementary Crossover [15] considering the particularities of the PLA design context.

4.1 Consistency of the Generated Solutions

When reasoning about proposing crossover operators for PLA optimization, we took into account that: (i) Harman and Tratt [8] state that is difficult to guarantee the accuracy of solutions after the application of the crossover operator, and (ii) incomplete solutions were obtained during the Feature-driven Crossover application in a previous study [1]. Obtaining incomplete solutions is possible due to the fact that attributes or methods associated with features that were tangled in the same class, end up being lost after the crossover. In this sense, we proposed two methods used in the modification of the Feature-driven Crossover and in the implementation of the Complementary Crossover, aiming at maintaining the consistency and completeness of the generated solutions. These methods are called *Diff* and *RemoveDuplicate*.

The *Diff* method searches for and adds the parent's elements that have not been added to the child yet. This procedure is performed for both parents. The *Diff* pseudocode is presented in Algorithm 1. A list (*diffListsElements*) with all elements of the parent (classes and interfaces) with methods and attributes that do not exist in the child (*offspring*) is created (line 1). For each element of *diffListElements*, it is checked if the child has the element corresponding to *element* (line 3). If so, methods and attributes from *element* are added to the corresponding element in the child (line 5). Otherwise, the entire element is added to the child.

Algorithm 1: Diff Method

```
   Input: parent, offspring
   Output: offspring
 1 diffListElements ← all elements from parent that do not exist in
     offspring;
 2 for each element ∈ diffListElements do
 3     elementFromOffspring ← offspring.findElement(element);
 4     if elementFromOffspring <> null then
 5         elementFromOffspring.addMethodsAndAttributes(element);
 6         else offspring.addElement(element) ;
 7     end if
 8 end for
```

After the *Diff* application, the *RemoveDuplicate* method is performed to remove duplicate elements (methods and attributes) from the child. The decision of which duplicate element should be removed is taken based on the feature modularization, which means that the element that has the worst feature modularization is removed. Algorithm 2 presents the *RemoveDuplicate* pseudocode. In lines 1 and 2, a list of elements (*listElems*) and a list of the child's elements (*elementsOffspring*) are initialized. *listElems* is an empty list and *elementsOffspring* contains all child's classes and interfaces. For each method and attribute of each element of the child, the algorithm checks whether the method or attribute is within *listElems* (lines 3 to 6). If so, it means that the method or attribute is duplicated. Then, the *getWorstMA* method (line 7) compares the current method or attribute (*ma*) with that previously stored in *listElems* in relation to the feature modularization of the elements (class or interface) to which the duplicate method or attribute belongs. The one with the worst modularization is selected to be removed from the *offspring* (line 8). In this context, modularization of each element is evaluated taking into account the feature interlacing and feature-driven cohesion. If the method or attribute is not in *listElems*, it is added to the list (line 10).

Algorithm 2: RemoveDuplicate Method

```
   Input: offspring
   Output: offspring
 1 listElems ← empty List;
 2 elementsOffspring ← offspring.getAllElements();
 3 for each element ∈ elementsOffspring do
 4     MAElements ← element.getAllMethodsAndAttributes();
 5     for each ma ∈ MAElement do
 6         if ma ∈ listElems then
 7             worstMA ← offspring.getWorstMA(ma, lstElems[ma]);
 8             offspring.removeMA(worstMA)
 9             else listElems.add(ma) ;
10         end if
11     end for
12 end for
```

Hence, the application of the proposed methods leads to the generation of complete and consistent solutions as the *Diff* method guarantees the inclusion of missing elements and the *removeDuplicate* method eliminates the duplicate elements in a solution.

4.2 Feature-driven Crossover

The Feature-driven Crossover operator aims at improving the feature modularization of a PLA design. This operator is motivated by the fact that a PLA with low feature modularization is likely to suffer early modifications to accommodate new products, making it difficult to maintain the design stability over time. Figure 1 depicts an illustrative example of this operator, where *Parent1* is represented by gray elements and *Parent2* by white elements, on the left side of the picture. The features that are realized by each element (*class*) are shown by stereotypes (e.g. «F_n»). The feature «$F1$» was chosen to be swapped between the parents. The elements of *Parent1* and *Parent2* that are not associated with «$F1$» are copied to *Child1* and *Child2*, respectively. The operator then reallocates the elements of *Parent1* associated with «$F1$» to *Child2* and the elements of *Parent2* to *Child1*. Due to lack of space, classes' details and other architectural elements were omitted.

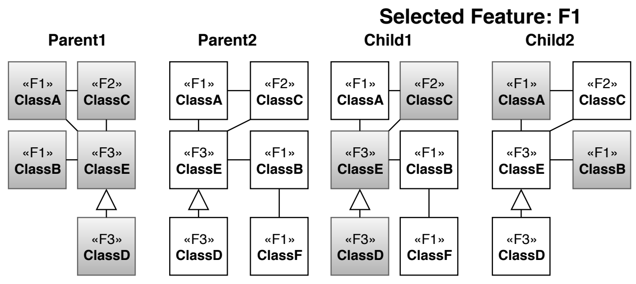


Figure 1: Feature-driven Crossover.

The Feature-driven Crossover pseudocode is presented in Algorithm 3. Initially, a list of all features of *parent1* is added in F (line 1). A feature *fx* is randomly selected from F (line 2). Then, all elements associated with *fx* of both parents are collected and placed in *c1* and *c2*. The two children are initialized as copies of *parent1* and *parent2* (lines 5 and 8). All elements of *offspring1* and *offspring2* associated with the *fx* feature are removed (lines 6 and 9). Then all elements (and their relationships) associated with *fx* and that belong to *parent2* are added in *offspring1* (line 7). Similarly, all elements of *parent1* associated with *fx* are added to *offspring2* (line 10). The improvements that we have made in this operator are highlighted in bold in lines 11-16. They consist in the calls for *Diff* and *RemoveDuplicate* methods. The *Diff* Method is applied in *offspring1* and *offspring2* regarding the parents to recover the elements potentially lost during the crossover operation (lines 11-12 and 15-17). Thus, the method to remove duplicate elements in *offspring1* and *offspring2* is applied (lines 13 and 16). Hence, the *Diff* and *RemoveDuplicate* methods solve the problem of consistency of the original version of Feature-driven Crossover [1] regarding missing methods and attributes and duplicated elements. Finally, the variabilities' links of each child are updated (lines 17-18).

4.3 Complementary Crossover

The main goal of the Complementary Crossover is to create offspring from parents that are complementary in different good fitness values. To do that, at the beginning of a generation cycle, all individuals are sorted in ascending order based on their objective fitness values. As MOA4PLA is a multi-objective approach, lists are created in the same proportion as the number of objectives, each one storing information related to a particular objective. Hence, if n objectives are selected, n lists will be created. For each list, the individuals are distributed in a decreasing way in relation to the fitness ranking. Then, two lists are randomly selected. Each parent is picked from one list by applying the standard roulette wheel selection, preferably selecting the best ones for each objective.

Algorithm 3: Feature-driven Crossover

```
Input: parent1, parent2
Output: offspring1, offspring2
1  F ← parent1.getAllFeatures();
2  fx ← a randomly selected feature from F;
3  c1 ← parent1.getGetAllElementsAssociatedWithFeature(fx);
4  c2 ← parent2.getGetAllElementsAssociatedWithFeature(fx);
5  offspring1 ← new Solution(parent1);
6  offspring1.removeElementsRealizingFeature(c1,fx);
7  offspring1.addElementsRealizingFeature(c2,fx);
8  offspring2 ← new Solution(parent2);
9  offspring2.removeElementsRealizingFeature(c2,fx);
10 offspring2.addElementsRealizingFeature(c1,fx);
11 diff (parent1, offspring1);
12 diff (parent2, offspring1);
13 removeDuplicate (offspring1);
14 diff (parent1, offspring2);
15 diff (parent2, offspring2);
16 removeDuplicate (offspring2);
17 offspring1.updateVariabilities();
18 offspring2.updateVariabilities();
```

Figure 2 illustrates the operation of the Complementary Crossover. To avoid duplication of elements in the children, the crossover point is determined only at one parent. In the picture, the crossover point (CP) is defined in *Parent1*, where the white part precedes the crossover point and the hachured part appears after the crossover point. *Parent2* is represented in green. Firstly, the child receives the part that precedes the crossover point of *Parent1* (white part in the child). Secondly, it receives elements from *Parent2* that are not yet in the child (green part in the child). Finally, the child receives the part after the crossover point (hachured part in the child) that is not yet in the child. In this way, the completeness and consistency of the generated solution is guaranteed.

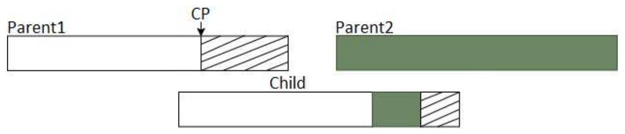


Figure 2: Complementary Crossover.

As the solution representation of a PLA design is not suitable to apply a crossover point, three lists are created to different kind of architectural elements (packages, classes and interfaces). Thus, it is necessary to create a crossover point for each one of them, that makes possible to divide an individual (representation of a PLA design) into two parts.

The pseudocode of our Complementary Crossover is presented in Algorithm 4. One of the two parents is randomly selected (*chosenParent* in line 2). A randomly crossover point is established in *chosenParent* called *cp* (line 3). Then, the child receives all elements of *chosenParent* located before the crossover point (line 4). After, the child receives elements from the other parent that are not yet in the child (line 5). Finally, the *chosenParent* elements that the child does not have yet, located after the crossover point, are added to the child (line 6). The operations performed in the lines 5 and 6 are the *Diff* method, adding elements from parents that aren't in the child. So there is no possibility of duplicate methods and attributes. Therefore, it is not necessary to apply the *removeDuplicate* method. In the last line, the variabilities' links are updated.

Algorithm 4: Complementary Crossover

```
Input: parent1, parent2
Output: offspring
1 offspring ← new Solution;
2 chosenParent ← a randomly selected parent;
3 cp ← a randomly selected point from chosenParent;
4 offspring ← all elements of chosenParent that are before cp;
5 diff (otherParent, offspring);
6 diff (chosenParent, offspring);
7 offspring.updateVariabilities();
```

5 EXPERIMENTAL STUDY DESIGN

Next we describe details of the experimental study conducted to evaluate the presented crossover operators.

Subject PLA Designs. Two PLA designs were used in the experimental study: Arcade Game Maker (AGM) [16] and Mobile Media (MM) [3]. AGM was created by the Software Engineering Institute (SEI). It is composed of three arcade games: Brickles, Bowling and Pong. MM is a mobile application composed of features that handle with music, video, and photo for portable devices [21]. Table 1 presents architectural elements numbers of both PLA designs.

Experimental Configurations. Four experimental configurations were considered in our study, named Exp_B, Exp_FdC, Exp_CC and Exp_All. Exp_B is the baseline experiment where only the six mutation operators of MOA4PLA are applied, as done in [2, 11]. Exp_FdC applies the Feature-driven Crossover and the six mutation operators. Exp_CC applies the Complementary Crossover and the six mutation operators. Exp_All applies both crossover operators and the six mutation operators.

Search-Based Algorithm. All experiments used NSGA-II [4]. We chose NSGA-II because it has been successfully used in many previous work [1, 2]. The objective functions were the same for all experiments: COE, ACLASS and FM (defined in Section 3), similarly to [11]. The parameters of NSGA-II were set up as described next.

NSGA-II Parameter Settings. To define the best configuration of crossover and mutation rates for each experiment we performed a calibration process. For the calibration NSGA-II was initially set with a population size of 100 individuals and 30,000 fitness evaluation (300 generations), which was the stopping criterion. The mutation operator was tuned by considering three rates: 0.7, 0.8, 0.9. For the crossover operator we considered four rates: 0.2, 0.4, 0.6, 0.8.

Table 1: PLA information

PLA	# Components	# Interfaces	# Classes	# Mandatory Features	# Variable Features
AGM	9	14	30	6	5
MM	8	15	14	7	7

Hence, for Exp_FdC, Exp_CC and Exp_All, 12 configurations were compared, which are combinations among the possible values. For Exp_B, that uses only mutation, 3 configurations were compared, one for each value of mutation rate. The calibration was performed for AGM and MM, executing 10 independent runs for each PLA in every configuration. The hypervolume of the Pareto fronts obtained for each configuration was used for the statistical comparison.

Considering statistical difference among the evaluated rates, the best calibration setting for all experimental configurations was crossover rate of 0.4 and mutation rate of 0.8. Recall that for Exp_B only mutation operator is applied. These operator rates and the same population size (100) and generations number (300) of the calibration were adopted for the actual evaluation of each experiment, where 30 independent runs were performed.

Quality Indicators. The quality indicators adopted to compare the results obtained by the experiments in terms of spread and convergence are Hypervolume and Coverage [10]. Hypervolume is a n-dimensional volume between a Pareto front and a specific reference point [24]. The higher the hypervolume value, the greater the coverage area, reflecting a better front. The Pareto Front (PF) is formed by the points in the objectives space that correspond to the set of all non-dominated solutions. Coverage (C) measures the dominance between two sets of solutions [23]. C(PFa, PFb) represents a value between 0 e 1 according to how much the set PFb is dominated by set PFa. Similarly, we compare C(PFb, PFa) to obtain how much PFa is dominated by PFb. Value 0 for C indicates that the solutions of the former set do not dominate any element of the latter set; on the other hand, value 1 indicates that all elements of the latter set are dominated by elements of the former set.

Statistical Tests. Shapiro-Wilk statistical test [9] was used to investigate if the sample sets have normal distribution. In order to statistically compare the results found by hypervolume, Kruskal-Wallis Pairwise test was applied since there are four data sets with non-normal distribution. We adopt a confidence of 95% (p-value $\leq$0.05) in order to verify the statistical difference between the sample sets. To further analysis we also compute the effect size with the Vargha-Delaney's $\hat{A}_{12}$ measure [19].

6 RESULTS AND DISCUSSION

In the following we present the quantitative and qualitative analysis of the results and answer the posed research question.

6.1 Quantitative and Qualitative Analysis

Figure 3 presents the surface that the solutions found for AGM cover in the search space. Since the problem of PLA design optimization is a minimization, we can see in the graph that Exp_All has better solutions, covering the search space below the other fronts. In this same graph we can see that the solutions of Exp_FdC are concentrated in the middle of the other fronts.

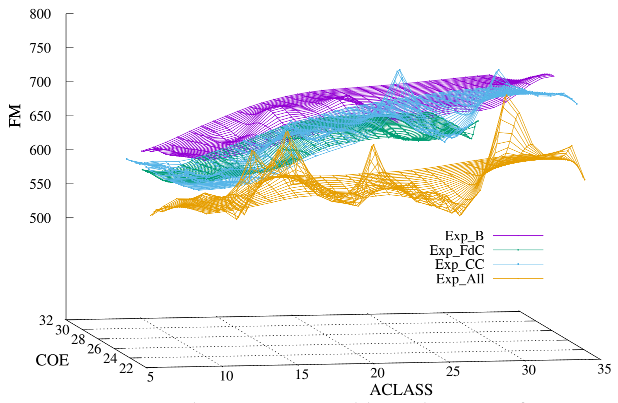


Figure 3: Search space covered by solutions of AGM.

Table 2: Results of the coverage indicator.

Experiment	AGM				MM			
	Exp_B	Exp_FdC	Exp_CC	Exp_All	Exp_B	Exp_FdC	Exp_CC	Exp_All
Exp_B	-x-	0.00	0.00	0.00	-x-	0.00	0.00	0.00
Exp_FdC	1	-x-	0.28	0.00	1	-x-	0.90	0.50
Exp_CC	1	0.66	-x-	0.00	1	0.42	-x-	0.00
Exp_All	1	1	1	-x-	1	0.66	0.90	-x-

Table 3: Results of the statistical test and the effect size of Hypervolume for the 30 independent runs of NSGA-II.

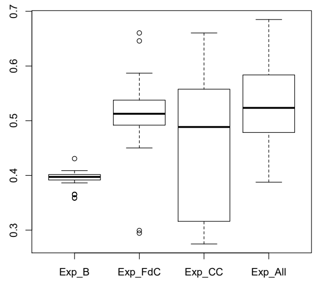
	AGM	MM
Experiments	p-value, effect size	p-value, effect size
Exp_B vs Exp_FdC	2.20E-05, 0.93 (large)	1.90E-10, 0.00 (large)
Exp_B vs Exp_CC	6.20E-02, 0.53 (negligible)	7.06E-01, 0.42 (small)
Exp_B vs Exp_All	4.70E-07, 0.97 (large)	3.80E-06, 0.13 (large)
Exp_FdC vs Exp_CC	1.40E-01, 0.43 (negligible)	1.60E-07, 0.95 (large)
Exp_FdC vs Exp_All	8.80E-01, 0.55 (negligible)	3.46E-01, 0.59 (small)
Exp_CC vs Exp_All	2.10E-02, 0.65 (small)	5.40E-04, 0.24 (large)

The results of coverage for AGM, presented in Table 2, corroborate the observation that the solutions obtained by Exp_All dominate every solution found by the other experiments (values 1 in the last row of the table). Solutions of Exp_CC dominate 66% of Exp_FdC solutions and the solutions of the latter dominate 28% of the solutions reach by the former. For MM, the solutions obtained by Exp_All are not dominated by solutions of Exp_CC and Exp_B, whereas half of its solutions are dominated by Exp_FdC. Solutions of Exp_FdC dominate almost all (90%) solutions obtained by Exp_CC. For both PLAs, Exp_All has the best performance and all solutions found by Exp_B are dominated by the other experiments.

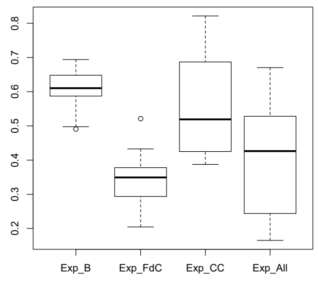
Regarding the hypervolume indicator, Table 3 presents the results of the Kruskal-Wallis test and the $\hat{A}_{12}$ effect size measure for the 30 independent runs of NSGA-II. The values of the second and the third columns are presented as follows: p-value of Kruskal-Wallis, value for $\hat{A}_{12}$ (effect's magnitude). We highlighted in gray the cells where there is significant difference between the experiments.

For AGM, the statistical test pointed significant difference, with confidence of 95% (p-value $\leq$ 0.05), between Exp_B vs Exp_FdC, Exp_B vs Exp_All and Exp_CC vs Exp_All. The effect size measure presented relevant difference for the same cases, where for the former two ones the magnitude is large and for the latter one the magnitude is small. To observe which of the experiments are better, we can analyze the boxplot of AGM presented in Figure 4(a). We can observe for this subject PLA that Exp_All is better than Exp_B and Exp_CC, and Exp_FdC is better than Exp_B.

Taking into account MM, the statistical testing pointed significant difference between the same experiments observed in AGM. In addition, there is also difference between Exp_FdC vs Exp_CC. Regarding the effect size, we can observe differences in all comparisons, with small magnitude in two cases, namely Exp_CC vs Exp_B and Exp_All vs Exp_FdC, and large magnitude for the other four cases. We noticed that the small magnitudes for effect size were pointed as not significant by the statistical test. To reason about the better experiments we can analyze the boxplot in Figure 4(b). Interestingly, according to effect size measure, the best values were obtained by Exp_B in comparison to all the other experiments. Exp_CC is better than Exp_FdC and Exp_All. Finally, Exp_All is better than Exp_FdC.



(a) AGM



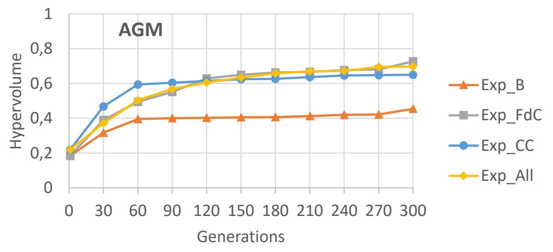
(b) MM

Figure 4: Boxplot of the hypervolume.

In an additional analysis of the boxplots presented in Figure 4, we can see that, for both PLAs, the Exp_B and Exp_FdC seem to behave inversely proportional. The operator Exp_CC reach values of hypervolume that seem to be more stable independent of the characteristics of the subject PLAs. As expected, Exp_All behaves as a combination of Exp_FdC and Exp_CC, reaching good results.

The summary of the hypervolume results is that the experiments that used the Feature-driven Crossover (Exp_FdC and Exp_All) performed better for AGM whereas the experiments that did not apply this operator were better for MM (Exp_B and Exp_CC). Thus, we performed an in-depth analysis in the AGM and MM original designs to understand the divergence in the experiments results. We could observe that the features of AGM are better modularized than MM. Since the rationale of our crossover operators are based on extracting better characteristics (e.g. feature modularization) of parents and merge in the offspring, we can infer that this operator can perform better when the PLA has good feature modularization. On the other hand, the mutation operators are responsible for making changes that improve the characteristics of the solutions.

Figures 5 and 6 present the behavior of each experiment over generations, in terms of hypervolume and FM fitness values, respectively. The slower convergence of the Exp_FdC and Exp_All experiments for MM corroborates our analysis that the original design of MM has worse feature modularization than AGM. In addition, the results of Exp_All are always the best (except in Figure 5 for AGM, where Exp_All achieved the second best value in the last generation), pointing that jointed application of Feature-driven Crossover and Complementary Crossover is more profitable.



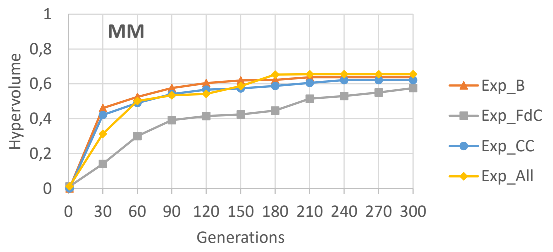
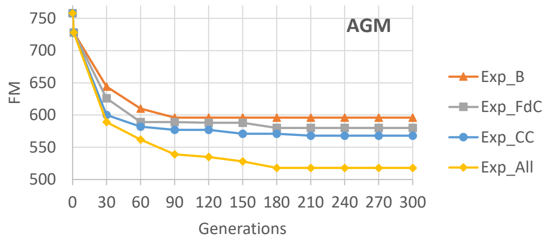


Figure 5: Hypervolume of each experiment over generations (the higher value, the better result).



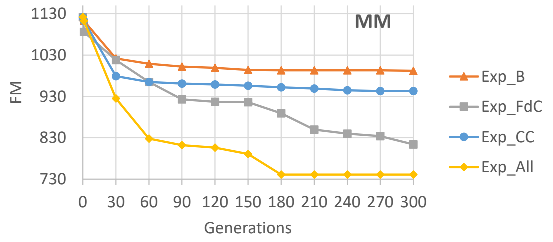


Figure 6: FM values over generations (the lower value, the better result).

To complement this evidence, we compared the solutions obtained by Exp_All and Exp_B that have the best trade-off among the objectives (solutions with the lowest Euclidean distance to the ideal solution [22]), whose values of fitness are presented in Table 4. The qualitative analysis was based on the architectural elements of the original PLAs compared to the elements of the obtained solutions, in terms of the properties optimized during the search process: feature modularization, relational cohesion and class coupling.

Table 4: Fitness values of the best trade-off solutions

PLA	Original PLA design (ACLASS, COE, FM)	Exp_B (ACLASS, COE, FM)	Exp_All (ACLASS, COE, FM)
AGM	(30, 26, 758)	(10, 32, 596)	(14, 28, 518)
MM	(14, 28, 1122)	(10, 35, 992)	(10, 30, 741)

During the analysis we noticed that the main changes are related to the improvement of feature modularization in accordance with the significant changes in FM objective values. For AGM, the packages `Package37278Ctrl` and `Package37359Ctrl` were added in the solution obtained by Exp_All to modularize the `movement` and `configuration` features, respectively. Originally, `movement` was only in the `GameBoardCtrl` package interlaced with many other features. In addition, one interface to realize the `play` feature was created in the `GameBoardMgr` package, leading to a better modularization of this feature in the obtained solutions than in the original PLA. Taking into account the solution obtained by Exp_B, the `movement` and `ranking` features were modularized into new packages, on the other hand, `configuration` and `play` are not so well modularized as in the solution obtained by Exp_All.

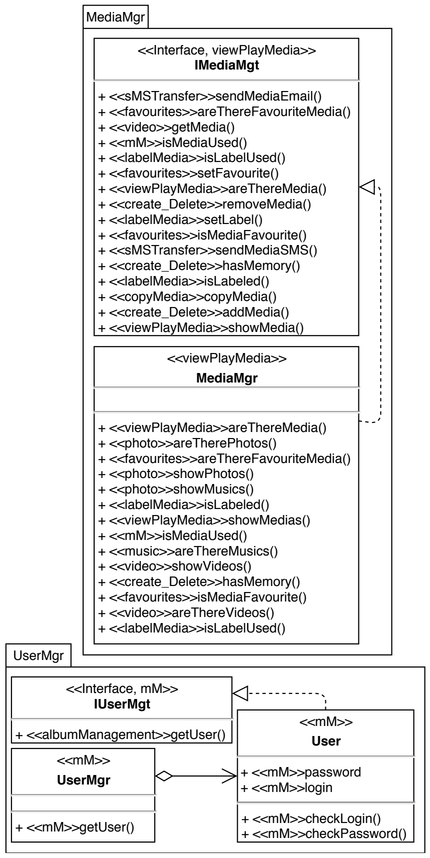
For MM, a greater improvement was noticed in the FM value of the solution obtained by Exp_All (around 30% - from 1122 to 741). In comparison with AGM, much more packages were created to modularize features in MM, what is expected as the original design is not well modularized. For the sake of illustration, Figure 7 depicts the elements associated with the `viewPlayMedia` and *mM* features in the original design (Figure 7(a)) and in the solutions obtained by Exp_B (Figure 7(b)) and Exp_All (Figure 7(c)). Features are assigned to each architectural element using stereotype (e.g. «*mM*»). Several relationships were omitted due to lack of space. Element names such as `Interface127853` were automatically generated and should be replaced by the architect before the PLA adoption.

In the solution of Exp_All, `viewPlayMedia` was modularized in an existing package in the original PLA, called `MediaMgr`. This package has an interface and a class (`MediaMgr` and `IMediaMgt`, respectively), both exclusively realizing `viewPlayMedia`. When compared to the original PLA, the `MediaMgr` package as well as its interface (`IMediaMgt`) contained many other architectural elements associated with other features, what represents high feature interlacing and low feature-driven cohesion. In the solution obtained by Exp_B, `viewPlayMedia` is less interlaced with other features in the `MediaMgr` package than in original design, however it is worse than in the solution of Exp_All. The `UserMgr` package of the three solutions shows the modularization of the `mM` feature. The solution obtained by Exp_B has the worst modularization of this feature. The interface named `Interface127853` has bad feature-driven cohesion and the `mM` feature is interlaced with `video` and `copyMedia`. This corroborates the importance of the presence of operators concerned with feature modularization to be applied during the search process in addition to the FM objective function.

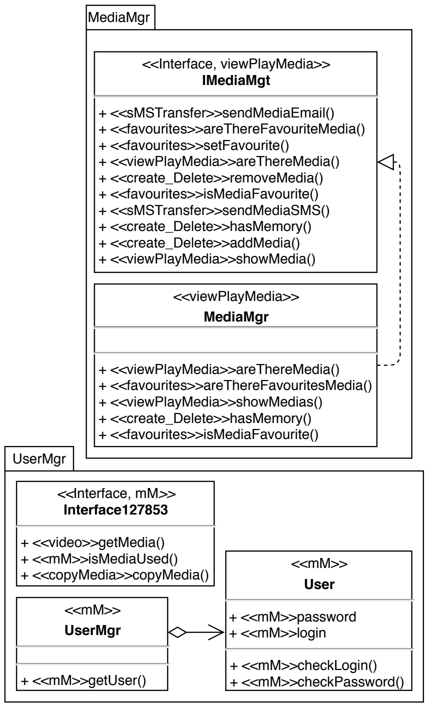
For both PLA designs and experiments, the main changes related to class coupling are due to the exclusion of some interfaces whose operations were modularized in other interfaces. Few other changes are similar: the movement of attributes and methods to different classes leading to the exclusion of some classes. The fitness values of COE presented in Table 4 show that the relational cohesion is compromised by the FM function. The COE values of the four obtained solutions are worse than the original value.

6.2 Answering the Research Question

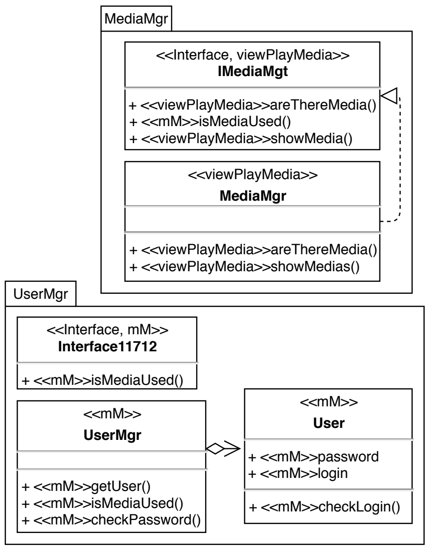
What is the benefit of using crossover operators in the optimization of PLA design in comparison to the state of the art?


MediaMgr
<<Interface, viewPlayMedia>>
IMediaMgt
+ <<sMSTransfer>>sendMediaEmail()
+ <<favourites>>areThereFavouriteMedia()
+ <<video>>getMedia()
+ <<mM>>isMediaUsed()
+ <<labelMedia>>isLabelUsed()
+ <<favourites>>setFavourite()
+ <<viewPlayMedia>>areThereMedia()
+ <<create_Delete>>removeMedia()
+ <<labelMedia>>setLabel()
+ <<favourites>>isMediaFavourite()
+ <<sMSTransfer>>sendMediaSMS()
+ <<create_Delete>>hasMemory()
+ <<labelMedia>>isLabeled()
+ <<copyMedia>>copyMedia()
+ <<create_Delete>>addMedia()
+ <<viewPlayMedia>>showMedia()
<<viewPlayMedia>>
MediaMgr
+ <<viewPlayMedia>>areThereMedia()
+ <<photo>>areTherePhotos()
+ <<favourites>>areThereFavouriteMedia()
+ <<photo>>showPhotos()
+ <<photo>>showMusics()
+ <<labelMedia>>isLabeled()
+ <<viewPlayMedia>>showMedias()
+ <<mM>>isMediaUsed()
+ <<music>>areThereMusics()
+ <<video>>showVideos()
+ <<create_Delete>>hasMemory()
+ <<favourites>>isMediaFavourite()
+ <<video>>areThereVideos()
+ <<labelMedia>>isLabelUsed()
UserMgr
<<Interface, mM>>
IUserMgt
+ <<albumManagement>>getUser()
<<mM>>
UserMgr
+ <<mM>>getUser()
<<mM>>
User
+ <<mM>>password
+ <<mM>>login
+ <<mM>>checkLogin()
+ <<mM>>checkPassword()

(a) Original MM design


MediaMgr
<<Interface, viewPlayMedia>>
IMediaMgt
+ <<sMSTransfer>>sendMediaEmail()
+ <<favourites>>areThereFavouriteMedia()
+ <<favourites>>setFavourite()
+ <<viewPlayMedia>>areThereMedia()
+ <<create_Delete>>removeMedia()
+ <<favourites>>isMediaFavourite()
+ <<sMSTransfer>>sendMediaSMS()
+ <<create_Delete>>hasMemory()
+ <<create_Delete>>addMedia()
+ <<viewPlayMedia>>showMedia()
<<viewPlayMedia>>
MediaMgr
+ <<viewPlayMedia>>areThereMedia()
+ <<favourites>>areThereFavouritesMedia()
+ <<viewPlayMedia>>showMedias()
+ <<create_Delete>>hasMemory()
+ <<favourites>>isMediaFavourite()
UserMgr
<<Interface, mM>>
Interface127853
+ <<video>>getMedia()
+ <<mM>>isMediaUsed()
+ <<copyMedia>>copyMedia()
<<mM>>
UserMgr
+ <<mM>>getUser()
<<mM>>
User
+ <<mM>>password
+ <<mM>>login
+ <<mM>>checkLogin()
+ <<mM>>checkPassword()

(b) Design obtained by Exp_B



(c) Design obtained by Exp_All

Figure 7: Excerpts of the obtained solutions with the best trade-off among the objectives for the MM PLA.

The empirical results of the experiments that applied the proposed crossover operators generated better PLA design solutions than the baseline experiment for AGM. For MM, whose original PLA has bad feature modularization, the application of the Complementary Crossover has tied with the baseline in terms of hypervolume, but it was better than the baseline in terms of coverage and feature modularization. The experiments that applied Feature-driven Crossover also found better solutions in terms of coverage and feature modularization than the baseline for MM. The method that verifies the completeness of the generated solutions is efficient to guarantee the completeness of the solutions. All search operators do not exclude attributes or methods in the classes. We manually verify several solutions searching for other inconsistencies, like unconnected or empty interfaces/classes, but none was found.

In summary, the benefit of using crossover operators in the PLA design optimization is to combine/merge parts already optimized in different individuals. These operators lead to more diversity in the population of potential PLA designs. In addition, crossover operators generated solutions with better feature modularization.

7 CONCLUDING REMARKS

In this paper we presented two crossover operators, namely an improved version of Feature-driven Crossover and the Complementary Crossover, with the goal of enhancing the search-based PLA design. The main purpose of both operators is to extract better characteristics (e.g. feature modularization) of different parents and merge in the offspring.

The methods proposed to guarantee the completeness and consistency of solutions are effective as no incomplete or inconsistent solutions was generated. Empirical results showed that in most of the cases there are significant differences among the use of only mutation (state of the art), and mutation with crossover. For AGM the crossover operators together with mutation operators were the best, and for MM the crossover operators achieved solutions with better feature modularization. Results also showed that the proposed operators complement each other as the experiment that applied both operators achieved better results.

Our ongoing work involves the application of both operators for other PLA designs, including an industrial one, aiming at increasing the sample of our empirical study. In future work we intend to perform a qualitative analysis of the generated solutions with practitioners in order to identify improvement needs.

ACKNOWLEDGMENTS

The work is supported by CNPq grants 428994/2018-0 and 408356/2018-9 and CAPES grant 448977/2019-01.

REFERENCES

[1] Thelma Elita Colanzi and Silvia Regina Vergilio. 2016. A feature-driven crossover operator for multi-objective and evolutionary optimization of product line architectures. *Journal of Systems and Software* 121 (2016), 126–143.

[2] Thelma Elita Colanzi, Silvia Regina Vergilio, Itana Gimenes, and Willian Nalepa Oizumi. 2014. A search-based approach for software product line design. In *Proceedings of the 18th International Software Product Line Conference (SPLC'14)*, Vol. 1. 237–241.

[3] Antonio C Contieri Jr, Guilherme G Correia, Thelma E Colanzi, Itana MS Gimenes, Edson A Oliveira Jr, Sandra Ferrari, Paulo C Masiero, and Alessandro F Garcia. 2011. Extending UML components to develop software product-line architectures: Lessons learned. In *Proceeding of the 5th European Conference on Software Architecture (ECSA)*. 130–138.

[4] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197.

[5] Édipo Luis Féderle, Thiago do Nascimento Ferreira, Thelma Elita Colanzi, and Silvia Regina Vergilio. 2015. OPLA-Tool: a support tool for search-based product line architecture design. In *Proceedings of the 19th International Software Product Line Conference*. 370–373.

[6] David E. Goldberg. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning* (1st ed.). Addison-Wesley Longman Publishing Co., Inc., USA.

[7] Mark Harman and Robert Hierons. 2002. A new representation and crossover operator for search-based optimization of software modularization. In *Proceedings of the Annual Conference on Genetic and Evolutionary Computation (GECCO'2002)*. 1351–1358.

[8] Mark Harman and Laurence Tratt. 2007. Pareto optimal search based refactoring at the design level. In *9th Annual Conference on Genetic and Evolutionary Computation*. ACM, New York, NY, USA, 1106–1113.

[9] Mirian T. Timpledon Lambert M. Surhone and Susan F. Marseken. 2010. *Shapiro-Wilk Test*. VDM Publishing. https://books.google.com.br/books?id=LsG-cQAACAAJ

[10] Miqing Li and Xin Yao. 2019. Quality Evaluation of Solution Sets in Multiobjective Optimisation: A Survey. *Comput. Surveys* 52 (03 2019), 1–38. https://doi.org/10.1145/3300148

[11] João Choma Neto, Cristiano Herculano da Silva, Thelma Elita Colanzi, and Aline Miotto Amaral. 2019. Are Memetic Algorithms profitable to search-based PLA design? *IET Software* 13, 6 (2019), 587–599. DOI: 10.1049/iet-sen.2018.5318.

[12] Camila Nunes, Uirá Kulesza, Cláudio Sant'Anna, Ingrid Nunes, Alessandro Garcia, and Carlos Lucena. 2009. Assessment of the Design Modularity and Stability of Multi-Agent System Product Lines. *Journal of Universal Computer Science* 15, 11 (jun 2009), 2254–2283.

[13] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology* 64 (2015), 1 – 18.

[14] Outi Räihä, Kai Koskimies, and Erkki Mäkinen. 2009. Empirical Study on the Effect of Crossover in Genetic Software Architecture Synthesis. In *Proceedings of World Congress on Nature and Biologically Inspired Computing*. 619–625.

[15] Outi Räihä, Kai Koskimies, and Erkki Mäkinen. 2010. Complementary crossover for genetic software architecture synthesis. In *Proceedings of the 10th International Conference on Intelligent Systems Design and Applications*. 266–271. DOI:10.1109/ISDA.2010.5687255.

[16] SEI. 2009. Arcade Game Maker pedagogical product line. (2009). https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=485941

[17] Christopher Simons, Ian C. Parmee, and Rhys Gwynllyw. 2010. Interactive, Evolutionary Search in Upstream Object-Oriented Class Design. *IEEE Transactions on Software Engineering* 36, 6 (nov.-dec. 2010), 798 –816.

[18] Frank J Van der Linden, Klaus Schmid, and Eelco Rommes. 2007. *Software product lines in action: the best industrial practice in product line engineering*. Springer Science & Business Media.

[19] András Vargha and Harold D. Delaney. 2000. A critique and improvement of the CL common language effect size statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics* 25, 2 (2000), 101–132.

[20] Yenisei Delgado Verdecia, Thelma E. Colanzi, Silvia R. Vergilio, and Marcelo C. Benitez Santos. 2017. An Enhanced Evaluation Model for Search-Based Product Line Architecture Design. In *Proceedings of the XX Ibero-American Conference on Software Engineering (CIbSE - ICSE 2017)*. 155–168.

[21] Trevor J Young. 2005. *Using aspectJ to build a software product line for mobile devices*. Master's thesis. University of British Columbia.

[22] Milan Zeleny and James L Cochrane. 1973. *Multiple criteria decision making*. University of South Carolina Press.

[23] Eckart Zitzler and Lothar Thiele. 1998. Multiobjective optimization using evolutionary algorithms — A comparative case study. In *Parallel Problem Solving from Nature — PPSN V*, Agoston E. Eiben, Thomas Bäck, Marc Schoenauer, and Hans-Paul Schwefel (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 292–301.

[24] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M Fonseca, and Viviane Grunert Da Fonseca. 2003. Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Trans. Evol. Comput.* 7, 2 (2003), 117–132.