This paper will be presented at GECCO-2017, Berlin, Germany

Mining Cross Product Line Rules with Multi-Objective Search and Machine Learning

Safdar Aqeel Safdar[1], Hong Lu[1], Tao Yue[1,2], Shaukat Ali[1]
[1]Simula Research Laboratory, Oslo, Norway
[2]University of Oslo, Oslo, Norway
{safdar, honglu, tao, shaukat}@simula.no

ABSTRACT

Nowadays, an increasing number of systems are being developed by integrating products (belonging to different product lines) that communicate with each other through information networks. Cost-effectively supporting Product Line Engineering (PLE) and in particular enabling automation of configuration in PLE is a challenge. Capturing rules is the key for enabling automation of configuration. Product configuration has a direct impact on runtime interactions of communicating products. Such products might be within or across product lines and there usually don't exist explicitly specified rules constraining configurable parameter values of such products. Manually specifying such rules is tedious, time-consuming, and requires expert's knowledge of the domain and the product lines. To address this challenge, we propose an approach named as SBRM that combines multi-objective search with machine learning to mine rules. To evaluate the proposed approach, we performed a real case study of two communicating Video Conferencing Systems belonging to two different product lines. Results show that SBRM performed significantly better than Random Search in terms of fitness values, Hyper-Volume, and machine learning quality measurements. When comparing with rules mined with real data, SBRM performed significantly better in terms of *Failed Precision* (18%), *Failed Recall* (72%), and *Failed F-measure* (59%).

CCS CONCEPTS

• Software and its engineering → Search-based software engineering; Software product lines

KEYWORDS

Rule Mining; Multi-Objective Search; Configuration; Machine Learning; Product Line.

1. INTRODUCTION

Product Line Engineering (PLE) is a well-acknowledged paradigm to improve the productivity of developing products with higher quality and at a lower cost. By benefiting from PLE, more and more systems are developed by integrating products, which belong to different product lines, and communicate and interact with each other through information networks [1, 2]. Examples of such systems include video conferencing systems (VCSs) [3] and material handling systems [4]. Such systems are highly configurable by presenting the users with configuration options. Consequently, at runtime, several products belonging to multiple product lines communicate (e.g., via information networks) with each other [1, 2] under various configurations. Thus, the runtime behavior of such systems not only depends on the configuration of these communicating products but is also influenced by the communication medium. Note that the configuration in our context indicates numerous configurable parameters exposed to users after the system is deployed.

Cost-effective PLE is challenging mainly because of the lack support of automation of the configuration process [5, 6]. Capturing rules is the key to enabling automation of various configuration functionalities (e.g., consistency checking, decision propagation, and decision ordering) [7-11]. In our context, such rules describe how configurations of communicating products belonging to different product lines influence their runtime interactions via information networks.

We name rules constraining configurations (values assigned to configurable parameters) of products belonging to different product lines as Cross Product Lines (CPL) rules. CPL rules are of significant importance for mainly two reasons. First, CPL rules can be used to identify invalid configurations where products may fail to interact with a confidence level due to, e.g., dependencies on external libraries and/or platforms. Identified invalid configurations can help developers to maintain current products or evolve future products. Second, CPL rules can provide support to enable (automated or semi-automated) configuration of products of future deployments. However, the literature does not provide sufficient support to mine such rules, as current practice mainly focuses on mining rules constraining product configurations within a single product line [6, 12].

CPL rules need to be captured by running the system due to the information only known at runtime, e.g., dependencies on external libraries and/or platforms. As mentioned in [13], rules that ensure correct runtime behaviors can be identified from either domain knowledge or testing of the system. Manually specifying such rules based on domain knowledge is tedious and time-consuming, and heavily relies on expert's knowledge of the domain and the product lines. Identifying CPL rules via testing has its own challenges, as the configuration space is typically very large and testing candidate configurations is often infeasible. Besides, in practice testers often use valid configurations to test the system [13]. Therefore, identifying CPL rules requires a dedicated approach that automatically obtains rules without exploring all possible configurations of the communicating products belonging to different product lines.

In [12], a rule mining approach is proposed that mines rules for a product line where product configurations belonging to one product line are generated randomly and labeled as faulty and non-faulty. Labeled product configurations are inputted to the classification algorithm of j48 [14] to mine rules. However, randomly generating configurations to mine rules is a brute-force way and time-consuming. In this work, we advance one step further by employing search to generate product configurations intelligently using three heuristics (Section 3.2), instead of randomly generating product configurations.

We propose an approach, named as Search-based Rule Mining (SBRM), which combines multi-objective search with machine-learning techniques, to mine CPL rules in an incremental and iterative way. SBRM obtains CPL rules with different degrees of confidence (i.e., the probability of being correct) with an emphasis on mining rules that can reveal invalid configurations

[15]. Instead of collecting a large amount of data required for machine learning all in once, we obtain the input data incrementally with multiple iterations. During each iteration, we use the rules mined from the previous iteration to guide the search for generating configuration data for the current iteration. The generated configuration data are combined together with those from all the previous iterations in order to incrementally refine the aforementioned rules. SBRM is validated using a real world case study of VCSs, where two products belonging to different product lines communicate (i.e., call) with each other.

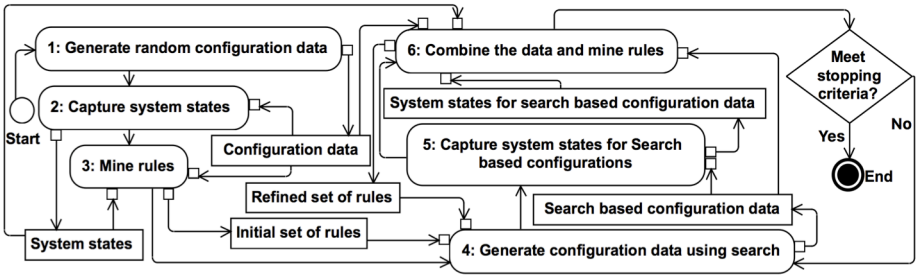

Figure 1: Overview of the proposed approach (SBRM)

We summarize the key contributions of the paper below:

- SBRM to mine CPL rules constraining configurations of communicating products across product lines.
- Three objectives to guide the search for generating configuration data in order to refine CPL rules.
- Evaluating SBRM by performing a real world case study of two communicating VCSs belonging to different product lines. With the case study, we compared the performance of NSGA-II with Random Search (RS) using fitness values, Hyper-Volume (HV), and machine learning quality measurements. Additionally, we compared the rules mined using SBRM with the rules mined with real data extracted from test case execution logs.

Evaluation results show that SBRM is effective to produce high-quality rules as compared to RS based rule mining approach (i.e., called RBRM). Results also indicate that SBRM produces better rules as compared to the rules mined based on real data extracted from test case execution logs.

The rest of the paper is organized as follows: In Section 2, we give an overview of SBRM followed by the search-based approach for generating configuration data in Section 3. In Section 4, we present the experiment design, execution, and results. Section 5 summarizes the literature review and finally, in Section 6, we conclude the work.

2. OVERVIEW

Figure 1 presents an overview of our proposed approach (SBRM), which relies on machine learning and multi-objective search to mine CPL rules. At the first step, an initial set of configuration data is generated randomly for the selected products belonging to different product lines. At the second step, selected products are configured with the randomly generated configuration data, and certain functionalities of the products are executed such that the selected products interact with each other via information networks (e.g., the Internet), and the states of the system are captured to know if they interact via communication network successfully. An interaction, in our context, can be defined as an action in which two or more objects (e.g., system, product, or component) are collaborating, communicating, or influencing each other. There does not exist a generic way of enabling interactions among various products of a system via communication networks as well as capturing the system states as it depends on the application domain of the system under study and its involved functionalities.

In step 3, we feed the set of generated configuration data (as Attributes) and their corresponding system states (as Classes) to Weka [14] as the initial input and apply the Pruning Rule-Based Classification algorithm (PART) [15] to mine the initial set of rules, which are consequently fed to NSGA-II for generating configuration data for the next iteration in step 4. Though C4.5 and RIPER are the two well-known algorithms, which generate rules based on decision trees [14, 15], C4.5 is expensive in terms of computation time since the process of generating/pruning rules is complex and requires global optimization. In the case of RIPPER, it suffers from over-pruning (hasty generalization) problem [16]. PART [15] combines these two paradigms while avoiding their shortcomings by generating partially pruned decision trees and inducing one rule corresponding the longest branch of each partial tree. In step 5, we repeat step 2 but take the configuration data generated from the search instead of the random one. In step 6, we combine all the configuration data generated from steps 1 and 4 and collected system states captured from steps 2 and 5, and feed all the data to Weka to mine a refined set of rules. This rule set is then used in the next iteration (starting from step 4) to generate more configuration data and further refine the rules.

In each iteration, newly generated configuration data with collected system states are added to the dataset from the previous iteration to mine a new set of rules. We repeat the process until we meet the stopping criteria, e.g., a fixed number of iterations and/or when the rules generated from two consecutive iterations are similar. Fixed number of iterations is useful when we have limited available resources for mining rules. Getting similar rules from consecutive iterations indicates that it is very unlikely to refine the rules further. We consider step 4, i.e., using search to generate configuration data, as the innovative part of the whole approach, i.e., SBRM. This is because using Weka to mine rules is a simple application of the PART algorithm and applying search requires carefully designing a fitness function. Therefore, in Section 3, we present how search is used for generating configuration data (step 4) and the evaluation of SBRM is presented in Section 4.

3. SEARCH-BASED APPROACH

Sections 3.1 presents definitions required to define the configuration data generation problem. Section 3.2 presents the objectives and measures, followed by the fitness function defined in Section 3.3.

3.1 Definition and Problem Representation

$CP = \{cp_1, cp_2, .., cp_{ncp}\}$ represents a set of configuration parameters with the total number being ncp. For each cp_i, CPV_i represents a set of possible values: ncpv is the total number of unique values (i.e., configuration space) for all the configuration parameters, which can be calculated as: $ncpv = \left|\left(\bigcup_{i=1}^{ncp} CPV_i\right)\right|$. Figure 2 shows four sanitized configuration parameters (cp1-cp4) from our case study. For example, cp1 represents the protocol (e.g., related to video conference over IP networks) of product P1, which can be configured with four different values (e.g., Pro-1).

```
cp1: P1_Protocol {Pro-1, Pro-2, Pro-3, Pro-4}
cp2: P1_Encryption {Enc-1, Enc-2, Enc-3}
cp3: P2_Encryption {Enc-1, Enc-2, Enc-3}
cp4: P2_ListenPort {LP-1, LP-2}
r1: P1_Protocol = Pro-2 AND P2_ListenPort = LP-2: Failed
r2: P1_Protocol = Pro-3 AND P2_Encryption = Enc-3: Connected
r3: P1_Encryption = Enc-1 AND P2_Encryption = Enc-2: Failed
```

Figure 2: Examples of sanitized configuration parameters and CPL rules

$R_N = \{r_{n1}, r_{n2}, r_{n3}, \dots, r_{nnr}\}$ represents nnr rules associated with normal states of the system, where the selected products interact as intended. $R_A = \{r_{a1}, r_{a2}, r_{a3}, \dots, r_{nar}\}$ represents nar rules related with abnormal states of the system where interactions between the selected products interact unexpectedly (Category-III). $Cf(r_i)$ represents the confidence of r_i, which is between 0 and 1. Confidence for a rule can be calculated as $Cf(r_i) = \frac{SP_i - V_i}{SP_i + V_i}$, where SP_i represents the number of instances for which r_i holds true (i.e., support) and V_i represents the number of instances that violate r_i (i.e., violation). An instance represents a set of values for configurable parameters of the selected products and corresponding system states. Based on confidence, support, and violation we further classify the R_N into two categories using two thresholds: High confidence rules (Category-I) where $Cf(r_i) > TH1$ and $(SP_i + V_i) > TH2$ and Low confidence rules (Category-II) where $Cf(r_i) \leq TH1$ or $(SP_i + V_i) \leq TH2$. Note that we used 0.9 (TH1) and 10 (TH2) for our experiment to classify CPL rules. Analyzing the effect of these thresholds on the performance of SBRM requires further investigation. In Figure 2, we present three sanitized CPL rules (r1-r3) mined for the case study. For example, r3 describes that if the encryptions of products P1 and P2 are set to Enc-1 and Enc-2 respectively, the call will fail. $S = \{s_1, s_2, \dots, s_{ns}\}$represents potential configuration solutions, where $ns = \prod_1^{ncp}(|CPV_i|)$, which is approximately 1.03^{e33} for our case study. Each solution s_j has a set of configuration values for ncp configuration parameters such that $s_j = \{cpvs_{j1} \dots cpvs_{jncp}\}$. $E_j = \{e_1, e_2, \dots, e_{ne}\}$ is a set of effectiveness measures for evaluating solution s_j.

We can then formulate the configuration generation problem as searching a non-dominant solution set S_R from ns solutions to obtain the highest effectiveness.

$$\forall_{s_r \in S_R} \forall_{i=1 \text{ to ns}} \forall_{j=1 \text{ to ne}} \exists \text{ Effect}(s_r, e_j) > \text{Effect}(s_i, e_j) \Lambda\ s_i \notin S_R \quad (1)$$

Effect (s_i, e_j) refers to the j^{th} effectiveness measure of solution s_i.

3.2 Objectives and Effectiveness Measures

The objectives are defined based on the three categories of rules (Section 3.1). Before presenting the objectives and effectiveness measures, we first define the distance function that is used to assess the effectiveness measures. The distance function indicates to what extent a configuration solution conforms to a rule.

$$D(r_i, s) = \frac{\sum_{i=0}^{CL} d(cl_i, cpv_i)}{MCL} \quad (2)$$

where $D(r_i, s)$ calculates the distance between rule r_i and solution s. In equation (2), $d(cl_i, cpv_i)$ calculates the branch distance between a clause cl_i from rule r_i and corresponding configuration value cpv_i from solution s. MCL is the maximum number of clauses in all the rules. To calculate the distance between cl_i and cpv_i as a branch distance, we use the distance calculation formula provided in [17, 18].

3.2.1 *Avoid configuration data satisfying or close to satisfying high confidence rules with normal states*

This objective is to avoid generating configuration data that completely or close to satisfy the rules in Category-I. The effectiveness measure (AHNS) corresponding to this objective can be calculated as:

$$AHNS(R_N, s) = \sum_{i=1}^{nnr} Cf(r_i) * D(r_i, s) \mid Cf(r_i) > TH1 \;\&\&\; (SP_i + V_i) > TH2 \quad (3)$$

where $AHNS(R_N, s)$ takes R_N (the set of rules related to the normal states) and one solution s as input and gives the effectiveness measure as output. To determine AHNS, we calculate the sum of weighted distances for all rules in Category-I, where the confidence of each rule is greater than threshold TH1 (i.e., 90%) and the sum number of support and violation instances for each rule is more than TH2 (i.e., 10). Weighted distance of r_i is calculated by multiplying $Cf(r_i)$ with $D(r_i, s)$.

3.2.2 *Generate configuration data satisfying or close to satisfying low confidence rules with normal states*

This objective is to generate configuration data within the configuration space that satisfy Category-II as well as its nearby space. The nearby space contains configuration data for which the distance to the rules in Category-II is close to 0 but not exactly 0. These configuration data might help to either improve the confidence of correct rules by increasing their support or filter out incorrect ones by increasing their violation and hence reducing their confidence. The effectiveness measure (NLNS) related to the second objective can be calculated as:

$$NLNS(R_N, s) = \sum_{i=1}^{nnr} Cf(r_i) * (1 - D(r_i, s)) \mid Cf(r_i) \leq TH1 \;||\; (SP_i + V_i) \leq TH2 \quad (4)$$

where $NLNS(R_N, s)$ takes R_N (the set of rules associated with the normal states) and solution s as input and outputs NLNS. Since we want to explore the configuration space near the configuration data satisfying the rules in Category-II, configuration data with a smaller distance to the rules in Category-II is preferred. Therefore, we use $(1 - D(r_i, s))$ in the $NLNS(R_N, s)$. To calculate NLNS, we calculate the sum of the weighted distance (i.e., calculated by multiplying $Cf(r_i)$ with $(1 - D(r_i, s))$) of a solution to all the rules in Category-II, where the confidence of each rule is less than or equals to TH1 (i.e., 90%) or the sum number of support and violation instances for each rule is less or equal to TH2 (i.e., 10).

3.2.3 *Generate configuration data satisfying or close to satisfying rules with abnormal states*

This objective is to generate configuration data within the configuration space that satisfy Category-III and its nearby space. The rules in Category-III are of high interest in our context because they indicate situations where interactions of the

Table 1: Overview of the experiment design*

RQ	Tasks	Description	Evaluation metrics	Algorithm's Parameters	Statistical tests
RQ1	T_1	Comparing fitness values and HV	– Individual objectives and Overall Fitness – HV	– Population size = 200 – maxEvaluations = 20K – Crossover rate = 0.9 – Mutation rate =1/(Total number of configuration parameters) – Total runs = 10	Man-Whitney U-test and Vargha and Delaney $\hat{A}_{12}$
RQ2 RQ3	T_2 T_3	Comparing rule sets based machine-learning quality measurements	– Accuracy (%) – F/C Precision (%) – F/C Recall (%) – F/C F-Measurement		

* F= Failed (Abnormal state), C=Connected (Normal state)

selected products fail. The effectiveness measure (NAS) for this objective can be calculated as:

$$NAS(R_A, s) = \sum_{i=1}^{nar} Cf(r_i) * (1 - D(r_i, s)) \quad (5)$$

where $NAS(R_A, s)$ takes rule set R_A (related to the abnormal states) and solution s as input. To calculate NAS, we calculate the sum of weighted distances for all the rules in R_A (Category-III).

3.3 Fitness Function

We first normalize the three effectiveness measures with $nor(F(x)) = \left(\frac{F(x)-F_{min}}{F_{max}-F_{min}}\right)$, where $F(x)$ is an effectiveness measure function, F_{max} and F_{min} are the maximum and minimum values of the effectiveness measure. For AHNS, F_{min} is 0 when the distance between all the rules in Category-I and solution s is 0. F_{max} can be calculated as $\sum_{i=1}^{nnr} Cf(r_i)$ where the distance between all the rules in Category-I and solution s is 1. For NLNS and NAS, F_{min} is 0 when the distance between all the rules in the corresponding category and solution s is 1. Corresponding to NLNS and NAS, F_{max} can be calculated as $\sum_{i=1}^{nnr} Cf(r_i)$ and $\sum_{i=1}^{nar} Cf(r_i)$ respectively, where the distance between all the rules and solution s is 0.

With the three effectiveness measures, we define the fitness function based on the three objectives as follow:

$$F(O_1) = 1 - Nor\ (AHNS(R_N, s)) \quad (6)$$

$$F(O_2) = 1 - Nor\ (NLNS(R_N, s)) \quad (7)$$

$$F(O_3) = 1 - Nor\ (NAS(R_A, s) \quad (8)$$

Note that, in the above equations, we define our search problem as a minimization problem by subtracting each normalized effectiveness measure from 1 to ensure that a solution with a value closer to 0 is better.

The fitness function with the three objectives is combined with NSGA-II to address the optimization problem. We implemented our problem in jMetal by encoding all the configuration parameters in the solution s as integer variables, where a variable cp_i holds a value cpv_{ij} such that $cpv_{ij} \in CPV_i$. Initially, all variables in s are initialized with random values. During the search, SBRM generates optimized solutions guided by the fitness function.

4. EVALUATION

We present experiment setup in Section 4.1, execution in Section 4.2, and results in Section 4.3. In Section 4.4, we present overall discussion and Section 4.5 presents threats to validity.

4.1 Experimental Setup

First, we present the experiment design including research questions (Section 4.1.1) followed by the case study (Section 4.1.2) and evaluation metrics (Section 4.1.3). Lastly, we present evaluation tasks, parameter settings, and statistical tests used for analysis (Section 4.1.4).

4.1.1 Research Questions

In SBRM, we apply commonly used NSGA-II [19-21] for generating configuration data as NSGA-II has proven to be effective for solving various software engineering problems such as test case prioritization and cost estimation [20, 22].

The goal of the evaluation is to assess if combining machine learning with NSGA-II in the rule mining process can improve the quality of rules. As RS is typically used as the comparison baseline [22, 23]; therefore, we investigate if NSGA-II is effective to solve the configuration generation problem and then compare the quality of rules produced from SBRM (with NSGA-II) with rules mined by RS based approach (i.e., called RBRM). To further assess the effectiveness of SBRM, we also compare rules mined from SBRM with rules mined from real data extracted from test case execution logs (i.e., called RDBRM). Thus, the evaluation is designed to answer the following three research questions:

RQ1. Is NSGA-II effective to solve the configuration generation problem as compared to RS?

RQ2. Does SBRM produce better quality rules than RBRM in terms of machine learning measurements?

RQ3. Does SBRM produce better quality rules than RDBRM in terms of machine learning measurements?

4.1.2 Case Study

Cisco Systems, Norway provides a variety of VCSs to facilitate high-quality virtual meetings [23]. Cisco has developed several product lines for VCS including C-Series, MX-Series, and SX-Series [3]. Each product from these different product lines has a large number of configuration parameters (e.g., Protocol and Encryption), which need to be configured before making calls. For each VCS we have a set of state variables representing the state of VCS (e.g., call status, camera connection status) that varies according to different hardware and software configurations. For our experiment, we used two real products C60 and MX300 developed by Cisco, which belong to C-series and MX-series, respectively. Simula Research Laboratory has a long-term collaboration with Cisco, Norway under Certus-SFI[1]. As part of our collaboration, we have access to several VCSs at our lab and thus we used these systems for our experiments. Therefore, our case study is real, but the experiment wasn't performed in the real industrial setting of Cisco.

For comparing the quality of rules produced using SBRM with ones mined by RDBRM, we obtained 9,989 test case execution logs from Cisco. Each test log contains a test case script and configurations and statuses representing the system states for all the products involved in the test case. The configurations and their corresponding system states (i.e., statuses) contained in the execution logs can be used to mine the rules. To extract the data, first, we obtained 3963 relevant (i.e., invoking the Dial command) logs from 9,989 test execution logs automatically, where the testing scenario is about making a call from one product to another. Second, corresponding to all relevant execution logs, we extracted configurations and statuses for the products involved in the test cases corresponding to execution logs. Finally, we use the extracted configurations and corresponding statuses to mine the rules.

4.1.3 Evaluation Metrics

To answer RQ1, we compared NSGA-II with RS in terms of the three objectives, and the overall fitness. Additionally, we also compared NSGA-II with RS in terms of HV, which is commonly used to measure the overall performance of multi-objective search algorithms (e.g., NSGA-II) [24]. HV is for obtaining the volume in the objective space covered by members of Pareto fronts for measuring both convergence and diversity [25].

To answer RQ2 and RQ3, we compared SBRM with RBRM and SBRM with RDBRM respectively, based on four machine-learning quality measurements (MLQMs): *Accuracy* of the classifier, *Precision*, *Recall*, and *F-measure* for each class (i.e., call status in our case), which are calculated with 10 times 10-fold cross-validation [26]. *Accuracy* indicates the overall performance of PART by specifying the percentage of instances that conforms to the mined rules [27], where one instance contains one specific set of configurations and its corresponding system states.

[1] www.certus-sfi.no

Precision represents the percentage of instances that are correctly classified divided by the total number of instances covered by rules associated with a specific system state (e.g., connected or failed in our case) [27]. For example, 98% *Precision* for the failed state means that, according to the mined rules, there are 2% of instances whose configurations are identified as invalid ones, which led to the failed state. But actually, they lead to the connected state. The *Recall* represents the percentage of instances that are correctly classified divided by the total number of instances corresponding to a particular system state [27]. For example, 90% *Recall* for the failed state means that configurations of 10% instances are not associated with the failed state according to the mined rules, but these instances actually lead to the failed state. *F-measure* is the harmonic mean of *Precision* and *Recall* [27].

4.1.4 Experimental Tasks, Parameter Settings, and Statistical Analysis

As shown in Table 1, we designed three tasks (T1-T3) for addressing RQ1-RQ3. T1 is to compare NSGA-II with RS in terms of HV, the three individual objectives, and the overall fitness. T2 and T3 are for comparing the quality of rules produced from SBRM with RBRM and RDBRM respectively, evaluated based on machine-learning quality measurements.

As shown in Table 1 (column 5), we used the default settings for NSGA-II as implemented in jMetal [28], which are typically recommended [29]. The single point crossover and bit-flip mutation, implemented in jMetal, were applied as crossover and mutation operators, respectively. The total number of configuration parameters is 17 for our case study. We used a population size of 200 where we select all the Pareto Non-dominated solutions for mining the rules. Since selecting the best set of parameters is application dependent [12], we used the default settings provided by Weka [14] for SBRM, RBRM, and RDBRM, which have been used in various contexts for applying the machine learning techniques [12, 30].

To compare SBRM (with NSGA-II) with RBRM and RDBRM, we use the non-parametric Mann-Whitney U-test as recommended in [31] using $\alpha = 0.05$ and the Vargha and Delaney's $\hat{A}_{12}$ statistics as an effect size measure [32]. For all MLQMs and HV, if $\hat{A}_{12}$ is less than 0.5, SBRM is better than RBRM/RDBRM, and a value greater than 0.5 means vice versa. Similarly, in the case of fitness values, if $\hat{A}_{12}$ is greater than 0.5, SBRM is better than RBRM otherwise RBRM is better than SBRM.

4.2 Experimental Execution

We selected the call status as the system state to classify the configurations. A failed call status represents the abnormal state and a connected call status represents a normal state. We selected the call functionality and its associated call status as it is the main functionality of a VCS and other functionalities depend on it.

To mine the initial set of the rules we randomly generate a set of 500 configurations corresponding to two selected products (i.e., C60 and MX300). To get the system state, we configure the two products with the generated configurations and make a call from product A to B for 20 seconds. We made the call for 20 seconds in order to give sufficient time for establishing the call connection. After waiting for 20 seconds we capture the call status and disconnect the call. We input these 500 configurations along with their corresponding system states to Weka [14] and apply PART [15] to mine the initial set of rules. To refine the rules, we use the initial set of rules to guide the search to generate 200 more configurations. To mine the refined set of rules we repeat the same process (i.e., configuring the products and making the call) to get the call status and mine a new set of rules based on 700 configurations (combining all the configurations generated so far) and corresponding system states. We repeat this incremental and iterative process for three iterations and mine the final set of rules based on a dataset containing 1100 configurations and their call statuses. We used three iterations as a stopping criterion. We also got more than 90% identical rules in the second and third iteration.

4.3 RESULTS AND ANALYSIS

In this section, we present the results of our evaluation and answer the research questions.

4.3.1 Effectiveness of search (RQ1)

To answer RQ1, from the results of the Man-Whitney U-test, we notice that p-values corresponding to all fitness values and HV are less than 0.05 showing a significant difference between NSGA-II and RS. $\hat{A}_{12}$ values corresponding to the three objectives are all greater than 0.5 and are less than 0.5 in the case of HV, which suggests that NSGA-II is significantly better than RS.

4.3.2 Comparing SBRM with RBRM (RQ2)

To answer RQ2, we compared SBRM and RBRM in terms of MLQMs based on rules from each iteration as well as overall (i.e., combined the results for all the three iterations) based on MLQMs (Section 4.1.3).

As shown in Table 2, for the first iteration, although all the $\hat{A}_{12}$ values indicate that SBRM has better performance for all the MLQMs, the *p* values show that the superiority of SBRM is not significant for all the MLQMs except for *Failed Recall*. In iteration-2, SBRM performed significantly better than RBRM with respect to *Accuracy*, *Failed Precision*, *Failed Recall*, and *Failed F-measure*. The results corresponding to iteration-3 and overall (Table 2) show that SBRM has performed significantly better than RBRM in terms of all the MLQMs. So, as moving from iteration-1 to iteration-3, SBRM starts to perform better than RBRM, which leads to the conclusion that SBRM produces better rules as compared to RBRM with respect to the MLQMs.

4.3.3 Comparing SBRM with RDBRM (RQ3)

To answer RQ3, we compared SBRM with RDBRM iteration-wise as well as overall (i.e., combined the values for all the three iterations) based on MLQMs (Section 4.1.3).

As shown in Table 3, the results related to all the MLQMs except for *Connected Recall* and *Connected F-measure* for all the iterations as well as overall show that SBRM performed significantly better than RDBRM. Results for *Connected Recall* corresponding to all the iterations as well as overall indicate that RDBRM performed significantly better than SBRM. In iteration-1, iteration-2, and

Table 2: Comparing the quality of rules produced with SBRM and RBRM – $\hat{A}_{12}$ and p-values for (RBRM VS SBRM)

Evaluation metric	Iteration-1		Iteration-2		Iteration-3		Overall		Overall Average	
	p-value	$\hat{A}_{12}$	p-value	$\hat{A}_{12}$	p-value	$\hat{A}_{12}$	p-value	$\hat{A}_{12}$	RBRM	SBRM
Accuracy	0.104	0.28	**0.010**	**0.16**	**0.002**	**0.10**	**<0.001**	**0.19**	95.7%	97.2%
Connected Precision	0.161	0.31	0.054	0.24	**0.026**	**0.20**	**0.002**	**0.27**	0.945	0.957
Connected Recall	0.173	0.32	0.150	0.31	**0.041**	**0.23**	**0.002**	**0.27**	0.955	0.971
Connected F-Measure	0.186	0.32	0.088	0.27	**0.025**	**0.20**	**0.001**	**0.25**	0.950	0.964
Failed Precision	0.063	0.25	**0.012**	**0.17**	**0.001**	**0.07**	**<0.001**	**0.19**	0.966	0.982
Failed Recall	**0.041**	**0.23**	**0.003**	**0.11**	**0.001**	**0.07**	**<0.001**	**0.16**	0.965	0.978
Failed F-Measure	0.104	0.28	**0.005**	**0.13**	**0.001**	**0.04**	**<0.001**	**0.18**	0.966	0.980

Table 3: Comparing the quality of rules produced with SBRM and RDBRM– $\hat{A}_{12}$ and p-values for (RDBRM VS SBRM)

Evaluation metric	Iteration-1		Iteration-2		Iteration-3		Overall		Actual values RDBRM
	p-value	$\hat{A}_{12}$	p-value	$\hat{A}_{12}$	p-value	$\hat{A}_{12}$	p-value	$\hat{A}_{12}$	
Accuracy	**<0.001**	**0.00**	**<0.001**	**0.00**	**<0.001**	**0.00**	**<0.001**	**0.00**	92.96%
Connected Precision	**0.001**	**0.10**	**<0.001**	**0.00**	**<0.001**	**0.00**	**<0.001**	**0.03**	0.934
Connected Recall	**<0.001**	**1.00**	**<0.001**	**1.00**	**<0.001**	**1.00**	**<0.001**	**1.00**	0.994
Connected F-Measure	0.418	0.40	0.418	0.60	**0.012**	**0.200**	0.135	0.400	0.963
Failed Precision	**<0.001**	**0.00**	**<0.001**	**0.00**	**<0.001**	**0.00**	**<0.001**	**0.00**	0.796
Failed Recall	**<0.001**	**0.00**	**<0.001**	**0.00**	**<0.001**	**0.00**	**<0.001**	**0.00**	0.260
Failed F-Measure	**<0.001**	**0.00**	**<0.001**	**0.00**	**<0.001**	**0.00**	**<0.001**	**0.00**	0.392

overall there is no significant difference between SBRM and RDBRM in terms of *Connected F-measure* whereas in iteration-3 SBRM outperformed RDBRM. Since for five out of the seven MLQMs, SBRM has performed significantly better than RDBRM whereas RDBRM outperformed SBRM in terms of *Connected Recall* only, it can be concluded that SBRM produces better rules than RDBRM.

4.4 Overall Discussion

For RQ1, we noticed that NSGA-II has outperformed RS in terms of HV, the three objectives as well as the combined. This suggests that our problem is not trivial and requires the search.

For RQ2 and RQ3, we observed that SBRM performed significantly better than RBRM and RSBRM for most of the MLQMs. This is because we guide the search using previously mined rules and generate specific configuration data that tend to either increase or decrease the confidence of a rule. In this way, SBRM converges more rapidly than RBRM to obtain high confidence rules. To further investigate the performance differences of SBRM with RBRM and RDBRM, we calculated the relative improvement (RI) due to SBRM for all MLQMs, across iterations. We calculated the RI with respect to RBRM as $RI\left(S(x_{ij}), R(x_{ij})\right) = \left(S(x_{ij}) - R(x_{ij})\right)$, where $S(x_{ij})$ and $R(x_{ij})$ give the average values corresponding to the i^{th} MLQM and j^{th} iterations for SBRM and RBRM, respectively. Similarly, to calculate RI with respect to RDBRM, we applied a similar formula as: $RI\left(S(x_{ij}), RD(x_i)\right) = \left(S(x_{ij}) - RD(x_i)\right)$, where $RD(x_i)$ gives the value of the i^{th} MLQM for RDBRM. Figure 3 and Figure 4 show the relative improvement in MLQMs due to SBRM in comparison to RBRM and RDBRM, respectively.

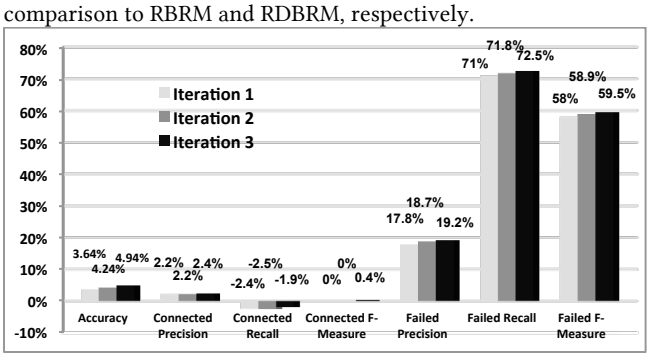

Figure 3: Relative improvement by SBRM in comparison to RDBRM

From Figure 3, one can observe that compared with RDBRM, the relative improvements of SBRM in terms of *Failed Precision*, *Failed Recall*, and *Failed F-measure* are much larger than the relative improvements of the other MLQMs, whereas it is negative in terms of *Connected Recall*. This can be justified by the fact that in SBRM we generate configurations that maximally conform to the rules with the abnormal state (i.e., the failed state). Also, we avoid generating configurations that conform to the high confidence rules with the normal state (i.e., the connected state), which justifies the negative RI value for *Connected Recall*.

Figure 4 shows that the relative improvement in MLQMs for SBRM as compared to RBRM is not large as it is in comparison to RDBRM, which is probably because the sample size used for mining the rules in SBRM and RBRM is small (i.e., 700, 900, and 1100 for iteration-1, iteration-2, and iteration-3, respectively). Moreover, in these small datasets, 500 initial configurations were the same across the datasets used for SBRM and RBRM, and only maximum 600 (i.e., in iteration-3) configurations were different. On the other hand, the relative improvement for SBRM with respect to RDBRM is large because the datasets used for RDBRM and SBRM were different. Also, the size of the dataset used for RDBRM was large (i.e., 3963). However, from Figure 4, we can observe an increasing trend of the relative improvement across the three iterations, suggesting that increasing the sample size can increase the relative improvement.

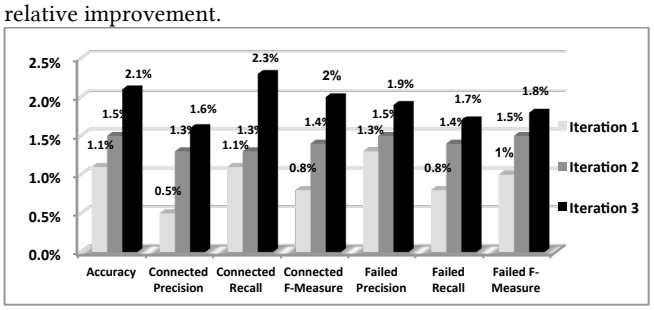

Figure 4: Relative improvement by SBRM in comparison to RBRM

4.5 Threats to Validity

The threat to internal validity of our study is the selection of parameter settings for the selected search algorithm, which may affect the performance of the algorithm. To mitigate this threat, we used default parameter settings, which have exhibited promising results [33]. Similarly, for the machine-learning algorithm, we also used default parameters settings, as selecting parameter settings is application dependent [12]. The threat to construct validity is the use of termination criteria for the search. We used the same stopping criterion (i.e., the number of fitness evaluations) for both NSGA-II and RS to find the optimal solutions. Another threat can be a selection of stopping criteria for the number of iterations and sample size used for mining the rules. We used three iterations and during each iteration added 200 more configurations to the dataset from the previous iteration due to practical challenges (i.e., the overall cost of the whole process was high particularly on executing configurations and getting corresponding call statuses, which was 50 seconds per configuration). To assess the effect of the sample size, the number of iterations, and different values for the thresholds used to classify the CPL rules, we plan to conduct dedicated empirical studies in the future.

The threat to conclusion validity is due to the random variation inherited in search algorithms. To minimize this threat, we repeated the experiment 10 times to reduce the effect caused by randomness, as recommended in [24, 29]. Moreover, we also applied the Mann-Whitney test to determine the statistical significance of the results and the Vargha and Delaney $\hat{A}_{12}$

statistics as the effect size measure, which are recommended for randomized algorithms [29]. The first threat to external validity is the selection of search algorithm for our study. To reduce this, we selected the most widely used NSGA-II algorithm that has shown promising results in different contexts [20, 22]. The second threat to external validity is the selection of algorithms for rule mining. To tackle this threat, we selected PART, which has proven to be more effective as compared to other well-known algorithms [15, 34]. The third threat to external validity is that we evaluated our approach using only one case study. To mitigate this, we used a real case study, the Cisco Video Conferencing Systems, which contains typical communicating products across multiple product lines. However, a generalization of the results requires additional experiments. In future, we plan to conduct an empirical study using several case studies to evaluate different search algorithms and machine-learning algorithms.

5. RELATED WORK

Search algorithms have been used to solve many problems in the context of PLE [35-37]. Since we are focusing on rule mining; therefore, we only discuss existing studies related to rule mining using machine-learning techniques in the context of PLE. In Section 5.1, we discuss dedicated approaches that focus on mining rules from different artifacts (e.g., source code, configuration file, feature model). In Section 5.2, we discuss approaches such as feature extraction, feature construction and feature recommendation, which mine crosstree constraints.

5.1 Dedicated Rule Mining Approaches

The work in [12] applies Binary Decision Tree-J48 (machine learning algorithm) to infer the constraints from a set of randomly generated product configurations. To classify the configurations as faulty and non-faulty, a computer vision algorithm was used as an oracle. To validate the approach, it was applied to an industrial video generator product line. Rules were evaluated based on expert's opinion and machine-learning measurements such as *Precision* and *Recall*. Results show that on average 86% *Precision* and 80% *Recall* rate can be achieved using the proposed approach.

In [38], Yi et al. proposed an approach to mine the crosstree binary constraints (i.e., requires, excludes) corresponding to a feature model. The approach takes a feature model as input containing the features, their descriptions, and some known crosstree binary constraints. First, it trains LIBSVM classifier (an extension of support vector machine) with existing crosstree binary constraints where the parameters of the classifier are optimized using the genetic algorithm to minimize the error rate of the classifier. Second, it extracts all the feature pairs, and finally, the optimized classifier finds the candidate features of binary constraints. The approach was validated using two feature models collected from SPLOT repository. Results show that rules with high *Recall* (i.e., close to 100%) and the variable low *Precision* (on average 42%) can be achieved using proposed approach.

In [39], another approach is presented for mining the crosstree constraints. It constructs configuration matrix (i.e., product-features matrix) from configuration files and extracts crosstree constraints using an association rule mining technique (i.e., Apriori algorithm). Rules are pruned using minimum support and minimum confidence thresholds. The approach was evaluated using a large-scale industrial software product line for embedded systems. The evaluation shows that a large number of rules with variable support (i.e., 80% to 99%) and confidence (i.e., 90% to 100%) can be identified. The majority of the rules were identified with support ranging from 80% to 85%.

In [40], an approach is presented to extract configuration constraints from existing C codebases using static analysis. It uses build time errors (e.g., pre-processor, parser, type, and link errors) as the oracle to classify the low-level system configurations (i.e., build and code files) and mine the constraints. To assess the accuracy of extracted rules, they were compared with the existing constraints specified in developer's created variability models. The approach was validated using four open source case studies (uClibc, BusyBox, eCos, and the Linux kernel). Results show that up to 19% of the total constraints can be recovered automatically from the source code, which assures successful build with the accuracy of 93%. In [13], an extension of [40] is presented in which the authors improved the static analysis and increased the recoverability rate by 9%. Additionally, an empirical study is also presented that identifies the sources of constraints.

5.2 Non-Dedicated Rule Ming Approaches

In [41], Czarnecki et al. proposed an extension of feature model called probabilistic feature model. To extract crosstree constraints from existing formally defined products, a rule mining process is presented that uses association-mining techniques to mine the constraints. The proposed mining process was applied on a small case study of Java Applets. Rules were evaluated based on machine-learning measurements (i.e., support and confidence).

In [42], an approach is proposed to model and recommend product features for any particular domain based on the product description provided by the domain expert. To mine association rules between product features, association rule mining techniques are applied to configuration matrix (i.e., product-features matrix). The proposed approach was validated with 20 different product categories using product descriptions available at SoftPedia [43]. Hariri et al. [44] extend the work presented in [42]. In [44], different clustering algorithms used to cluster the features and construct products by feature matrix were compared. The evaluation was also improved by applying the approach on diverse domains as well as a large project of a software suite for remote collaboration. Results show that rules with different *Precision* and *Recall* rates can be mined according to the threshold set for the confidence.

The work in [6] presents an approach to synthesize attributed feature models (AFM) from a set of product descriptions in the form of tables (i.e., configuration matrix). An algorithm is proposed that uses implication graph and mutex graph constructed from configuration matrix to extract the crosstree constraints. For extracting the relational constraints defined on values of attributes, the algorithm uses domain knowledge or selects the boundary values of attributes randomly when domain knowledge is not provided. The approach was validated using random configuration matrices as well as a real-world case study. Results show that the proposed algorithm can be used to mine a large number of rules for large-scale case studies.

Davril et al. [45] proposed an approach to construct a feature model automatically from informal product descriptions available over the Internet. To mine the implication rules of features, CFP-growth algorithm and Apriori algorithm are applied on configuration matrix (i.e., product-features matrix). The proposed approach was applied to a case study of antivirus software using the product descriptions available at SoftPedia [43].

5.3 Summary

All the approaches discussed above focus on mining binary crosstree constraints (requires and excludes) between different features of a product line or constraints on features' attributes. In our study, we focus on mining rules between configuration

parameters and system behaviors of interacting products across product lines. Additionally, we defined three objectives (Section 3.2) for generating configuration data, which are fed to the machine-learning tool in order to refine rules. To evaluate the quality of rules, all the approaches discussed above have used machine-learning quality measurements (e.g., *Precision*). We also evaluated the quality of constraints based on machine-learning quality measurements. Additionally, we also compared the rules produced using SBRM with the rules mined based on real data.

6. CONCLUSION

We presented an incremental and iterative approach (named as SBRM) for mining rules for configurations of communicating products belonging to different product lines. To mine rules, we combine multi-objective search with machine learning techniques. To use the search in the rule mining process, we defined three objectives and integrated them with the widely used multi-objective optimization algorithm—NSGA-II. We compared SBRM with RS based approach (RBRM) in terms of the three objectives, HV, and machine learning quality measurements. The results of the statistical tests show that SBRM performed significantly better than RBRM for all the three objectives, HV, and machine learning quality measurements. In comparison with the rules mined based on real data (RDBRM), SBRM has performed significantly better particularly for *Failed Precision*, *Failed Recall*, and *Failed F-measure* where SBRM improved them by 18%, 72%, and 59% respectively, when compared with RDBRM.

Acknowledgement. This work was supported by the Zen-Configurator project funded by the Research Council of Norway (grant no. 240024/F20) under the category of Young Research Talents of the FRIPO funding scheme. Tao Yue and Shaukat Ali are also supported by RCN funded MBT4CPS project, RFF Hovedstaden funded MBE-CR project, EU Horizon 2020 funded U-Test project, RCN funded Certus SFI, and the EU COST action MPM4CPS.

REFERENCES

[1] Holl, G., Grünbacher, P., and Rabiser, R., 2012. A systematic review and an expert survey on capabilities supporting multi product lines. *Information and Software Technology 54*, 8, 828-852.

[2] Rosenmüller, M. and Siegmund, N., 2010. Automating the Configuration of Multi Software Product Lines. In *VaMoS* (2010), 123-130.

[3] Video Conferencing Systems. http://www.cisco.com/.

[4] ULMA Handling Systems. http://www.ulmahandling.com.

[5] Yue, T., Ali, S., and Selic, B., 2015. Cyber-Physical System Product Line Engineering: Comprehensive Domain Analysis and Experience Report. In *Software Product Line Conference* (2015), ACM, 338-347.

[6] Bécan, G., Behjati, R., Gotlieb, A., and Acher, M., 2015. Synthesis of attributed feature models from product descriptions. In *19th International Conference on Software Product Line* (2015), ACM, 1-10.

[7] Nie, K., Yue, T., Ali, S., Zhang, L., and Fan, Z., 2013. Constraints: the core of supporting automated product configuration of cyber-physical systems. In *Model-Driven Engineering Languages and Systems,* Springer, 370-387.

[8] Safdar, S.A., Yue, T., Ali, S., and Lu, H., 2016. Evaluating Variability Modeling Techniques for Supporting Cyber-Physical System Product Line Engineering. In *International Conference on System Analysis and Modeling* (2016), Springer, 1-19.

[9] Lu, H., Yue, T., Ali, S., and Zhang, L., 2015. Model-based Incremental Conformance Checking to Enable Interactive Product Configuration. *Information and Software Technology*, 72, 68-89.

[10] Lu, H., Yue, T., Ali, S., and Zhang, L., 2016. Nonconformity Resolving Recommendations for Product Line Configuration. In *International Conference on Software Testing* (2016), IEEE, 57-68.

[11] Lu, H., Yue, T., Ali, S., Kunming, N., and Li, Z., 2014. Zen-CC: An Automated and Incremental Conformance Checking Solution to Support Interactive Product Configuration. In *25th International Symposium on Software Reliability Engineering* (2014), IEEE, 13-22.

[12] Temple, P., Duarte, J.a.G., Acher, M., and Jézéquel, J.-M., 2016. Using Machine Learning to Infer Constraints for Product Lines. In *Software Product Line Conference* (2016), ACM, 209-218.

[13] Nadi, S., Berger, T., Kästner, C., and Czarnecki, K., 2015. Where do configuration constraints stem from? an extraction approach and an empirical study. *IEEE Transactions on Software Engineering 41*, 8, 820-841.

[14] Witten, I.H., Frank, E., and Hall, M.A., 2011. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

[15] Frank, E. and Witten, I.H., 1998. Generating accurate rule sets without global optimization. In *15th International Conference on Machine Learning* (1998), 144-151.

[16] Satti, A., Cercone, N., and Keselj, V., 2004. Experiments in Web Page Classification for Semantic Web. In *Workshop on Web-based Support Systems* (2004), 137-141.

[17] Mcminn, P., 2004. Search-based software test data generation: a survey. *Software Testing Verification and Reliability 14*, 2, 105-156.

[18] Ali, S., Iqbal, M.Z., Arcuri, A., and Briand, L.C., 2013. Generating test data from OCL constraints with search techniques. *IEEE Transactions on Software Engineering 39*, 10, 1376-1402.

[19] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T., 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation, 6*, 2, 182-197.

[20] Sarro, F., Petrozziello, A., and Harman, M., 2016. Multi-objective software effort estimation. In *38th International Conference on Software Engineering* (2016), ACM, 619-630.

[21] Pradhan, D., Wang, S., Ali, S., and Yue, T., 2016. Search-Based Cost-Effective Test Case Selection within a Time Budget: An Empirical Study. In *Genetic and Evolutionary Computation Conference* (2016), ACM, 1085-1092.

[22] Pradhan, D., Wang, S., Ali, S., Yue, T., and Liaaen, M., 2016. STIPI: Using Search to Prioritize Test Cases Based on Multi-objectives Derived from Industrial Practice. In *International Conference on Testing Software and Systems* (2016), Springer, 172-190.

[23] Wang, S., Buchmann, D., Ali, S., Gotlieb, A., Pradhan, D., and Liaaen, M., 2014. Multi-objective test prioritization in software product line testing: an industrial case study. In *18th International Software Product Line Conference* (2014), ACM, 32-41.

[24] Wang, S., Ali, S., Yue, T., Li, Y., and Liaaen, M., 2016. A Practical Guide to Select Quality Indicators for Assessing Pareto-based Search Algorithms in Search-Based Software Engineering. In *International Conference on Software Engineering* (2016), ACM, 631-642.

[25] Nebro, A.J., Luna, F., Alba, E., Dorronsoro, B., Durillo, J.J., and Beham, A., 2008. AbYSS: Adapting scatter search to multiobjective optimization. *IEEE Transactions on Evolutionary Computation, 12*, 4, 439-457.

[26] Sokolova, M. and Lapalme, G., 2009. A systematic analysis of performance measures for classification tasks. *Information Processing & Management 45*, 4, 427-437.

[27] Han, J., Pei, J., and Kamber, M., 2011. *Data mining: concepts & techniques*. Elsevier.

[28] Durillo, J.J. and Nebro, A.J., 2011. jMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software 42*, 10, 760-771.

[29] Arcuri, A. and Briand, L., 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *33rd International Conference on Software Engineering* (2011), IEEE, 1-10.

[30] Ali, S. and Smith, K.A., 2006. On learning algorithm selection for classification. *Applied Soft Computing 6*, 2, 119-138.

[31] Mann, H.B. and Whitney, D.R., 1947. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, 50-60.

[32] Vargha, A. and Delaney, H.D., 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, 101-132.

[33] Arcuri, A. and Fraser, G., 2011. On parameter tuning in search based software engineering. In *Search Based Software Engineering,* Springer, 33-47.

[34] Holmes, G., Hall, M., and Prank, E., 1999. Generating rule sets from model trees. In *Australasian Joint Conference on Artificial Intelligence* (1999), Springer, 1-12.

[35] Lopez-Herrejon, R.E., Linsbauer, L., and Egyed, A., 2015. A systematic mapping study of search-based software engineering for software product lines. *Information and Software Technology 61*, 33-51.

[36] Harman, M., Jia, Y., Krinke, J., Langdon, W.B., Petke, J., and Zhang, Y., 2014. Search based software engineering for software product line engineering: a survey and directions for future work. In *18th International Software Product Line Conference* (2014), ACM, 5-18.

[37] Wang, S., Ali, S., and Gotlieb, A., 2014. Cost-effective test suite minimization in product lines using search techniques. *Journal of Systems and Software, 370-391*.

[38] Yi, L., Zhang, W., Zhao, H., Jin, Z., and Mei, H., 2012. Mining binary constraints in the construction of feature models. In *20th International Requirements Engineering Conference* (2012), IEEE, 141-150.

[39] Zhang, B. and Becker, M., 2013. Mining complex feature correlations from software product line configurations. In *7th International Workshop on Variability Modelling of Software-intensive Systems* (2013), ACM, 19-25.

[40] Nadi, S., Berger, T., Kästner, C., and Czarnecki, K., 2014. Mining configuration constraints: Static analyses and empirical results. In *36th International Conference on Software Engineering* (2014), ACM, 140-151.

[41] Czarnecki, K., She, S., and Wasowski, A., 2008. Sample spaces and feature models: There and back again. In *12th International Software Product Line Conference* (2008), IEEE, 22-31.

[42] Dumitru, H., Gibiec, M., Hariri, N., Cleland-Huang, J., Mobasher, B., Castro-Herrera, C., and Mirakhorli, M., 2011. On-demand feature recommendations derived from mining public product descriptions. In *33rd International Conference on Software Engineering* (2011), IEEE, 181-190.

[43] Softpedia. http://www.softpedia.com.

[44] Hariri, N., Castro-Herrera, C., Mirakhorli, M., Cleland-Huang, J., and Mobasher, B., 2013. Supporting domain analysis through mining and recommending features from online product listings. *IEEE Transactions on Software Engineering 39*, 12, 1736-1752.

[45] Davril, J.-M., Delfosse, E., Hariri, N., Acher, M., Cleland-Huang, J., and Heymans, P., 2013. Feature model extraction from large collections of informal product descriptions. In *9th Joint Meeting on Foundations of Software Engineering* (2013), ACM, 290-300.