Feature Selection Optimization in Software Product Lines

UZMA AFZAL¹, TARIQ MAHMOOD², AYAZ H. KHAN³, SADEEQ JAN⁴, RAIHAN UR RASOOL⁵, ALI MUSTAFA QAMAR⁶,⁷,⁸, (Member, IEEE), AND REHAN ULLAH KHAN⁹,¹⁰

¹Computer Science Department, Federal Urdu University of Arts Science and Technology, Karachi 75300, Pakistan
²Computer Science Department, Institute of Business Administration, Karachi 75270, Pakistan
³Computer Science Department, Dhanani School of Science and Engineering, Habib University, Karachi 75290, Pakistan
⁴Department of Computer Science and IT, University of Engineering and Technology, Peshawar 25000, Pakistan
⁵Centre for Applied Informatics (CAI), Institute for Sustainable Industries & Liveable Cities, Engineering and Science, Victoria University, Melbourne, VIC 3011, Australia
⁶Department of Computer Science, College of Computer, Qassim University, Buraydah, Saudi Arabia
⁷Department of Computing, School of Electrical Engineering and Computer Science, National University of Sciences and Technology, Islamabad 44000, Pakistan
⁸BIND Research Group, College of Computer, Qassim University, Buraydah, Saudi Arabia
⁹Department of Information Technology, College of Computer, Qassim University, Buraydah, Saudi Arabia
¹⁰Intelligent Analytics Group, College of Computer, Qassim University, Buraydah, Saudi Arabia

Corresponding author: Ali Mustafa Qamar (al.khan@qu.edu.sa)

This work was supported by a national doctoral research grant from the Higher Education Commission (HEC), Pakistan under the Indigenous PhD Fellowship Program.

ABSTRACT Feature modeling is a common approach for configuring and capturing commonalities and variations among different Software Product Lines (SPL) products. This process is carried out by a set of SPL design teams, each working on a different configuration of the desired product. The integration of these configurations leads to inconsistencies in the final product design. The typical solution involves extensive deliberation and unnecessary resource usage, which makes SPL inconsistency resolution an expensive and unoptimized process. We present the first comprehensive evaluation of swarm intelligence (using Particle Swarm Optimization) to the problem of resolving inconsistencies in a configured integrated SPL product. We call it o-SPLIT (optimization-based Software Product Line Tool) and validate o-SPLIT with standard ERP, SPLIT (Software Product Lines Online Tools), and BeTTy (BEnchmarking and TesTing on the analyYsis) product configurations along with diverse feature set sizes. The results show that Particle Swarm Optimization can successfully optimize SPL product configurations. Finally, we implement o-SPLIT as a decision-support tool in a real, local SPL setting and acquire subjective feedback from SPL designers which shows that the teams are convinced of the usability and high-level decision support provided by o-SPLIT.

INDEX TERMS Software product line, inconsistencies, optimization, feature models, particle swarm optimization.

I. INTRODUCTION

A Software Product Line (SPL) is a collection of software-intensive information systems for configuring similar software products, which share common features to satisfy the need of a particular business market [1], [2]. The success of an SPL is highly dependent on how well the problem domain is modeled along with its commonalities and variations. A well-known approach to model the SPL is Feature Model (FM) which captures and models the commonalities and differences among SPL products.

SPL product configuration is a labor-intensive and time-consuming process [3] and requires high interaction between the developers and users to identify and select the FM compliant feature set, which also fulfills the user requirements. Not complying with the requirements of FM leads to inconsistent product configuration.

The product configuration problems such as product complexity and product inconsistency are highlighted in different case studies and reported by various SPL industries [1], [4]–[7]. General Motors has discussed the challenges and issues of the product line engineering and has highlighted FM inconsistency, product complexity, and variation richness as major problems of their SPL configurations [8]. Moreover, White et al. [9] consider a consistent product configuration as a primary goal of developers. Hubaux [3] discusses the inconsistency in the context of contradictory choice of features.

To solve the inconsistency issue, researchers proposed different solutions such as description logic [10], abductive reasoning [11], a knowledge-based solution [12], and
ontologies [13]. The description logic-based framework is evaluated using a small product configuration with a limited feature set (almost 35). In a real-world SPL product configuration scenario with thousands of features, this solution does not show the same result [10]. The semantic web approach is used to identify and resolve the inconsistencies from FM, which is also a potential research issue for SPL researchers. However, this approach does not apply to the product configuration because of the different dynamics (explained in detail in the Background Section) [13]. Similarly, the main focus of the knowledge-based approach [12] is to solve the inconsistencies in FM. The abductive reasoning approach [11] only identifies an inconsistency with the possible reason but it does not solve inconsistencies. Moreover, these works are validated using exemplary FMs with a limited set of features (30-1000), which raises questions on their applicability to large-scale FMs.

Considering the scale of a typical SPL (thousands of features), researchers also explored Constraint Satisfaction Problem (CSP) and optimization to cater to different SPL problems. Many researchers [14]–[17] present optimization-based solution to solve the cost-based feature selection problem. Trinidad et al. [18] present a CSP-based solution to identify the inconsistency of FM and generate a consistent FM from the given inconsistent one. FM inconsistency issues cannot be mapped with the product inconsistency issue because of their different dynamics. The research by Bagheri et al. [19] and Cruz et al. [20] optimize features based on user requirements and user segments. They optimize the given FM and generate predefined and static product configurations, which satisfy the objective functions. These predefined configuration techniques are interesting to explore the different research directions, but in industrial setups and real-world environments, feature selection in product configurations is not static and predefined. Features select/ deselected dynamically during the configuration as per the user demand. Therefore, it is not practically feasible to lock the feature selection during the actual process of configuration. Similarly, researchers have also explored optimization to SPL inconsistency problem [21], but the focus of this research is to drive a consistent predefined configuration from the given FM. It does not cater to the real-time scenario in which a product becomes inconsistent during the configuration.

The limited applicability (on small-scale FMs) of existing research work to cater the large-scale SPL product inconsistencies also shows its reflection on the SPL industry. Hence, SPL industrial configuration tools also show the limited support to inconsistency resolution, with an increased focus on identifying the inconsistencies rather than solving them [22]. This situation motivates the need to provide SPL developers with appropriate decision support to solve the real-world inconsistencies of large-scale SPLs (Section VII presents a detailed comparison of the existing solutions and their limitations).

From the existing research literature, we discovered that the most effective and scalable research domain to solve large-scale SPL configuration issues is optimization, which has seen widespread applications in a large number of domains (including SPL) of varying complexity [23], [24]. Our study also revealed optimization-related research works such as [14], [25], [26], but we did not find any work which addresses the issue of inconsistent product configuration. Only, [21] caters to the inconsistency in SPL, but is limited to FM inconsistency.

Optimization is a good solution to product inconsistency problem because typically an SPL product configuration contains a large number of features or search space, and hence can be conveniently optimized. Moreover, feature selection is an NP-hard problem [27], where a large number of decisions are required to select the feature set from an inconsistent configuration to make it consistent. A product configuration has multi-modal search space, i.e., multiple possible solutions without any standard process of resolving feature inconsistencies. Before setting a real-world controlled setup for our research work, a Proof of Concept (PoC) is designed to validate the potential practicality of optimization to the inconsistency issue in SPL product configuration [28]. In this PoC, a small testing setup is designed with industrial control knobs to evaluate the validity of optimization to inconsistency issues in SPL product configuration. Genetic Algorithm (the selection of the algorithm is justified in the paper) is employed to minimize the inconsistencies. Three datasets (one small-sized and two large-sized) are used to run the experiments. The results verify the applicability of optimization to the product inconsistency issue. We also explored other optimizations algorithms, such as Particle Swarm Optimization (PSO). The same experimental setup is used to run the experiments with PSO. Promising results strengthened our idea with better performance (published within our research group).

After a successful PoC, our next challenge was to select the appropriate optimization technique. For this, we found that Swarm Intelligence (SI) is a standard optimization technique, with Particle Swarm Optimization (PSO) being a well-known, widely applied, and recognized algorithm [29], [30]. Moreover, the initial results of our testing setup support this selection. In this context, we posed and addressed a question in this article i.e.,

**RQ1:** Can we experimentally validate an application of SI through PSO to resolve SPL product inconsistencies?

To answer the research question, we conducted experiments using different representative SPLs which varied in the size of feature repositories. These SPLs were provided by an anonymous multinational SPL vendor and focused on the ERP business domain. We designed and implemented an SPL tool, which is labeled as $o$-SPLIT (optimization-based Software Product Line Tool), and implements a module related to SI. Finally, we attempted to validate the different results of $o$-SPLIT for the SPL industry concerning usability, effectiveness, efficiency, and user satisfaction. For this, we posed and answered the following research question:
RQ2: What is the opinion of the SPL designers regarding the optimization-based decision support provided by o-SPLIT in the SPL product configuration process?

We compare the performance and efficiency of SI by implementing o-SPLIT in a testing environment of an SPL company, whose data was used to run the experiments. We acquired anonymous feedback from SPL designers of this company for the results of o-SPLIT through a subjective questionnaire.

This article is organized as follows: In Section II, the background information pertinent to this article is presented. Section III presents the architecture of o-SPLIT framework and Section IV discusses the experimental methodology. Section V presents the detailed results of o-SPLIT over different experimental settings, whereas Section VI describes its controlled evaluation. Finally, we compare o-SPLIT with state-of-the-art to prove its novelty.

II. BACKGROUND

In this section, we first describe the Software Product Line (SPL) concepts. Later, we discuss the feature modeling, wherein we present an exemplary FM of Vendor Master (VM) to describe the features and constraints. We then discuss the process of a product configuration using VM as a reference FM and present some consistent and inconsistent product configurations. Finally, we present the concepts of optimization.

A. SOFTWARE PRODUCT LINES

An SPL is a collection of software-intensive systems for configuring similar software products which share common features to satisfy the need of a particular business market. An SPL has two development processes, i.e., Domain Engineering (DE) and Application Engineering (AE). DE focuses on the problem domain and defines the commonalities and variabilities of the SPL products. AE reuses the domain artifacts and exploits the SPL variability to develop an SPL product. The transition from DE to AE is done through a configuration, that adapts a domain model to define an application product. Each unique product derivation is labeled as a variant, i.e., the representation of a unique SPL configuration that differs from other variants on specific variation points.

To manage the complex SPL dynamics, analytical skills are required to identify, model, and encode domain and product knowledge into artifacts that can be reused across the development lifecycle. The success of an SPL is highly dependent on how well the domain is modeled along with its commonalities and variations. A well-known approach to model the SPL is feature modeling.

1) FEATURE MODELING

Feature Modeling captures the commonalities and variations among SPL products during DE. It represents all possible SPL products. The features are the primary distinguishing characteristics of a product [31], [32]. A product can be configured by selecting a subset of all features. The rules that govern the entire configuration process are derived from constraints. The list of standard constraints is as follows (modified from the list in [32]):

- Mandatory: The existence of feature $F$ in product $P$ is mandatory.
- Optional: The existence of feature $F$ in product $P$ is optional.
- OR: In product $P$, there is a feature set from which one or more features can be selected.
- Alternative: In product $P$, there is a feature set $\{F_1, F_2, F_3, \cdots, F_n\}$ from which only one feature can be selected.
- Exclude: If feature $F_1$ excludes feature $F_2$, both features cannot be configured in the same product $P$.
- Include: If feature $F_1$ includes a feature $F_2$, the inclusion of $F_1$ in a product $P$ implies the inclusion of $F_2$ in $P$.

2) AN EXAMPLE OF FEATURE MODELING

Figure 1 presents a feature model of a VM, module of an ERP SPL which integrates numerous configuration units of multiple departments of an organization into a single streamlined system. Each unit targets a particular business process, like product development, purchasing, sales, and marketing.
VM manages the information about vendors that supply an enterprise. The VM in Figure 1 is derived from the work done in [33], [34] in which the authors discuss the concepts of ERP. The description of the feature model in Figure 1 is as follows:

- **F1** is a mandatory feature, whereas **F6**, **F8**, and **F11** are optional ones. **F1** conveys Vendor Information in account and has two children **F2** and **F3**. **F2** and **F3** are “Anded” and represent Vendor Name and Vendor ID respectively. **F3** has two children: **F4** and **F5**. **F4** generates Manual ID and **F5** generates System Generated ID. **F4** and **F5** exclude each other, since System Generated and Manual IDs cannot co-exist in a valid product. **F6** is an optional feature to inactivate a vendor after a specific time period, described by **F7**. **F6** includes **F7** to be a meaningful feature. **F8** takes the Temporary Vendor feature in account and has two children **F9** and **F10**. **F9** and **F10** exclude each other; **F9** allows temporary vendor while **F10** does not allow it. **F11** arranges the Vendor List and has three alternative children **F12**, **F13**, and **F14**. **F12**, **F13**, and **F14** arrange vendors by their Name, Code, and Postal Code respectively.

3) CONFIGURING THE SPL PRODUCT

Software product configuration selects and de-selects features from the FM according to user preferences. Software product configuration needs a strong interaction between developers and users to identify a FM compliant configuration feature set [3].

Using Figure 1 as a reference, following are some consistent product configurations which are compliant with the constraints describe in VM feature model:

- \[ P = \{F_1(F_2, F_3(F_4)), F_6(F_7)\} \]
- \[ P = \{F_1(F_2, F_3(F_5)), F_6(F_7)\} \]

A product configuration becomes inconsistent if the configured feature set is not compliant with the FM and violates the predefined constraints. Using Figure 1 as a reference, following are some inconsistent product configurations:

- \[ P = \{F_6(F_7)\} \] is an inconsistent product because a mandatory feature **F1** along with its children features is missing.
- \[ P = \{F_1(F_2, F_3(F_4, F_3))\} \] is also an inconsistent product because **F4** and **F5** cannot coexist in the same product.
- \[ P = \{F_1(F_2, F_3(F_6)), F_6\} \] is also an inconsistent product because **F6** includes **F7**, which is missing from the current feature selection.

4) FEATURE MODEL GENERATORS

SPLOT (Software Product Lines Online Tools) FM generator [35] and BeTTTy (BEEnmarking and TeSting on the analysis (of FMs)) online FM generator [36] provide standardized information, tools, and datasets for both SPL researchers and practitioners; and support the generation of FM test data to evaluate the performance of analysis tools. In this article, we used them to generate the FMs for validating o-SPLIT (our proposed SPL tool).

SPLOT FM generator is a simple, yet robust Java-based visual editor which supports FM creation, configuration, and editing. SPLOT FM generator generates FMs based on several input parameters such as feature-set size, percentage distribution of mandatory, optional, alternative and exclusive constraints, and minimum and maximum branching factors of the FM. BeTTTy online FM generator provides a web-based interface to generate random feature models. It supports the generation of FMs on the basis of very few parameters, such as the number of features and user information. However, one can also select the advanced parameters to generate FM such as percentage distribution of mandatory, optional, or alternative constraints, maximum branching factor and a maximum number of sub-features in a feature set. BeTTTy is released under LGPL3 licence and distributed as a jar file.

B. OPTIMIZATION

Optimization is the selection of an optimal candidate concerning predefined conditions from a set of available candidates. Optimization has seen wide acceptability to several research domains, such as designing the aircraft, bioinformatics, and control engineering [23], [24].

1) PARTICLE SWARM OPTIMIZATION (PSO): SWARM INTELLIGENCE

Swarm Intelligence is based on the study and analysis of collective behavior in decentralized and self-organized systems. Examples of these types of natural systems are fish schooling, ant colonies, and animal herds. PSO is a swarm-based stochastic computational algorithm, that is influenced by the social behavior of social organisms, such as fish schools and bird flocks [37]. PSO has many similarities with evolutionary algorithms, like GA in that, it initializes with a population of random candidates and searches for the optimal solution by updating the generations. However, it does not have evolutionary operators such as crossover and mutation and requires the setting of only a few parameter types [38], [39].

Figure 2 shows the standard working of PSO and its associated pseudocode is displayed in Algorithm 1. PSO starts working by initializing an initial population of particles \( P \). Each particle \( P_i \) is randomly placed in the search space as a candidate solution to the optimization problem. The change in particle’s position is defined as velocity \( V \), and the movement of particle is based on the interaction of particle’s personal experience and social experience. Each particle adjusts its trajectory based on its own personal best position experience \( p_{id} \) and the best position held by any particle \( p_{bd}(p_{bd}) \) of the swarm. Equation 1 and 2 are used to update the velocity \( V \) and position \( S \) of each particle.

\[
V_{id}(t + 1) = V_{id}(t) + C_1R_1(P_{id} - S_{id}(t)) + C_2R_2(P_{bd} - S_{id}(t)) \tag{1}
\]

\[
S_{id}(t + 1) = S_{id}(t) + V_{id}(t + 1) \tag{2}
\]

where \( t \) is the counter, \( C_1 \) and \( C_2 \) are the acceleration coefficients (Cognitive and Social attractions), and \( R_1 \) and \( R_2 \)
**Algorithm 1** Particle Swarm Optimization - Pseudo Code

1. For each Particle \( P_i \)
2. Initialize \( P_i \);
3. End For
4. Do
5. For each Particle \( P_i \)
6. Compute fitness;
7. If fitness > its personal best
8. Update current value as the new personal best;
9. End If
10. End For
11. Select the particle \( P \) with the best fitness value of all as the global best;
12. For each Particle \( P_i \)
13. Compute \( V_{id} \) using Equation 3;
14. Compute \( F_{id} \) using Equation 2;
15. End For
16. While (the termination criteria is not attained;)

are two random numbers in the range \([0,1]\). The updated particles are evaluated by the objective function. After several iterations, the best particle is returned as the optimal solution.

In [40], the authors introduce the concept of the inertia weight \( w \) to the standard velocity update equation, which supervises the effect of previous velocities on current velocity and controls the convergence of the algorithm. The modified velocity is depicted in Equation 3.

\[
V_{id}(t + 1) = wV_{id}(t) + C_1R_1(P_{id} - S_{id}(t)) + C_2R_2(P_{gd} - S_{id}(t)) \quad (3)
\]

and the inertia weight is updated according to Equation 4.

\[
W = W_{mx} - ((W_{mx} - W_{mi})/I_{mx}) \times I \quad (4)
\]

where \( W_{mx}, W_{mi} \) are the maximum and minimum values respectively, that \( W \) can take; \( I \) is the current iteration and \( I_{mx} \) represents the total number of iterations.

Several encoding schemes have been proposed to represent the particles, such as binary and natural encoding.

### III. o-SPLIT: OPTIMIZATION-BASED SOFTWARE PRODUCT LINE TOOL

In this section, we describe our optimization-based Software Product Line Tool (o-SPLIT) to deal with the SPL product inconsistencies. o-SPLIT provides automated support to resolve inconsistencies in the form of decision support to SPL developers while configuring critical feature sets. o-SPLIT successfully resolves inconsistencies by employing Swarm Intelligence (SI).

The o-SPLIT architecture comprises a Swarm Intelligence Module (SIM) as shown in Figure 3.

#### A. SWARM INTELLIGENCE MODULE (SIM)

SIM implements Particle Swarm Optimization (PSO) to generate consistent product configurations. In Figure 3, we show the operational flowchart of SIM. It fetches an inconsistent product configuration from Product-Rep and uses the objective function explained in Section III-B, to encode configuration as an optimization problem. SIM encodes the inconsistent product configuration as a particle. For the given configuration with \( n \) number of features, it maps them to a particle with \( n \) dimensions. The bit strings (0 and 1) are used to encode the particle, where 0 and 1 represent the de-selection and selection of a feature respectively.

As our ultimate goal is to generate a consistent configuration from an inconsistent one, SIM uses the given inconsistent product configuration as an initial seed to generate the initial swarm with multiple particles. It tunes swarm size, inertia weight, cognitive and social attraction parameters to acquire their optimal values. SIM, then employs PSO to the encoded configuration with the optimal values of the parameters and finally, returns the optimized configurations. We have implemented the SIM module in Matlab by inheriting and modifying the PSO code available on MathWorks website.\(^1\)

#### B. OBJECTIVE FUNCTION

Here, we describe our objective function which is used by SIM. This function must address two challenges: 1) minimize inconsistencies while avoiding the deselection of all features

\(^1\)https://www.mathworks.com/matlabcentral/fileexchange/25986-constrained-particle-swarm-optimization/all_files
in a given configuration, and 2) maintain prioritization of the different types of inconsistencies. We solve them as follows:

To understand the first challenge, assume an inconsistent product configuration $P = \{F_1(F_2, F_3(F_4, F_5))\}$, where $F_4$ and $F_5$ cannot coexist in the same product. The easiest way to make it consistent is the deselection of both inconsistent features, i.e., $F_4$ and $F_5$. This is logical but not a good approach, because developers have to make their efforts again in selecting between $F_4$ and $F_5$. To better understand the problem, assume we scale up a product configuration to a large number of features $n$ containing a high number of inconsistencies $m$. If we turn off all inconsistent features, then the consistent configuration can contain a minimum of $n - m$ features, which can potentially be a very small subset of $n$. Obviously, this doesn’t give much choice for developing products having diverse features. A better approach, for instance, could be to deselect any one of $F_4$ or $F_5$ to generate a consistent product configuration with $n-1$ features. To formalize similar solutions automatically, we modify our objective function to meet the threshold value of the number of selected features, which can be set by the developers along with the main objective of minimizing inconsistencies.

Our second challenge was to assign weights to different constraint types that introduce product inconsistencies. In a real-world scenario, developers give importance to those constraint violations which have a high priority. To model our optimization approach according to the real world, we assigned weights to different types of constraints. For this, we targeted four primary types, i.e., mandatory, include, exclude, and alternative. We first prioritized them based on their severity, i.e., Mandatory $>$ Exclude and Alternative $>$ Include. Then, we normalized the weight values on a scale of 0-1. We arbitrarily assigned 0.4 weight to a mandatory constraint violation, 0.3 to an alternative or an exclude constraint violation, and 0.3 to an include constraint violation.

We now mathematically formalize our objective function. Suppose that:

- $FS$ is a set of all possible Features (F) of a Software Product Line (SPL). It represents the complete domain of a particular SPL.
- $\gamma$ is an inconsistent product configuration derived to meet the demands of a specific SPL customer. It contains $\gamma$ features.
- $\Psi = \{F_1, F_2, \ldots , F_n\}$ is a subset of $FS$ which is configured in $\gamma$.
- $\Omega = \{C_{t_1}, C_{t_2}, \ldots , C_{t_n}\}$ is the set of applicable constraints on $F_i$ of $\gamma$. For instance, mandatory, include etc.
- $\omega = \{0.4, 0.3, 0.3\}$ is the set of weights assigned to constraints $C_{t_n} \in \Omega$. As already discussed, 0.4 for mandatory, 0.3 for exclude and include constraint violations.
Φ = {ω₁C₁F₁, ω₂C₂F₂, ⋅⋅⋅, ωₘCₘFₘ} is the set of \( j \) inconsistencies, where \( j \) is not necessarily equal to \( n \) (total number of features in \( Ψ \)), because the given product configuration \( γ \) contains \( Ψ \); and \( Ψ \) can also have consistent features which do not introduce any inconsistencies. In a nutshell, \( n – j \) are those features which are consistent. If a mandatory feature \( F_i \) is missed in \( γ \), the representation of this \( j \) inconsistency is \( (0.4(\text{Mandatory})^* F_i) \). Note that, we use the textual terms to increase the readability of the user, although an encoding scheme (explained in Section IIIA and IV) is used to represent the \( Φ, ω, Ψ, γ, \) and \( FS \).

Then, Equations 5 and 6 are used to represent the objective of the optimization, i.e., maximization of selected features and minimization of the inconsistencies.

\[
\text{Min} \sum_{j=1}^{n} Φ(ω_jC_jF_j) \quad (5)
\]

\[
s.t. \quad P \rightarrow Ω \sum_{i=1}^{n} Ψ(F_i) \sim \text{Threshold} \quad (6)
\]

For a given inconsistent product configuration \( γ \) of a SPL with \( Φ \) inconsistencies, the goal of the optimization problem is to minimize the \( Φ \) by complying \( Ω \) along with maximization of the features \( F_j \) up to a given threshold value. This threshold value is set by a consensus between developers and users. As shown in Equations 5 and 6, a consistent SPL product configuration is a multi-objective optimization problem where the two objectives, i.e., minimization of inconsistencies and maximization of feature selection are competing. Therefore, there is usually no single optimal solution. A tradeoff of these multiple objectives is calculated as a solution (Pareto optimal solution). For more details on multi-objective optimization and pareto optimal solution, please refer to [41].

As we already mentioned, we use PSO to optimize this problem which tries to search a consistent product configuration in a search space. We have \( n \) dimensional search space which contains a swarm of possible candidates of \( m \) dimensional configurations. The values of \( n \) and \( m \) are based on swarm size and configuration size (number of features) respectively. For instance, if we have small-scale product configuration with 100 (\( m \)) features and 20 (\( n \)) swarm size, then the search space has \( 20^*100 \) dimensions. Every possible candidate configuration is encoded in a particle. The position (movement) of these particles in the search space is based on particle personal experience and social experience (detailed in section IIB). An objective function is used to evaluate how good or bad is the position of the particle. After several iterations, PSO generates an optimal product configuration with lesser number of inconsistencies.

C. O-SPLIT: A WORKING EXAMPLE

In this subsection, a working example of the o-SPLIT is presented to give an idea of how it can provide SPL developer teams a decision support. Vendor Master (VM) example (explained in the Background section) is used for the illustration. The details are as follows:

- **Actors and Roles:** Assume the following actors in the working example:
  - SPL Vendor (SV): Representing the company involved in selling SPL products.
  - Developer Teams: Representing the developer teams involved in SPL product configuration.
  - R-Industries: Representing the client company interested in purchasing the VM module.

- **Domain Engineering Module (DEM):** o-SPLIT’s DEM provides interfaces for SPL initialization along with its associated modules, features, and constraints definitions for SPL. These interfaces are described as follows:
  - **SPL Initialization:** This interface is used to store SPL profile information including SPL ID, SPL Name, and SPL Description which helps in tracking a particular SPL. Figure 4 shows the initialization of an SPL, with an ID T-ERP, which is a Test ERP.
  - **Add Modules:** This interface allows the developer teams to add and define modules to an SPL. It stores Module ID, Module Name, and Module Description for an SPL. Figure 5 shows the definition of a VM module, i.e., ERP-VM for T-ERP SPL.
  - **Add Constraints:** This interface facilitates the developer teams to define SPL constraints. Figure 6 shows the definition of a unary constraint (single feature involved) mandatory to the T-ERP.
  - **Add Features:** This interface helps the developer teams in defining features to the SPL. The process of feature definition starts with the selection of an SPL followed by the selection of a particular module for which the features are defining or being defined. This interface stores Feature ID, Feature Name, and Feature Description. It also allows a feature to be stored as a Root Feature (with no parent feature) or as a Final Feature (with no child feature). In case a feature is not a Final one, a user can associate a parent feature. Figure 7 shows the
definition of $F_1$, a root feature, which stores Vendor Information of VM-ERP.

- Add Constraints to Features: This interface facilitates the developer teams to create association between SPL features and constraints, shown in Figure 8. A mandatory constraint is being assigned to feature $F_1$, already defined in the T-ERP SPL domain.

- Application Engineering Module (AEM): o-SPLIT provides a runtime configuration support to developer teams in AE process of SPL by offering interface for SPL product configuration.

- AE Initialization: This interface facilitates the developers to start an SPL product configuration process by initializing a client profile along with the modules they want in the configuration. Figure 9 shows the initialization of product configuration process for R-Industries. The ID assigned to the SPL product is CL-T-ERP, which is a test configuration of T-ERP. CL-T-ERP contains only a single module, i.e., VM.

- Product Configuration: This is the core interface of an SPL product configuration. Figure 10 shows the configuration process of VM-ERP module of CL-T-ERP. The top left of Figure 10 shows the tracking information of VM-ERP, i.e., CL-T-ERP (product name) and T-ERP (SPL name). Similarly, the top right shows the information of the features which are already configured within VM-ERP. The
bottom of Figure 10 shows a list of potential features (along with description), which are not part of VM-ERP but can still be selected.

- **Optimization Module** – o-SPLIT provides decision support through its optimization module, i.e., SIM. Figure 11 shows the optimization of the SPL product CL-T-ERP which contains the configuration of VM feature model. After clicking the input configuration button, the developer can browse the file which contains inconsistent CL-T-ERP configuration. The inconsistent configuration used to run this example is as follows (Φ represents the % inconsistencies and Ψ shows the number of features in the given configuration):

  - CL-T-ERP = F_1(F_3(F_4, F_5)), F_6, F_8(F_9, F_{10}), F_{11}(F_{12}, F_{13}); \ Φ = 0\%, \ Ψ = 10

After this, parameter values are selected to run the experiment. For this example, the following parameters setting for the small-scale configurations is used:

- Swarm size = 20, inertia = 0.5, cognitive attraction = 1.5, and social attraction = 1.5

The experiments are run 10 times to acquire the three best configurations on the basis of performance measure, i.e., minimization of inconsistencies. The results containing the details of these 3 consistent configurations are finally exported to the text file. This file is presented to the client who can select a configuration for the implementation. Three consistent configurations, SIM generated for CL-T-ERP are as follows (Φ represents the % inconsistencies and Ψ shows the number of features in the given configuration):

  - CL-T-ERP = F_1(F_2, F_3(F_4)), F_6(F_7), F_8(F_{10}), F_{11}(F_{13}); \ Φ = 0\%, \ Ψ = 10

  - CL-T-ERP = F_1(F_2, F_3(F_4)), F_6(F_7), F_8(F_9), F_{11}(F_{12}); \ Φ = 0\%, \ Ψ = 10

  - CL-T-ERP = F_1(F_2, F_3(F_4)), F_6(F_7), F_8(F_{10}), F_{11}(F_{13}); \ Φ = 0\%, \ Ψ = 10

These CL-T-ERP configurations are 100% consistent with the same number of selected features.

The source code of the main features of o-SPLIT and datasets are available at this URL.\(^2\)

### IV. EXPERIMENTAL SETUP

In this section, we first define the data collection procedure, then we describe the problem encoding scheme and the process of generating the initial population. Finally, we discuss the experimental configurations of the SIM module.

#### A. DATA COLLECTION

We acquired real-world data of a local ERP SPL from a well known multi-national organization, that has a large customer base and a good repute in our local market. The datasets contain small-scale (containing 100 features), medium-scale (containing 500 features), large-scale (containing 1000 features), and very large-scale (containing 5000 features) FMs with different complexities (measured in terms of the number of features and constraints). Table 1 presents the details of the total features and constraints in each of the FM. It shows that the number of constraints in the given FM is proportional to the FM size. Moreover, in each FM except the small-scale one, the occurrence of include constraint is the highest followed by mandatory, exclude, and alternative constraints. In small-scale FM, the occurrence of mandatory constraint is the highest followed by include, exclude, and alternative constraints.

| SPL Scale | Features | M | T | E | A | Total |
|-----------|----------|---|---|---|---|-------|
| Small     | 100      | 24 | 20 | 10 | 10 | 64    |
| Medium    | 500      | 110| 118| 48 | 34 | 310   |
| Large     | 1000     | 212| 226| 104| 75 | 617   |
| Very Large| 5000     | 900| 984| 734| 400| 3018  |

We then generated 10 inconsistent product configurations for each FM, containing all real-world inconsistencies including mandatory, exclude, alternative, and include constraint.

\(^2\)https://sites.google.com/site/afzaluzmaa/research/i-split
TABLE 2. Small-Scale Configurations Set; CID = Configuration ID, \( F \) = Number of Features, \( M \) = Mandatory Inconsistencies, \( I \) = Include Inconsistencies, \( E \) = Exclude Inconsistencies, \( A \) = Alternative Inconsistencies.

| CID | F | Inconsistencies | ERP Configurations (ES) | SPLOT Configurations (SS) | BeTTy Configurations (BS) |
|-----|---|-----------------|-------------------------|----------------------------|---------------------------|
|     |   | M | I | E | A | T | M | I | E | A | T | M | I | E | A | T |
| C1  | 45| 18| 20| 8 | 8 | 54| 42| 11| 19| 18| 12| 60| 49| 20| 22| 10| 8 | 60 |
| C2  | 32| 22| 17| 9 | 9 | 57| 39| 14| 12| 19| 12| 59| 41| 21| 13| 12| 10| 56 |
| C3  | 48| 20| 14| 6 | 8 | 48| 46| 23| 18| 10| 16| 67| 53| 27| 19| 12| 8 | 66 |
| C4  | 40| 10| 13| 8 | 8 | 39| 41| 20| 18| 20| 16| 74| 34| 15| 17| 4 | 6 | 42 |
| C5  | 35| 23| 18| 8 | 9 | 58| 43| 24| 16| 11| 21| 72| 51| 26| 21| 6 | 15| 68 |
| C6  | 45| 15| 16| 9 | 7 | 47| 48| 16| 20| 14| 10| 60| 39| 18| 16| 10| 14| 58 |
| C7  | 40| 12| 12| 5 | 4 | 33| 45| 17| 22| 19| 11| 69| 56| 15| 25| 11| 13| 64 |
| C8  | 33| 19| 15| 6 | 8 | 48| 47| 13| 19| 21| 14| 67| 37| 19| 14| 6 | 10| 49 |
| C9  | 25| 13| 17| 9 | 9 | 48| 46| 19| 13| 9 | 13| 54| 49| 21| 19| 10| 19| 69 |
| C10 | 40| 8 | 18| 9 | 9 | 44| 43| 12| 21| 10| 16| 59| 58| 20| 22| 15| 10| 67 |
| Avg.| 39| 16| 16| 8 | 8 | 48| 44| 17| 18| 14| 15| 64| 47| 20| 19| 10| 11| 60 |

To analyze the applicability of \( o \)-SPLIT to different types of FMs with different complexities, we also generated FMs using the SPLOT FM generator and BeTTy online FM generator. SPLOT and BeTTy are the two well-known names in the SPL industry, they aim to put SPL research into practice by providing standardized information, tools, and datasets.

SPLOT FM generator generated small, medium, and large-sized FMs efficiently, but got stuck in a continuous loop of FM rejections (due to inconsistent FM generation) while generating very large-scale consistent FM. We attempted this process five times but SPLOT FM generator did not produce very large-scale FM (containing 5000 features). On average, SPLOT FM generator rejected 19,500 FMs an hour due to inconsistency. On the other hand, BeTTy online FM generator did not encounter any issues and easily generated the FMs for each scale (small to very large).

The number of constraints in these automated FMs (SPLOT and BeTTy) are normally distributed, that’s why we are not mentioning them in a separate table. Similar to ERP configurations, we generated 10 inconsistent product configurations for each FM and named them as SS-C1, SS-C2, \( \cdots \), SS-C10 (SPLOT Small-scale Configurations), SM-C1, SM-C2, \( \cdots \), SM-C10 (SPLOT Medium-scale Configurations), SL-C1, SL-C2, \( \cdots \), SL-C10 (SPLOT Large-scale Configurations), and BS-C1, BS-C2, \( \cdots \), BS-C10 (BeTTy Medium-scale Configurations), and BL-C1, BL-C2, \( \cdots \), BL-C10 (BeTTy Very Large-scale Configurations), respectively.

Table 2, 3, 4, 5 show the small, medium, large, and very large-scale configurations derived from the feature models generated through ERP, SPLOT FM generator, and BeTTy online tool, respectively. Figure 12 shows a comparison of these configurations. In all of the scale configurations, except very large-scale ones, SPLOT configurations contain the highest number of inconsistencies followed by BeTTy and ERP configurations. In very large-scale configurations, ERP configurations contain the highest occurrence of inconsistencies followed by the BeTTy configurations. The overall number of inconsistencies in all configurations is proportional to the size of the given configurations. Inconsistencies wise, the occurrence of mandatory and include inconsistencies are the highest in numbers, followed by exclude and alternative inconsistencies.

To check the performance of \( o \)-SPLIT in both cases, i.e., the best (containing a fewer number of inconsistencies) and the worst (containing a high number of inconsistencies) situations.

We statistically computed the configurations details, such as the number of inconsistencies and features, which helped us in the selection process. Based on the above discussion, Table 6 lists the selected configurations, which we used to run our optimization related experiments.

### B. PARAMETER TUNING

In order to obtain the optimal values for the different parameters of SIM, we used the parameter tuning method, which is a traditional way of testing and comparing different values before the actual test run [42]. We randomly selected ES-C2 from the small-scale configurations set, SM-C3 from the medium-scale configurations set, BL-C10 from the large-scale configurations set, and EVL-C8 from very large-scale configurations set. For SIM, we tuned the swarm size, cognitive and social attraction, and inertia weight. For each parameter setting, we ran the experiments 10 times.
For cognitive (C1) and social (C2) attraction, we used three standard settings, i.e., C1=C2 (C1=1.5, C2=1.5), C1 > C2 (C1=2.0, C2=1.0), C2 > C1 (C2=2.0, C1=1.0) [44], [45]. For swarm size parameter, we tested the selected configurations with a small (20, 100, 200, 1000), medium (40, 200, 400, 2000), large (80, 400, 800, 4000), and very large (160, 800, 1600, 8000) swarm sizes. For the tuning of inertia weight, we used a value ranging between 0.4-1.0 [46]. The results of these experiments are described in Section V.

C. OPTIMIZING AND EVALUATING INCONSISTENT CONFIGURATIONS

After obtaining the optimal parameter values for SIM, we passed these parameters to SIM for their optimization task. For selected configurations set (Table 6), we ran the optimization experiments 10 times (based on recommendations of [43]) to obtain the best consistent configuration. We performed all these experiments on a Windows 8 machine with Intel Core i7 CPU, 2.4 GHz processor, and 16GB RAM.
it reached 1.0. To find the optimal value for cognitive and social attractions, we tested SIM with three standard settings, i.e., $C_1 = C_2$, $C_1 > C_2$ and $C_2 > C_1$.

Table 7 shows the optimal values of SIM parameters. SIM produced the optimal results with equal values of cognitive and social attractions across all the given configurations, i.e., $C_1 = 1.5$ and $C_2 = 1.5$. We also found 0.95 as the optimal value for inertia except for small-scale configurations, which produced optimal configurations with inertia as 0.5. Similar to the value of inertia, small-sized swarm is the optimal setting across all the given configurations except large-scale and very large-scale configurations.

### Table 6. Selected Inconsistent Configurations.

| CID | ERP Configurations | F | Inconsistencies | M | I | E | A | T |
|-----|-------------------|---|-----------------|---|---|---|---|---|
| C1  | 2991              | ES-C2 | High Inconsistencies | ES-C2, BS-C5, SS-C4, SS-C5, BS-C5, BS-C9 | 2913 | 822 | 902 | 658 | 356 | 2738 | 2873 | 859 | 997 |
| C2  | 2867              | SM-C6, SS-C5 | Low Inconsistencies | BM-C4, BM-C10 | 878 | 614 | 596 | 325 | 2413 | 2539 | 829 | 975 |
| C3  | 2893              | BM-C4, BM-C10 | SS-C4, BS-C8 | 835 | 522 | 701 | 398 | 2456 | 2149 | 645 | 825 |
| C4  | 2972              | BM-C4, BM-C10 | SS-C4, BS-C8 | 789 | 767 | 665 | 345 | 2566 | 2504 | 798 | 765 |
| C5  | 2983              | BM-C4, BM-C10 | SS-C4, BS-C8 | 801 | 845 | 706 | 298 | 2650 | 2667 | 575 | 609 |
| C6  | 2863              | BM-C4, BM-C10 | SS-C4, BS-C8 | 557 | 960 | 698 | 258 | 2419 | 2598 | 667 | 698 |
| C7  | 2879              | BM-C4, BM-C10 | SS-C4, BS-C8 | 789 | 756 | 584 | 276 | 2405 | 2541 | 891 | 548 |
| C8  | 2950              | BM-C4, BM-C10 | SS-C4, BS-C8 | 654 | 852 | 592 | 301 | 2399 | 2935 | 765 | 529 |
| C9  | 2932              | BM-C4, BM-C10 | SS-C4, BS-C8 | 892 | 957 | 731 | 394 | 2974 | 2824 | 913 | 713 |
| C10 | 2991              | BM-C4, BM-C10 | SS-C4, BS-C8 | 895 | 975 | 726 | 391 | 2991 | 2798 | 916 | 762 |
| Avg. | 2919              | BM-C4, BM-C10 | SS-C4, BS-C8 | 810 | 810 | 666 | 334 | 2601 | 2670 | 786 | 742 |

### Table 7. Large-Scale Configurations Set; CID = Configuration ID, F = Number of Features, M = Mandatory Inconsistencies, I = Include Inconsistencies, E = Exclude Inconsistencies, A = Alternative Inconsistencies.

| CID | ERP Configurations | F | Inconsistencies | M | I | E | A | T |
|-----|-------------------|---|-----------------|---|---|---|---|---|
| C1  | 609               | BS-C5, BS-C9 | High Inconsistencies | BS-C5, BS-C9 | 188 | 190 | 98 | 63 | 539 | 231 | 178 | 180 |
| C3  | 615               | BS-C5, BS-C9 | High Inconsistencies | BS-C5, BS-C9 | 184 | 202 | 74 | 43 | 462 | 598 | 132 | 142 |
| C5  | 627               | BS-C5, BS-C9 | High Inconsistencies | BS-C5, BS-C9 | 199 | 199 | 77 | 59 | 498 | 602 | 198 | 165 |
| C6  | 617               | BS-C5, BS-C9 | High Inconsistencies | BS-C5, BS-C9 | 144 | 203 | 99 | 53 | 499 | 569 | 188 | 174 |
| C7  | 603               | BS-C5, BS-C9 | High Inconsistencies | BS-C5, BS-C9 | 189 | 207 | 101 | 61 | 555 | 553 | 210 | 204 |
| C8  | 623               | BS-C5, BS-C9 | High Inconsistencies | BS-C5, BS-C9 | 209 | 220 | 103 | 70 | 602 | 564 | 178 | 202 |
| C9  | 599               | BS-C5, BS-C9 | High Inconsistencies | BS-C5, BS-C9 | 175 | 200 | 69 | 48 | 492 | 589 | 120 | 189 |
| C10 | 623               | BS-C5, BS-C9 | High Inconsistencies | BS-C5, BS-C9 | 135 | 206 | 83 | 72 | 496 | 610 | 140 | 178 |
| Avg. | 616               | BS-C5, BS-C9 | High Inconsistencies | BS-C5, BS-C9 | 174 | 199 | 89 | 60 | 523 | 587 | 172 | 176 |

### Table 8. Small-Scale Configurations with SIM.

| CID | ERP Configurations | F | Inconsistencies | M | I | E | A | T |
|-----|-------------------|---|-----------------|---|---|---|---|---|
| C1  | 2991              | ES-C2 | High Inconsistencies | ES-C2, BS-C5, SS-C4, SS-C5, BS-C5, BS-C9 | 2913 | 822 | 902 | 658 | 356 | 2738 | 2873 | 859 | 997 |
| C2  | 2867              | SM-C6, SS-C5 | Low Inconsistencies | BM-C4, BM-C10 | 878 | 614 | 596 | 325 | 2413 | 2539 | 829 | 975 |
| C3  | 2893              | BM-C4, BM-C10 | SS-C4, BS-C8 | 835 | 522 | 701 | 398 | 2456 | 2149 | 645 | 825 |
| C4  | 2972              | BM-C4, BM-C10 | SS-C4, BS-C8 | 789 | 767 | 665 | 345 | 2566 | 2504 | 798 | 765 |
| C5  | 2983              | BM-C4, BM-C10 | SS-C4, BS-C8 | 801 | 845 | 706 | 298 | 2650 | 2667 | 575 | 609 |
| C6  | 2863              | BM-C4, BM-C10 | SS-C4, BS-C8 | 557 | 960 | 698 | 258 | 2419 | 2598 | 667 | 698 |
| C7  | 2879              | BM-C4, BM-C10 | SS-C4, BS-C8 | 789 | 756 | 584 | 276 | 2405 | 2541 | 891 | 548 |
| C8  | 2950              | BM-C4, BM-C10 | SS-C4, BS-C8 | 654 | 852 | 592 | 301 | 2399 | 2935 | 765 | 529 |
| C9  | 2932              | BM-C4, BM-C10 | SS-C4, BS-C8 | 892 | 957 | 731 | 394 | 2974 | 2824 | 913 | 713 |
| C10 | 2991              | BM-C4, BM-C10 | SS-C4, BS-C8 | 895 | 975 | 726 | 391 | 2991 | 2798 | 916 | 762 |
| Avg. | 2919              | BM-C4, BM-C10 | SS-C4, BS-C8 | 810 | 810 | 666 | 334 | 2601 | 2670 | 786 | 742 |

In this subsection, we present the results of SIM application with tuned parameters on the given SPL product configurations. We first present the results of small-scale SPL product configurations, followed by medium, large, and very large-scale configurations. For each of the different scale configuration results, initially, we discuss the results of ERP configurations, followed by SPLOT and BeTTy configurations. Later, we compare the results of all these configurations.
Table 7: SIM: Optimal Values for Parameters.

| Configurations   | Swarm Size | Inertia | Cognitive Attraction | Social Attraction |
|------------------|------------|---------|----------------------|-------------------|
| Small Scale      | 20         | 0.5     | 1.5                  | 1.5               |
| Medium Scale     | 100        | 0.95    | 1.5                  | 1.5               |
| Large Scale      | 800        | 0.95    | 1.5                  | 1.5               |
| Very Large Scale | 1000       | 0.95    | 1.5                  | 1.5               |

Table 8: SIM: Small-Scale Configurations Results; \(I = \) Number of Inconsistencies, \(F = \) Number of Features, \(\%\text{Dec} = \% \) Decrease, \(\%\text{Inc} = \% \) Increase.

| CID   | \(I\) \((\text{Org.})\) | \(F\) \((\text{Org.})\) | \(I\) \((\text{o-SPLIT})\) | \(F\) \((\text{o-SPLIT})\) | \% Dec. \((I)\) | \% Inc. \((F)\) | Time \((\text{sec.})\) |
|-------|-----------------|-----------------|-----------------|-----------------|----------------|----------------|----------------|
| ES-C2 | 57              | 32              | 0               | 68              | 100            | 112.5          | 1.98           |
| ES-C5 | 58              | 35              | 0               | 67              | 100            | 91.42          | 2.03           |
| ES-C4 | 39              | 42              | 0               | 68              | 100            | 61.90          | 2.05           |
| ES-C7 | 33              | 40              | 0               | 66              | 100            | 65             | 2.15           |
| SS-C4 | 74              | 41              | 0               | 62              | 100            | 51.21          | 2.25           |
| SS-C5 | 72              | 43              | 0               | 61              | 100            | 41.86          | 2.28           |
| SS-C2 | 59              | 39              | 0               | 63              | 100            | 61.53          | 2.32           |
| SS-C9 | 54              | 46              | 0               | 62              | 100            | 34.78          | 2.22           |
| BS-C5 | 68              | 51              | 0               | 62              | 100            | 21.56          | 2.4            |
| BS-C9 | 69              | 49              | 0               | 60              | 100            | 22.44          | 2.42           |
| BS-C4 | 42              | 34              | 0               | 60              | 100            | 76.47          | 2.38           |
| BS-C8 | 49              | 37              | 0               | 59              | 100            | 59.45          | 2.36           |

Table 9: SIM: Medium Scale Configurations Results; \(I = \) Number of Inconsistencies, \(F = \) Number of Features, \(\%\text{Dec} = \% \) Decrease, \(\%\text{Inc} = \% \) Increase.

| CID   | \(I\) \((\text{Org.})\) | \(F\) \((\text{Org.})\) | \(I\) \((\text{o-SPLIT})\) | \(F\) \((\text{o-SPLIT})\) | \% Dec. \((I)\) | \% Inc. \((F)\) | Time \((\text{Min.})\) |
|-------|-----------------|-----------------|-----------------|-----------------|---------------- |---------------- |---------------- |----------------|
| EM-C1 | 294             | 198             | 3               | 341             | 98.97          | 72.22          | 7.10            |
| EM-C6 | 303             | 192             | 1               | 347             | 99.66          | 80.72          | 6.98            |
| EM-C4 | 222             | 201             | 2               | 345             | 99.09          | 71.64          | 6.99            |
| EM-C10| 216             | 204             | 3               | 341             | 98.61          | 67.15          | 7.09            |
| SM-C6 | 384             | 209             | 2               | 328             | 99.47          | 56.93          | 6.29            |
| SM-C7 | 363             | 210             | 2               | 325             | 99.44          | 54.76          | 6.41            |
| SM-C3 | 264             | 200             | 3               | 323             | 98.86          | 61.50          | 6.20            |
| SM-C8 | 275             | 206             | 2               | 334             | 99.27          | 62.13          | 6.38            |
| BM-C3 | 338             | 215             | 2               | 332             | 99.40          | 54.31          | 6.35            |
| BM-C5 | 312             | 214             | 1               | 346             | 99.67          | 61.68          | 6.65            |
| BM-C4 | 268             | 226             | 2               | 339             | 99.25          | 50.00          | 6.78            |
| BM-C10| 269             | 231             | 2               | 344             | 99.25          | 48.917         | 6.46            |

to generate a consistent configuration for all configurations is almost similar (2.22-2.32 seconds). For BeTTy configurations, SIM was able to resolve 100% of the inconsistencies and increased the number of selected features by more than 76% in BS-C4, followed by BS-C8 (59%), BS-C9 (22%), and BS-C5 (23%). The time required to generate a consistent configuration for all configurations is almost similar (2.36-2.42 seconds).

Table 9 shows the medium-scale configuration results. For ERP configurations, SIM resolved almost 98-99% of the inconsistencies. Furthermore, it also increased the number of selected features by more than 80% in EM-C6, 72% in EM-C1 and EM-C4, and 67% in EM-C10. The time required to generate a consistent configuration is 6.98-7.10 minutes. For SPLOT configurations, SIM resolved 98-99% of the inconsistencies; it also increased the number of selected features by more than 2-3%. The time taken by SIM to generate a consistent configuration ranged between 147 to 153 minutes. Similar to ERP configurations, SIM removed 98% of the inconsistencies from the given SPLOT configurations. It also increased the number of selected features by more than 11% in SL-C7 and SL-C6, followed by SL-C3 (7%) and SL-C2 (3%). The time taken by SIM to generate a consistent configuration ranged between 6-7 minutes.

Table 10 shows the SIM results with large-scale configurations. SIM removed almost 98% of the original inconsistencies from the given ERP configurations and increased the number of selected features by more than 2.3%. The time taken by SIM to generate a consistent configuration ranged between 147 to 153 minutes. Similar to ERP configurations, SIM removed 98% of the inconsistencies from the given SPLOT configurations. It also increased the number of selected features by more than 11% in SL-C7 and SL-C6, followed by SL-C3 (7%) and SL-C2 (3%). The time taken by SIM to generate a consistent configuration ranged between
TABLE 10. SIM: Large-Scale Configurations Results; I = Number of Inconsistencies, F = Number of Features, %Dec = % Decrease, %Inc = % Increase.

| CID   | I (Org.) | F (Org.) | I (o-SPLIT) | F (o-SPLIT) | % Dec. (I) | % Inc. (F) | Time (Min.) |
|-------|----------|----------|-------------|-------------|------------|------------|-------------|
| EL-C5 | 606      | 620      | 7           | 634         | 98.84      | 2.25       | 151         |
| EL-C8 | 602      | 623      | 10          | 641         | 98.33      | 2.88       | 153         |
| EL-C2 | 478      | 624      | 9           | 645         | 98.11      | 3.36       | 149         |
| EL-C3 | 462      | 618      | 8           | 639         | 98.26      | 3.39       | 147         |
| SL-C7 | 671      | 553      | 8           | 618         | 98.80      | 11.75      | 146         |
| SL-C6 | 660      | 569      | 7           | 632         | 98.93      | 11.07      | 138         |
| SL-C2 | 501      | 619      | 6           | 636         | 98.80      | 2.74       | 139         |
| SL-C3 | 494      | 598      | 9           | 637         | 98.17      | 6.52       | 143         |
| BL-C4 | 570      | 559      | 4           | 615         | 99.29      | 10.01      | 132         |
| BL-C10| 570      | 575      | 2           | 617         | 99.64      | 7.30       | 135         |
| BL-C9 | 478      | 581      | 3           | 612         | 99.37      | 5.33       | 141         |
| BL-C1 | 464      | 566      | 4           | 621         | 99.13      | 9.72       | 129         |

TABLE 11. SIM: Very Large-Scale Configurations Results; I = Number of Inconsistencies, F = Number of Features, %Dec = % Decrease, %Inc = % Increase.

| CID    | I (Org.) | F (Org.) | I (o-SPLIT) | F (o-SPLIT) | % Dec. (I) | % Inc. (F) | Time (Days) |
|--------|----------|----------|-------------|-------------|------------|------------|-------------|
| EVL-C9 | 2974     | 2932     | 117         | 3201        | 96.06      | 7.48       | 2.1-3.9     |
| EVL-C10| 2991     | 2991     | 115         | 3215        | 96.15      | 5.86       | 3.0-5.4     |
| EVL-C7 | 2405     | 2879     | 96          | 3152        | 96.00      | 9.48       | 2.9-5.3     |
| EVL-C8 | 2399     | 2950     | 85          | 3123        | 96.45      | 5.86       | 2.3-5.0     |
| BVL-C1 | 2618     | 2873     | 127         | 3189        | 95.14      | 10.99      | 3.0-5.6     |
| BVL-C2 | 2517     | 2539     | 119         | 3034        | 95.27      | 19.49      | 2.7-5.8     |
| BVL-C8 | 1907     | 2667     | 101         | 3137        | 94.70      | 17.62      | 3.1-5.9     |
| BVL-C5 | 1873     | 2935     | 93          | 3201        | 95.03      | 9.06       | 2.8-5.7     |

C. DISCUSSION OF SIM RESULTS AND ANSWER TO RESEARCH QUESTION

In this subsection, we present the analysis of SIM results with ERP, SPLOT, and BeTTy configurations. We also analyze the effects of configuration size, configuration type, and a number of inconsistencies in the configurations on SIM.

Figure 13 shows a comparison of the number of features selected in optimized configurations generated by SIM to the original configurations. An increase or decrease in the number of selected features is dependent on two factors, the number of features in original configurations, and the distribution of different inconsistency types in the original configurations. The results of SIM validate that the existence of fewer features in original configurations increases the chances of selection of a greater number of features in optimized configurations. For instance, EM-C6 (from ERP medium-scale configurations set) contained the least number of features, i.e., 192. Furthermore, EM-C6 showed the highest features selection increment, i.e., 80% (shown in Table 9) during optimization. The optimization of SM-C3 (from the medium-scale SPLOT configurations set) validates the effects of inconsistency types on the feature selection, i.e., the existence of a greater number of mandatory and include inconsistencies increases the chances of a higher number of feature selection, while alternative and exclude inconsistencies decreases this possibility. Moreover, as shown in Figure 13, the large and very large-scale configurations show a little increase in feature selection as compared to small and medium-scale configurations. This shows an equal trade-off between features selection (due to mandatory and include inconsistencies) and features deselection (due to the existence of exclude and alternative inconsistencies).

SIM resolved inconsistencies from all given configuration sizes. SIM resolved 100% inconsistencies from small-scale configurations, followed by medium (98-99%), large (98-99%) and very large-scale configurations (94-96%). These results show that the performance of SIM in terms of inconsistency resolution decreases, i.e., 1-5% with an
increase in the size of configurations. However, this decrease is independent of the number of inconsistencies in the original configurations. For instance, SIM produced almost equally optimized configurations with highly inconsistent (EVL-C9, EVL-C1) and the least inconsistent configurations. This is true across all the given configurations as shown in Figure 14. SIM not only produces the optimal solutions with the least inconsistent configurations, but also generates almost equally optimized solutions for the configurations having a higher number of inconsistencies. Thus, SIM optimizes both types of configurations, i.e., highly inconsistent and least inconsistent ones.

Figure 15 (A) shows the inconsistencies resolution comparison of SIM results for the configurations generated through ERP, BeTTy, and SPLOT. For small-scale configurations, SIM resolved 100% of the inconsistencies from ERP, BeTTy, and SPLOT configurations. For all medium-scale configurations, SIM resolved 99% of the inconsistencies, followed by large-scale configurations (98%). For very large-scale configurations, SIM resolved 96% of the inconsistencies from ERP configurations, followed by BeTTy configurations, i.e., 95%. Figure 15 (A) also depicts a little decline in the optimality of the solutions with respect to the configuration sizes, i.e., small to very large. SIM does not show a particular trend in feature selection (shown in Figure 15 (B)). As discussed earlier, the number of features is a trade-off between feature selection and deselection, which is dependent on different inconsistency types. These results validate the flexibility of SIM across different configuration domains and scales.

Figure 16 shows the efficiency of SIM for different scales of configuration. For efficiency comparison, we averaged...
the time taken by SIM to generate small, medium, and large sized optimized configurations for ERP, BeTTy, and SPLOT. For very large-scale configurations, we rather measured the efficiency of SIM by recording the two time limits, i.e., the time to generate the most efficient and the least efficient configurations. These results show that the efficiency of SIM is proportional to the configuration size. For small and very large-scale configurations, SIM generated the most efficient results with ERP configurations, while for medium and large-scale configurations, SPLOT and BeTTy configurations produced the most efficient results respectively.

Based on SIM results, we answer RQ1 as follows: SIM optimizes all given configurations of different scales and domains. For all small-scale SPL product configurations, SIM produced 100% consistent configurations, and for medium-scale, large-scale and very large-scale configurations, SIM resolved 95-99% inconsistencies. Hence, we can experimentally validate the application of Swarm Intelligence (SI) to resolve the SPL product inconsistencies.

VI. O-SPLIT: A CONTROLLED EVALUATION

In this section, we discuss the implementation and evaluation of o-SPLIT and answer RQ2. We implemented o-SPLIT in our client organization, whose FM and datasets were used to run our experiments. It is important to mention here that the evaluation is a controlled one, where we took professionals from the industry to participate in the study.

We developed a testing environment to configure a medium-scale ERP product for a team of 25 developers (10 juniors and 15 seniors). We setup 10 test servers and a database server, where a test server was assigned to every developer. All test servers were connected to the database server for sharing SPL repositories and were equipped with o-SPLIT interface, while the database server was populated with the configuration repositories of o-SPLIT. o-SPLIT repositories were also populated with the test FM data for medium-scale configuration.

The testing process started with the domain engineering of medium-scale FM. After that, a product was configured for an exemplary client. The final product configuration contained inconsistencies because of the involvement of junior developers, who were not an expert of the ERP domain. Optimized configurations using o-SPLIT were then generated from the medium-scale inconsistent product configuration.

After the successful execution of o-SPLIT in the testing environment, we acquired the feedback from the developers involved in the test configuration through a subjective questionnaire. We designed this questionnaire according to the standard guidelines for questionnaire design [47]. This feedback was acquired anonymously from developers, marked on a scale of 1 (strongly disagree) to 5 (strongly agree). The questions are as follows:

- Q1: o-SPLIT reduces the overall complexity of the configuration process.
- Q2: The features set generated through o-SPLIT is optimized (performance of o-SPLIT in terms of inconsistencies resolution).
- Q3: The inconsistencies are removed and the configuration is consistent.
• Q4: It is efficient as compared to manual configurations.
• Q5: It has a practical applicability to the business domain.

We circulated the questionnaire to the developers and calculated the average response for each question:
• Q1, Q2, and Q4 received an average rating of 5, i.e., all developers unanimously concurred with the efficiency of -SPLIT, as compared to their manual efforts.
• Q3 received an average rating of 4, i.e., developers endorsed the consistency of configurations generated through -SPLIT.
• Q5 received a normal response with an average rating of 4.5, i.e., developers are sure about the industrial practicability of -SPLIT.

Considering these results, we believe that -SPLIT has the potential to significantly and positively impact the SPL product configuration problems.

VII. -SPLIT: STATE-OF-THE-ART COMPARISON
We compare -SPLIT with two industrial tools, i.e., Gears and Purevariants as shown in Table 12. -SPLIT provides additional pre-configuration support by generating multiple optimized solutions based on previously configured product. Clients can select an optimized configuration according to their requirements. -SPLIT also generates consistent configurations from a given inconsistent configuration. On the other hand, Purevariants resolves mandatory and include inconsistencies by automatically selecting the relevant features. It identifies exclude and alternative constraints, but does not resolve them automatically; it only lists them for further manual actions. Similar to Purevariants, the Gears tool also identifies inconsistencies, i.e., automatically resolve mandatory and include constraints, and list the remaining ones for manual actions. Nevertheless, -SPLIT not only identifies and resolves the mandatory and include inconsistencies but also identifies and resolves the alternative inconsistencies. Table 12 shows this comparison in detail.

| Features                        | Gears | Purevariants | -SPLIT |
|---------------------------------|-------|--------------|--------|
| Generate Predefined Configuration | N/A   | N/A          | ✓      |
| Identify Mandatory Constraints   | ✓     | ✓            | ✓      |
| Resolve Mandatory Constraints    | ✓     | ✓            | ✓      |
| Identify Include Constraints     | ✓     | ✓            | ✓      |
| Resolve Include Constraints      | ✓     | ✓            | ✓      |
| Identify Alternative Constraints | ✓     | ✓            | ✓      |
| Resolve Alternative Constraints  | ×     | ✓            | ✓      |
| Generate Consistent Configurations | N/A  | N/A          | ✓      |

Table 13 presents a comparison of -SPLIT with other state-of-the-art techniques. First, we picked our problem domain, i.e., inconsistencies in SPL configurations, for this, we selected only Artificial Intelligence (AI) solutions since the base of -SPLIT is optimization, which is a sub-domain of AI. Then, we expand the comparison by picking our solution domain, i.e., optimization, for this, we selected those research works which are based on optimization to solve the SPL configuration issues.

In [10], the authors propose a framework to identify and resolve the inconsistencies by proposing a description logic-based solution. The SPL product which is used to test the framework has a limited feature set, i.e., 35 features. The framework takes an inconsistent configuration; identifies and corrects the inconsistencies, and returns a minimal set of consistent features. This solution resolves inconsistencies for all given configurations but shows higher identification and resolution times for large-scale models. Similarly, in [13], the authors use ontology to propose a semantic web approach to identify the inconsistencies. This work only focuses on the inconsistencies present in FM, rather than the product configuration. It takes an inconsistent FM as an input and generates a consistent FM. For testing, an in-house FM with 1000 features is used.

A knowledge-based approach to solve the inconsistencies is also presented in [12]. Here, the focus is on inconsistent FM due to dead and inconsistent features. For this, the authors convert the FM into a Knowledge Base (KB) and generate a list of inconsistent and dead features. The FM with 35 features is used to validate the work. Trinidad et al. [18] also present a CSP (Constraint Satisfaction Problem) based framework to diagnose the FM inconsistencies. The experiments are performed on large-scale FM with 5000 features.

Similar to our work, [21] and [48] use optimization to solve the inconsistencies; both works take an FM as input and generate different consistent configurations from this given FM. These consistent predefined configurations can be implemented for a SPL client as-is. However, in a real-world scenario, the users have their own wish list of features and they want the product configuration according to their choice. In this context, the predefined configuration cannot be a good solution, and the developers need an automated support to generate the consistent configuration from a given inconsistent product, configured at runtime. -SPLIT fills this gap as we mentioned in the results section. Besides that, -SPLIT can also generate a predefined set of consistent configurations (mentioned in Table 12).

Similar to [21] and [48]; [14]–[17], [19], [20], [49] also generate optimized predefined configurations from the given FM based on the given objective (listed in Table 13). They all are different from -SPLIT in terms of objective functions and the input criterion, i.e., FM.

VIII. THREATS TO VALIDITY
In this section, we explain the potential threats to validity of our research work. Reference [50] proposes a systematic approach to evaluate the validity threats for empirical software engineering. We adapted this approach to analyse the possible threats to our work. We also discuss the actions that we have taken to reduce the effects of these threats.

A. INTERNAL THREATS TO VALIDITY
These threats refer to any confounding element that can affect the outcomes. -SPLIT implements PSO. For this, we modified the PSO code available on MathWorks...
TABLE 13. o-SPLIT: Comparison with state-of-the-art; IncId = Inconsistencies Identification, IncRe = Inconsistencies Resolution.

| Research Work | Technique Used        | Product Size | IncId | IncRe    | Input    | Output                      |
|---------------|-----------------------|--------------|-------|----------|----------|-----------------------------|
| [10] A. O. Elfaki, S. Phon-Amnuaisuk, and C. K. Ho, “Knowledge based feature model debugging based on description logic reasoning,” in Proc. DMS, vol. 9, 2009, pp. 9–15. | Description Logic | 35 Features | Y     | Y        | Inconsistent Configuration | Consistent Configuration |
| [11] P. Trinidad and A. R. Cortés, “Abductive reasoning and automated analysis of large software product lines,” in Proc. 16th Int. Softw. Product Line Conf., 2010, pp. 337–346. | Knowledge base | 35 Features | Y     | N        | Inconsistent FM             | List of Inconsistencies |
| [12] M. Noorian, A. Ensan, E. Bagheri, H. Boley, and Y. Biletskiy, “Feature model debugging based on description logic reasoning,” in Proc. DMS, vol. 11, 2011, pp. 158–164. | Ontology | 1000 Features | Y     | Y        | Inconsistent FM             | Consistent FM            |
| [13] N. Niu, J. Savolainen, and Y. Yu, “Variability modeling for product line viewpoints integration,” in Proc. IEEE 34th Annu. Comput. Softw. Appl. Conf., Jul. 2010, pp. 337–346. | Constraint Satisfaction Problem | 5000 Features | Y     | Y        | Inconsistent FM             | Consistent FM            |
| [14] R. Flores, C. Krueger, and P. Clements, “Mega-scale product line engineering at general motors,” in Proc. Software Product Line Eng., vol. 8142. Saarbrücken, Germany: Dagstuhl Seminar, 2008. | Optimization | 6000 Features | N/A   | N/A      | FM                    | Consistent Configuration |
| [15] J. Syst. Softw., vol. 30, no. 7, pp. 825–847, 2000. | Optimization | 2500 Features | N/A   | N/A      | FM                    | Consistent Configuration |
| [16] J. Van Gurp and C. Prehofer, “From sps to open, compositional platforms,” in Combining the Advantages of Product Lines and Open Source Forms, vol. 8142. Saarbrücken, Germany: Dagstuhl Seminar, 2008. | o-SPLIT | Optimization | 5000 Features | Y     | Y        | Inconsistent Configuration | Consistent Configuration |

B. EXTERNAL THREATS TO VALIDITY

We conducted experiments with 44 datasets of different sizes. 14 datasets are derived from real-world industrial ERP feature models, while the rest are generated through automated feature model generators (SPLOT-FM and BeTTy). These system-generated datasets pose a threat to validity because they do not contain real-world inconsistencies. However, SPLOT and BeTTy are two big names in SPL industry and many researchers use them to run their SPL experiments [36], [51]. To reduce this threat to generalizability, we generated 10 configurations for every configuration group (small, medium, large, and very large) and selected two configurations with the highest number of inconsistencies and two configurations with the least number of inconsistencies. This configuration selection process is appropriate to select the configurations similar to the real-world configurations. Moreover, we developed and evaluated o-SPLIT in a controlled testing environment rather than an artificial environment.

IX. CONCLUSION AND FUTURE WORK

In this article, we comprehensively explore the application of swarm intelligence algorithms to resolve the product inconsistencies in SPLs. We name our tool as o-SPLIT. We select and fine-tune the widely applied Particle Swarm Optimization (PSO) algorithm. We also select standardized ERP, SPLOT, and BeTTy feature models, along with four different feature set size configurations. Our results show that PSO has the potential to generate almost 100% optimized feature models in an incomparably lesser time as compared to the manual feature model configuration. We then implement o-SPLIT as a decision-support tool in a real-life SPL setting, and obtain subjective responses regarding its performance from representative feature model designers. The results show that the designers are convinced of the high-level decision support provided by o-SPLIT.

These results have motivated us, as a future work, to perfect our technology at the enterprise level so that it can be seamlessly integrated in a standard industrial SPL/ERP setting. We plan to implement this offering initially using a cloud-based SaaS model.

REFERENCES

[1] K. Pohl, G. Böckle, and F. Van Der Linden, Software Product Line Engineering, vol. 10. Berlin, Germany: Springer-Verlag, 2005, pp. 540–543.
[2] A Framework for Software Product Line Practice, Version 5.0, SEI, Oaks, PA, USA, 2012.
[3] A. Hubaux, “Feature-based configuration: Collaborative, dependable, and controlled,” Ph.D. dissertation, Dept. Comput. Sci., Univ. Namur, Namur, Belgium, 2012.
[4] M. Ardis, N. Daley, D. Hoffman, H. Suy, and D. Weiss, “Software product lines: A case study,” Softw.-Pract. Exper., vol. 30, no. 7, pp. 825–847, 2000.
[5] M. Noorian, A. Ensan, E. Bagheri, H. Boley, and Y. Biletskiy, “Feature model debugging based on description logic reasoning,” in Proc. DMS, vol. 11, 2011, pp. 158–164.
[6] J. Van Gurp and C. Prehofer, “From sps to open, compositional platforms,” in Combining the Advantages of Product Lines and Open Source Forms, vol. 8142. Saarbrücken, Germany: Dagstuhl Seminar, 2008.
[7] J. White, D. Benavides, D. C. Schmidt, P. Trinidad, B. Dougherty, and A. Ruiz-Cortes, “Automated diagnosis of feature model configurations,” J. Syst. Softw., vol. 83, no. 7, pp. 1094–1107, Jul. 2010.
[8] M. Noorian, A. Ensan, E. Bagheri, H. Boley, and Y. Biletskiy, “Feature model debugging based on description logic reasoning,” in Proc. DMS, vol. 11, 2011, pp. 158–164.
[9] P. Trinidad and A. R. Cortés, “Abductive reasoning and automated analysis of feature models: How are they connected,” in Proc. 17th Int. Softw. Product Line Conf. SPLC, vol. 1, 2012, pp. 259–268.
A. S. Sayyad, J. Ingram, T. Menzies, and H. Ammar, "A genetic algorithm for optimized feature selection with resource constraints in software product lines," J. Syst. Softw., vol. 84, no. 12, pp. 2088–2091, Dec. 2011.

Z.-Q. Wu, T. Jia-fu, and W. Li-yan, "An optimization framework for reuse component selection in software product line," in Proc. Chin. Control Decis. Conf., Jun. 2009, pp. 1880–1884.

J. Muller, "Value-based portfolio optimization for software product lines," in Proc. 15th Int. Softw. Product Line Conf., Aug. 2011, pp. 15–24.

Y.-L. Wang and J.-W. Pang, "Ant colony optimization for feature selection in software product lines," J. Shanghai Jiaotong Univ. (Sci.), vol. 19, no. 1, pp. 50–58, Feb. 2014.

P. Trinidad, D. Benavides, A. Durán, A. Ruiz-Cortés, and M. Toro, "Automated error analysis for the agilization of feature modeling," J. Syst. Softw., vol. 81, no. 6, pp. 883–896, Jun. 2008.

E. Bagheri, M. Asadi, D. Gasevic, and S. Soltani, "Stratified analytic hierarchy process: Prioritization and selection of software features," in Proc. International Conference on Software Product Lines, 2010, pp. 300–315.

J. Cruz, P. S. Neto, R. Britto, R. Rabelo, W. Ayala, T. Soares, and M. Mota, "Toward a hybrid approach to generate software product line portfolios," in Proc. IEEE Conf. Evol. Comput., Jun. 2013, pp. 2229–2236.

A. A. Sayyad, J. Ingram, T. Menzies, and H. Ammar, "Scalable product line configuration: A straw to break the camel's back," in Proc. 28th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE), Nov. 2013, pp. 465–474.

U. Afzal, T. Mahmood, and Z. Shaikh, "Intelligent software product line configurations: A literature review," Comput. Standards Interface, vol. 48, pp. 30–48, Nov. 2016.

K.-F. Man, K.-S. Tang, and S. Kwong, "Genetic algorithms: Concepts and applications," IEEE Trans. Ind. Electron., vol. 43, no. 5, pp. 519–534, Oct. 1996.

R. T. Marler and J. S. Arora, "Survey of multi-objective optimization methods for engineering," Struct. Multidisciplinary Optim., vol. 26, no. 6, pp. 369–395, Apr. 2004.

A. A. Sayyad, J. Ingram, T. Menzies, and H. Ammar, "Optimum feature selection in software product lines: Let your model and values guide your search," in Proc. 1st Int. Workshop Combining Modeling Search-Based Softw. Eng. (CMSBSE), May 2013, pp. 22–27.

H. I. Alsawalqah, S. Kang, and J. Lee, "A method to optimize the scope of a software product platform based on end-user features," J. Syst. Softw., vol. 98, pp. 79–106, Dec. 2014.

J. White, B. Dougherty, and D. C. Schmidt, "Selecting highly architectural feature sets with filtered Cartesian flattening," J. Syst. Softw., vol. 82, no. 8, pp. 1268–1284, Aug. 2009.

U. Afzal, T. Mahmood, I. Rauf, and Z. A. Shaikh, "Minimizing feature model inconsistencies in software product lines," in Proc. 17th IEEE Int. Multi Topic Conf., Dec. 2014, pp. 137–142.

A. E. Eiben and J. E. Smith, Introduction to Evolutionary Computing, vol. 53. Berlin, Germany: Springer-Verlag, 2003.

C. Blum and X. Li, "Swarm intelligence in optimization," in Swarm Intelligence, Berlin, Germany: Springer-Verlag, 2008, pp. 43–85.

D. Luo and S. Diao, "Feature dependency modeling for software product line," in Proc. Int. Conf. Comput. Eng. Technol., Jan. 2009, pp. 256–260.

C. Thao, E. V. Munson, and T. N. Nguyen, "Software configuration management for product derivation in software product families," in Proc. 15th Annu. IEEE Int. Conf. Workshop Eng. Comput. Based Syst. (ECBS), Mar. 2008, pp. 265–274.

E. Shehab, M. Sharp, L. Supramaniam, and T. A. Spedding, "Enterprise resource planning: An integrative review," Bus. Process Manage. J., vol. 10, no. 4, pp. 359–386, 2004.

A. Hawari and R. Heeks, "Explaining ERP failure in a developing country: A Jordanian case study," J. Enterprise Inf. Manage., vol. 23, no. 2, pp. 135–160, Feb. 2010.

SPLC. (2014). Software Product Line Online Tools. [Online]. Available: http://www.splot-research.org/

S. Segura, J. A. Galindo, D. Benavides, J. A. Parejo, and A. Ruiz-Cortés, "BeTTy: Benchmarking and testing on the automated analysis of feature models," in Proc. 6th Int. Workshop Variability Modeling Softw.-Intensive Syst. ViMoS, 2012, pp. 63–71.

J. Kennedy and R. Eberhart, "Particle swarm optimization," in Proc. IEEE ICNN, vol. 4, Nov./Dec. 1995, pp. 1942–1948.

A. Bajaj and O. P. Sangwan, "A systematic literature review of test case prioritization using genetic algorithms," IEEE Access, vol. 7, pp. 126355–126375, 2019.

Z. Wang, J. Li, K. Fan, W. Ma, and H. Lei, "Prediction method for low speed characteristics of compressor based on modified similarity theory with genetic algorithm," IEEE Access, vol. 6, pp. 36834–36839, 2018.

Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in Proc. IEEE Int. Conf. Evol. Comput. World Congr. Comput. Intell., May 1998, pp. 69–73.

K. Deb, Multi-Objectvie Optimization Using Evolutionary Algorithms, 1st ed. Hoboken, NJ, USA: Wiley, 2001.

C. Belzal, J. A. Fernández del Poozo, and P. Larrañaga, "Parameter control of genetic algorithms by learning and simulation of Bayesian networks—A case study for the optimal ordering of tables," J. Comput. Sci. Technol., vol. 28, no. 4, pp. 720–731, Jul. 2013.

A. Arcuri and L. Briand, "A Hitchhiker’s guide to statistical tests for assessing randomized algorithms in software engineering," Softw. Test., Verification Rel., vol. 24, no. 3, pp. 219–250, May 2014.

P. K. Tripathi, S. Bandyopadhyay, and S. K. Pal, "Multi-objective particle swarm optimization with time variant inertia and acceleration coefficients," Inf. Sci., vol. 177, no. 22, pp. 5033–5049, Nov. 2007.

A. Rataweera, S. K. Halgamuge, and H. C. Watson, "Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients," IEEE Trans. Evol. Comput., vol. 8, no. 3, pp. 240–255, Jun. 2004.

Y. Shi and R. C. Eberhart, "Empirical study of particleswarm optimization," in Proc. Congr. Evol. Comput.-CEC, Jul. 1999, pp. 1945–1950.

I. Brace, Questionnaire Design: How to Plan, Structure and Write Survey Material for Effective Market Research. London, U.K.: Kogan Page Publishers, 2008.

A. S. Sayyad, T. Menzies, and H. Ammar, "On the value of user preferences in search-based software engineering: A case study in software product lines," in Proc. 35th Int. Conf. Softw. Eng. (ICSE), May 2013, pp. 492–501.

J. White, B. Dougherty, and D. C. Schmidt, "Filtered Cartesian flattening: An approximation technique for optimally selecting features while adhering to resource constraints," in Proc. SPLC, vol. 2, 2008, pp. 209–216.

M. de Oliveira Barros and A. C. Dias-Neto, "0006/2011-threats to validity in search-based software engineering empirical studies," RelaTe-DIA, vol. 5, no. 1, pp. 1–12, 2011.

C. Camillieri, L. Parisi, M. Blay-Fornarino, F. Precioso, M. Rivelle, and J. Cancela-Vaz, "Towards a software product line for machine learning workflows: Focus on supporting evolution," in Proc. 10th Workshop Mod- els Evol. Co-Located ACM/IEEE 19th Int. Conf. Model Driven Eng. Lang. Syst. (MODELS), Oct. 2016, pp. 1–6.

UZMA AFZAL received the Ph.D. degree in computer science from the National University of Computer and Emerging Sciences, Pakistan. She is currently an Assistant Professor with the Federal Urdu University of Arts Science and Technology, Karachi, Pakistan. Her research interests include software engineering, software product line, artificial intelligence, big data, and product configuration. She also worked on research projects with Federal Urdu University. She has authored many research papers in conferences and journals of international repute, including Elsevier and IEEE.

TARIQ MAHMOOD received the M.S. degree in statistical machine learning from Universite Pierre et Marie Curie (Paris 6), France, and the Ph.D. degree in machine learning from the University of Trento, Italy. He is currently an Associate Professor with the Faculty of Computer Science, Institute of Business Administration (IBA), Karachi, Pakistan. He has published around 20 international journal and 35 conference publications with total 691 citations and H-index of 12 (Google Scholar). His research interests include BDA, deep learning, and machine learning/data science. He heads the Big Data Analytics Laboratory, IBA, with a focus on imparting data science and big data certifications to students and industry professionals, implementing BDA-related industrial projects and researching in BDA technology stack, particularly to develop BDA architectures for different types of streaming and non-streaming data. He also consults in various local industries regarding business intelligence, data governance, BDA, and machine learning.
AYAZ H. KHAN received the bachelor’s degree (Hons.) in computer science and information technology from NED University, Pakistan, the M.S. degree in computer science (Hons.) from the Lahore University of Management Sciences, and the Ph.D. degree in computer science and engineering from the King Fahd University of Petroleum and Minerals (KFUPM), Saudi Arabia. He is currently working as an Assistant Professor with the Dhanani School of Science and Engineering, Habib University, Karachi, Pakistan. In addition to these, he possesses a dozen of professional certifications for various technical and soft skills. As a Computer Scientist with interest in high-performance computing, parallel programming, and deep learning, he is the author of more than 20 publications in reputed journals and conferences along with a book on parallel processing. He has successfully completed several research funded projects and few more are in process of accumulated worth of about half a million dollars.

SADEEQ JAN received the B.Sc. degree in engineering from the University of Engineering and Technology, Peshawar, Pakistan, in 2004, the M.Sc. degree from the KTH, Royal Institute of Technology, Sweden, in 2007, and the Ph.D. degree (Hons.) from the University of Luxembourg, in 2017, with the thesis entitled Automated and Effective Security Testing for XML-Based Vulnerabilities. He is currently an Assistant Professor and the Scientific Director of the National Centre for Cyber Security (NCCS-UETP), University of Engineering and Technology, Peshawar. He has worked as an Information Security Consultant in Sweden for several years providing services in the areas of penetration testing, PCI testing, ISO27001: 2005 compliance process, and developing security policies for organizations. His research interests include information security, software testing and verification, search-based software engineering, and security testing. He was awarded the Presidential Award (IZAZ-E-SABQAT) and the University Gold Medal for academic excellence in Pakistan.

RAIHAN UR RASOOL is currently a Senior IT Consultant with the largest Australian technology company. He is also affiliated with Victoria University as a Research Scientist. He is also a Fulbright alumnus of the University of Chicago, USA. His research interests include large-scale systems, security, and computer architecture. His research work, comprising over 80 papers is published in various international conferences and journals, such as ISCA, HiPEC, CCGrid, ACM SIGARCH, the IEEE TRANSACTIONS ON CLOUD COMPUTING, JNCA, and FGCS.

ALI MUSTAFA QAMAR (Member, IEEE) received the B.E. degree (Hons.) in computer software engineering from the National University of Sciences and Technology, Pakistan, in 2005, the M.S. degree in computer science from University Joseph Fourier (UJF), Grenoble, France, in 2007, and the Ph.D. degree in computer science from the University of Grenoble, France, in 2010. He has worked as a Temporary Assistant Professor with UJF, from November 2010 to June 2011. He has been an Assistant Professor of computer science with the College of Computer, Qassim University, Buraydah, Saudi Arabia, since 2014, and an Assistant Professor of computer science with the School of Electrical Engineering and Computer Science, National University of Sciences and Technology, Islamabad, Pakistan, since 2011. His research interests include machine learning, data mining, deep learning, social networks, and information filtering.

REHAN ULLAH KHAN received the B.Sc. degree in information systems from the University of Engineering and Technology Peshawar, in 2004, the M.Sc. degree in information systems in 2006, and the Ph.D. degree from the Vienna University of Technology, Austria, in 2011. He is currently an Assistant Professor and the Director Research and Development at CoC, Qassim University, Saudi Arabia. His current research interest includes pattern analysis and its applications in multiple disciplines.

***