Information Sciences 474 (2019) 33–47

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Constraint programming heuristics for configuring optimal products in multi product lines

Lina Ochoa[a,c], Oscar González-Rojas[a,*], Nicolás Cardozo[a], Alvaro González[a], Jaime Chavarriaga[a], Rubby Casallas[a], Juan Francisco Díaz[b]

[a] Systems and Computing Engineering Department, Universidad de los Andes, Bogotá, Colombia
[b] AVISPA, Universidad del Valle, Cali, Colombia
[c] Centrum Wiskunde & Informatica, Amsterdam, The Netherlands

ARTICLE INFO

Article history:
Received 8 March 2018
Revised 17 September 2018
Accepted 20 September 2018
Available online 20 September 2018

Keywords:
Variability modeling
Feature modeling
Multi product lines
Configuration management
Cross-model constraints
Constraint programming

ABSTRACT

Nowadays, complex application domains require configuring multi-product lines where product features and constraints among them are specified in several variability models. These variability models are enriched with inter-model constraints representing the existing relations among domain concerns, and with non-functional properties modeled as attributes attached to product features. Multiple techniques use constraint programming to automate the cumbersome task of manually configuring a suitable product. Currently, there are some proposals to improve the performance of constraint solvers when configuring single-model product lines, however configuration scenarios with multiple interrelated and attributed models are not yet targeted. This paper proposes and evaluates three search heuristics used to configure optimal products regarding multi-objective criteria. Results are compared against the default search strategy of the Choco constraint solver. We evaluated the performance for configuring optimal products in four state-of-the-art product lines and 130 generated variability models representing multi-product lines that scale up to 6400 features and 960 constraints. As a result, we observe that the proposed heuristics perform better than the default solver strategy when the variability models scale in terms of features. In contrast, the default strategy and one of the proposed heuristics perform better as the number of interdependencies between variability models increases.

1. Introduction

Product line engineering is a paradigm that supports mass customization in a given domain, by exploiting commonalities and variations in a family of products [32]. *Feature models* are widely used to represent product lines variability. These models are enriched with constraints that describe which features can be selected to derive a product. The selection and deselection of the set of features of a product line is known as a *product configuration*, which turns to be valid if it satisfies the constraints specified in the feature model (see Section 2).

* Corresponding author.
E-mail addresses: lm.ochoa750@uniandes.edu.co (L. Ochoa), o-gonza1@uniandes.edu.co (O. González-Rojas), n.cardozo@uniandes.edu.co (N. Cardozo), ad.gonzalez1@uniandes.edu.co (A. González), ja.chavarriaga908@uniandes.edu.co (J. Chavarriaga), rcasalla@uniandes.edu.co (R. Casallas), juanfco.diaz@correounivalle.edu.co (J.F. Díaz).

https://doi.org/10.1016/j.ins.2018.09.042
0020-0255/

Multi-Product Line (MPLs) have been proposed to support the increasing need to represent complex domains [26]. In a MPL, the variability is specified using multiple feature models, custom attributes in the features, and relationships among the models. Although these additional elements support the specification of non-functional requirements and interactions among different domains, they also demand the improvement of performance and scalability aspects during product configuration [26,27]. This is especially important when the introduced non-functional requirements are translated into optimization problems where a predefined set of numerical non-functional values related to product features should be minimized or maximized. To face this challenge, some approaches based on *constraint programming* have been developed to solve single-objective optimization [8,9,20,24,41], and multi-objective optimization [16,29,35] problems. These approaches provide a product line configuration in between 1.8 s [20] and 90 min [9] depending on the product line size (models with up to 5000 features), the number of optimization objectives (e.g., 1 to 5), and other confounding variables that are meant to be explored and discovered by researchers in the field (e.g., type of involved constraints). A detailed analysis of the related work is presented in Section 3.

Even though there are well known solutions that tackle optimal products configuration, we consider two open research challenges that, to the best of our knowledge, have not been yet considered by the research community: **(1)** scenarios where product line representations are enhanced with *attributes* (to represent non-functional properties) and MPLs (to represent modular lines) should be further explored; and **(2)** most of the analyzed constraint programming approaches do not consider the characteristics of the configuration problem (e.g., types of constraints, optimization objectives) to guide an efficient search in the solution space. We define the following research questions to address the aforementioned gaps.

RQ1 How to improve performance of constraint programming solutions for finding optimal configurations of attributed MPLs as they scale?

RQ2 How to identify which improvement technique (from those defined in **RQ1**) is best suited (in terms of performance and scalability) in different scenarios for configuring optimal products in attributed MPLs?

To answer RQ1 we define three constraint programming heuristics, namely the *constraint instantiation degree, cross-tree and model constraints*, and *optimization in or-constraints* heuristics. Such heuristics are used as part of the search strategy of a *constraint satisfaction problem* resolution. The proposed heuristics are incorporated into the CoCo program synthesis [23–25], enriching the existing constraint programming encoding. This contribution is presented in Section 4.

Regarding **RQ2**, we conduct three different experiments to evaluate heuristics' performance and scalability when configuring products in attributed MPLs. The first and second experiment consider 130 generated variability models representing attributed MPLs that scale up to 6400 features and 960 constraints. The first experiment varies the number of features per MPL, whilst the second experiment varies the density of constraints among product lines. Lastly, the third experiment considers four product lines taken from the current state-of-the-art, namely DecISIonAL [10], Drupal [30], E-Shop [15], and Web Portal [21] models. In all cases we measure the resolution time of product configuration. We compare the performance of the proposed heuristics against the default search heuristic of an off-the-shelf solver, namely the *weighted degree* heuristic.

As a result, we observe that the proposed heuristics perform better than the default solver strategy when the variability models scale in terms of features. In contrast, the default strategy and the *constraint instantiation degree* heuristic perform better as the number of interdependencies between variability models increases. The evaluation results are presented in Section 5. Finally, conclusions and future work are presented in Section 6.

2. Configuration of multi-product lines: Core concepts

This section gives an overview of the core concepts involved in Product Line (PL) configuration: feature models used to represent a PL, feature-solution graphs used to represent relationships and dependencies among MPLs, and the processes that support the configuration of optimized solutions in these models.

2.1. Feature models

A Feature Model (FM) describes the common and variable features of a family of products along with constraints that rule which feature selections and deselections are valid in a PL [32]. A FM is represented in a tree structure where the root feature, known as the *concept*, represents the domain of the PL; the nodes represent *features* that can be included in a product configuration; and edges represent *feature relationships*, also known as Tree Constraints (TCs), that valid configurations must satisfy. Additional propositional formulas, called Cross-Tree Constraints (CTCs), can be used to create relations between non-connected features in the tree. In each CTC, features are encoded as propositions and the logical value of the formula must remain true.

MPLs interrelate several FMs that are enriched with non-functional properties modeled as attributes attached to features, and with inter-model constraints representing the existing relations among domain concerns. These relationships among features in two or more domains are called Cross-Model Constraints (CMCs) and can be represented using a Feature Solution Graph (FSG) [7]. In a FSG, features in different FMs can be related by defining propositional formulas and specific relationships such as *forces* (where the selection of a feature in a FM implies the selection of another feature in other FM) and *prohibits* (where the selection of one feature in a FM impedes the selection of another feature in other FM). As consequence, FSGs are used to support modularity and to interrelate independently modeled PLs allowing the representation

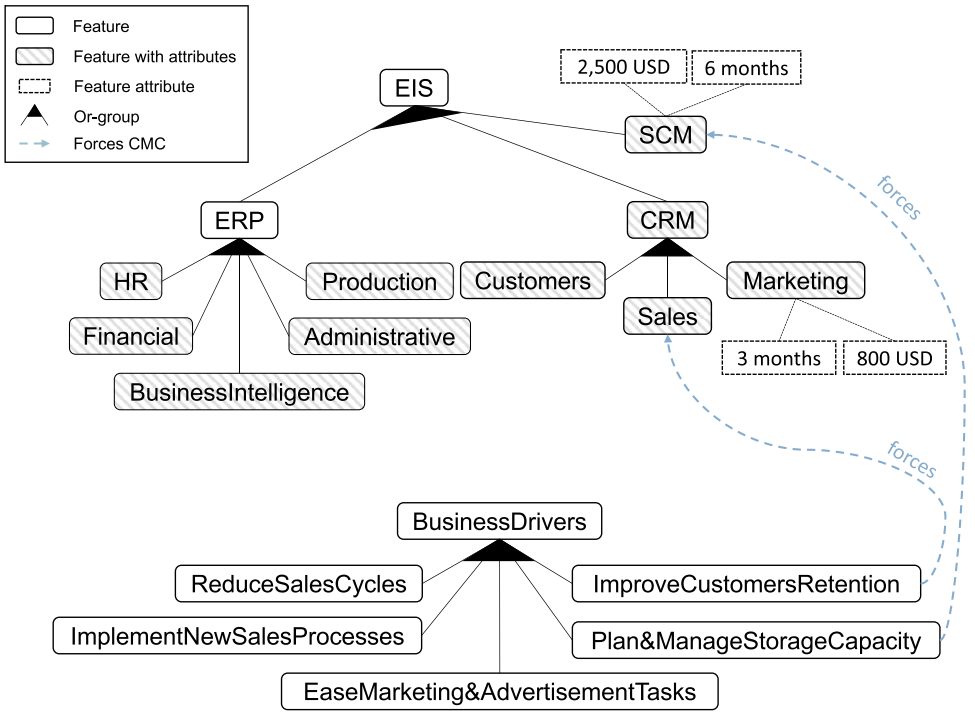

Fig. 1. FSG representation of the DecISIonAL MPL (adapted from [10]).

of MPLs. Fig. 1 illustrates a MPL interrelating two concerns for configuring investing decisions on Enterprise Information System (EIS).

A PL representing the variability of *EIS modules* is the first domain concern to be configured (*cp.* top FM in Fig. 1). The root has three features attached as an *or* feature group (at least one of the features should be selected in a product configuration): *Customer Relationship Management (CRM), Extended Feature Model (ERP)*, and *Supply Chain Management (SCM)*. Both *CRM* and *ERP* are related through an *or* constraint group to a set of features that represent EIS modules (e.g., *Marketing* and *Human Resources*). In this PL, *SCM* does not have any related sub-modules. Furthermore, this FM is enriched with *feature attributes* representing non-functional properties related to each EIS module. This type of models is known as Extended Feature Models(EFMs). In the figure we can observe that all leaf features have two types of feature attributes: one related to the *Costs* of configuring the corresponding module, and a second one related to the *Time* taken to achieve the module configuration goal. For example, the *SCM* module has a configuration cost of 2500 USD and a configuration time of 6 months, whilst the *CRM Marketing* module costs 800 USD and has a configuration time of 3 months.

The *Business Drivers* PL is considered as other concern that may support EIS modules selection (*cf.* bottom FM in Fig. 1). The root feature has a set of child features related through an *or* TC. *Improve Customers Retention, Implement New Sales Process, Ease Marketing and Advertisement Tasks, Reduce Sales Cycles*, and *Plan and Manage Storage Capacity* are some of the represented business drivers.

Both FMs (i.e., *EIS* and *Business Drivers*) are interrelated through CMCs representing how decisions in the first model affect the possible decisions in the second model. When managers decide on the business drivers for an EIS project, they must decide later on how to configure the available EIS modules that support the implementation of the chosen business drivers. These relationships among features in these two domains are represented through a *forces CMC*. For instance, selecting the *Improve Customers Retention* driver forces the implementation of the *CRM Sales* module; as well as the *Plan and Manage Storage Capacity* forces the selection of the *SCM* module.

2.2. Configuration process and optimization

In a *configuration process* one or more users select or deselect features of a FM or a FSG. A configuration is complete if there is a decision (i.e., selection or deselection) for all the features. A configuration is partial if there is no decision for at least one feature. In the FSG configuration, even if users define a complete configuration for one or more (but not all) models, that configuration is partial for the FSG. The configuration process usually implies taking a partial or complete configuration from the first model and determining the set of valid configurations for the remaining models. A software system can be used to automate a FSG configuration process, particularly when the FSG is too large in terms of features and constraints. For instance, an automated process may consider a set of previously selected business drivers and determine the

corresponding set of EIS modules to configure as shown in Fig. 1. In this example, the whole MPL has more than 1 billion valid configurations [10].

Additionally, users can define a set of *configuration constraints* orthogonal to the configuration process, and usually related to non-functional requirements. For instance, a user can set a budget limit over the cost of the expected *EIS* product (e.g., the cost should be less than 8000 USD), or she can also specify an optimization objective (e.g., the cost of the product should be minimized). A software system optimizes a partial configuration when feature selection or deselection is based on one or more optimization objectives that describe the users' non-functional requirements (e.g., minimize *Cost*, minimize *Time*). The optimization is *single-objective* if it considers only one optimization objective, or *multi-objective* if it considers more than one. Optimization is usually applied to EFMs. For instance, an optimization process may find which configuration exhibits the minimum configuration *Cost*, the minimum configuration *Time*, or which configurations exhibit the best *Cost-Time* relation for the FSG described in Fig. 1.

2.3. Configuration process as a constraint satisfaction problem

In order to support a configuration process and, for instance, perform a *configuration optimization*, many authors have proposed solutions based on Constraint Programming(CP).

A Constraint Satisfaction Problem (CSP) is a three-tuple $\langle V, D, \Phi \rangle$ problem defined in terms of a set of n variables V, the domains D in which variables take their values, and a set of m constraints Φ on variables domains. A solution for the problem comprises an assignment $\psi: V_i \rightarrow D_i$ that maps each variable i to a value in its domain, such that all constraints in Φ are satisfied.

A CSP solver is a software artifact that finds a solution for a given CSP specification. Solutions to the CSP are found in an iterative process, where the solver builds the solution set, typically using a search tree. In each iteration step, the solver gives a value to a given variable, so that the constraints are not violated. A CSP is said to be feasible, if it is possible to build a solution set assigning values to all variables such that all constraints are satisfied. Note that a CSP may have multiple solution sets (i.e., variables taking different values in their domain). Usually, CSP solvers can determine the total number of solutions and the corresponding values related to each solution. In addition, some solvers offer options to define decisions and goals to seek the best solution based on a given set of optimization criteria. Finding a solution under these circumstances normally includes heuristics that accelerate the search process. The use of heuristics still encompasses the *exploration* and *traversing* phases of the search –that is, building the search tree and traversing it, respectively. Heuristics optimize the search time to find a solution to a CSP by defining the order in which variables are instantiated in the search process, the order in which values are given to instantiated variables, or both.

There are many approaches to process FMs and configurations. These approaches usually encode the elements in the models and the user decisions as a CSP that can be processed by a solver. For instance, [3,5] encode an EFM with cardinalities into a CSP and use a solver to validate and perform some analyses on the FMs. Karataş et al. [13,14] extend that approach to encode complex CTCs related to an EFM, which involve feature-feature, feature-attribute, and attribute-attribute relations; and to introduce a larger set of analysis operations. Similarly, Salinesi et al. [35] encode an EFM into a CSP by considering arithmetic, boolean, and reified constraints. Ochoa and González-Rojas [23] encode EFMs within two different Java-based CSP solvers while considering stakeholder preferences, CTCs, and configuration constraints.

Most of the existing techniques to process FM configurations are based on the seminal work from Mannion [17] and Batory [2]. Such techniques define mappings to represent a FM as a set of expressions in propositional logic. In addition, they describe how to use Logic Truth Maintenance Systems (LTMS) and off-the-shelf SAT solvers [33] to process these expressions and detect conflicts in the FM and the configuration process.

3. Challenges of finding optimal configurations in MPLs

After encoding a MPL as a CSP, it may be processed by a CSP solver to find an optimal configuration based on stakeholders' objectives. The solver looks for a solution in the search space defined by the variables and domains in the problem. The resolution time depends on the size of the search space and on the search strategy. Solvers include predefined strategies to find a solution. However, programmers may provide custom heuristics based on domain experts' knowledge to speed up the process. This section reviews the state-of-the-art in CP-based approaches that aim at finding configurations for EFMs. We only consider EFMs given that, to the best of our knowledge, there are no solutions targeting configuration of extended MPLs. We consider two main aspects: **(1)** performance when configuring PLs; and **(2)** definition of heuristics to improve the PL configuration process.

3.1. Performance of CP-based approaches

Current CP-based approaches support the configuration of products for a PL by considering single-objective optimization [8,9,20,24,41], and multi-objective optimization [16,29,35]. These approaches have evaluated performance and/or scalability concerns, by considering the number of configuration constraints and the size of the related FM. Table 1 summarizes these evaluations.

Table 1
Performance tests of CP-based approaches.

Article	FM	F	CC	ET
Gamez et al. [8]	Booking	200	5 operations	> 10 s
White et al. [41]	Generated	2000	3 steps	> 12 s
García-Galán et al. [9]	AWS EC2	35	5	< 1 h 30 min
Mazo et al. [20]	Generated	5000	Undefined	1.8 s
White et al. [41]	Generated	5000	1	3 min
Ochoa et al. [24]	IT Investment	140	1	296 ms-18 min
Leite et al. [16]	Industrial	46	1	10 min

FM: feature model, F: features, CC: configuration constraints, ET: execution time

These PLs group between 35 and 5000 features, with the number of configuration constraints varying from 1 to 5. It is important to notice that the nature and semantics of the considered constraints change significantly from one approach to another (e.g., optimization objects against budget limit), then a direct and general comparison cannot be claimed. The FMs with a larger number of features are automatically generated by a software tool. Some approaches like the ones presented by Gamez et al. [8] and White et al. [41] analyze other concerns considered during the PL configuration, such as the number of internal operations [8] and the number of steps in a staged-configuration [41]. The execution time to find a configuration varies depending on the considered constraints and the size of the PL. For all cases, solutions are obtained in at least 1.8 s [20] and at most 1 h 30 min [9]. However, if the complexity and size of the FM increase, or the number of considered configuration constraints is higher, these values can be drastically affected. In fact, for the FM based on the AWS EC2 service, which has a total of 35 features and 5 configuration constraints, the execution time was around 1 h 30 min for generating 4 configurations [9].

3.2. Heuristic definition to solve product configuration problems

CP-based heuristics have been proposed to speed up the analysis or configuration of PL models. For instance, Mendonça et al. [22] propose two heuristics for encoding FMs to Binary Decision Diagram(BDDs), which are acyclic graphs with internal nodes representing variables and external nodes representing two constant functions, 0 and 1. This representation problem is known as a NP-hard problem. The main objective of these heuristics consists on reducing the size of BDDs by setting a *good* order to improve configuration performance. To achieve this goal, heuristics are based on sorting, clustering, and FM pre-order traversals. Scalability results for FMs with up to 2000 features show an execution time of 3.7 s and FM with up to 2500 features show a time of 9.9 s [22]. This solution is only applicable to FMs without attributes.

Other heuristics have been proposed for CSP, where solvers can be used to process EFMs. Sanchez et al. [36] aim at selecting the most suitable product of a PL based on a non-functional property. A *best-first search* algorithm is proposed to minimize a given set of attributes based on linear weighted functions. This algorithm is enhanced with strategies and heuristics to improve the configuration performance. The configuration search is done over a *state-space graph*, where some heuristics are defined to guide the process over *relaxed models*. Nodes in the directed-graph can be visited based on a Best-First search Star (BF*) or a Greedy Best-First Search (GBFS) strategy. The GBFS strategy derives results for FMs with less than 150 features in less than 0.1*s*. However, these heuristics do not consider configuration processes involving MPLs.

Mazo et al. [19] present six heuristics aimed at improving stakeholders' satisfaction and supporting scalability during large PL configuration. The PL is encoded in a CSP by using both logic and arithmetic operations. The proposed configurator finds the best configuration order to minimize the number of steps to obtain a product. It also needs to minimize the computation time taken to propagate decisions. Thus, the CP heuristics contemplate the selection of: **(1)** variables with the smallest domain; **(2)** most constrained variables; **(3)** variables appearing in most products; **(4)** automatic completion when no choice is provided; **(4)** variables related to the last configured variables; and **(6)** variables that split the problem space. Nonetheless, most heuristics except for the third one, are applicable to a generic CSP and do not consider the nature of the PL configuration problem, nor its particular constraints (e.g., TCs, CTCs, CMCs).

Finally, Pesant et al. [31] propose a set of algorithms that support counting-based search in CP. This approach does not directly address the PL configuration problem; however, a set of global heuristics are presented to tackle any generic problem in a fairly robust way. Contrary to our study, they rely on information provided at the level of CP constraints (e.g.*all different, global cardinality, regular*, and *knapsack* constraints), instead of only considering information attached to variable-value pairs. They validate their solution with eight known problems, comparing the obtained results against other generic heuristics. In most cases their algorithms outperform existing solutions. Although their findings are not included in this research, we plan to check their proposed heuristics and test their performance in MPL configuration scenarios.

3.3. Shortcomings for configuring multi product lines

Even though some heuristics have been proposed to deal with the exponential behavior of CP-based PL configuration, there are some shortcomings that must be addressed by current and future work. In this section we highlight two gaps that are addressed in the following sections.

First, there are two configuration scenarios that must be further explored: EFMs and MPLs. In the former, feature attributes do not only play an informative role, they are also considered in configuration constraints (e.g., optimization objectives) to derive the most suitable configurations based on particular needs. These attributes and their related configuration constraints must have a particular treatment in the CSP search strategies. Thus, a clever search can be performed by the selected CSP solver. In the latter, configuration scenarios that consider MPLs instead of single FMs must manage CMCs, which are another type of constraint that increases the complexity of the problem. Now, the increase in not only CTCs but CMCs may affect the scalability and performance of the solution.

Second, most state-of-the-art heuristics for CP-based PL configuration, do not consider the nature of the problem and its corresponding representation (e.g., FM). Generic heuristics may improve the resolution time of a MPL configuration, however search strategies must benefit from the particularities of the represented problem; these strategies must consider TCs, CTCs, CMCs, features, and feature attributes nature. Tests and systematic bench-markings should be performed to confirm this hypothesis.

4. Customized search heuristics

CP search strategies can affect the performance of a CSP resolution. Search strategies may consider: **(1)** *variable selection*, which takes place during the labeling process; **(2)** *value selection* of a variable in its given domain; and/or **(3)** the selection of the *operation to execute* when the desired variable and its corresponding value are procured [1].

To tackle **RQ1** (i.e., to improve solving performance) we focus on the definition of three different search heuristics for selecting a variable that follows the *fail-first principle*, namely (CID) (a generic CSP heuristic), (Cross*C), and (2OC) heuristics. The latter two respond to the nature of MPLs and the configuration problem (e.g., the use of CTC and CMC in the problems at hand).

Hereafter we present our proposal. First, we describe the Weighted Degree (WD) heuristic, which is the default search strategy of the Choco CSP solver used as part of the evaluation of our approach. Then, we present the proposed heuristics and their corresponding algorithms[1]

4.1. Weighted degree heuristic

This heuristic is taken from the default search strategy of the Choco solver [34], which is based on the work done by Boussemart et al. [6]. The heuristic selects the variable with the smallest value of the ratio defined in Eq. (1), where $|D_i|$ refers to the size of the ith variable domain, and w_i is the *weighted degree* of the same variable. w_i adds the weight of constraints where the ith variable is present and at least two more variables are not instantiated[2] [34].

$$\frac{|D_i|}{w_i} \tag{1}$$

4.2. Constraint instantiation degree heuristic

The first proposed heuristic is a generic heuristic that is independent of the MPL nature. The rationale is that we should first consider variables involved in constraints where there is a higher number of other instantiated variables; then, there is a higher chance to instantiate the remaining variables thanks to constraint propagation. The CID heuristic selects the variable with the highest value of the ratio defined in Eq. (2). This heuristic considers, for each variable, the m constraints where it is involved. $|V_j|$ is the total number of variables involved in the jth constraint, and $|V'_j|$ is the number of instantiated variables in the same constraint. In this way, the variable with the highest ratio of instantiated variables in its related constraints is selected.

$$\frac{\sum_{j=1}^{m} |V'_j|}{\sum_{j=1}^{m} |V_j|} \tag{2}$$

Algorithm 1 presents the CID heuristic. First, a *globalRatio* is initialized with a value of −1 (Line 1), and an iteration over the set of *n variables* representing features in a FSG is performed (Line 2). For each uninstantiated variable, the algorithm considers the m constraints where the variable is present (i.e., *localConstraints*) and it performs a loop to count the number of other variables involved in those constraints (i.e., *localVariables*), as well as the cardinality of the instantiated variables subset (i.e., *instVariables*) (Lines 4–8). Once the inner loop ends, the algorithm calculates the ratio given in Eq. (2) (Lines 9–14). If the *localRatio* is bigger than the previous *globalRatio*, we consider this new ratio and the ith variable related to it as the new best candidate for the CSP labeling. The process ends after all variables have been visited.

[1] Source code available at https://github.com/CoCoResearch/PLConfiguration.

[2] Implementation available at https://goo.gl/eEkqZB.

Algorithm 1 Select variable with the CID heuristic.

Require: Initialized set of feature *variables*
Ensure: Return the selected *variable* or null if no variable is found
1: Initialize *globalRatio* and *variable*
2: **for** i = 0 **to** n **do**
3: **if** *variables*[i] is uninstantiated **then**
4: Initialize *localVariables*, *instVariables*, *localConstraints*, and *localRatio*
5: **for** j = 0 **to** m **do**
6: *localVariables* += variables involved in *localConstraints*[j]
7: *instVariables* += instantiated variables involved in *localConstraints*[j]
8: **end for**
9: **if** *localVariables* > 0 **then**
10: *localRatio* = *instVariables* / *localVariables*
11: **if** *localRatio* > *globalRatio* **then**
12: *variable* = *variables*[i]
13: **end if**
14: **end if**
15: **end if**
16: **end for**
17: **return** *variable*

4.3. Cross-tree and model constraints heuristic

Our second proposed heuristic takes advantage of the FSG's structure. In this case, the selected variable corresponds to the first variable involved in a *mandatory* TC, *requires* or *excludes* CTC, or *forces* or *prohibits* CMC, where the other variable in the constraint is already instantiated. Notice that these constraints are binary, only two variables compose them, this means that if one of the variables is instantiated then there is a high chance of instantiating the other one by means of constraint propagation. This behavior is subject to the type of considered constraint.

Algorithm 2 presents the Cross*C heuristic. It starts by iterating over the set of *n variables* representing features in a FSG (Line 2). For each uninstantiated variable, the algorithm iterates over the set of *m* constraints (i.e., *localConstraints*) where the *i*th variable is involved (Lines 4–5). If the *j*th constraint has one of the previously mentioned types, and the other variable present in the constraint is already instantiated, then the *i*th variable is defined as the preferred CSP variable to be labeled, and the algorithm stops (Lines 6–8).

4.4. Optimization in or-constraints heuristic

The last proposed heuristic also considers the structure of FSGs. This heuristic is applicable to feature attribute optimization problems. The rationale for it is that variables are selected based on the type of optimization: if a maximization is expected then variables with the largest upper domains are selected, otherwise the ones related to the smallest domains are picked. Thus, the variable to be labeled corresponds to the first variable V_j involved in the *j*th *or* TC, which is also associated to the optimized feature attribute *A*. If there exist a *minimization* function over *A*, the selected variable must have the

Algorithm 2 Select variable with the Cross*C heuristic.

Require: Initialized set of feature *variables*
Ensure: Return the selected *variable* or null if no variable is found
1: Initialize *variable*
2: **for** i = 0 **to** n AND *variable* == null **do**
3: **if** *variables*[i] is uninstantiated **then**
4: Initialize *localConstraints*
5: **for** j = 0 **to** m **do**
6: **if** *localConstraint* type == *ManadatoryTC* || *RequiresCTC* || *ExcludesCTC* || *ForcesCMC* || *ProhibitsCMC*AND sibling variable is instantiated **then**
7: *variable* = *variables*[i]
8: **end if**
9: **end for**
10: **end if**
11: **end for**
12: **return** *variable*

lowest upper bound of the siblings considered in the or-group (Eq. (3)). If there exists a *maximization* function over A, the selected variable V_j must have the *highest* upper bound of the siblings (Eq. (4)). Notice that we consider the upper bound of the variable because in the studied MPL configuration scenarios a feature attribute is encoded as a variable, whose lower bound corresponds to 0 (in the case the associated feature is not selected) and its upper bound to u (representing its default value). In optimized encodings a variable will be defined as a constant, thus both lower and upper bounds will point to the same value u and the selection of one of the two bounds will be indifferent for the searching strategy.

$$\sup_{A}\{V_j\} \tag{3}$$

$$\max_{A}\{V_j\} \tag{4}$$

Algorithm 3 presents the 2OC heuristic implementation. As with the other heuristics, the algorithm starts by iterating over the set of *n variables* representing features in a FSG (Line 2). For each uninstantiated variable, the algorithm iterates over the set of *m* constraints (i.e., *localConstraints*) where the *i*th variable is involved (Line 4–5). If the *j*th constraint is an *or* TC, the variables related included in the constraint are assigned to the *orVariables* array (Lines 6–8). For the same variable, if there are other constraints that consider feature attributes related to a given *Attribute* type, then the *hasAttributes* flag is marked with a *true* value (Lines 9–11). Once the loop over the *i*th variable constraints ends, the algorithm verifies if *orVariables* is not null and the *hasAttributes* variable is true (Line 14). An iteration is performed over the *orVariables* array, and the element with the smallest (in the case of a *minimization* function) or highest upper bound (in the case of a *maximization* function) is set as the preferred variable to be labeled in the CSP (Line 15); the algorithm immediately stops.

5. Assessing the performance for configuring multi product lines

To answer **RQ2**, we evaluate the performance of each of the three proposed heuristics (see Section 4) on different generated FSGs and also on a subset of FMs taken from the current state-of-the-art. In the case of generated FSGs, we produce multiple scenarios scaling up to MPLs containing 6400 features and 960 constraints among such features.

The three proposed heuristics were implemented into CoCo's program synthesis [23,24], which automates the configuration process by generating a specific CSP based on a given FSG. Heuristics, as well as PLs and configuration constraints specifications, are translated to Choco [34], a Java-based CSP solver. The motivation behind using Choco responds to: **(1)** the maturity of the project, which has been developed and maintained for around seven years and has been awarded in multiple categories of the CP MinZinc competition; **(2)** the underlying programming language, Java, which is also used to build CoCo's program synthesis; and **(3)** the strong community and documentation surrounding the library. CoCo's CP-based encoding corresponds to the algorithmic mapping presented by Salinesi et al. [35]. Therefore, with the definition of new heuristics a user can specify its preferred search strategy in a DSL, and it will be automatically incorporated in the CP implementation.

Algorithm 3 Select variable with the 2OC heuristic.

Require: Initialized set of feature *variables*
Ensure: Return the selected *variable* or null if no variable is found

```
1: Initialize variable, orVariables, and hasAttributes = false
2: for i = 0 to n AND variable == null do
3:     if variables[i] is uninstantiated then
4:         Initialize localConstraints
5:         for j = 0 to m AND orVariables == null AND !hasAttributes do
6:             if localConstraint type == OrTC then
7:                 orVariables = variables involved in localConstraints[j]
8:             else
9:                 if Involved variables type == Attribute then
10:                    hasAttributes = true
11:                end if
12:            end if
13:        end for
14:        if orVariables != null AND hasAttributes then
15:            variable = get variable with (smallest|highest) upper bound from orVariables
16:        end if
17:    end if
18: end for
19: return variable
```

Table 2
FSG configuration scenarios for Experiment 1.

Features per FM	No. of features per FSG	No. of CTCs	No. of CMCs	No. of constraints
40	{40, 80, 120, 160, 200}	{4, 8, 12, 16, 20}	{2, 4, 6, 8, 10}	{6, 12, 18, 24, 30}
80	{80, 160, 240, 320, 400}	{8, 16, 24, 32, 40}	{4, 8, 12, 16, 20}	{12, 24, 36, 48, 60}
160	{160, 320, 480, 640, 800}	{16, 32, 48, 64, 80}	{8, 16, 24, 32, 40}	{24, 48, 72, 96, 120}
320	{320, 640, 960, 1280, 1600}	{32, 64, 96, 128, 160}	{16, 32, 48, 64, 80}	{48, 96, 144, 192, 240}
640	{640, 1280, 1920, 2560, 3200}	{64, 128, 192, 256, 320}	{32, 64, 96, 128, 160}	{96, 192, 288, 384, 480}
1280	{1280, 2560, 3840, 5120, 6400}	{128, 256, 384, 512, 640}	{64, 128, 192, 256, 320}	{192, 384, 576, 768, 960}

5.1. Subjects, design, and variables

We design three experiments that compare the time to find optimal MPL configurations in a CSP by using a default heuristic (i.e., WD), against the time obtained with the three proposed heuristics (i.e., CID, Cross*C, and 2OC).[3]

The first two experiments use EFMs generated by the BeTTy[4] model generator, which implements the algorithm proposed by Thüm et al. [40]. The FSGs are built using a M2M transformation, which considers different densities of CTCs and CMCs to relate features in a single and multiple FMs, respectively. The third experiment considers existing FMs used in the literature. Only valid FMs and FSGs were considered as subjects in the three experiments –that is, models for which a configuration can be found. To guarantee validity, all models were previously checked with the FaMa Framework tool [4]. In addition, only binary CTCs and CMCs are considered in the study.

Features in all studied FMs and FSGs are enriched with two feature attributes in the domain [0, 100]. In all test cases, each of the two feature attributes is related to an arbitrary non-functional property (in real scenarios these non-functional properties could represent concerns such as costs, time, etc.). We refer to these non-functional properties as *NFP 1* and *NFP 2*. Accordingly, two optimization criteria are followed in all configuration scenarios: **(1)** minimize *NFP 1*; and **(2)** maximize *NFP 2*.

Furthermore, for the purpose of these experiments we restrict the number of CSP solutions to 10 in each scenario. This decision answers to the human involvement in a decision process, where overwhelming decision-makers with too many solutions may be counterproductive. Each configuration scenario is repeated three times and the *harmonic mean* is used to obtain the average resolution time to find 10 solutions for each configuration, avoiding the impact of outliers. To check the overall behavior of the studied heuristics we compute the *Spearman rank correlation* between resolution time and number of features in the FSG (*cf.* Experiments 1 and 3), and the same type of correlation between the resolution time and the CTC (*cf.* Experiment 3) or CMC density (*cf.* Experiment 2). This correlation is a non-parametric measure that does not have strong assumptions such as *linearity*, which is not met in our experiments. On the contrary, it assumes an ordinal, interval, or ratio level of measurement; and the existence of a *monotonic* relation among the correlated variables. When these assumptions hold we compute the corresponding value. Lastly, when running the proposed heuristics, we follow a process for both CID and Cross*C heuristics in which, if solutions are not found with the given heuristics, each will revert to use the WD heuristic. In the case of 2OC heuristic, if no solution is found it will revert first to use Cross*C heuristic, and then to use WD heuristic.

All the experiments were executed on a 64bit virtual machine running Windows 7 Enterprise with a 2.59GHz Intel(R) Xeon(R) processor and 8GB RAM. The experiments run on Eclipse Oxigen 2 (4.7.0) using Java 1.8 with a standard memory allocation (-Xms256m, -Xmx1024m).

Scope of Experiment 1: Varying number of features per FSG. This experiment uses generated FSGs where the number N of involved features per FM, and the number M of involved FMs per FSG vary (see Table 2).

In this context, $N \in \{40 \times 2^n \mid n \in [0, 5]\}$, and $M \in [1, 5]$. Each feature in the model has a branching factor of 10, and its descendant features are defined with a 25% probability of having either a *mandatory, or, optional*, or *alternative* TC. Additionally, each FM contains a CTC density of $0.1N$, and each FSG a density of $0.05(M*N)$. As a result of combining variables M and N we obtain a set of 30 experimental scenarios. Lastly, we execute three times each scenario with a different FSG to obtain reliable results, generating 90 different FSGs for the experiment. For each FSG we compare the CSP resolution time when using each of the four mentioned search heuristics, including the default heuristic.

Scope of Experiment 2: Varying CMC density per FSG. This experiment also uses generated FSGs where the CMC density and the number M of involved FMs per FSG vary. However, the number of features N per FM remains constant $N = 40$. Moreover, $M \in [2, 5]$ and the CMC density is $\rho = \gamma(M * N)$ for $\gamma \in \{0.05, 0.1, 0.15, 0.2\}$ (see Table 3).

As in the first experiment, each feature in the FSG has a branching factor of 10, and descendant features have a 25% probability of having a *mandatory, or, optional*, or *alternative* TC. Each FM also contains a CTC density of $0.1N$. As a result of combining variables M and ρ we derive a set of 20 experimental scenarios. Three executions with different FSGs are also considered per scenario to obtain reliable results, generating 60 different FSGs for the experiment. For each FSG we compare the CSP resolution time when using each of the four studied search heuristics.

[3] Models and results available at https://github.com/CoCoResearch/PLConfiguration.
[4] Tool available at http://www.isa.us.es/betty/betty-online.

Table 3
FSG configuration scenarios for Experiment 2.

CMC density	*0.05*	*0.1*	*0.15*	*0.2*
FMs per FSG	[2,5]	[2,5]	[2,5]	[2,5]
No. of features per FSG	{80, 120, 160, 200}	{80, 120, 160, 200}	{80, 120, 160, 200}	{80, 120, 160, 200}
No. of CTCs	{8, 12, 16, 20}	{8, 12, 16, 20}	{8, 12, 16, 20}	{8, 12, 16, 20}
No. of CMCs	{4, 6, 8, 10}	{8, 12, 16, 20}	{12, 18, 24, 30}	{16, 24, 32, 40}
No. of constraints	{12, 18, 24, 30}	{16, 24, 32, 40}	{20, 30, 40, 50}	{24, 36, 48, 60}

Table 4
FMs characterization for Experiment 3.

FM	#Features	#CTC	CTC Density	#Products
DecISIonAL [10]	143	88	~0.615	More than one billion
Drupal [30]	21	9	~0.429	442
E-Shop [15,38]	290	22	~0.076	More than one billion
Web Portal [21]	43	6	~0.14	2,120,800

Scope of Experiment 3: Application to existing case studies. We consider a set of FMs found in the state-of-the-art. Despite the absence of MPLs in the literature, this experiment is done in order to test our results against case studies that are usually considered in the experimentation phase of other performance-concerned approaches [11,12,28,37–39].

The selected FMs correspond to diverse PLs. First, the Investing Decisions on Enterprise Information Systems (DecISIonAL)[5] is a FM employed to evaluate risks of EIS implementation projects [10] (an adapted fragment of the model is presented in Section 2). [10]. Second, the Electronic Shopping (E-Shop)[6] FM captures variability in e-commerce systems [15,38]. The Web Portal[7] FM represents the variability in the implementation of web systems [21]. Finally, the Drupal[8] FM represents the variability of a content management system [30]. Studied FMs with their corresponding number of features, CTCs, CTC density, and number of products are presented in Table 4.

As aforementioned, all these FMs are extended with feature attributes related to the generic non-functional properties *NFP 1* and *NFP 2*. The same two optimization criteria are applied to the configuration problem.

5.2. Analysis of results

5.2.1. Results for Experiment 1

Fig. 2 presents the resolution time for solving the experimental configuration scenarios while varying the number of features and FMs per FSG. The total number of features in the complete FSG are shown in the *x*-axis, while the *y*-axis shows the resolution time in logarithmic scale. Note that the number of features is proportional to the number of CTCs and CMCs according to the corresponding densities.

As it can be seen, there is an initial exponential growth of the CSP resolution time as the number of features in the model increases, however after configuring FSGs with over 1000 features the slopes of the series decrease. Although this trend is observable for all four heuristics, in almost all cases the proposed heuristics outperform the WD heuristic. We compute the average Spearman rank correlation between the resolution time and number of features, in order to discover a relation between both variables. This computation results in a correlation $\varrho = 0.988$ with a standard deviation $\sigma = 0.004$, which denote a strong positive association between the CSP resolution time and the number of features of the model.

The graphs in Fig. 3 zoom in the resolution time behavior of each heuristic when facing FSGs with one to five FMs, and with different number of features. The *x*-axis presents the number of features per FM in the FSG, and the *y*-axis represents the resolution time in logarithmic scale. In all the graphs, we observe a tendency to have a higher resolution time as models grow in features (and intrinsically in FMs). Note that as the number of FMs increases, the resolution time between the proposed heuristics against the default heuristic is sparser. For instance, the minimum and maximum configuration time taken by the three proposed heuristics oscillate between 6.56 ms (for FSGs with 40 features and 1 FM in the case of CID) and 13.11 s (for FSGs with 1280 features and 5 FMs in the case of 2OC); whilst in the case of the default heuristic the boundary times correspond to 14.61 ms (for FSGs with 40 features and 1 FM) and 209.51 s (for FSGs with 1280 features and 5 FMs). Moreover, CID and Cross*C tend to outperform 2OC in almost all configurations. Regarding the behavior of the heuristics when configuring FSGs with 4 FMs, there is a higher difference between the resolution time of the CID heuristic with that of the Cross*C and 2OC heuristics.

The erratic behavior of the CID heuristic leads us to hypothesize the existence of confounding variables in the generated models that favor this heuristic over the other. Factors like the type of TCs, CTCs, and CMCs constraints in the model, may

[5] FM found at http://www.splot-research.org/models/model_20170405_957299568.xml.
[6] FM found at http://www.splot-research.org/models/REAL-FM-4.xml.
[7] FM found at http://www.splot-research.org/models/REAL-FM-3.xml.
[8] FM found at http://exemplar.us.es/demo/SanchezJSS2016.

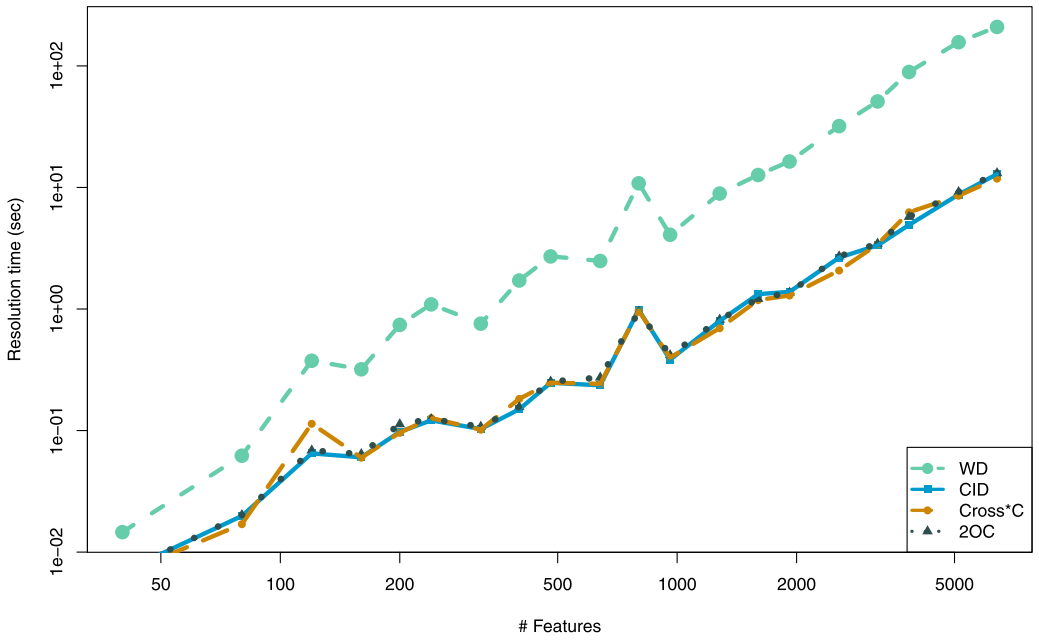

Fig. 2. Logarithmic view of the resolution time against number of features per FSG.

influence the results. Experimentation with other generated models may lead to a similar behavior as in the case of 1, 2, or 3 FMs.

5.2.2. Results for Experiment 2

Fig. 4 shows the results obtained when configuring optimal products in generated FSGs while varying the CMC density and the number of FMs. Each of the four graphs presents performance results for the four search heuristics while configuring FSGs with $M \in [2, 5]$. The CMC density ρ is shown in the x-axis, whilst the y-axis shows the obtained resolution time in logarithmic scale.

Contrary to the first experiment results, there is a decrease in terms of resolution time as ρ increments. Highest values are obtained when $\rho = 0.05$, however it is not the case when configuring FSGs with $M = 2$; in this case the maximum values are obtained when $\rho = 0.1$.

With regards to the independent performance behavior of the search heuristics, the WD and CID heuristics tend to outperform Cross*C and 2OC heuristics; and on the whole, the CID heuristic tends to beat all the alternatives. This affirmation is particularly strong for configuration scenarios where $\rho \in \{0.05, 0.1\}$; but when $\rho \in \{0.15, 0.2\}$ all heuristics tend to align towards a similar resolution time value. After reaching $\rho = 0.15$ (in some scenarios $\rho = 0.1$) the slope tends to 0. Nonetheless, there is a slight difference in terms of resolution time when configuring FSGs with $M = 2$. In this case, the WD heuristic is beaten by the other heuristics.

Lastly, on the way to elucidate a relation between FSG configuration time and CMC density, we compute the average Spearman rank correlation between these two variables in all configuration scenarios except when $M = 2$. In the latter there is no monotonic trend between the two variables, thus the use of the corresponding relation would yield to wrong conclusions. For the remaining scenarios, $\varrho = -0.916$ with a standard deviation $\sigma = 0.102$. We obtain a strong negative correlation between the two variables for scenarios where $M > 2$.

5.2.3. Results for Experiment 3

Fig. 5 presents the resolution time obtained with the four studied heuristics, when configuring each of the four state-of-the-art FMs.

As it can be seen, the E-Shop FM is the one that takes the longest time to configure 10 optimal solutions; whilst Drupal, Web Portal, and DecISIonAL tend to get the requested solutions in the order of milliseconds. In the case of Drupal and DecISIonAL, the 2OC heuristic outperforms the other heuristics. WD heuristic has a slightly better resolution time compared to the proposed heuristics when configuring the Web Portal FM. Lastly, the CID heuristic outperforms remaining ones during the E-Shop model configuration, and WD heuristic shows the worst performance behavior.

When reviewing the existent relations between resolution time and both number of features and CTC density, we get the following results: **(1)** the average Spearman rank correlation between resolution time and number of features is $\varrho = 0.6$ and there is a standard deviation $\sigma = 0.231$. This denotes a moderate positive association between the two variables. We see that the type of dependency obtained in this experiment endorses the behavior observed in the first experiment

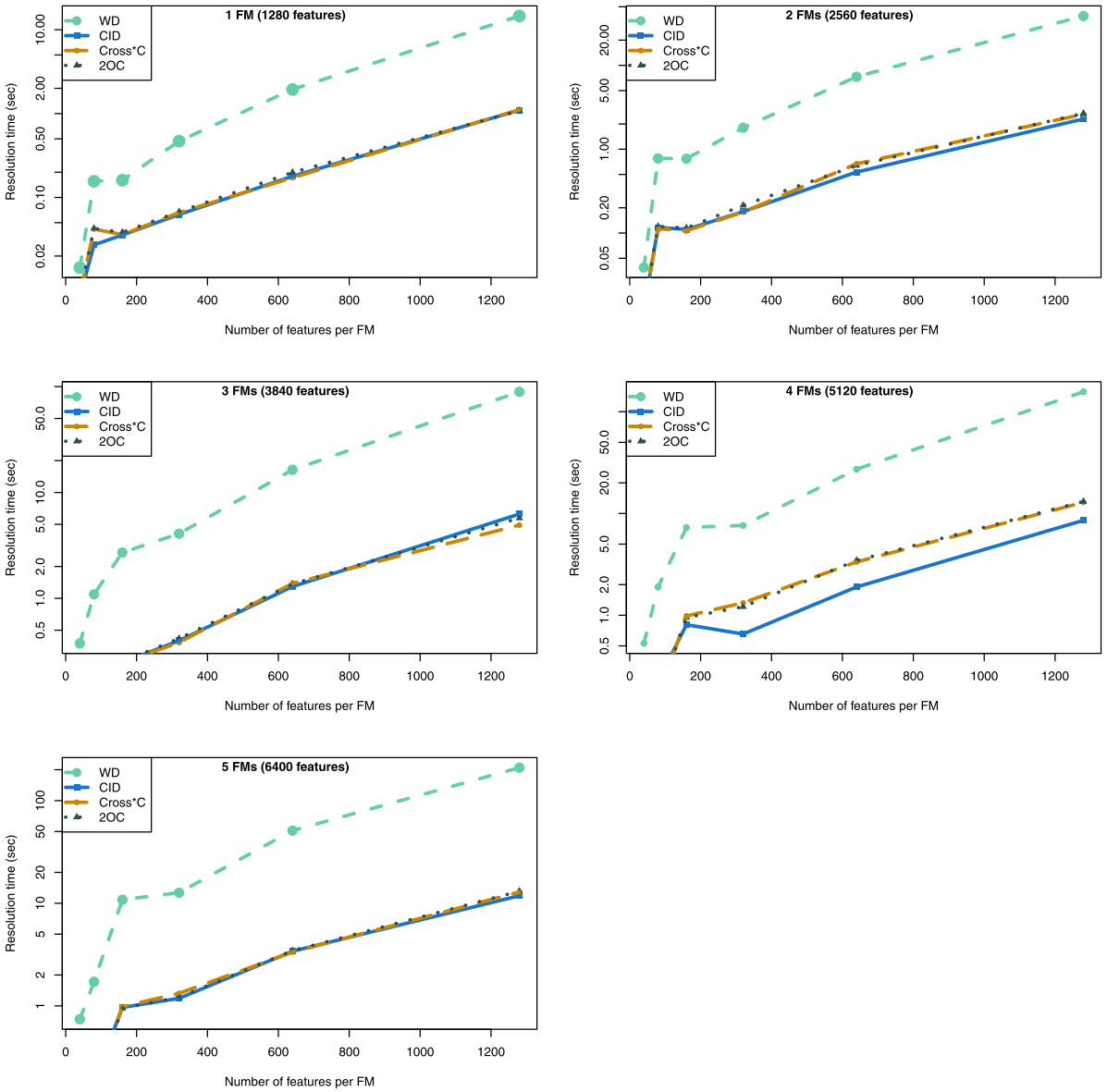

Fig. 3. Evaluation results per number of involved FMs and features per FM.

(i.e., $\varrho = 0.988$); and **(2)** the average Spearman rank correlation between resolution time and CTC density is $\varrho = -0.45$ and there is a standard deviation $\sigma = 0.225$; which results in a moderate negative association between the variables. These results are not completely aligned to the ones obtained in the second experiment (i.e., $\varrho = -0.916$), where a strong negative relation was obtained.

5.3. Discussion

Figs. 2 and 3 evidence that heuristics that do not consider the nature of the MPL configuration problem, such as the default heuristic (i.e., WD) provided by the Choco solver, are unfit to efficiently solve this challenge. This is due to the generality of the heuristic with respect to the types of used models. As a consequence, as we initially posit, specialized heuristics to the PL domain are required in order to improve the performance of finding a solution.

Considering the tendency of the resolution time observed in Fig. 3 for the different scenarios, Cross*C provides the best performance improvement (between 1.16x and 4.66x with respect to the default strategy) in the average case. Therefore, the heuristic to use for average uses is the Cross*C heuristic. The CID and 2OC heuristics have oscillating behavior. As a

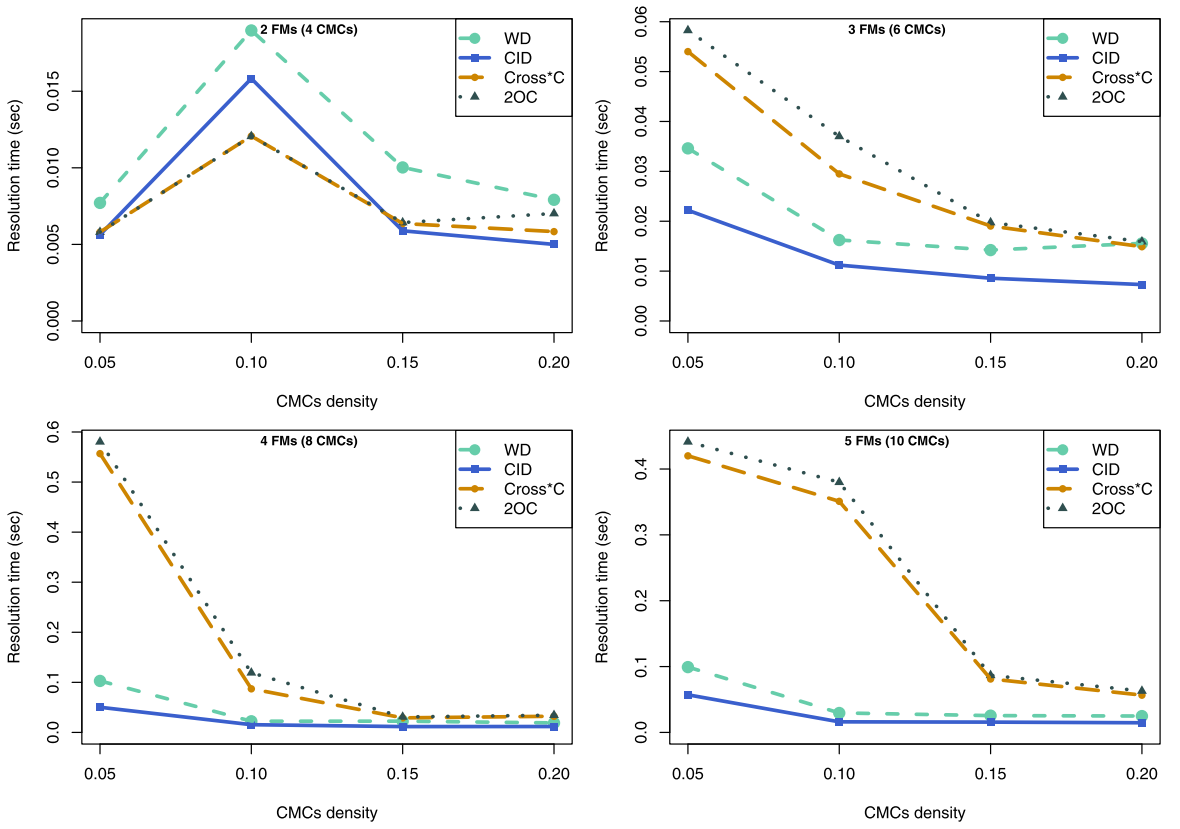

Fig. 4. Evaluation results per number of involved FMs and CMC density per FM.

consequence, the choice of which heuristic to use is left to the domain experts, based on the size of the models and the types of constraints defined across them.

Certain types of MPL configurations could benefit from the use of a particular heuristic. Note that for larger scenarios (with over 200 features) CID outperforms the other heuristics across all configuration sizes, thus it should be use in large configuration scenarios. Additionally, for configurations consisting of less than $\sim$40 features per FM, and a CMC density in $\sim$[0,0.15), CID and WD heuristics are preferable to the other heuristics. For a configuration consisting of 40 features per FM and a CMC density in $\sim$[0.15,0.2], CID is preferable. However, for the specific configuration consisting of four FMs, each containing 40 features, 2OC is more appropriate. The decision of which heuristic to use for a configuration problem given a specific model is still left to the domain expert. Further evaluation and improvements to automate the selection of a heuristic given a generic model (within a domain) is left as part of our future work. For all other configurations, there is not a significant difference in resolution time between the heuristics, and any of them could be used.

5.4. Threats to validity

Subjects selection, understood as the chosen characteristics of FMs and FSGs generation, may affect the experiments' resolution time. The structure among MPLs could be similar, defining a behavior that could bias the obtained results. In order to overcome this threat that affects both internal (i.e., variables dependency) and external validity (i.e., results generalization), TCs definition were defined with equal probabilities, and the number of features and FMs in FSGs varied. However, CTCs and CMCs, while maintaining the same proportion in all configurations, are defined randomly. Such definition may ignore tree branches and FMs (respectively), which may lead to different results in other configuration scenarios.

An additional threat to the results has to do with the generated nature of FSGs, which may have a different behavior from real-world MPL configurations, and therefore affect the results. Further evaluations are required involving both industrial and academic case studies to allow the generalization and tuning of the results.

6. Conclusion and future work

The definition and incorporation of customized heuristics to a CP-based program synthesis improves the performance and scalability of product configuration in large scale MPLs. The implemented heuristics are used according to the nature

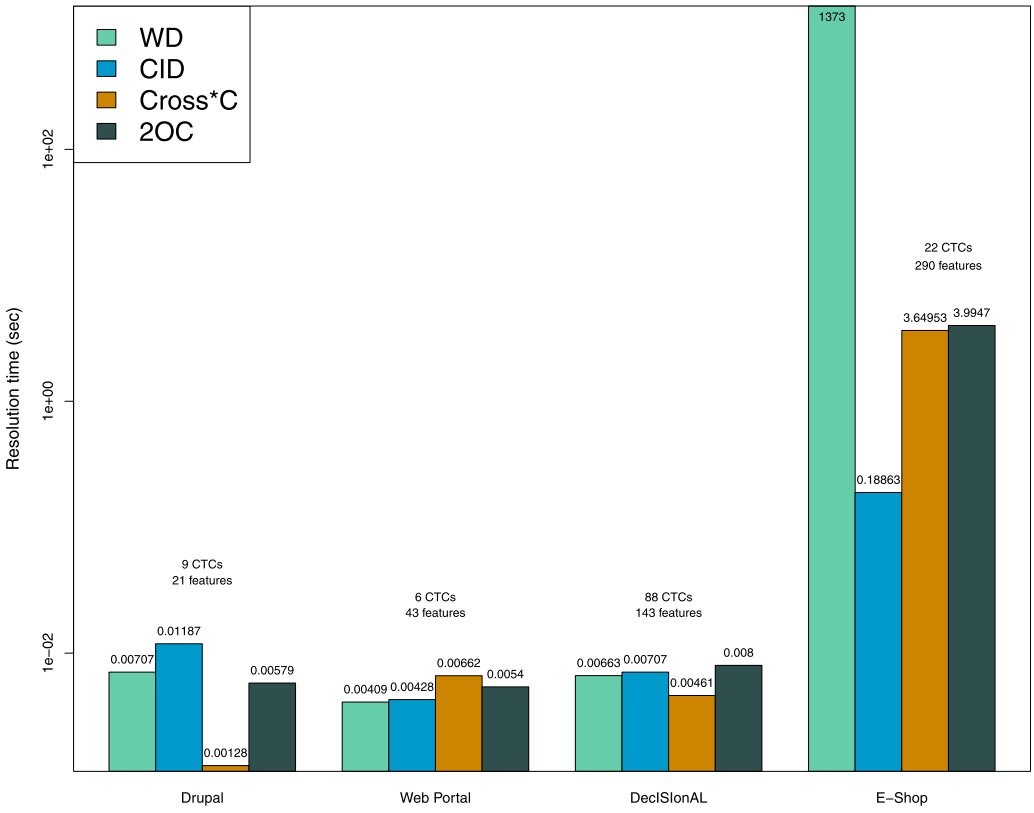

Fig. 5. Configuration time per state-of-the-art FM.

of the MPL configuration problem. Therefore, the most efficient heuristic for a product configuration can now be selected based on the number of features in a model, the number of FMs per FSG, the density of MPL particular constraints (i.e., CTC and CMC), and the type of configuration constraints.

The implemented heuristics achieved a resolution time improvement that oscillates between 14% and 40% when configuring a small MPL (for FSGs with 20 to 40 features, respectively). An improvement between 33% and 80% was achieved when scaling the number of features and constraints in the FSG (for FSGs with over 200 features). The results show a better performance for large scale FSGs. Note that the heuristics maximize the performance improvement proportional to the number of FMs involved in the configuration. Therefore, the best performance improvement with respect to the default strategy is obtained using 5FMs with 200 features each.

Multiple experiments are considered for further research. First, other techniques as well as other CP heuristics like the ones proposed by Pesant et al. [31], must be studied to improve performance and scalability when analyzing MPLs. In concrete, we are considering hybrid algorithms for Stochastic Local Search [18], that have proven to be highly competitive for solving a range of hard computational problems. Second, the extended FSG can be used to configure products in operational domains (e.g., cloud services, medical treatments), or real case studies to allow the generalization of the obtained results. Third, FSGs with different types of CTCs and CMCs should be considered to identify how performance and scalability are affected under certain circumstances. Outliers should also be further explored to contribute to this understanding.

References

[1] K. Apt, Principles of constraint programming, Cambridge University Press, New York, 2003.
[2] D. Batory, Feature models, grammars, and propositional formulas, in: H. Obbink, K. Pohl (Eds.), SPLC 2005, LNCS, 3714, Springer, Berlin, Heidelberg, 2005, pp. 7–20, doi:10.1007/11554844_3.
[3] D. Benavides, S. Segura, P. Trinidad, A. Ruiz-Cortés, Using Java CSP solvers in the automated analyses of feature models, in: 2005 Int. Conf. on Generative and Transformational Techniques in Software Engineering, Springer, Berlin, Heidelberg, 2006, pp. 399–408, doi:10.1007/11877028_16.
[4] D. Benavides, S. Segura, P. Trinidad, A. Ruiz-Cortés, FAMA: tooling a framework for the automated analysis of feature models, in: 1st Int. Workshop on Variability Modelling of Software-intensive Systems, The Irish Software Engineering Research Centre, Limerick, 2007, pp. 129–134.
[5] D. Benavides, P. Trinidad, A. Ruiz-Cortés, Automated reasoning on feature models, in: 17th Int. Conf. on Advanced Information Systems Engineering, Springer, 2005, pp. 491–503, doi:10.1007/11431855_34.
[6] F. Boussemart, F. Hemery, C. Lecoutre, L. Sais, Boosting systematic search by weighting constraints, in: 16th European Conf. on Artificial Intelligence, IOS Press, Amsterdam, The Netherlands, 2004, pp. 146–150.

[7] J. Chavarriaga, C. Noguera, R. Casallas, V. Jonckers, Propagating decisions to detect and explain conflicts in a multi-step configuration process, in: J. Dingel, W. Schulte, I. Ramos, S. Abrahão, E. Insfran (Eds.), MODELS 2014, LNCS, 8767, Springer, Cham, 2014, pp. 337–352, doi:10.1007/978-3-319-11653-2_21.

[8] N. Gamez, J.E. Haddad, L. Fuentes, SPL-TQSSS: a software product line approach for stateful service selection, in: IEEE Int. Conf. on Web Services, IEEE, 2015, pp. 73–80, doi:10.1109/ICWS.2015.20.

[9] J. García-Galán, O.F. Rana, P. Trinidad, A.R. Cortés, Migrating to the cloud - a software product line based analysis, in: 3rd Int. Conf. on Cloud Computing and Services Science, SciTePress, 2013, pp. 416–426, doi:10.5220/0004357104160426.

[10] O. González-Rojas, L. Ochoa-Venegas, A decision model and system for planning and adapting the configuration of enterprise information systems, Comput. Ind. 92–93 (2017) 161–177, doi:10.1016/j.compind.2017.08.004.

[11] C. Henard, M. Papadakis, M. Harman, Y. Le Traon, Combining multi-objective search and constraint solving for configuring large software product lines, in: 37th Int. Conf. on Software Engineering, IEEE, 2015, pp. 517–528, doi:10.1109/ICSE.2015.69.

[12] R.M. Hierons, M. Li, X. Liu, S. Segura, W. Zheng, SIP: Optimal product selection from feature models using many-Objective evolutionary optimization, ACM Trans. Softw. Eng. Methodol. 25 (2) (2016) 17:1–17:39, doi:10.1145/2897760.

[13] A.S. Karataş, H. Oğuztüzün, A. Doğru, Mapping extended feature models to constraint logic programming over finite domains, in: 14th Int. Conf. on Software Product Lines, Springer, Berlin, Heidelberg, 2010, pp. 286–299, doi:10.1007/978-3-642-15579-6_20.

[14] A.S. Karataş, H. Oğuztüzün, A. Doğru, From extended feature models to constraint logic programming, Sci. Comput. Program. 78 (12) (2013) 2295–2312, doi:10.1016/j.scico.2012.06.004.

[15] S.Q. Lau, Domain analysis of e-commerce systems using feature-based model templates, Master's Thesis, Electrical and Computer Engineering. University of Waterloo, Canada, 2006.

[16] A.F. Leite, V. Alves, G.N. Rodrigues, C. Tadonki, C. Eisenbeis, A.C.M.A.d. Melo, Automating resource selection and configuration in inter-clouds through a software product line method, in: 8th Int. Conf. on Cloud Computing, IEEE, 2015, pp. 726–733, doi:10.1109/CLOUD.2015.101.

[17] M. Mannion, Using first-order logic for product line model validation, in: G.J. Chastek (Ed.), SPLC 2002, LNCS, Springer, London, 2002, pp. 176–187, doi:10.1007/3-540-45652-X_11.

[18] M.-E. Marmion, F. Mascia, M. López-Ibáñez, T. Stützle, Automatic design of hybrid stochastic local search algorithms, in: M.J. Blesa, C. Blum, P. Festa, A. Roli, M. Sampels (Eds.), Hybrid Metaheuristics, LNCS, 7919, Springer Berlin Heidelberg, 2013, pp. 144–158, doi:10.1007/978-3-642-38516-2_12.

[19] R. Mazo, C. Dumitrescu, C. Salinesi, D. Diaz, Recommendation heuristics for improving product line configuration processes, in: M.P. Robillard, W. Maalej, R.J. Walker, T. Zimmermann (Eds.), Recommendation Systems in Software Engineering, Springer, Berlin, Heidelberg, 2014, pp. 511–537, doi:10.1007/978-3-642-45135-5_19.

[20] R. Mazo, C. Salinesi, D. Diaz, O. Djebbi, A. Lora-Michiels, Constraints: the heart of domain and application engineering in the product lines engineering strategy, Int. J. Inf. Syst. Model. Des. 3 (2) (2012) 33–68, doi:10.4018/jismd.2012040102.

[21] M. Mendonça, T.T. Bartolomei, D. Cowan, Decision-making coordination in collaborative product configuration, in: ACM Symposium on Applied Computing, ACM, 2008, pp. 108–113, doi:10.1145/1363686.1363715.

[22] M. Mendonça, A. Wasowski, K. Czarnecki, D.D. Cowan, Efficient compilation techniques for large scale feature models, in: 7th Int. Conf. on Generative Programming and Component Engineering, ACM, 2008, pp. 13–22, doi:10.1145/1449913.1449918.

[23] L. Ochoa, O. González-Rojas, Program synthesis for configuring collaborative solutions in feature models, in: I. Ciuciu, C. Debruyne, H. Panetto, G. Weichhart, P. Bollen, A. Fensel, M.-E. Vidal (Eds.), OTM 2016 Workshops, LNCS, 10034, Springer, Cham, 2017, pp. 98–108, doi:10.1007/978-3-319-55961-2_10.

[24] L. Ochoa, O. González-Rojas, T. Thüm, Using decision rules for solving conflicts in extended feature models, in: 2015 ACM SIGPLAN Int. Conf. on Software Language Engineering, ACM, 2015, pp. 149–160, doi:10.1145/2814251.2814263.

[25] L. Ochoa, O. González-Rojas, M. Verano, H. Castro, Searching for optimal configurations within large-scale models: a cloud computing domain, in: S. Link, J.C. Trujillo (Eds.), MoBiD 2016, LNCS, 9975, Springer, 2016, pp. 65–75, doi:10.1007/978-3-319-47717-6_6.

[26] L. Ochoa, O. González-Rojas, A.P. Juliana, H. Castro, G. Saake, A systematic literature review on the semi-automatic configuration of extended product lines, J. Syst. Software 144 (2018) 511–532, doi:10.1016/j.jss.2018.07.054.

[27] L. Ochoa, J.A. Pereira, O. González-Rojas, H. Castro, G. Saake, A survey on scalability and performance concerns in extended product lines configuration, in: Eleventh Int. Workshop on Variability Modelling of Software-intensive Systems, ACM, 2017, pp. 5–12, doi:10.1145/3023956.3023959.

[28] R. Olaechea, D. Rayside, J. Guo, K. Czarnecki, Comparison of exact and approximate multi-objective optimization for software product lines, in: 18th Int. Software Product Line Conference, ACM, New York, 2014, pp. 92–101, doi:10.1145/2648511.2648521.

[29] S.K. Ong, Q. Lin, A.Y.C. Nee, Web-based configuration design system for product customization, Int. J. Prod. Res. 44 (2) (2006) 351–382, doi:10.1080/00207540500244153.

[30] J.A. Parejo, A.B. Sánchez, S. Segura, A. Ruiz-Cortés, R.E. Lopez-Herrejon, A. Egyed, Multi-objective test case prioritization in highly configurable systems, J. Syst. Software 122 (C) (2016) 287–310, doi:10.1016/j.jss.2016.09.045.

[31] G. Pesant, C. Quimper, A. Zanarini, Counting-based search: branching heuristics for constraint satisfaction problems, J. Artif. Int. Res. 43 (1) (2012) 173–210, doi:10.1613/jair.3463.

[32] K. Pohl, G. Böckle, F.J.v.d. Linden, Software product line engineering: foundations, principles and techniques, Springer-Verlag, Berlin Heidelberg, 2005.

[33] M.R. Prasad, A. Biere, A. Gupta, A survey of recent advances in sat based formal verification, Int. J. Softw. Tools Technol. Transfer 7 (2) (2005) 156–173, doi:10.1007/s10009-004-0183-4.

[34] C. Prud'homme, J.-G. Fages, X. Lorca, Choco documentation, TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016.

[35] C. Salinesi, R. Mazo, D. Diaz, O. Djebbi, Using integer constraint solving in reuse based requirements engineering, in: 18th IEEE Int. Requirements Engineering Conference, IEEE, 2010, pp. 243–251, doi:10.1109/RE.2010.36.

[36] L.E. Sanchez, S. Moisan, J.-P. Rigault, Metrics on feature models to optimize configuration adaptation at run time, in: 1st Int. Workshop on Combining Modelling and Search-Based Software Engineering, IEEE, 2013, pp. 39–44, doi:10.1109/CMSBSE.2013.6604435.

[37] A.S. Sayyad, J. Ingram, T. Menzies, H. Ammar, Scalable product line configuration: a straw to break the camel's back, in: 28th IEEE/ACM Int. Conf. on Automated Software Engineering, IEEE, Piscataway, 2013, pp. 465–474, doi:10.1109/ASE.2013.6693104.

[38] A.S. Sayyad, T. Menzies, H. Ammar, On the value of user preferences in search-based software engineering: a case study in software product lines, in: Int. Conf. on Software Engineering, IEEE, Piscataway, 2013, pp. 492–501, doi:10.1109/ICSE.2013.6606595.

[39] T.H. Tan, Y. Xue, M. Chen, J. Sun, Y. Liu, J.S. Dong, Optimizingselection of competing features via feedback-directed evolutionary algorithms, in: 2015 Int. Symposium on Software Testing and Analysis, ACM, New York, 2015, pp. 246–256, doi:10.1145/2771783.2771808.

[40] T. Thüm, D. Batory, C. Kastner, Reasoning about edits to feature models, in: 31st Int. Conf. on Software Engineering, IEEE, Washington, 2009, pp. 254–264, doi:10.1109/ICSE.2009.5070526.

[41] J. White, J.A. Galindo, T. Saxena, B. Dougherty, D. Benavides, D.C. Schmidt, Evolving feature model configurations in software product lines, J. Syst. Software 87 (2014) 119–136, doi:10.1016/j.jss.2013.10.010.