Applications of #SAT Solvers on Feature Models

Chico Sundermann
University of Ulm, Germany

Michael Nieke
TU Braunschweig, Germany

Paul Maximilian Bittner
University of Ulm, Germany

Tobias Heß
University of Ulm, Germany

Thomas Thüm
University of Ulm, Germany

Ina Schaefer
TU Braunschweig, Germany

ABSTRACT

Product lines are ubiquitous for managing variable systems. The variability of a product line is typically described in terms of a feature model. Analyzing a feature model gives insight into various aspects, such as the validity of a configuration of features. Several analyses have been considered that require computing the number of valid configurations which proves highly inefficient when using regular SAT solvers. A #SAT solver computes the number of satisfying variable assignments of a propositional formula and is specifically optimized for the aforementioned analyses. In this work, we summarize and unify the state-of-the-art on #SAT-based feature-model analyses and propose a variety of new #SAT-based analyses. We provide an exhaustive catalogue for applications of #SAT for feature models serving as a reference for researchers and practitioners. Furthermore, we show that all these 21 applications are based on only three different #SAT queries. Thus, future research can focus on providing solutions and optimizations for those three queries to accelerate #SAT-based applications.

CCS CONCEPTS

• **General and reference** → *Surveys and overviews*; • **Software and its engineering** → **Software configuration management and version control systems**; *Constraint and logic languages*; *Automated static analysis*; *Software evolution*; • **Hardware** → *Theorem proving and SAT solving*.

KEYWORDS

Feature Models, Product Lines, Model Counting, Configuration Counting, #SAT, #SAT Applications, Feature-Model Analysis

ACM Reference Format:
Chico Sundermann, Michael Nieke, Paul Maximilian Bittner, Tobias Heß, Thomas Thüm, and Ina Schaefer. 2021. Applications of #SAT Solvers on Feature Models. In *15th International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS'21), February 9–11, 2021, Krems, Austria.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3442391.3442404

VaMoS'21, February 9–11, 2021, Krems, Austria

ACM ISBN 978-1-4503-8824-5/21/02…$15.00
https://doi.org/10.1145/3442391.3442404

1 INTRODUCTION

Product lines are widely used to manage families of similar products [10, 13, 32]. Each product is composed from a set of distinct units, called *features*, that are reused across products [56]. A product can be identified by the set of features it implements, referred to as its *configuration*. The most common way to specify the set of valid configurations are *feature models* [8, 10, 20].

Managing a product line is challenging due to the high number of constraints imposed by the feature model [10]. Various analyses have been proposed, such as checking the validity of a configuration [8, 10, 20, 25, 43, 51–54, 57]. Industrial feature models, such as the Linux kernel, contain thousands of features and constraints [38, 59]. Furthermore, these systems often grow significantly over time [26, 33, 38, 40, 59]. To properly manage a product line over time, it is necessary to control its variability. Tom DeMarco amongst others stated "*you can't control what you can't measure*" [22]. We propose using the number of valid configurations as a metric to measure the variability of a product line. Several analyses dependent on the number of valid configurations have been proposed, such as uniform random sampling [44, 47, 48, 55]. Typically, a feature model is translated to propositional logic to analyze it [8, 10, 61]. While it is possible to count assignments using satisfiability-based tools, such as SAT solvers, this typically comes with major scalability issues [48, 52, 59, 63].

Counting satisfying assignments of a propositional formula is known as #SAT and well researched [14, 21, 35, 39, 62]. Yet, possible applications of #SAT technology for feature models have been researched sparsely [16, 23, 24, 28, 32, 38]. Heradio et al. [28, 32] propose several applications based on computing the number of valid configurations limited to estimations of the economic benefits of a product line. Fernandez et al. [24] provide a smaller scale survey with only five applications which are limited to computing the number of valid configurations that contain a certain feature. Other research that proposes using the number of valid configurations just considers very few applications [10, 16, 23, 30, 38, 41, 52, 59]. Gathering an overview for applications based on the number of valid configurations currently requires a large time commitment for researchers and practitioners. Furthermore, proposed applications are often vaguely defined, lack formal definitions, or miss out on feasible algorithms to compute them. This induces additional effort to (1) generally interpret the application, (2) resolve ambiguities, and (3) compute actual results. We consider each of those aspects in our work to make applications of #SAT solvers on feature models more accessible with the following three major contributions.

First, we survey applications of #SAT on feature models considered in the literature. The goal is to gather existing applications with detailed and consistent descriptions in order to significantly

reduce the effort for research. Overall, we identified twelve distinct applications considered in the literature.

Second, we propose nine new #SAT-based applications to show additional possibilities and further reveal the currently unused potential of applying #SAT for feature-model analysis. The newly considered applications are mainly inspired by the cooperation with industry partners.

Third, we identified three different queries to a #SAT solver which provide the basis for every single of the 21 considered applications. In particular, these queries consist of computing (1) the number of valid configurations for the entire feature model (cf. Section 4), (2) the number of valid configurations including a particular feature (cf. Section 5), and (3) the number of remaining valid configurations of a partial configuration (cf. Section 6). For each of the three queries, we provide a solution that can easily be applied to off-the-shelf #SAT solvers. As these three queries form the basis for all identified applications, optimizations can focus on these three queries alone.

In summary, we provide an expert survey for applications dependent on computing the number of valid configurations of a feature model. Our work motivates further research and applying #SAT technology to feature models and makes required information vastly more accessible as there has not been a large-scale survey of #SAT-based applications before.

2 RUNNING EXAMPLE

In this section, we introduce a feature model that we use in the following sections to explain and motivate applications for #SAT. Fig. 1 shows the feature model of a simplified car. Each car requires a `Carbody` (denoted by MANDATORY) and exactly one `Gearbox`-type (denoted by ALTERNATIVE). Cars may contain a `Radio` that can be further configured (denoted by OPTIONAL). Each `Radio` may contain a `USB`-port, `CD`-port, or both (denoted by the OR group of `Radio's` OPTIONAL child `Ports`). The cross-tree constraint `Navigation` ⇒ `USB` ensures that each car with `Navigation` has an `USB` port.

Both, the tree hierarchy and the cross-tree constraints limit the set of valid configurations. Computing the cardinality of the set of valid configurations is trivial for feature models without cross-tree constraints [32]. For such models, each sub-tree can be analyzed separately as there are no interdependencies between features from different sub-trees. Thus, the number of valid configurations can be computed with a linear-time traversal of the feature model by applying certain rules for each relationship type (e.g., the number of valid configuration of an alternative group is the sum of its features) [32]. However, for feature models with cross-tree constraints simply traversing the tree typically leads to faulty results as the restrictions imposed by the cross-tree constrains are disregarded.

3 BACKGROUND

SAT describes the problem of computing the satisfiability of a propositional formula. A propositional formula F is defined over a set of variables $vars(F)$ and consists of literals and logical operators $\neg$ (not), $\vee$ (or), and $\wedge$ (and). Each literal corresponds to exactly one variable $v \in vars(F)$. An assignment $\alpha : vars(F) \rightarrow \{\top, \bot\}$ maps variables $v \in vars(F)$ to the truth values $\top$ (true) and $\bot$ (false). We denote the number of variables which α assigns to truth values by $|\alpha|$. If $|\alpha| = |vars(F)|$, we call α *total*. Otherwise, we call it *partial*.

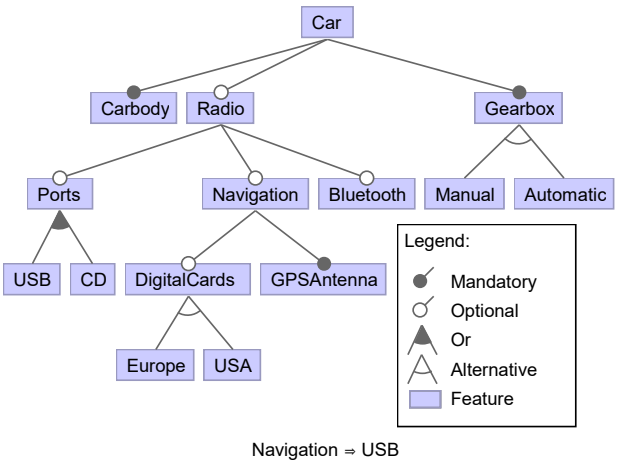

Figure 1: Simplified Car Feature Model (adapted from [5])

We lift assignments α to formulas F such that $\alpha(F)$ evaluates the formula F under α. If $\alpha(F) = \top$, we call α a *satisfying* assignment for F. Formally, the SAT problem corresponds to deciding whether an α exists with $\alpha(F) = \top$ for a given formula F.

#SAT, instead, is the problem of computing the number of satisfying assignments for a propositional formula F. Let A be the set of all total assignments over $vars(F)$. As each variable can be assigned either to $\top$ or $\bot$, there are $|A| = 2^{|vars(F)|}$ total assignments. Typically, not every assignment α satisfies the formula F. A #SAT solver computes the number of assignments in A that satisfy F (i.e., $\#F = \left|\{\alpha \in A \mid \alpha(F) = \top\}\right|$). #SAT is widely believed to be a computationally harder problem than SAT [14, 65]. It is obvious that #SAT is at least as hard as SAT considering it is trivial to determine whether a formula is satisfiable after computing the number of satisfying assignments [34].

We define a feature model as a tuple $FM = (FEATS, CONST)$ over a set of features $FEATS$ and a set of boolean constraints $CONST$ which includes tree and cross-tree constraints. A configuration $C = (FM, I, E)$ consists of a set of included features I and excluded features E. A feature cannot be included and excluded in the same configuration (i.e., $I \cap E = \emptyset$). If each feature is either included or excluded in C (i.e., $I \cup E = FEATS$), we call C *total* [10]. Otherwise, C is *partial*. We consider a configuration $C = (FM, I, E)$ to be *valid* if the sets of included I and excluded E features violate no constraints in $CONST$. Checking whether a configuration violates a subset of the constraints is typically infeasible by hand and, thus, requires efficient tool support [16, 31, 57].

Instead of developing new ad-hoc algorithms to analyze feature models, they are commonly translated to equivalent propositional formulas [8, 20, 25, 43, 51–54]. A formula F_{FM} is *equivalent* to a feature model FM ($F_{FM} \equiv FM$) if the following two properties hold: (1) for each configuration $C = (FM, \{i_1, ...i_n\}, \{e_1, ..., e_m\})$ and an assignment α, with $\alpha(i_{k=1,...,n}) = \top$ and $\alpha(e_{j=1,...,m}) = \bot$, $\alpha(F_{FM}) = \top$ iff C is valid and (2) $vars(F_{FM}) = FEATS$. For each feature model an equivalent propositional formula exists [10, 43]. The number of satisfying assignments of an equivalent formula F_{FM} is equal to the number of valid configurations by definition (i.e., $\#F_{FM} = \#FM$). Thus, invoking a #SAT solver with F_{FM} as input can be used to compute the number of valid configurations $\#FM$. In

Application	Source
Void feature model	Benavides et al. [10]
Variability factor	Benavides et al. [9], Fernandez et al. [24], Heradio et al. [28, 32]
Variability of feature subsets	**New**
Maintainability prediction	Bagheri et al. [7]
Variability reduction	**New**
Rating errors	Kübler et al. [38]
Degree of orthogonality	Czarnecki et al. [19] Benavides et al. [10]
Configuration relevance	**New**
Cost savings of a product line	Clements et al. [17], Heradio et al. [28]

Table 1: Applications - Cardinality of Feature Models

the following, we present three queries which we use as input for #SAT solvers to acquire results for all 21 considered applications.

4 CARDINALITY OF FEATURE MODELS

The most basic query to analyze feature models using #SAT solvers is to compute the number of valid configurations of the entire feature model. We refer to the cardinality of the set of valid configurations as the *cardinality of the feature model*.

Definition 1 (Cardinality of a Feature Model). *Let FM be a feature model and VC the set of valid configurations for FM. The cardinality of FM is defined by the cardinality of VC:*

$$\#FM = |VC|$$

For a feature model FM translated to an equivalent propositional formula F_{FM} (i.e., $FM \equiv F_{FM}$), $\#FM = \#F_{FM}$ holds by definition. Thus, $\#FM$ can be computed by invoking a #SAT solver on F_{FM} without further adaptations. As the translation of feature models to propositional formulas is well-researched [8, 10, 43, 61] and we propose using off-the-shelf #SAT solvers, this is trivially applicable. Table 1 provides an overview of applications based on the cardinality of feature models.

Void Feature Model. A feature model is considered void if it induces no valid configurations [10]. Computing its cardinality implicitly detects whether the feature model is void.

$$\#FM = 0 \Leftrightarrow FM \text{ is void} \tag{1}$$

In general, computing the voidness of a feature model with a SAT solver should be more efficient. However, if a feature model's cardinality is required anyways, an additional SAT call can be saved.

Variability Factor. The variability factor of a feature model describes the relative share of configurations that are valid compared to all configurations [9, 24, 28, 32]. Disregarding all restrictions, every combination of features induces a valid configuration. In this case, there are $2^{|FEATS|}$ configurations. The variability factor of a feature model FM is computed as follows:

$$VariabilityFactor(FM) = \frac{\#FM}{2^{|FEATS|}} \tag{2}$$

The variability factor lies between zero and one. A variability factor close to zero indicates a highly restricted feature model, while a feature model with no restrictions has a variability factor of one [28, 32]. Benavides et al. [9] argue that the variability factor can be used to estimate the benefits of a product-line approach as a small variability factor may indicate that developing standalone products is more beneficial. An unexpected variability factor may indicate design errors.

Our running example contains 15 features, only 1 cross-tree constraint, and induces 42 valid configurations. The variability factor $\frac{42}{2^{15}} = \frac{42}{32768} = 0.0013$ indicates that only a small fraction of all configurations is valid. This shows that the feature-model hierarchy limits the set of valid configurations by a large margin.

The strong limitations of the tree structure may make it difficult to grasp the impact of cross-tree constraints. We propose to simplify grasping the impact by introducing the *cross-tree variability factor* which compares the cardinality of a feature model with cross-tree constraints (FM) and without ($FM_{\overline{CTC}}$).

$$VariabilityFactor_{CTC}(FM) = \frac{\#FM}{\#FM_{\overline{CTC}}} \tag{3}$$

A larger cross-tree variability factor indicates that approximating the cardinality of the feature model by disregarding cross-tree constraints provides more accurate results. Without cross-tree constraints, our running example induces 66 valid configurations. Thus, the variability factor of cross-tree constraints is $\frac{42}{66} = 0.636$ which shows that the cross-tree constraints do not limit the number of valid configurations by a large margin.

Variability of Feature Subsets. Analyzing large feature models is computationally expensive. In some instances, it may be sufficient to focus analyses on specific parts of the feature model. We propose using the cardinality of a feature model to compute the variability of only a feature subset (e.g., a feature model sub-tree). For instance, this can be used to compute the cardinality of a feature model disregarding *abstract* features. Abstract features are irrelevant for the actual products and are used to structure the feature model [61]. The variability of a feature subset can also be used to find parts with a particularly high or low variability. Other analyses can then focus on identified parts. Furthermore, an unexpected variability degree for a subset may indicate modeling errors.

Given a set of features $S \subseteq FEATS$, the idea is to compute the number of different combinations of S that appear in the set of valid configurations. Consider the following example: Let $S = \{A, C\}$ be a feature subset and $I_1 = \{A, B, D\}$, $I_2 = \{A, C, D\}$, $I_3 = \{A, B, C\}$, and $I_4 = \{A, B, C, D\}$ the valid configurations induced by the feature model. While there are overall four valid configurations, only two different combinations of the features A, C appear ($\{A\}$ and $\{A, C\}$). In this case, the variability of the feature subset S is two. To compute the variability of a feature subset, we propose to use a combination of feature-model slicing and #SAT. Slicing can be used to remove features while preserving the restrictions that were imposed by the removed feature [1, 36]. After slicing out all features that are not part of the subset (i.e., $FEATS \setminus S$), we obtain a feature model that only contains features that appear in S but remains all original restrictions. Computing the cardinality of the resulting feature model is equal to the variability of S in the feature model FM.

$$SubsetVariability(FM, S) = \#(Slice(FM, FEATS \setminus S)) \tag{4}$$

In our example, the sub-tree induced by the feature `Radio` may be our subset S. The sub-tree induces 21 different variants. With this number, we can see that the main variability results from S, as the rest of the feature model only induces $\frac{42}{21} = 2$ variants which result from the alternative between `Automatic` and `Manual`.

Maintainability Prediction. Bagheri et al. [6] predict the maintainability of a product line using different metrics about the feature model such as the number of features or its cardinality. Such predictions using structural metrics allows assessing the maintainability of a product line in early stages of development. The insights can be used to preserve maintainability in later stages by effectively reducing the complexity early. Bagheri et al. examine statistical correlations between ten metrics and the maintainability. According to their evaluation, a high cardinality of the feature model strongly indicates a high complexity of the product line. With the gathered insights, they propose a prediction model to estimate the maintainability. The model includes the cardinality of a feature model as one of four metrics.

Variability Reduction. We propose to use the cardinality of a feature model to examine the impact of a change to the feature model. We call the change in cardinality from FM to an adaptation FM' variability reduction.

$$VariabilityReduction(FM, FM') = \#FM - \#FM' \tag{5}$$

One use-case of this metric is to identify undesired changes to the product line. If the cardinality of a feature model changes in an unexpected way, this is an indicator for a faulty edit. For example, an additional constraint that does not change the cardinality of the feature model is redundant [38]. Also, new constraints may unexpectedly reduce the cardinality by a large margin. In this case, the edits can be re-evaluated and design flaws can be prevented. For instance, for our running example, a company sells cars with `Automatic` exclusively to the USA. Thus, developers want to introduce a new cross-tree constraint `Automatic` $\Rightarrow$ `USA`. The resulting feature model FM' induces 25 valid configurations. The original 42 valid configurations can be separated in 21 configurations with `Automatic` and 21 configurations with `Manual`. As the introduced constraint does not remove any valid configuration with `Manual`, there are only 4 valid configurations with `Automatic` left. This information can be used by developers to reconsider the changes. A possible alternative may be `DigitalCards` $\wedge$ `Automatic` $\Rightarrow$ `USA`. The resulting model induces 38 valid configurations. Even though the originally proposed constraint caused an unintended high variability reduction, it did not introduce any traditional anomalies, like dead or false-optional features [10]. Thus, such design flaws are difficult to detect using satisfiability-based analyses.

Reducing the variability of a product line may simplify quality assurance and maintenance [6, 38], as fewer products need to be considered. To achieve this effectively, it is necessary to know the impact of an edit regarding the variability reduction. However, it is difficult or even impossible to estimate the difference in the number of configurations without automated support. The cardinalities before and after multiple changes can be used as one of the criteria to select a possible change. Consider the following two changes to our running example: remove the gearbox type `Manual` or make `Radio` MANDATORY. These result in a remaining cardinality of 21 or 40, respectively. Thus, removing `Manual` is more effective regarding the variability reduction.

Thüm et al. [60] propose computing the added and removed configurations explicitly. While this provides the variability reduction implicitly, it is not feasible for large differences. In previous work, we showed that there are up to $\approx 10^{1500}$ added configurations after a new version of a feature model [59]. If we store the 10^{1500} configurations with a size of one byte each, we need 10^{1488} terabytes of memory. Enumerating and storing is not feasible but we can compute the change in the cardinality.

Rating Errors. Kübler et al. [38] propose using the cardinality of a feature model specialization for rating an error. A specialization is an adaptation of the feature model which adds additional constraints removing valid configurations [60]. The idea is that an erroneous subset of the configuration space included in more valid configurations is potentially more critical. Let FM_{ERR_1}, FM_{ERR_2} be two specializations of the feature model FM that are erroneous.

$$\begin{gathered}\#FM_{ERR_1} > \#FM_{ERR_2} \Rightarrow \\ FM_{ERR_1} \text{ is potentially more critical than } FM_{ERR_2}\end{gathered} \tag{6}$$

Suppose there are two specializations of FM that cause errors in the implementation of our car product line: FM' which always contains `Bluetooth` and `USB` and FM'' which always contains `Automatic` and at least one of `Radio` and `CD`. Those specializations induce 16 and 20 valid configurations, respectively. This can be used as an indicator to argue that the error induced by FM'' is more critical and that the developers should focus on this error.

Degree of Orthogonality. Feature-model analyses are typically complex because of sub-tree interdependencies specified by cross-tree constraints. If the impact of these interdependencies is small, the sub-trees can be evaluated separately to get a reasonable estimate. To analyze a single sub-tree, it is possible to use only local constraints [10, 19]. Czarnecki et al. [19] and Benavides et al. [10] define constraints as local if they are cross-tree constraints with only a single feature, or tree constraints. The degree of orthogonality describes the ratio between the cardinality of the feature model with all and with only local constraints [20]. A high degree of orthogonality indicates that interdependencies between subtrees have a low impact regarding the cardinality of the feature model. In this case, analyses can be performed locally [20]. Let FM be a feature model and FM_{local} a generalization that only considers the tree hierarchy and local cross-tree constraints. The degree of orthogonality is computed as follows:

$$DegreeOfOrthogonality(FM) = \frac{\#FM}{\#FM_{local}} \tag{7}$$

For our example, the degree of orthogonality is equal to the cross-tree variability factor (0.636) as there are no local cross-tree constraints. This indicates that analyses only considering local constraints may provide a reasonable estimate depending on the accuracy required for the application.

Configuration Relevance. We propose using the cardinality of a feature model to rate a configuration's relevance. For example, consider 1,000 products that were ordered from a product line. One product based on configuration C_1 was ordered 20 times. Without knowing the size of the configuration space, it is difficult to tell whether C_1 is more or less represented than an arbitrary configuration. The product line may contain 42 valid configurations like in our motivating example or even 10^{11} valid configurations like a smaller industrial model which we evaluated in previous work [59]. In the first case, C_1 has a below-average representation, but in the second case a highly above-average representation. Let L be the list of configurations, C_1 the configuration in question, and $|L_C|$ the

number of configurations equivalent to C in L.

$$RelevanceOfConfiguration(FM, L, C_1) = \#FM \cdot \frac{|L_{C_1}|}{|L|} \quad (8)$$

$RelevanceOfConfiguration(FM, L, C_1) = a$ states that C_1 appears a times as often in L compared to an arbitrary configuration. A relevance score a below one indicates, that C_1 has an below-average representation. A relevance score a higher than one indicates, that C_1 has an above-average representation which can be used to prioritize C_1 for testing and maintenance.

Cost Savings of a Product Line. Clements et al. [17] and Heradio et al. [28] use the cardinality of feature models to evaluate whether it is beneficial to develop a product family as product line (PL) or not (i.e., as standalone products) (NPL). The costs $CNPL$ to develop standalone products can be estimated by multiplying $\#FM$ with a cost estimation CP to develop a single product.

$$CNPL(FM) = \#FM \cdot CP \quad (9)$$

For the cost of a product-line approach, the authors consider first building one standalone product [17, 28]. Afterwards, the common features (i.e., features that appear in more than one configuration) are changed to be reusable. The relative costs for developing the features for reusability are referenced by CR. A $CR = 2$ indicates that building the features for reuse is twice as much effort than developing it as a standalone product. Then, the rest of the products $\#FM - 1$ can be built with a reduced cost CPR [28]. The details on how to compute CPR are omitted in the following formula as they are not related to the cardinality of a feature model.

$$CPL(FM) = CR \cdot CP + (\#FM - 1) \cdot CPR \quad (10)$$

Overall, we are interested in the difference between the costs of implementing the family as a product line $CPL(FM)$ and the costs of all standalone products $CNPL(FM)$. This difference indicates the benefit of a product-line approach [17, 28]. Consider the following example where developing a standalone car costs $CP = 3$ and it costs twice as much to develop for reuse (i.e., $CR = 2$). Each additional car then costs $CPR = 2$ to be built. Overall, this results in $CNPL(FM) - CPL(FM) = (42 \cdot 3) - (2 \cdot 3 + 41 \cdot 2) = 39$ which indicates that it requires less effort to build all 42 possible cars with a product-line approach.

Summary. In this section, we presented six applications from the literature and three new proposals based on the cardinality of a feature model. After computing the cardinality of a feature model once, the result can be reused across all applications presented in this section. This can be used for deriving configurations, collecting statistical inferences from the feature model, estimate economical benefits, or to support developers. We examined the scalability of several #SAT solvers on computing cardinalities of feature models in our previous work and found that multiple solvers scale to a vast majority of industrial feature models [59].

5 CARDINALITY OF FEATURES

For several applications, it is important to only consider a subset of the configuration space, such as configurations that contain a certain feature. We propose the *cardinality of a feature* which we define as the number of valid configurations that contain that feature. A similar concept has been proposed with the *commonality of a feature* which describes the relative share of valid configurations that contain the feature [24, 49, 64]. In consequence, commonality is a value between zero and one that indicates the probability of the feature to appear in a given valid configuration [49].

Application	Source
Core, dead & false-optional	Trinidad et al. [64], Perez [50]
Atomic set candidates	**New**
Feature prioritization	**New**
CTC restrictiveness for features	**New**
Payoff threshold of a feature	Heradio et al. [28, 32]
Degree of reuse	Cohen et al. [18, 24, 28]
Homogeneity	Clements et al. [17], Fernandez et al. [24], Heradio et al. [28, 32]
Optimize configuration process	Chen et al. [16], Mazo et al. [41]

Table 2: Applications - Cardinality of Features

Definition 2 (Cardinality of a Feature). *Let $FM = (FEATS, CONST)$ be a feature model, $f \in FEATS$ a feature, and VC the set of valid configurations. The cardinality of f is defined as the number of valid configurations that contain f:*

$$\#FM_f = \left|\{C = (FM, I, E) \in VC \mid f \in I\}\right|$$

Definition 3 (Commonality of a Feature). *Let $\#FM$ and $\#FM_f$ be the cardinality of the feature model FM and a feature f, respectively. The commonality of f is defined as:*

$$Commonality(FM, f) = \begin{Bmatrix} \frac{\#FM_f}{\#FM} & \#FM \neq 0 \\ 0 & \#FM = 0 \end{Bmatrix}$$

Similar to the cardinality of a feature model, it is possible to compute the cardinality of a feature using a #SAT solver. Let FM be a feature model and F_{FM} a propositional formula with $FM \equiv F_{FM}$. The cardinality $\#FM_f$ of the feature f is equal to $\#(F_{FM} \wedge f)$ which can be used as input for a #SAT solver. In our running example, the feature USB appears in 20 out of the 42 valid configurations. Thus, its cardinality is 20 and its commonality is 0.476. Table 2 provides an overview for applications based on the cardinality of features.

Core, Dead & False-Optional Features. The commonality of a feature can be used to identify anomalies. Given a feature model FM, a feature f, and the commonalities of f and its parent p, the anomalies dead, core, and false-optional feature can be computed in the following way. If the entire feature model is void, we consider each feature to be dead as our definition of commonality implies.

$$Commonality(FM, f) = 1 \Leftrightarrow f \text{ is a core feature [50]} \quad (11)$$

$$Commonality(FM, f) = 0 \Leftrightarrow f \text{ is a dead feature [50, 64]} \quad (12)$$

$$Commonality(FM, p) = Commonality(FM, f) \text{ and } f \text{ is not mandatory} \Leftrightarrow f \text{ is false-optional} \quad (13)$$

We expect that existing analyses for anomalies that are based on satisfiability are less time-consuming. However, if the commonality of features is computed anyway, these defects can be identified without additional SAT calls. In our running example, `Carbody` has a cardinality of 42 which is equal to the cardinality of the feature model. Thus, its commonality is one, which indicates its core status.

Atomic Set Candidates. We propose to use the cardinality of features to find possible candidates for atomic sets. Features in the same atomic set always appear in the same number of valid configurations. Otherwise, there would be at least one configuration that contains only a subset of the features. However, two features

with the same cardinality may still be part of distinct valid configurations. Thus, they are not necessarily part of the same atomic set. The following formulas can be used to find atomic set candidates.

$$\#FM_f = \#FM_g \Rightarrow f, g \text{ are candidates for an atomic set} \quad (14)$$

$$\#FM_f \neq \#FM_g \Rightarrow f, g \text{ are not part of the same atomic set} \quad (15)$$

In our example, both `Navigation` and `GPSAntenna` appear in 24 valid configurations. Thus, they may be part of the same atomic set. A further analysis can examine whether the features are part of the same atomic set. `USB` appears in 32 valid configurations and, thus, is not part of this atomic set.

Feature Prioritization. We argue that the cardinality of a feature can be used as an indicator for the importance of this feature. For example, a developer may have to decide between two alternative features f_1, f_2 to implement next. To potentially create more distinct products, it is more effective to develop the feature with a higher cardinality first.

$$\begin{gathered}\#FM_{f_1} > \#FM_{f_2} \text{ and prioritize } f_1 \text{ over } f_2 \\ \Rightarrow \text{ Build more distinct products}\end{gathered} \quad (16)$$

In our running example, a product-line engineer may have to decide to implement `USB` or `CD` first. Those appear in 32 and 20 valid configurations, respectively. Therefore, it may be more beneficial to implement `USB` first.

In testing, it may be also interesting to prioritize features with a low cardinality as features with a high cardinality are covered with a high chance anyway. It is typically more likely to oversee a bug in a feature that appears in fewer valid configurations. Knowledge about feature's cardinalities can be used to lower the chances of missing an uncommon feature during testing. In our example, suppose the quality assurance tests five randomly configured cars. `Manual` is not part of the sample with a probability of 3.12%. `USA` has a 34.77% chance to not appear in the sample. Thus, prioritizing `USA` over `Manual` increases the odds for a wider coverage as `Manual` is more likely to be included in the sample anyways.

CTC Restrictiveness for Features. We argue that the cardinality of features can be used to identify features which are heavily restricted by cross-tree constraints. For this, we consider a feature model FM and an adaption $FM_{\overline{CTC}}$ without the cross-tree constraints. Then, we compare the cardinalities of a feature f in FM and in $FM_{\overline{CTC}}$. This information can be used to identify design errors.

$$CTCRestrictiveness(FM, f) = 1 - \frac{\#FM_f}{\#FM_{\overline{CTC},f}} \quad (17)$$

The *CTCRestrictiveness* lies between zero and one. Zero indicates that the feature is not restricted at all by cross-tree constraints. A value close to one indicates that the feature is heavily restricted through cross-tree constraints. Due to modernization of cars with `Automatic`, `CD` is only available for `Manual` cars. The developers introduce a new constraint `CD` $\Leftrightarrow$ `Manual` which unintentionally also requires every car with `Manual` to have a `CD`. These changes result in a $CTCRestrictiveness(\texttt{Manual}) = 1 - \frac{10}{66} = 0.85$ which indicates that `Manual` is heavily restricted. This information can be used to alert the developers to reconsider the added cross-tree constraint and change it to `CD` $\Rightarrow$ `Manual` instead. The adapted constraint results in $CTCRestrictiveness = 1 - \frac{21}{33} = 0.36$.

Payoff Threshold of a Feature. Typically, it is more expensive to develop a feature such that it can be reused for multiple products. However, the required adaptions to use the feature in other products is typically cheaper than to develop it from scratch. For a feature that appears only in a very small number of products it can be inefficient to engineer it for generic usage. The cardinality of a feature may indicate whether it is beneficial to develop it for reuse.

To achieve a return of investment, a feature that is developed for reuse must be reused a certain number of times. This break-even threshold depends on the added cost of developing a feature for reuse and the reduced cost of reusing it. If the number of valid configurations that contain a feature f is larger than the threshold, it is considered to be efficient to develop f for reuse [28, 32]. Let $Cost(f) = 1$ be the relative cost of implementing f from scratch. $CostToDevelopForReuse(f) = a$ indicates that its a times as expensive to develop f for reuse in the product line in contrast to develop it for standalone products with $Cost(f)$. After developing f for reuse, implementing f for a specific product only needs a proportion of the original cost. This reduced cost is described by $CostToReuse_f$.

$$PayoffThreshold(f) = \frac{CostToDevelopForReuse(f)}{1 - CostToReuse(f)} \quad (18)$$

$$\begin{gathered}\#FM_f < PayoffThreshold(f) \Rightarrow \\ \text{It is not efficient to develop } f \text{ for reuse.}\end{gathered} \quad (19)$$

Several authors proposed concrete procedures to compute the $CostToReuse(f)$ and $CostToDevelopForReuse(f)$ [11, 45]. However, the exact computations are beyond the scope of this work.

In our running example, suppose the cost of developing the feature `USA` for reuse is doubled. Furthermore, reusing `USA` costs $2/3$ of the regular cost of implementing it. This results in a threshold of $\frac{2}{1-2/3} = 7$. Thus, `USA` needs to be reused at least seven times to break even. `USA` appears in eight valid configurations which indicates that it is beneficial to develop `USA` for reuse. The benefit potentially even increases further over time, as the configurable systems are typically growing [38, 59].

We expect that payoff thresholds only provide limited insights for industrial feature models. Boehm et al. [11] empirically evaluate the analysis but consider product lines with only a single digit number of valid configurations. Industrial feature models typically induce more than 10^{10} of valid configurations [38, 59].

Degree of Reuse. When implementing a product line, the degree of reuse can provide insight into the benefit of a product-line approach. The degree of reuse indicates the portion of products that is provided by common features (i.e., features that are part of at least two valid configurations). For example, a degree of reuse of 0.4 indicates that a typical product consists of 40% common features [18, 24, 28]. Eq. 20 quantifies the degree of reuse for the feature model FM with the set of common features $COM \subseteq FEATS$ [24, 28]. $Cost(f)$ indicates the effort to develop the feature f.

$$DegreeOfReuse(FM, COM) = \frac{\sum_{f \in COM}(Cost(f) \cdot \#FM_f)}{\sum_{f \in FEATS}(Cost(f) \cdot \#FM_f)} \quad (20)$$

Industrial feature models typically contain no features that appear in only one valid configuration as indicated by the data set used in our previous work [59]. In this case, the degree of reuse does not provide usable results as it is always one. The analysis is more suited for comparing a small number of products that only share some common features.

Suppose there exist three cars of the product family described by our running example that each contain a unique feature that is not part of the feature model. Every feature that directly appears in the feature model is part of at least two configurations and, thus, is common. For the sake of simplicity, we assume that every common feature has a cost of one. The three unique features have a cost of two. This leads to: $DegreeOfReuse = \frac{15}{15+6} = \frac{15}{21} = 0.71$. The result shows that the main effort to build the products comes from common features. This indicates a great benefit of product lines.

Homogeneity. The homogeneity of a product line describes the similarity of the valid configurations [17, 24, 28, 30, 32]. The homogeneity is a value between zero and one. A value close to zero indicates that the valid configurations are dissimilar (i.e., share a small number of features) while a value close to one indicates that the configurations are similar [24]. Clements et al. [17] propose to compute the homogeneity of a product line using the number of features that appear only in one valid configuration. However, Fernandez et al. [24] argue that this formula computes unexpected results for some cases in which highly heterogeneous configuration spaces are identified as homogeneous. Therefore, Fernandez et al. propose the commonality mean of all features [24].

$$Homogeneity(FM) = \frac{\sum_{f \in FEATS} Commonality(FM, f)}{|FEATS|} \quad (21)$$

With this formula, the homogeneity of our running example is 63% which indicates that the valid configurations have a high number of features in common.

Configuration Derivation Optimizations. Deriving a configuration can be improved with tool support, such as selection propagation [57]. Another option is ordering the features with different strategies to accelerate the derivation. First, a different feature order may reduce the number of configuration steps by performing a selection that implies a high number of other selections early. Second, assigning a feature that appears in many constraints may simplify the formula that is used for background analyses (e.g., selection propagation). Selecting such variables early in the configuration process may also accelerate the derivation.

Mazo et al. [41] propose six different heuristics to order features to accelerate the derivation. Two of the heuristics depend on #SAT. The first heuristic orders the features by their cardinality. The idea is to process features first that appear in a large variety of products [41]. The authors argue that this decreases the time required by the solver to find a valid configuration. However, we assume that this is neglected by the higher number of configuration steps required as the selection of a feature with high cardinality reduces the remaining possible selections by a smaller margin. The second heuristic prioritizes features that split the remaining configuration space in two partitions of similar size. Selecting such a feature halves the number of remaining valid configurations. Thus, following this procedure should reduce the number of required steps and the cost of analyses [41].

Chen et al. [16] propose to sort features by their selectivity to optimize product derivation. The selectivity of a feature depends on its cardinality. A highly selective feature has a high cardinality and implies the selection of a high number of other features which reduces the number of required configuration steps.

Application	Source
Uniform Random Sampling	Oh et al. [47, 48], Achlioptas et al. [2], Munoz et al. [44], Sharma et al. [55], Charkchakraborty et al. [15]
Atomic Sets	**New**
Rating Feature Interactions	**New**
Interactive Configuration Support	**New**

Table 3: Applications - Cardinality of Partial Configurations

Summary. In this section, we presented five applications from the literature and three own proposals based computing the cardinality or commonality for each feature of interest. The considered applications can be used for deriving configurations, collecting statistical inferences from the feature model, economic estimations, and support of developers.

6 CARDINALITY OF CONFIGURATIONS

For some applications, it is necessary to consider subsets of the configuration space that must contain multiple features and also allow to exclude some. We propose to compute the number of remaining valid configurations of a partial configuration for such applications which we define as the *cardinality of a partial configuration.* In addition, several applications considered for the cardinality of a feature can be extended for partial configurations. For example, instead of comparing the cardinality of a single feature in order to find priorities, a developer may also consider larger feature sets.

Definition 4 (Cardinality of a Partial Configuration). *Let $FM = (FEATS, CONST)$ be a feature model, $P = (I_P, E_P)$ a partial configuration, and VC the set of valid configurations. Then, the cardinality of P is defined as the number of valid configurations with all features included in P and without all features excluded in P:*
$\#FM_P = |\{C \in VC \mid I_P \subseteq I_C,\ E_P \subseteq E_C,\ I_P \cap E_C = \emptyset,\ E_P \cap I_C = \emptyset\}|$

The cardinality of a partial configuration can be computed by invoking a #SAT solver similarly as for a feature. For each included or excluded feature, we conjoin a corresponding positive or negative literal to the formula representing a feature model. The cardinality of a partial configuration $P = (FM, I, E)$ is $\#(F_{FM} \wedge (\bigwedge_{i \in I} i) \wedge (\bigwedge_{e \in E} \neg e))$. For our running example, there are four valid configurations where `Europe` and `GPSAntenna` are selected and `Manual` is deselected. Table 3 provides an overview of applications based on the cardinality of a partial configuration.

Uniform Random Sampling. While some anomalies can be found for an entire product line (e.g., dead features [10, 57]), a majority of operations for quality assurance, such as testing, only works on specific configurations [4, 42, 66]. Typically, these configurations are computed by sampling. Random configurations are useful as they can be used for representative testing, providing accurate statistical data, and evaluating other sampling methods [15, 48]. However, it is not trivial to create uniformly distributed random configurations. Randomly including or excluding features to create uniformly distributed configurations typically results in a vast majority of invalid configurations [46, 47, 59]. Thus, applying a filter to remove all invalid configurations is generally infeasible [47].

Uniform random sampling generates samples that guarantee true randomness of the included configurations. The idea is to create a one-to-one mapping between integers and the valid configurations of FM [44, 47]. Then, given a random number $r \in [1, \#FM]$, exactly one configuration can be selected. The resulting configurations should all be valid, uniformly distributed, and independent of the order variables are processed [44, 47].

Multiple authors [2, 15, 44, 47, 48, 55] propose algorithms for performing uniform random sampling. All of those rely on computing the cardinality of a partial configuration to decide about the inclusion or exclusion of features. The results of multiple empirical evaluations indicate that uniform random sampling scales for various industrial feature models [29, 48, 55].

Atomic Sets. In Section 5, we described how to use the cardinality of features to find candidates for atomic sets. Here, we extend this procedure to not only compute candidates but actual atomic sets by exploiting the following property. Consider two features f_1, f_2 that share the same commonality $\#FM_f$. If the number of valid configurations that contain both f_1 and f_2 is equal to $\#FM_f$, then f_1 and f_2 are part of the same atomic set. Let $V \subseteq FEATS$ be a set of features. The number of valid configurations that contain V is equivalent to $\#FM_{C_V}$ with $C_V = (FM, V, \emptyset)$.

$$f_i \in V : \#FM_{f_1} = \#FM_{f_2} = \ldots = \#FM_{f_n} = \#FM_{C_V} \Rightarrow \text{V is an atomic set.} \tag{22}$$

In our running example, `Navigation` and `GPSAntenna` appear in 24 valid configurations. Thus, they are potential candidates for an atomic set. Computing $\#FM_{C_V}$ with $V = \{$`Navigation`, `GPSAntenna`$\}$ also results in $\#FM_{C_V} = 24$ valid configurations. Thus, `Navigation` and `GPSAntenna` are part of the same atomic set. The features `Manual` and `Automatic` both are part of 21 valid configurations but $\#FM_{C_V}$ results in zero for $V = \{$`Manual`, `Automatic`$\}$. Therefore, the features are not part of the same atomic set.

Rating Feature Interactions for Sampling. We propose to use the cardinality of a partial configuration to rate feature interactions when sampling. Krieter et al. [37] propose a sampling algorithm that iteratively computes sets of configurations and then removes configurations with low scores. The authors propose to use the number of feature interactions that appear only in this configuration as the configuration's score. The goal is to reach a sample covering a high number of unique interactions after a number of iterations specified by the user. It may be beneficial to add the number of valid configurations that contain the interactions as a criterion for the configuration scores. Consider two configurations C_1, C_2 that both cover one interaction i_1, i_2 that does not appear in any other configuration of the sample. Suppose i_1 appears in more valid configurations. Then, i_1 is more likely to be covered by another configuration in the next iteration. Thus, C_2 should have a higher score than C_1 to increase the probability of covering more unique feature interactions.

$$\#FM_{i_1} > \#FM_{i_2} \Rightarrow score(i_1) < score(i_2) \tag{23}$$

In our running example, the interaction between `Radio` and `Manual` is in 20 of the 42 valid configurations. Thus, when creating samples for testing, it is very likely that this interaction is covered by some configuration of the sample. However, an interaction between `CD` and `DigitalCards` that appears in 8 valid configurations is less likely to be included in the sample. Thus, a configuration that contains the latter interaction should have a higher score.

Interactive Support for Derivation of Configurations. Deriving valid configurations for a feature model is complex because of the constraints imposed by the feature model. It is typically impossible for users to grasp the impact of a feature selection. We propose to interactively display the number of valid configurations for the current selection and the numbers that result from selecting unassigned features. For each currently unselected feature, the number of valid configurations that remain after the selection should be shown next to the feature. On one hand, this information can be used to grasp the impact of a feature selection. On the other hand, unexpected resulting numbers may help to find design flaws.

Summary. In this section, we presented one application from the literature and three own proposals based on the cardinality of a partial configuration opposed to single features or only once for the feature model. The considered applications can be used for sampling, deriving configurations, and to support developers.

7 CONCLUSION & FUTURE WORK

We conclude that applying #SAT technology to feature models has great potential, as we identified a variety of applications that are dependent on counting the number of valid configurations. Thus, applying #SAT for the analysis of feature models is beneficial for several aspects of the development process, such as economical estimations, guidance for developers, detecting design errors, sampling, and gathering statistical inferences. Each considered application depends on one of three metrics, namely the cardinalities of (1) a feature model (i.e., number of overall valid configurations), (2) features (i.e., number of valid configurations containing a specific feature), or (3) partial configurations (i.e., number of valid configurations that include/exclude the features specified in a configuration). Thus, it suffices to develop and optimize algorithms for these queries, to compute results for the 21 applications we considered.

In this work, we only considered naive invocations of #SAT solvers. We expect that optimizing the three considered queries yields performance improvements. Optimizations may exploit the similarity of multiple queries on the same feature model or reuse available information. Another important aspect for the usability of #SAT technology is its scalability. The empirical evaluation in our previous work [59] strongly indicated that computing the cardinality of feature models with off-the-shelf #SAT solvers scales to a vast majority of industrial feature models. The performance of computing the cardinality of features and partial configurations is currently largely unknown for industrial feature models. Therefore, an empirical evaluation on industrial feature models is paramount to make a strong conclusion about the usability of such analyses. There may be large product lines for which no off-the-shelf #SAT solver scales. In this case, evaluating the trade-off of approximate between accuracy and performance of #SAT solvers [3, 12, 27] may provide insights on their benefits.

ACKNOWLEDGMENTS

This work is based on the master's thesis of Sundermann [58].

REFERENCES

[1] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert B. France. 2011. Slicing Feature Models. In *Proc. Int'l Conf. on Automated Software Engineering (ASE)*. IEEE, Washington, DC, USA, 424–427. https://doi.org/10.1109/ASE.2011.6100089

[2] Dimitris Achlioptas, Zayd S Hammoudeh, and Panos Theodoropoulos. 2018. Fast Sampling of Perfectly Uniform Satisfying Assignments. In *Proc. Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*. Springer, Berlin, Heidelberg, 135–147.

[3] Dimitris Achlioptas and Panos Theodoropoulos. 2017. Probabilistic Model Counting With Short XORs. In *Proc. Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*. Springer, Berlin, Heidelberg, 3–19.

[4] Bestoun S. Ahmed, Kamal Z. Zamli, Wasif Afzal, and Miroslav Bures. 2017. Constrained Interaction Testing: A Systematic Literature Study. *IEEE Access* 5 (2017), 25706–25730. https://doi.org/10.1109/ACCESS.2017.2771562

[5] Sofia Ananieva. 2016. Explaining Defects and Identifying Dependencies in Interrelated Feature Models.

[6] Ebrahim Bagheri and Dragan Gasevic. 2011. Assessing the Maintainability of Software Product Line Feature Models Using Structural Metrics. *Software Quality Journal (SQJ)* 19, 3 (2011), 579–612.

[7] Ebrahim Bagheri, Tommaso Di Noia, Dragan Gasevic, and Azzurra Ragone. 2012. Formalizing Interactive Staged Feature Model Configuration. *J. Software: Evolution and Process* 24, 4 (2012), 375–400.

[8] Don Batory. 2005. Feature Models, Grammars, and Propositional Formulas. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. Springer, Berlin, Heidelberg, 7–20.

[9] David Benavides, Antonio Ruiz-Cortés, and Pablo Trinidad. 2005. Automated Reasoning on Feature Models. In *Proc. Int'l Conf. on Advanced Information Systems Engineering (CAiSE)*. Springer, Berlin, Heidelberg, 491–503.

[10] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. 2010. Automated Analysis of Feature Models 20 Years Later: A Literature Review. *Information Systems* 35, 6 (2010), 615–708.

[11] Barry Boehm, A Winsor Brown, Ray Madachy, and Ye Yang. 2004. A Software Product Line Life Cycle Cost Estimation Model. In *Proc. Int'l Symposium on Empirical Software Engineering (ISESE)*. IEEE, Washington, DC, USA, 156–164.

[12] Michele Boreale and Daniele Gorla. 2019. Approximate Model Counting, Sparse XOR Constraints and Minimum Distance. In *The Art of Modelling Computational Systems: A Journey from Logic and Concurrency to Security and Privacy*. Springer, Berlin, Heidelberg, 363–378.

[13] Jan Bosch, Gert Florijn, Danny Greefhorst, Juha Kuusela, J Henk Obbink, and Klaus Pohl. 2001. Variability Issues in Software Product Lines. In *Proc. Int'l Workshop on Software Product-Family Engineering (PFE)*. Springer, Berlin, Heidelberg, 13–21.

[14] Jan Burchard, Tobias Schubert, and Bernd Becker. 2015. Laissez-Faire Caching for Parallel# SAT Solving. In *Proc. Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*. Springer, Berlin, Heidelberg, 46–61.

[15] Supratik Chakraborty, Daniel J Fremont, Kuldeep S Meel, Sanjit A Seshia, and Moshe Y Vardi. 2015. On Parallel Scalable Uniform SAT Witness Generation. In *Proc. Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Springer, Berlin, Heidelberg, 304–319.

[16] Sheng Chen and Martin Erwig. 2011. Optimizing the Product Derivation Process. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. IEEE, Washington, DC, USA, 35–44. https://doi.org/10.1109/SPLC.2011.47

[17] Paul C Clements, John D McGregor, and Sholom G Cohen. 2005. *The Structured Intuitive Model for Product Line Economics (SIMPLE)*. Technical Report. Carnegie-Mellon University, USA.

[18] Sholom Cohen. 2003. *Predicting When Product Line Investment Pays*. Technical Report. Carnegie-Mellon University, USA.

[19] Krzysztof Czarnecki and Chang Hwan Peter Kim. 2005. Cardinality-Based Feature Modeling and Constraints: A Progress Report. In *Proc. Int'l Workshop on Software Factories (SF)*. ACM, San Diego, California, USA, 16–20.

[20] Krzysztof Czarnecki and Andrzej Wąsowski. 2007. Feature Diagrams and Logics: There and Back Again. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. IEEE, Washington, DC, USA, 23–34.

[21] Adnan Darwiche. 2004. New Advances in Compiling Cnf to Decomposable Negation Normal Form. In *Proc. European Conf. on Artificial Intelligence*. IOS press, Amsterdam, Netherlands, 318–322.

[22] Tom DeMarco. 1986. *Controlling Software Projects: Management, Measurement, and Estimates*. Prentice Hall PTR, USA.

[23] David Fernández-Amorós, Ruben Heradio Gil, and José Cerrada Somolinos. 2009. Inferring Information From Feature Diagrams to Product Line Economic Models. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. Springer, Berlin, Heidelberg, 41–50.

[24] David Fernández-Amorós, Ruben Heradio, José Antonio Cerrada, and Carlos Cerrada. 2014. A Scalable Approach to Exact Model and Commonality Counting for Extended Feature Models. *IEEE Trans. on Software Engineering (TSE)* 40, 9 (2014), 895–910.

[25] José A Galindo, Mathieu Acher, Juan Manuel Tirado, Cristian Vidal, Benoit Baudry, and David Benavides. 2016. Exploiting the Enumeration of All Feature Model Configurations: A New Perspective With Distributed Computing. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, New York, NY, USA, 74–78.

[26] Michael Godfrey and Qiang Tu. 2001. Growth, Evolution, and Structural Change in Open Source Software. In *Proc. Int'l Workshop on Principles of Software Evolution*. ACM, New York, NY, USA, 103–106.

[27] Carla P. Gomes, Ashish Sabharwal, and Bart Selman. 2006. Model Counting: A New Strategy for Obtaining Good Bounds. In *Proc. Conf. on Artificial Intelligence (AAAI)*. AAAI Press, 54–61.

[28] Ruben Heradio, David Fernández-Amorós, José Antonio Cerrada, and Ismael Abad. 2013. A Literature Review on Feature Diagram Product Counting and Its Usage in Software Product Line Economic Models. *Int'l J. Software Engineering and Knowledge Engineering (IJSEKE)* 23, 08 (2013), 1177–1204.

[29] Ruben Heradio, David Fernández-Amorós, José Antonio Galindo, and David Benavides. 2020. Uniform and Scalable SAT-Sampling for Configurable Systems. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, New York, NY, USA, 17:1–17:11.

[30] Ruben Heradio, David Fernández-Amorós, Christoph Mayr-Dorn, and Alexander Egyed. 2019. Supporting the statistical analysis of variability models. In *Proc. Int'l Conf. on Software Engineering (ICSE)*. IEEE, Montreal, Quebec, Canada, 843–853.

[31] Ruben Heradio, Hector Perez-Morago, David Fernández-Amorós, Roberto Bean, Francisco Javier Cabrerizo, Carlos Cerrada, and Enrique Herrera-Viedma. 2016. Binary Decision Diagram Algorithms to Perform Hard Analysis Operations on Variability Models. In *Proc. Int'l Conf. on Intelligent Software Methodologies, Tools and Techniques (SOMET)*. IOS Press, 139–154.

[32] Rubén Heradio-Gil, David Fernández-Amorós, José Antonio Cerrada, and Carlos Cerrada. 2011. Supporting Commonality-Based Analysis of Software Product Lines. *IET software* 5, 6 (2011), 496–509.

[33] Ayelet Israeli and Dror G. Feitelson. 2010. The Linux kernel as a case study in software evolution. *J. Systems and Software (JSS)* 83, 3 (2010), 485–501.

[34] David S Johnson. 1992. The NP-Completeness Column: An Ongoing Guide. *J. Algorithms* 13, 3 (1992), 502–524.

[35] Roberto J. Bayardo Jr. and Joseph Daniel Pehoushek. 2000. Counting Models Using Connected Components. In *Proc. Conf. on Artificial Intelligence (AAAI)*. AAAI Press / The MIT Press, 157–162.

[36] Sebastian Krieter, Reimar Schröter, Thomas Thüm, and Gunter Saake. 2016. *An Efficient Algorithm for Feature-Model Slicing*. Technical Report FIN-001-2016. University of Magdeburg, Germany.

[37] Sebastian Krieter, Thomas Thüm, Sandro Schulze, Gunter Saake, and Thomas Leich. 2020. YASA: Yet Another Sampling Algorithm. In *Proc. Int'l Working Conf. on Variability Modelling of Software-Intensive Systems (VaMoS)*. ACM, New York, NY, USA, Article 4, 10 pages. https://doi.org/10.1145/3377024.3377042

[38] Andreas Kübler, Christoph Zengler, and Wolfgang Küchlin. 2010. Model Counting in Product Configuration. In *Proc. Int'l Workshop on Logics for Component Configuration (LoCoCo)*. Open Publishing Association, Waterloo, Australia, 44–53. https://doi.org/10.4204/EPTCS.29.5

[39] Jean-Marie Lagniez and Pierre Marquis. 2017. An Improved Decision-DNNF Compiler. In *Proc. Int'l Joint Conf. on Artificial Intelligence (IJCAI)*. ijcai.org, 667–673.

[40] Rafael Lotufo, Steven She, Thorsten Berger, Krzysztof Czarnecki, and Andrzej Wąsowski. 2010. Evolution of the Linux Kernel Variability Model. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. Springer, Berlin, Heidelberg, 136–150.

[41] Raúl Mazo, Cosmin Dumitrescu, Camille Salinesi, and Daniel Diaz. 2014. Recommendation Heuristics for Improving Product Line Configuration Processes. In *Recommendation Systems in Software Engineering*. Springer, Berlin, Heidelberg, 511–537.

[42] Flávio Medeiros, Christian Kästner, Márcio Ribeiro, Rohit Gheyi, and Sven Apel. 2016. A Comparison of 10 Sampling Algorithms for Configurable Systems. In *Proc. Int'l Conf. on Software Engineering (ICSE)*. ACM, New York, NY, USA, 643–654. https://doi.org/10.1145/2884781.2884793

[43] Marcílio Mendonça, Andrzej Wąsowski, and Krzysztof Czarnecki. 2009. SAT-Based Analysis of Feature Models is Easy. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. Software Engineering Institute, Pittsburgh, PA, USA, 231–240.

[44] Daniel-Jesus Munoz, Jeho Oh, Mónica Pinto, Lidia Fuentes, and Don Batory. 2019. Uniform Random Sampling Product Configurations of Feature Models That Have Numerical Features. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, New York, NY, USA, 289–301. https://doi.org/10.1145/3336294.3336297

[45] Jarley Palmeira Nóbrega, Eduardo Santana de Almeida, and Sílvio Romero Lemos Meira. 2008. Income: Integrated Cost Model for Product Line Engineering. In *Proc. of Euromicro Conf. on Software Engineering and Advanced Applications*. IEEE, Washington, DC, USA, 27–34.

[46] Jeho Oh, Don Batory, Margaret Myers, and Norbert Siegmund. 2016. *Finding Product Line Configurations With High Performance by Random Sampling*. Technical Report. University of Texas at Austin, USA.

[47] Jeho Oh, Don Batory, Margaret Myers, and Norbert Siegmund. 2017. Finding Near-Optimal Configurations in Product Lines by Random Sampling. In *Proc. Int'l Symposium on Foundations of Software Engineering (FSE)*. ACM, 61–71. https://doi.org/10.1145/3106237.3106273

[48] Jeho Oh, Paul Gazzillo, Don Batory, Marijn Heule, and Maggie Myers. 2019. *Uniform Sampling from Kconfig Feature Models.* Technical Report TR-19-02. The University of Texas at Austin, Department of Computer Science.

[49] Joaquin Peña, Michael G Hinchey, and Antonio Ruiz-Cortés. 2006. Building the Core Architecture of a Multiagent System Product Line: With an example from a future NASA Mission. In *Proc. Int'l Workshop on Agent-Oriented Software Engineering (AOSE)*. Springer, Berlin, Heidelberg.

[50] Héctor José Pérez Morago. 2016. *BDD Algorithms to Perform Hard Analysis Operations on Variability Models.* Ph.D. Dissertation. Universidad Nacional de Educación a Distancia (UNED), Spain.

[51] Gilles Perrouin, Sagar Sen, Jacques Klein, Benoit Baudry, and Yves Le Traon. 2010. Automated and Scalable T-Wise Test Case Generation Strategies for Software Product Lines. In *Proc. Int'l Conf. on Software Testing, Verification and Validation (ICST)*. IEEE, Washington, DC, USA, 459–468. https://doi.org/10.1109/ICST.2010.43

[52] Richard Pohl, Kim Lauenroth, and Klaus Pohl. 2011. A Performance Comparison of Contemporary Algorithmic Approaches for Automated Analysis Operations on Feature Models. In *Proc. Int'l Conf. on Automated Software Engineering (ASE)*. IEEE, Washington, DC, USA, 313–322. https://doi.org/10.1109/ASE.2011.6100068

[53] Reimar Schröter, Sebastian Krieter, Thomas Thüm, Fabian Benduhn, and Gunter Saake. 2016. Feature-Model Interfaces: The Highway to Compositional Analyses of Highly-Configurable Systems. In *Proc. Int'l Conf. on Software Engineering (ICSE)*. ACM, New York, NY, USA, 667–678. https://doi.org/10.1145/2884781.2884823

[54] Sergio Segura. 2008. Automated Analysis of Feature Models Using Atomic Sets. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*, Vol. 2. 201–207.

[55] Shubham Sharma, Rahul Gupta, Subhajit Roy, and Kuldeep S Meel. 2018. Knowledge Compilation Meets Uniform Sampling. In *Proc. Int'l Conf. on Logic for Programming, Artificial Intelligence, and Reasoning*. EasyChair, 620–636.

[56] Stefan Sobernig, Sven Apel, Sergiy Kolesnikov, and Norbert Siegmund. 2016. Quantifying Structural Attributes of System Decompositions in 28 Feature-Oriented Software Product Lines. *Empirical Software Engineering (EMSE)* 21, 4 (Aug. 2016), 1670–1705. https://doi.org/10.1007/s10664-014-9336-6

[57] Joshua Sprey, Chico Sundermann, Sebastian Krieter, Michael Nieke, Jacopo Mauro, Thomas Thüm, and Ina Schaefer. 2020. SMT-Based Variability Analyses in FeatureIDE. In *Proc. Int'l Working Conf. on Variability Modelling of Software-Intensive Systems (VaMoS)*. ACM, New York, NY, USA, Article 6, 9 pages. https://doi.org/10.1145/3377024.3377036

[58] Chico Sundermann. 2020. *Applications of #SAT Solvers on Product Lines.* Master's thesis. TU Braunschweig, Germany. https://doi.org/10.24355/dbbs.084-202009161329-0

[59] Chico Sundermann, Thomas Thüm, and Ina Schaefer. 2020. Evaluating #SAT Solvers on Industrial Feature Models. In *Proc. Int'l Working Conf. on Variability Modelling of Software-Intensive Systems (VaMoS)*. ACM, New York, NY, USA, Article 3, 9 pages. https://doi.org/10.1145/3377024.3377025

[60] Thomas Thüm, Don Batory, and Christian Kästner. 2009. Reasoning about Edits to Feature Models. In *Proc. Int'l Conf. on Software Engineering (ICSE)*. IEEE, Washington, DC, USA, 254–264. https://doi.org/10.1109/ICSE.2009.5070526

[61] Thomas Thüm, Christian Kästner, Sebastian Erdweg, and Norbert Siegmund. 2011. Abstract Features in Feature Modeling. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. IEEE, Washington, DC, USA, 191–200. https://doi.org/10.1109/SPLC.2011.53

[62] Marc Thurley. 2006. sharpSAT - Counting Models with Advanced Component Caching and Implicit BCP. In *Proc. Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*. Springer, Berlin, Heidelberg, 424–429.

[63] Takahisa Toda and Takehide Soh. 2016. Implementing Efficient All Solutions SAT Solvers. *J. Experimental Algorithmics* 21 (2016), 1–12.

[64] Pablo Trinidad, David Benavides, and Antonio Ruiz Cortés. 2006. Isolated Features Detection in Feature Models. In *Proc. Int'l Conf. on Advanced Information Systems Engineering (CAiSE)*. Springer, Berlin, Heidelberg.

[65] Leslie G Valiant. 1979. The Complexity of Enumeration and Reliability Problems. *SIAM J. Computing* 8, 3 (1979), 410–421.

[66] Mahsa Varshosaz, Mustafa Al-Hajjaji, Thomas Thüm, Tobias Runge, Mohammad Reza Mousavi, and Ina Schaefer. 2018. A Classification of Product Sampling for Software Product Lines. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, New York, NY, USA, 1–13. https://doi.org/10.1145/3233027.3233035