Monte Carlo Simulations for Variability Analyses in Highly Configurable Systems

Jose-Miguel Horcas and **A. Germán Márquez** and **José A. Galindo** and **David Benavides**[1]

Abstract. Highly configurable systems expose numerous variation points to be configured by the stakeholders. Deciding which variant to select for a given variation point is hard to know a priori because each variant affects the configuration properties (*e.g.,* performance, efficiency, fault tolerance) differently, and evaluating all configurations is practically infeasible. This paper introduces an approach based on Monte Carlo simulations to analyze the influence of each feature selection when configuring a variability model. We split the whole configuration space into step-wise decisions driven by the variation points that a user normally needs to face/decide during the configuration process. Monte Carlo simulations approximate the influence of each feature variant evaluating as few configurations as possible. Our solution complements existing sampling techniques to analyze colossal configuration spaces improving the understanding of the influence of each feature selection. It can be part of a decision-making tool to assist the user by means of recommendation systems and interactive configuration processes.

1 INTRODUCTION

A Software Product Line (SPL) [1] defines a set of common and variable characteristics that can be customized according to the specific needs of the stakeholders in a particular application. To represent and model those commonalities and variabilities, feature models [25] stand out as the de-facto standard in SPL. The automated analysis of feature models (AAFM) [6, 11] is challenging due to the large number of variants and configurations.

A feature model represents the variation points of an application and its possible variants in terms of features, which can be optional, mandatory, required or excluded by other features. For example, the persistence feature is a recurrent variation point in many systems requiring to select a specific database variant (*e.g.,* SQL, MongoDB) to be used in the target application. Deciding which variant to select depends on several factors such as the user's requirements, performance, energy efficiency, or fault tolerance of the complete configuration. The analysis of how each variant influences the application properties (*e.g.,* performance, efficiency, fault tolerance) is hard to know because of the high number of possible alternatives for a specific feature, logical constraints between the features, and interactions among them. Evaluating all configurations for highly configurable systems is practically infeasible because the number of configurations grows exponentially in terms of the number of features.

Configuration sampling [38] is a technique used to avoid exhaustive analysis, providing a subset of all valid configurations. Several sampling strategies have been proposed in the literature about SPLs and configurable systems [45]: random sampling [18, 30, 33] to select configurations uniformly; coverage-based sampling [9, 10, 34] to select configurations that cover all combinations of t selected features (*e.g.,* pair-wise sampling for $t = 2$); or distance-based sampling [24] to select configurations according to a given probability distribution and a distance metric; among other techniques reviewed in [45]. Despite the fact that these techniques have shown great results in SPL testing [13] and learning configuration spaces [35], sampling large-scale SPLs poses a challenge [38]. Existing techniques present scalability issues as they tend to run out of memory, do not terminate, or produce samples which are too large to be tested [38]. In addition, analyzing and making decisions from a sample of configurations that considers the whole configuration space can be difficult for the user that configures a product. For instance, the performance or efficiency of the database variant (*e.g.,* SQL) used in a system's configuration is not influenced, in most cases, by the back end technology (*e.g.,* Gradle, Maven) used to build the application. Thus, analyzing the feature interactions (*e.g.,* by using coverage-based sampling) between both variation points may be useless for the user configuring the product. Moreover, from the analysis of a particular complete configuration, it is challenging to comprehend a priori the influence of each feature variant in such configuration and in the rest of configurations of the SPL [34].

In this paper, we propose an approach based on Monte Carlo simulations to analyze the influence of the variation points and each of their variants in the configuration properties (*e.g.,* performance, fault tolerance). Our approach relies on statistical analysis [19] and random sampling [18] to analyze large-scale configuration spaces improving the understanding of the influence of each feature selection. We split the whole configuration space into step-wise decisions driven by the variation points that a user normally needs to face and decide during the configuration process. Monte Carlo simulations approximate the influence of each feature variant evaluating as few configurations as possible. We demonstrate the applicability of our approach with a large-scale real-world SPL: the JHipster Web development stack [16], the configuration space of which has been exhaustively evaluated in [16] serving us as an appropriate candidate to verify the results of the Monte Carlo simulations. Our approach can be part of a decision-making tool to assist the user in the process of configuring a feature model improving the recommendation systems and interactive configuration processes of the current state-of-the-art.

The paper is organized as follows. Section 2 motivates our approach by discussing related work and its limitations. Section 3 describes our approach to analyze feature models using Monte Carlo simulations. In Section 4, we apply our approach to the jHipster SPL. Finally, Section 5 concludes the paper and presents future work.

[1] University of Seville, Spain, email: jhorcas@us.es, antmartru@alum.us.es, jagalindo@us.es, benavides@us.es

2 RELATED WORK AND MOTIVATION

This section reviews related work about product configuration of feature models and evaluation of configurations. We also expose its limitations and compare it with our approach.

2.1 Product configuration of feature models

Product configuration is a decision-making process involving selecting a valid combination of features from a feature model [37]. This process takes place during the application engineering phase of an SPL where the stakeholder (the application engineer) elicits the product's requirements of a particular application and derives a concrete product satisfying those requirements [36]. There exist many mechanisms to support the product configuration of large and complex SPLs by means of recommender systems [29, 41], interactive processes [5], automated propagation strategies [2, 46], optimization of quality requirements [15, 22, 26], or visualization techniques [32, 36]. These systems present a collection of heuristics to prioritize choices and recommend candidate features to be configured [29]. They guide the users through the product configuration using decision models [5], proposing a feature ranking approach to support decision makers [4, 41, 42], predicting the configuration likability based on users' votes [28] or questionaries [12], or ordering the selection of the features [32, 36, 37].

Limitations. Automated configuration systems like those based on propagation strategies [29, 46] or search-based optimization techniques [15, 26] do not propose any mechanism to guide the users choosing among the candidate features. Also, most of the recommender systems that provide user assistance can be overwhelming to users due to the amount and complexity of options presented by the configurator [12, 28]. Moreover, interactive assistants [5] are constantly asking the user to choose between two competing features, even when the user is not interested in a particular decision. Finally, without the user's input, those algorithms usually make poor decisions [5] because, they usually rely on historical data from previous users' configurations to make decisions [28].

Our approach. *We propose to split the whole configuration space into step-wise decisions driven by the variation points of the feature model that require to make a choice (*i.e., *select a variant). So that, the user can focus on a particular and concise decision at a specific time.* However, variation points of the applications usually have dependencies between them, so the selection of a variant can affect others variation points. To infer the influence of each feature selection and assist the user when making a decision, we have to evaluate the influence of each feature selection in the configuration properties of interest such as performance, energy efficiency, or defects and faults present in the configurations.

2.2 Evaluation of configurations

Evaluating all configurations to study the influence of each feature in the configuration properties (*e.g.,* performance) is not feasible for large-scale feature models. Researchers often rely on configuration sampling techniques [38] to obtain a sample of configurations, evaluate them and make predictions of the whole configuration space [35]. Many sampling strategies have been proposed in the literature for SPL configuration, most of them taking as input a feature model [45]. Schaefer et al. [45] review product sampling for SPL and provide a classification of techniques. Most used sampling techniques include random sampling [18, 30, 33] that aims to cover the configuration space uniformly; solver-based sampling [17] that relies on SAT and constraint solvers; feature coverage-based sampling [9, 10, 34] that aims to optimize the sample with regards to a coverage criterion, as for example the pair-wise sampling to consider every interaction between two features; or distance-based sampling [24] that covers the configuration space according to a given probability distribution and a distance metric.

Limitations. On the one hand, sampling techniques present scalability issues when dealing with large-scale SPLs [38]. They usually run out of memory, take long execution times, or produce samples which are too large to be analyzed. There even exist a specific challenge to cope with those limitations [38]. On the other hand, analyzing and making decisions from a sample of configurations that considers the whole configuration space is challenging [35]. For example, from the analysis of a particular complete configuration randomly obtained, it is difficult to comprehend a priori how each feature variant influence the configuration properties. Moreover, pairwise sampling [34] consider every two feature interactions in the feature model, but not all interactions are of user's interest when configuring a product.

Our approach. *We propose to use Monte Carlo simulations [27] to approximate the influence of each feature variant in the configuration properties (*e.g., *performance) evaluating as few configurations as possible.* Monte Carlo simulations [27] is a randomness model used to predict the probability of different outcomes in a process that cannot easily be analyzed because of the intervention of complex inter-related variables. The basis of Monte Carlo simulations involve constantly repeating an experiment many times (*e.g.,* a random sampling in our case) to approximate the expected results (*e.g.,* the configuration properties) using the *law of large numbers* theorem [14] and other methods of statistical inference [27]. *As we split the configuration space into individual variation points, a Monte Carlo simulation in our approach will be a uniform random sampling considering only the variants of a specific variation point. We use statistical analysis of feature models [19] to determine the number of simulations/evaluations to be performed for each feature variant. As results, our approach provides an estimation of the influence of each feature in the complete configuration to assist the user when configuring a product.*

Monte Carlo simulations have been used in many real-world problems with great success [27], including combinatorial optimization [43], constraint satisfaction problems (CSP) [3] and boolean satisfiability (SAT) [40], model checking [39], scheduling problems [31], and feature selection problems [7]. Typical uses of Monte Carlo simulations are (i) *sampling* to gather information about a random object by observing many realizations of it (*e.g.,* simulation of a system's behavior [27]); (ii) *estimation* of certain numerical quantities (*e.g.,* the expected throughput in an SPL [20]); and (iii) *optimization* of complicated objective functions (*e.g.,* to improve a search-based technique [26]).

3 SIMULATION-BASED ANALYSIS OF FEATURE MODELS

Figure 1 overviews our approach based on Monte Carlo simulations to analyze the influence of each feature selection when configuring a

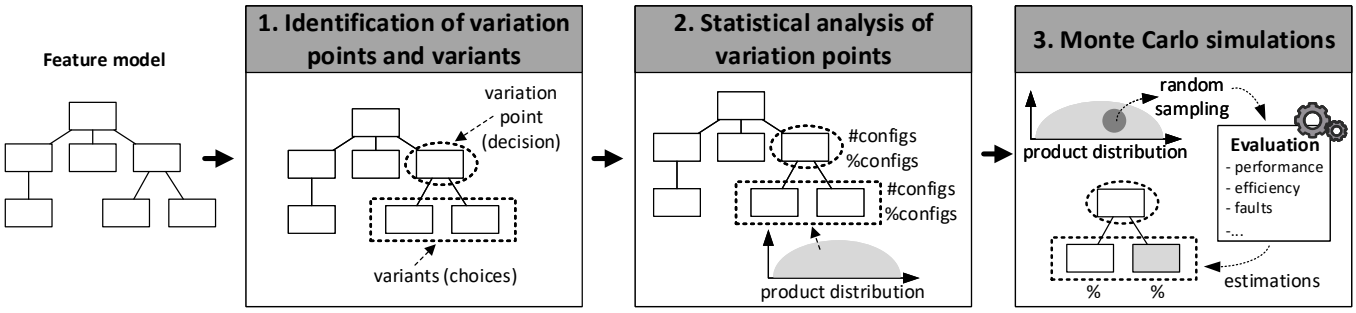

Figure 1: Overview of our step-wise decisions analysis of feature models based on Monte Carlo simulations and driven by the variation points.

feature model. Our approach is divided in three steps detailed in the following subsections:

1. **Identification of variation points and variants.** To overcome the scalability issues [38] discussed in the previous section, we split the whole configuration space into step-wise decisions driven by the variation points of the feature model that require to make a choice (*i.e.,* select a variant). So that, the user can focus on a particular and concise decision (*i.e.,* a variation point) at a specific time. Therefore, the first step in our approach consists in identifying the variation points, their variants, and the possible variant combinations given by the variability type of the variation point (Section 3.1).
2. **Statistical analysis of variation points.** The second step of our approach is characterizing the complexity of every decision that the user needs to face. To do that, we rely on the statistical analysis [19] of the feature model, and specifically on the product distribution of each variation point. The product distribution provides us an idea about the complexity of each decision (variation point) in terms of the number of products or configurations in which that variation point, and each of its variant respectively, are involved (Section 3.2).
3. **Monte Carlo simulations.** The statistical analysis is useful to determine the number of Monte Carlo simulations to be performed in the last step of our approach. It consists on taking a small size random sampling of configurations for a specific variant of a given variation point, and evaluating those configurations by measuring the non-functional properties (*e.g.,* performance, energy efficiency, faults, etc.). We then average the results and obtain an approximation of the influence of the feature variant. We repeat this process for each possible variant combination of the variation point, so that the user can obtain an estimation about how each feature variant affects its configuration (Section 3.3).

3.1 Variation points and variants

The first step in our approach consists in identifying (1) the variation points (*i.e.,* the decisions), (2) its variants (*i.e.,* the choices), and also (3) the possible variant combinations of each variation point (*i.e.,* the allowed choice combinations according to the variability type). Variation points are represented by those features that require to make a decision in order to obtain a valid configuration of the feature model. The variants of a variation point are represented by direct sub-features (*i.e.,* the possible choices), which may also represent other variation points. Finally, the possible variant combinations of a variation point are the distinct choice combinations of the variants that are allowed to select in a configuration of the feature model, and it depends on the variability type of the features as well as on the

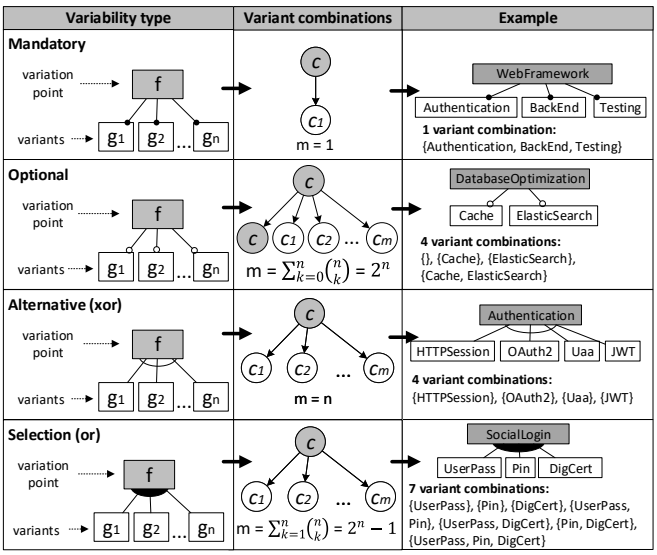

Figure 2: Variations points, variants, and possible variant combinations, according to the feature variability type.

dependencies between features. Figure 2 illustrates these concepts for each variability type:

Mandatory features. For variation points whose variants are mandatory features, there is only one variant combination which is the result of selecting all the mandatory features. So, the user does not have to make any decision as all mandatory features need to be selected in any configuration where that variation point is selected. For example, the `WebFramework` variation point requires to select each of its variants `Authentication`, `BackEnd`, and `Testing`, and thus, there is only one possible variant combination including the three variants (see third column in Figure 2).

Optional features. For variation points whose variants are optional features, the user may choose to include or not include each of the variant, leading to a total of $\sum_{k=0}^{n}\binom{n}{k} = 2^n$ variant combinations, where n is the number of variants (sub-features) and k goes from 0 (none variant is selected) to n (all variants are selected). $\binom{n}{k}$ is the binomial coefficient (read as "n choose k") indicating the ways to choose an unordered subset of k variants from the n available options. For instance, for the `DatabaseOptimization` variation point with 2 optional variants (`Cache` and `ElasticSearch`), the user has to decide between four possible combinations: (1) none of the variants is selected, (2) only `Cache` is selected, (3), only `ElasticSearch` is selected, and (4) both `Cache` and `ElasticSearch` are selected. The variant combinations grows exponentially, for 5 vari-

ants there are 32 possible combinations, while for 7 variants there are 128 combinations.

Variation points that contains both mandatory and optional variants can be seen as if they only contain optional variants because the mandatory features will be always selected in every variant, and the user only needs to take care of the optional features.

Alternative (xor) group features. Variation points represented by an alternative-group feature requires to select only one of its n variants. Thus, there are n possible variant combinations to be decided by the user. For instance, the `Authentication` variation point offers four variants: `HTTPSession`, `OAuth2`, `Uaa`, and `JWT`; and only one of them is allowed to be selected in a valid configuration, leading to four possible variant combinations.

Selection (or) group features. Variation points represented by an or-group feature requires to select at least one of its variants and at most n. Similarly to optional features (but requiring at least one variant), there are $\sum_{k=1}^{n} \binom{n}{k} = 2^n - 1$ possible variant combinations to be decided. For instance, the `SocialLogin` variation point allows to select between three variants: `UserPass`, `Pin`, or `DigCert`; and the user can select any of them, leading to a total of 7 variant combinations (see third column in Figure 2).

Note that in the feature model literature, there exist models that allow *multiple decomposition type* for features [8]. That is, a variation point may have more than one variability type — *e.g.,* optional and mandatory sub-features together with one or more xor or selection groups under the same feature. Czarnecki and Eisenecker [8] describe a normalization strategy for such models by introducing additional abstract features. This does not affect our approach, as it only increases the number of variation points and simplifies the variant combinations by mapping them to the explained cases in Figure 2. Moreover, current existing tools for modeling and managing feature models do not support complex decomposition types [23]. Note also that, at this step, we have not considered the cross tree constraints that may exist between variants of a variation point, which may reduce the number of possible choices. Cross tree constraints are considered in the following step where we analyze the valid configurations for each decision.

3.2 Statistical analysis of variation points

For each variation point identified in the previous step, we calculate the product distribution of its variant combinations. The product distribution is defined by Heradio et al. [19] as the number of valid products in the whole configuration space having a given number of features. Thus, in this research we define the product distribution as the number of valid products (or configurations) having the features of a variant combination.

The product distribution provides us a characterization of the complexity of the whole configuration space. It allows us to determine the number of configurations containing a specific variation point, as well as the number of configurations containing each of the variants and variant combinations for a variation point. The product distribution will depend on the cross tree constraints defined in the feature model. For a given variation point, all its variants will have the same probability of being part of a configuration if there are not cross tree constraints involving those variants. In other case, there will be variants present in more configurations than others. To compute the product distribution it is necessary either to enumerate all configurations of the feature model, or to encode the feature model as a binary decision diagram (BDD) [21]. The former is infeasible for large configuration spaces. In contrast, using a BDD has been demonstrated to scale for large feature models [21, 44]. Then, the product distribution can be computed by traversing the BDD and accounting for how many configurations have the specified variant features [19].

In our approach, we use the product distribution to determine the number of Monte Carlo simulations that will be performed for each variant combination in order to evaluate as few as possible products.

3.3 Monte Carlo simulations

The basis of Monte Carlo simulations involve constantly repeating an experiment many times to approximate the expected results [27]: First, a Monte Carlo simulation takes the variable with uncertainty and assigns it a random value. Then, the model is evaluated and a result is provided. This process is repeated a number of times (simulations) while assigning the variable in question with many different values. Once all simulations have finished, the results are averaged together to provide an estimation.

In our approach, a Monte Carlo simulation consists in a uniform random sampling of configurations [18], where the variable with uncertainty is the feature variant being analyzing, and the assignment of a random value to that variable corresponds with the evaluation of a sample of configurations containing that feature variant. The sample of configurations are evaluated by measuring their properties (*e.g.,* performance, memory consumption, number of defects,...), and the obtained results are averaged obtaining an estimation of the feature expected influence. To simplify the implementation of this process, we consider a Monte Carlo simulation as an individual configuration randomly sampled and evaluated. Therefore, performing N simulations corresponds with a random sampling of N configurations.

The number of simulations to be performed can be defined according to several criteria such as a fixed number, a percentage of configurations, or a computational budget (*e.g.,* time or memory). In this paper, we use the product distribution calculated in the statistical analysis to determine the number of simulations. We specify a shared percentage of configurations to be sampled for all variants, so that the same ratio of configurations are evaluated for each variant. Then, the number of simulations will depends on the number of configurations containing a specific feature variant. Note that we can define the variable (feature) of interested to be analyzed at any level: (1) at the variation point level to analyze it independently of its variants, (2) to the variant level to study the influence of each variant, and (3) to the variant combinations to analyze the interaction between the different variants. The latter can be seen as a t-wise coverage sampling considering only the variant features of a specific variation point.

Next section illustrates our approach with a real-world SPL.

4 APPLICABILITY OF OUR APPROACH

To show the applicability of our approach we use the JHipster Web development stack [16]. The jHipster SPL (Figure 3) is a popular code generator for web applications with 45 features which lead a total of 26,256 configurations. We choose jHipster because its configuration space has been exhaustively evaluated in [16] to find those configurations that present errors or defects when they are deployed (*i.e.,* configurations that make the application fails). Therefore, we can use it to compare and verify the results of the Monte Carlo simulations. In this SPL, we focus on evaluating the configurations to find whether they provoke an error or some other misbehavior when they are deployed, but our approach can be used to evaluate any other configuration property like performance or energy efficiency. Complete

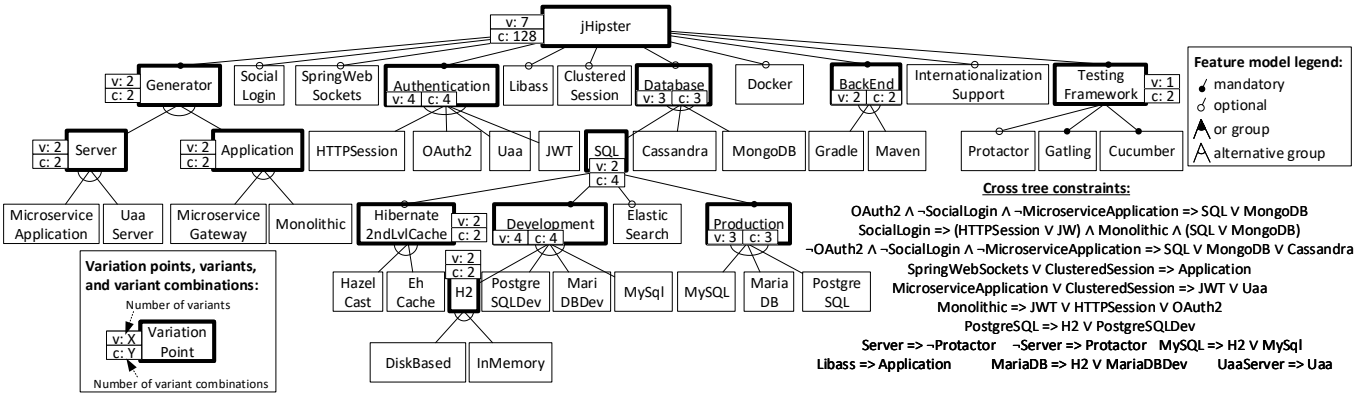

Figure 3: Feature model of the jHipster product line with the variation points highlighted.

analysis results can be found online[2].

Identifying variation points, variants, and variant combinations. Following our approach, we first identify the variation points in the jHipster feature model, as well as the variants of each variation point and the possible variant combinations. Figure 3 presents the jHipster feature model where we have identified 13 variation points (highlighted features). The variants of each variation points are their direct sub-features. For instance, the `Database` variation point is a xor-group feature with three alternative variants: `SQL`, `Cassandra`, and `MongoDB`, being the `SQL` variant also a variation point itself. For each variation point we explicitly show the number of variants and the number of possible variant combinations. This gives us an idea about the number of decisions and options the user needs to face in order to configure a valid product. For example, to configure the `Database` variation point the user needs to make, at least, one decision with three possible choices (`SQL`, `Cassandra`, `MongoDB`), but if the `SQL` variant is selected the user needs to face, in addition, three more decisions (and possibly more): the variants of the `SQL` variation point, and the two mandatory variation points that require to select a database variant to be used in `Development` and `Production`, respectively. Left-hand side of Table 1 details these decisions that the user needs to consider. For space reasons, we only show 5 illustrative variation points of the jHipster feature model. The complete analysis of all variation points can be found online.

Statistical analysis of variation points. Along with the variation points, Table 1 shows the product distribution of each variant combination. That is, the number of configurations in the feature model containing such variant features and the percentage of configurations that it represents with regard the whole configuration space. We can observe that for the `Database` variation point, there are 25,488 valid configurations with the `SQL` variant (that means 97.13% of all configurations of the feature model) in contrast to 280 valid configurations with the `Cassandra` variant (0.01%), and `472` with MongoDB (0.02%). This evidences that applying a uniform random sampling over the whole configuration space with lead to the majority of the resulting configurations to have selected the `SQL` variant, to the detriment of the other alternative variants (`Cassandra` or `MongoDB`). Therefore analyzing those random configurations does not provide enough insight about the influence of each variant in a configuration. To avoid that, our approach considers the product distribution of each variant in order to calculate the number of Monte Carlo simulations, and thus, obtain the configurations to be sampled and evaluated.

Monte Carlo simulations. Middle of Table 1 summarizes the results of the Monte Carlo simulations with the number of configurations sampled and evaluated for each variant combination, and the median, mean, standard deviation, and percentage of defective configurations found in the sample. In all cases, we set the number of simulations to be performed as 1% of the configurations of each variant combination.

We can observe in Table 1 that for the `Authentication` variation point, the `JWT` variant (highlighted) is the less likely to lead to a configuration with defects (19.64%), in contrast to the `Uaa` variant which is the most likely to achieve a defective configuration (92.22%). So, the user will prefer to select the JWT variant when configuring a product that does not provoke errors. Similarly, the `SQL` variant for the `Database` variation point is the most safety variant with 34.31% of defective configurations, being also the variant that appears in most configurations: 25,488 (97.13% of the whole configuration space). In contrast, `Cassandra` and `MongoDB` appears in a few number of configurations: 280 (0.02%), and 472 (0.02%), respectively; but their configurations are more likely to present defects (66.67% and 40% respectively). The `SQL` variant also provides two optional features to be decided (`ElasticSearch` and `Hibernate2ndLvlCache`) and two more xor-group variation points (`Development` and `Production`). In the former, the solely selection of the `ElasticSearch` provides the combination of variants that lead to less defective configurations (32.56%) versus the decision of selecting the `Hibernate2ndLvlCache` variant (36.47%), selecting both variants together (33.53%) or not selecting any variant at all (`None` with 38.37% defective configurations). For `Development` and `Production`, `MySQL` and `PostgreSQL` are the most desirable variants with similar probability of achieving a configuration with defect. Figure 4 summarizes the influence of each variant feature regarding configurations with defects for every variation point identified in the jHipster SPL. A broader feature indicates a major influence, that is, a higher probability of leading to a configuration with defect.

With this information, we can provide a decision-making tool to assist the user in the process of configuring a feature model or to assist the developer when testing and maintaining the SPL (see Figure 5). The expected probabilities obtained with Monte Carlo simulations help the stakeholder to make decisions for each variation point. On the one hand, the user (*e.g.,* an application engineer) is aware about the influence of each feature decision in the configura-

[2] https://github.com/diverso-lab/montecarlo_analysis

Table 1: Simulation-based analysis of the variation points in the jHipster product line.

	Statistical analysis			Monte Carlo simulations					Real probability
				Defective configurations					
VP (type)	Variant combination	#Conf	%Conf	#Sim	Med	Mean	Std	%Conf	Defective conf.
Authentication (xor-group)	HHTPSession	7104	27.06%	72	17.0	16.5	3.7	23.61%	1586 (22.33%)
	OAth2	3520	13.41%	36	12.5	13.3	3.4	34.72%	1397 (39.69%)
	Uaa	4488	17.09%	45	41.5	40.8	2.7	92.22%	4114 (91.67%)
	JWT	11144	42.44%	112	22.0	22.1	4.3	19.64%	2279 (20.45%)
Database (xor-group)	SQL	25488	97.13%	255	87.5	89.2	9.7	34.31%	8992 (35.28%)
	Cassandra	280	0.01%	3	2.0	2.0	0.8	66.67%	176 (62.86%)
	MongoDB	472	0.02%	5	2.0	2.2	1.3	40.00%	200 (42.37%)
SQL (optional)	None	4248	16.67%	43	16.5	16.3	2.7	38.37%	1607 (37.83%)
	Hibernate2ndLvlCache	8496	33.33%	85	31.0	31.9	4.8	36.47%	3218 (37.88%)
	ElasticSearch	4248	16.67%	43	14.0	14.4	2.8	32.56%	1390 (32.72%)
	Hibernate2ndLvlCache, ElasticSearch	8496	33.33%	85	28.5	28.6	5.0	33.53%	2777 (32.69%)
Development (xor-group)	H2	16992	66.67%	170	60.5	59.7	5.9	35.59%	5970 (35.13%)
	PostgreSQLDev	2832	11.11%	29	6.0	5.9	2.2	20.69%	551 (19.46%)
	MariaDBDev	2832	11.11%	29	19.0	19.5	2.0	65.52%	1913 (67.55%)
	MySql	2832	11.11%	29	6.0	5.8	1.6	20.69%	558 (19.70%)
Production (xor-group)	MySQL	8496	33.33%	85	15.0	15.7	4.0	17.65%	1660 (19.54%)
	MariaDB	8496	33.33%	85	55.5	56.0	4.9	65.29%	5708 (67.18%)
	PostgreSQL	8496	33.33%	85	16.5	16.7	3.0	19.41%	1624 (19.11%)

For each variation point (VP), we show the configuration distribution of its variants: number of configurations (#Conf) and its percentage (%Conf) *wrt.* the whole configuration space. For each variant, we perform a number of simulations (#Sim) corresponding to a uniform random sampling with sample sizes of 1% *wrt.* the product distribution (number of configurations) of the variant. We show the median, mean and standard deviation of defective configurations found in the simulations, as well as the percentage (%Conf) using the median. To calculate the statistics we performed 30 executions. Finally, we show real total values of defective configurations in the whole configuration space of the jHipster feature model to verify the results of the simulations.

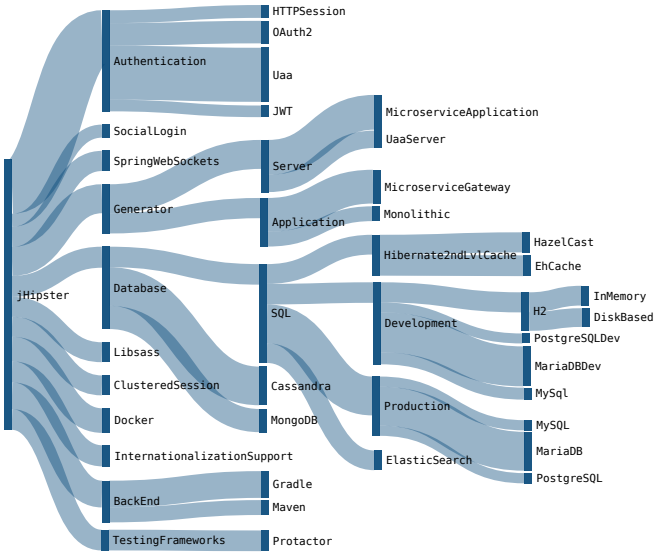

Figure 4: Influence of the variant features in each variation point in the jHipster SPL considering configurations with defects.

tion properties (*e.g.,* configurations with defects in this case), and can use this knowledge to configure a product with a lower probability of contain a defect. On the other hand, the developer of the SPL can easily identify the problematic features and act accordingly by changing their implementation or modify the feature model by updating the feature relationships to avoid the defective configurations. Figure 5 shows the estimated probabilities for achieving configurations with faults in the jHipster feature model, and a valid selection (shaded features) for the most probable configuration without defects.

Verifying the results. To verify our results, we compare them with the real probability of finding defective configurations (right-hand side of Table 1). We calculate the real probability of finding defective configurations by considering all configurations for each variation point and variant combination. In order to be able to obtain the real values, the whole configuration space needs to be evaluated as Halin et al [16] did for the jHipster SPL. However, this is not feasible for large-scale feature models, and in those cases our approach can provide robust approximations to the real probabilities (check the small standard deviation of the results in Table 1).

As observed in Table 1, only sampling and evaluating 1% of the configurations we approximate to the real probability of finding defective configurations with less than 1% of error in most cases. We can improve the approximations of the Monte Carlo simulations by increasing the number of simulations. Figure 6 illustrates how the Monte Carlo simulations approximate to the real probability as the number of simulations increases. Considering the complete jHipster feature model which contains 9,376 defective configurations from a total of 26,256 (35.71% of defective configurations), a sample of 1% (260 configurations) leads to 95 defective configurations (36.54%) with an error of 0.0083, while a sample of 20% (5,000 configurations) lead to a total of 1,787 defective configurations (35.74%) with an error of 0.0003. We can conclude that using samples of 1% for the Monte Carlo simulations are enough to approximate the probability of finding defective configurations in the jHipster feature model.

Threats to validity. There are two main threats, first, population validity, as we only have tested it with the jHipster dataset. However, we aim at demonstrating that this approach is feasible and are planning to perform a more exhaustive evaluation in future work. While the external validity, in general, focuses on the generalization of the results to other contexts (*e.g.,* using other models), the ecological va-

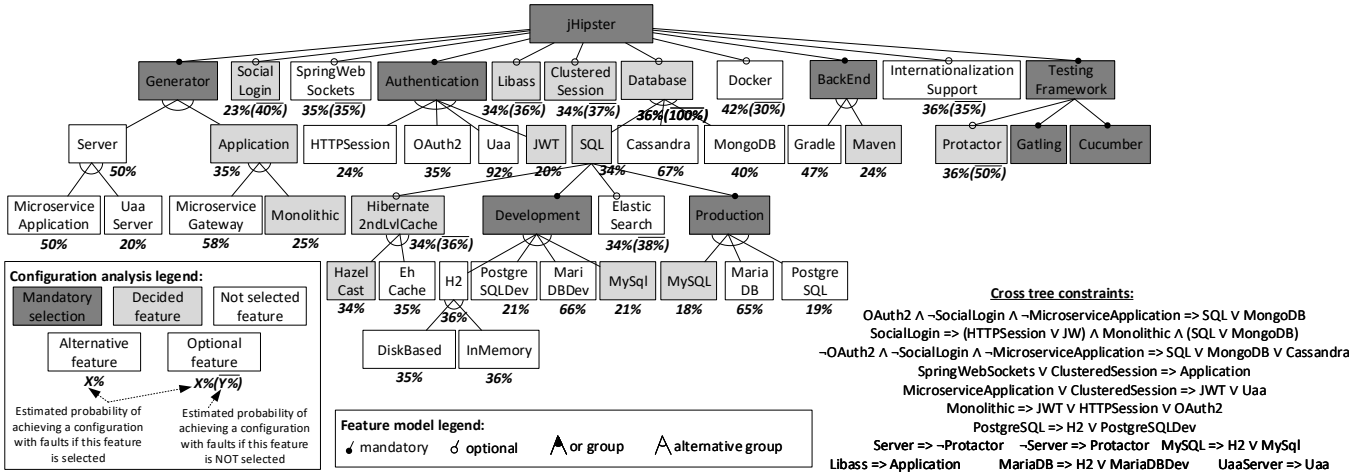

Figure 5: Estimated probabilities for achieving configurations with faults in the jHipster feature model, and a valid selection (shaded features) for the most probable configuration without defects.

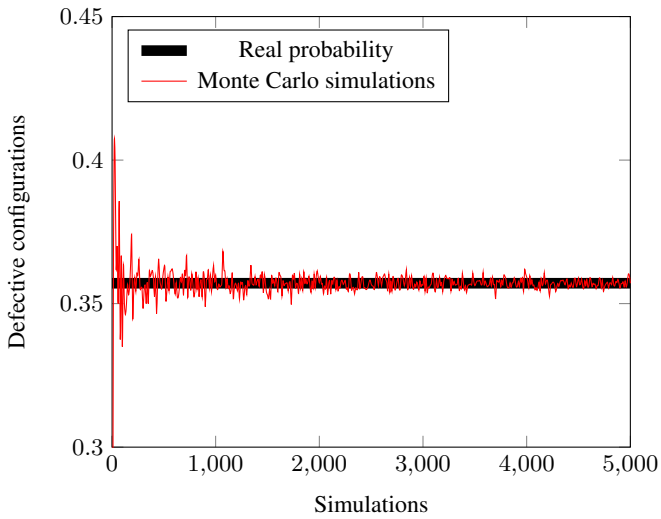

Figure 6: Approximation of the Monte Carlo simulations to the real probability of finding defective configurations in the jHipster SPL.

lidity focuses on possible errors in the experiment materials and tools used. To mitigate this threat we have relied on well-known tools for automated analysis of feature models like BDD solvers [18].

5 CONCLUSIONS AND FUTURE WORK

We have presented a step-wise decision analysis based on Monte Carlo simulations and driven by the variation points of the feature model. Our approach scales for large configuration spaces by splitting the analysis into the main decisions a user usually faces when configuring a product. Monte Carlo simulations approximate the influence of each feature variant in the configuration properties evaluating as few configurations as possible. In particular, we have applied our approach in a real-world feature model like the jHipster SPL to identify those features that have a higher probability of provoking a defect in a configuration. Our solution complements existing sampling techniques to analyze colossal configuration spaces improving the understanding of the influence of each feature selection. It can be part of a decision-making tool to assist the user in the process of configuring a feature model, to assist the developer when testing and maintaining an SPL, or to improve reasoner module of recommender systems and interactive configuration processes.

As future work, we plan to quantitatively compare our approach with existing sampling techniques that consider the whole configuration space (*e.g.,* feature coverage-based sampling), in order to evaluate how differ the features' influence in the configuration properties between those techniques. We also plan to extend our evaluation considering a broaden corpus of large-scale feature models and non-functional properties of the configuration to be measured like performance, memory footprint, or energy efficiency.

ACKNOWLEDGEMENTS

This work was supported by the Project (RTI2018-101204-B-C22, OPHELIA), funded by: FEDER/Ministry of Science and Innovation — State Research Agency; the TASOVA network (MCIU-AEI TIN2017-90644-REDT); the Junta de Andalucia COPERNICA and METAMORFOSIS projects; and the Spanish Government under Juan de la Cierva—Formación 2019 grant. We would also like to thank José A. Troyano for having inspired us in the usage of Monte Carlo methods in software product line analyses.

REFERENCES

[1] Sven Apel, Don S. Batory, Christian Kästner, and Gunter Saake, *Feature-Oriented Software Product Lines - Concepts and Implementation*, Springer, 2013.

[2] Mohsen Asadi, Samaneh Soltani, Dragan Gasevic, Marek Hatala, and Ebrahim Bagheri, 'Toward automated feature model configuration with optimizing non-functional requirements', *Inf. Softw. Technol.*, **56**(9), 1144–1165, (2014).

[3] Satomi Baba, Yongjoon Joe, Atsushi Iwasaki, and Makoto Yokoo, 'Real-time solving of quantified csps based on monte-carlo game tree search', in *22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 655–661, Barcelona, Spain, (2011).

[4] Ebrahim Bagheri, Mohsen Asadi, Dragan Gasevic, and Samaneh Soltani, 'Stratified analytic hierarchy process: Prioritization and selection of software features', in *14th International Conference on Software Product Lines (SPLC)*, volume 6287 of *LNCS*, pp. 300–315, (2010).

[5] Ebrahim Bagheri and Faezeh Ensan, 'Dynamic decision models for staged software product line configuration', *Requir. Eng.*, **19**(2), 187–212, (2014).

[6] David Benavides, Sergio Segura, and Antonio Ruiz Cortés, 'Automated analysis of feature models 20 years later: A literature review', *Inf. Syst.*, **35**(6), 615–636, (2010).

[7] Muhammad Umar Chaudhry and Jee-Hyong Lee, 'MOTiFS: Monte carlo tree search based feature selection', *Entropy*, **20**(5), (2018).

[8] Krzysztof Czarnecki and Ulrich W. Eisenecker, *Generative programming - methods, tools and applications*, Addison-Wesley, 2000.

[9] Krzysztof Czarnecki, Steven She, and Andrzej Wasowski, 'Sample spaces and feature models: There and back again', in *12th International Conference on Software Product Lines (SPLC)*, pp. 22–31, (2008).

[10] Stefan Fischer, Roberto E. Lopez-Herrejon, Rudolf Ramler, and Alexander Egyed, 'A preliminary empirical assessment of similarity for combinatorial interaction testing of software product lines', in *9th Workshop on Search-Based Software Testing (SBST@ICSE)*, pp. 15–18, (2016).

[11] José Angel Galindo, David Benavides, Pablo Trinidad, Antonio Manuel Gutiérrez-Fernández, and Antonio Ruiz-Cortés, 'Automated analysis of feature models: Quo vadis?', *Computing*, **101**(5), 387–433, (2019).

[12] José Angel Galindo, Deepak Dhungana, Rick Rabiser, David Benavides, Goetz Botterweck, and Paul Grünbacher, 'Supporting distributed product configuration by integrating heterogeneous variability modeling approaches', *Inf. Softw. Technol.*, **62**, 78–100, (2015).

[13] José Angel Galindo, Hamilton A. Turner, David Benavides, and Jules White, 'Testing variability-intensive systems using automated analysis: an application to android', *Softw. Qual. J.*, **24**(2), 365–405, (2016).

[14] Carl Graham and Denis Talay, *Strong Law of Large Numbers and Monte Carlo Methods*, 13–35, Springer Berlin Heidelberg, 2013.

[15] Jianmei Guo, Jia Hui Liang, Kai Shi, Dingyu Yang, Jingsong Zhang, Krzysztof Czarnecki, Vijay Ganesh, and Huiqun Yu, 'SMTIBEA: a hybrid multi-objective optimization algorithm for configuring large constrained software product lines', *Softw. Syst. Model.*, **18**(2), 1447–1466, (2019).

[16] Axel Halin, Alexandre Nuttinck, Mathieu Acher, Xavier Devroey, Gilles Perrouin, and Benoit Baudry, 'Test them all, is it worth it? assessing configuration sampling on the jhipster web development stack', *Empir. Softw. Eng.*, **24**(2), 674–717, (2019).

[17] Christopher Henard, Mike Papadakis, Gilles Perrouin, Jacques Klein, Patrick Heymans, and Yves Le Traon, 'Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines', *IEEE Transactions on Software Engineering*, **40**(7), 650–670, (2014).

[18] Ruben Heradio, David Fernández-Amorós, José A. Galindo, and David Benavides, 'Uniform and scalable sat-sampling for configurable systems', in *24th ACM International Systems and Software Product Line Conference (SPLC)*, volume A, pp. 17:1–17:11. ACM, (2020).

[19] Ruben Heradio, David Fernández-Amorós, Christoph Mayr-Dorn, and Alexander Egyed, 'Supporting the statistical analysis of variability models', in *41st International Conference on Software Engineering (ICSE)*, pp. 843–853. IEEE/ACM, (2019).

[20] Ruben Heradio, David Fernández-Amorós, Luis Torre-Cubillo, and Alberto Pérez García-Plaza, 'Improving the accuracy of COPLIMO to estimate the payoff of a software product line', *Expert Syst. Appl.*, **39**(9), 7919–7928, (2012).

[21] Ruben Heradio, Hector Perez-Morago, David Fernández-Amorós, Roberto Bean, Francisco Javier Cabrerizo, Carlos Cerrada, and Enrique Herrera-Viedma, 'Binary decision diagram algorithms to perform hard analysis operations on variability models', in *15th International Conference on New Trends in Software Methodologies, Tools and Techniques (SoMeT)*, volume 286 of *Frontiers in Artificial Intelligence and Applications*, pp. 139–154, (2016).

[22] Jose Miguel Horcas, Mónica Pinto, and Lidia Fuentes, 'Variability models for generating efficient configurations of functional quality attributes', *Inf. Softw. Technol.*, **95**, 147–164, (2018).

[23] José-Miguel Horcas, Mónica Pinto, and Lidia Fuentes, 'Software product line engineering: a practical experience', in *23rd International Systems and Software Product Line Conference (SPLC)*, pp. 25:1–25:13. ACM, (2019).

[24] Christian Kaltenecker, Alexander Grebhahn, Norbert Siegmund, Jianmei Guo, and Sven Apel, 'Distance-based sampling of software configuration spaces', in *41st International Conference on Software Engineering (ICSE)*, pp. 1084–1094. IEEE/ACM, (2019).

[25] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson, 'Feature-oriented domain analysis (FODA) feasibility study', Technical report, (1990). CMU/SEI-90-TR-21.

[26] Reza Karimpour and Guenther Ruhe, 'Evolutionary robust optimization for software product line scoping: An explorative study', *Comput. Lang. Syst. Struct.*, **47**, 189–210, (2017).

[27] Dirk P. Kroese, Tim Brereton, Thomas Taimre, and Zdravko I. Botev, 'Why the monte carlo method is so important today', *WIREs Computational Statistics*, **6**(6), 386–392, (2014).

[28] Jabier Martinez, Gabriele Rossi, Tewfik Ziadi, Tegawendé François D. Assise Bissyandé, Jacques Klein, and Yves Le Traon, 'Estimating and predicting average likability on computer-generated artwork variants', in *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1431–1432. ACM, (2015).

[29] Raúl Mazo, Cosmin Dumitrescu, Camille Salinesi, and Daniel Diaz, 'Recommendation heuristics for improving product line configuration processes', in *Recommendation Systems in Software Engineering*, 511–537, Springer, (2014).

[30] Daniel-Jesus Munoz, Jeho Oh, Mónica Pinto, Lidia Fuentes, and Don Batory, 'Uniform random sampling product configurations of feature models that have numerical features', in *23rd International Systems and Software Product Line Conference (SPLC)*, pp. 289–301, (2019).

[31] Hootan Nakhost and Martin Müller, 'Monte-carlo exploration for deterministic planning', in *21st International Jont Conference on Artifical Intelligence (IJCAI)*, p. 1766–1771, (2009).

[32] Alexander Nöhrer and Alexander Egyed, 'Optimizing user guidance during decision-making', in *15th International Conference on Software Product Lines (SPLC)*, pp. 25–34. IEEE Computer Society, (2011).

[33] Jeho Oh, Don S. Batory, Margaret Myers, and Norbert Siegmund, 'Finding near-optimal configurations in product lines by random sampling', in *11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, pp. 61–71, (2017).

[34] Juliana Alves Pereira, Mathieu Acher, Hugo Martin, and Jean-Marc Jézéquel, 'Sampling effect on performance prediction of configurable systems: A case study', in *ACM/SPEC International Conference on Performance Engineering (ICPE)*, pp. 277–288, (2020).

[35] Juliana Alves Pereira, Hugo Martin, Mathieu Acher, Jean-Marc Jézéquel, Goetz Botterweck, and Anthony Ventresque, 'Learning software configuration spaces: A systematic literature review', *CoRR*, **abs/1906.03018**, (2019).

[36] Juliana Alves Pereira, Jabier Martinez, Hari Kumar Gurudu, Sebastian Krieter, and Gunter Saake, 'Visual guidance for product line configuration using recommendations and non-functional properties', in *33rd Symposium on Applied Computing (SAC)*, pp. 2058–2065, (2018).

[37] Juliana Alves Pereira, Pawel Matuszyk, Sebastian Krieter, Myra Spiliopoulou, and Gunter Saake, 'A feature-based personalized recommender system for product-line configuration', in *ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE)*, pp. 120–131. ACM, (2016).

[38] Tobias Pett, Thomas Thüm, Tobias Runge, Sebastian Krieter, Malte Lochau, and Ina Schaefer, 'Product sampling for product lines: the scalability challenge', in *23rd International Systems and Software Product Line Conference (SPLC)*, pp. 14:1–14:6, Paris, France, (2019). ACM.

[39] Simon Poulding and Robert Feldt, 'Heuristic model checking using a monte-carlo tree search algorithm', in *Annual Conference on Genetic and Evolutionary Computation (GECCO)*, p. 1359–1366, (2015).

[40] Alessandro Previti, Raghuram Ramanujan, Marco Schaerf, and Bart Selman, 'Monte-carlo style UCT search for boolean satisfiability', in *Artificial Intelligence Around Man and Beyond*, pp. 177–188, (2011).

[41] Jorge Rodas-Silva, José Angel Galindo, Jorge García-Gutiérrez, and David Benavides, 'Selection of software product line implementation components using recommender systems: An application to wordpress', *IEEE Access*, **7**, 69226–69245, (2019).

[42] Lei Tan, Yuqing Lin, and Li Liu, 'Quality ranking of features in software product line engineering', in *2014 21st Asia-Pacific Software Engineering Conference*, volume 2, pp. 57–62, (2014).

[43] Y. Tanabe, K. Yoshizoe, and H. Imai, 'A study on security evaluation methodology for image-based biometrics authentication systems', in *3rd IEEE International Conference on Biometrics: Theory, Applications, and Systems*, pp. 1–6, (2009).

[44] Thomas Thüm, 'A BDD for linux?: the knowledge compilation challenge for variability', in *24th ACM International Systems and Software Product Line Conference (SPLC)*, volume A, pp. 16:1–16:6, (2020).

[45] Mahsa Varshosaz, Mustafa Al-Hajjaji, Thomas Thüm, Tobias Runge, Mohammad Reza Mousavi, and Ina Schaefer, 'A classification of product sampling for software product lines', in *22nd International Systems and Software Product Line Conference (SPLC)*, pp. 1–13, (2018).

[46] Cristian Vidal-Silva, José A. Galindo, Jesús Giráldez-Cru, and David Benavides, 'Automated completion of partial configurations as a diagnosis task using fastdiag to improve performance', in *Intelligent Systems in Industrial Applications*, pp. 107–117, (2021).