

Towards Learning-Aided Configuration in 3D Printing: Feasibility Study and Application to Defect Prediction

Benoit Amand, Maxime Cordy, Patrick Heymans, Mathieu Acher, Paul Temple, Jean-Marc Jézéquel

▶ **To cite this version:**

Benoit Amand, Maxime Cordy, Patrick Heymans, Mathieu Acher, Paul Temple, et al.. Towards Learning-Aided Configuration in 3D Printing: Feasibility Study and Application to Defect Prediction. VAMOS 2019 - 13th International Workshop on Variability Modelling of Software-Intensive Systems, Feb 2019, Leuven, Belgium. pp.1-9, 10.1145/3302333.3302338 . hal-01990767

HAL Id: hal-01990767
https://inria.hal.science/hal-01990767

Submitted on 23 Jan 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Towards Learning-Aided Configuration in 3D Printing

Feasibility Study and Application to Defect Prediction

Benoit Amand
University of Namur, Belgium
amand.benoit@gmail.com

Maxime Cordy
SnT, University of Luxembourg*
maxime.cordy@uni.lu

Patrick Heymans
University of Namur, Belgium
patrick.heymans@unamur.be

Mathieu Acher
Univ Rennes, Inria, CNRS, IRISA
Rennes, France
mathieu.acher@irisa.fr

Paul Temple
Univ Rennes, Inria, CNRS, IRISA
Rennes, France
paul.temple@irisa.fr

Jean-Marc Jézéquel
Univ Rennes, Inria, CNRS, IRISA
Rennes, France
jean-marc.jezequel@irisa.fr

ABSTRACT

Configurators rely on logical constraints over parameters to aid users and determine the validity of a configuration. However, for some domains, capturing such configuration knowledge is hard, if not infeasible. This is the case in the 3D printing industry, where parametric 3D object models contain the list of parameters and their value domains, but no explicit constraints. This calls for a complementary approach that learns what configurations are valid based on previous experiences. In this paper, we report on preliminary experiments showing the capability of state-of-the-art classification algorithms to assist the configuration process. While machine learning holds its promises when it comes to evaluation scores, an in-depth analysis reveals the opportunity to combine the classifiers with constraint solvers.

KEYWORDS

Configuration, Machine Learning, Sampling, 3D printing

ACM Reference Format:
Benoit Amand, Maxime Cordy, Patrick Heymans, Mathieu Acher, Paul Temple, and Jean-Marc Jézéquel. 2019. Towards Learning-Aided Configuration in 3D Printing: Feasibility Study and Application to Defect Prediction. In *Proceedings of 13th International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS'19),* Gilles Perrouin and Danny Weyns (Eds.). ACM, New York, NY, USA, 9 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

Configurators are software applications typically used to tailor products (i.e. goods or services) to specific requirements [12]. They comprise a user interface wherein users specify their needs and preferences by selecting options and setting parameter values. In the end, this process results in a *configuration* encoded as the set of selected options (the others being considered as discarded) and the set of values assigned to parameters. Not all configurations are valid, though, because of, e.g., technical, marketing or legal reasons. In order to assess the validity of a configuration, configurators rely on some *configuration knowledge* that consists of logical rules over the options and parameters. Capturing and maintaining this knowledge are critical activities for building successful and reliable configurators [4, 14]. One can then encode this knowledge in standard source code (e.g., as state invariants of abstract data types) – with the risk of more errors and a decreased maintainability – or encode it in a dedicated, declarative model that can be processed by configuration engines [5, 11, 13, 27, 31, 32]. In feature-oriented software development and software product lines, the most common models to encode configuration knowledge are feature models (FMs) [6–10, 22].

Regardless of the actual syntax, a critical assumption behind the development of configurators is that configuration knowledge is *available, correct and complete.* This assumption does not always hold, though. In particular, this is the case in the 3D printing industry. Community websites like Thingiverse [39] provide parametric 3D object models that one can customize by means of a configurator named *Customizer* (see Figure 1). Customizer thus allows one to set the parameters of the object to print. However, it does not give any guarantee that the parameterized object can be printed correctly. There are two main reasons behind this. First, the 3D object model syntax does not allow the definition of constraints over the parameters. Second, printing instructions are given in a programming-like language and as such, it is hard to automatically identify all cases where the printing will fail or yield inappropriate results. This leads to a waste of time – as 3D printing processes are time-consuming – and more importantly, a waste of resources.

Clearly, 3D printing is but one of the many application domains that would benefit from configurators that can prevent errors. Given the absence of (correct and complete) configuration knowledge, an alternative is to rely on tested configurations – which are known to be valid or not – in order to *predict* whether an unseen configuration is also valid. To achieve this, one can rely on state-of-the-art classification algorithms such as, e.g., decision trees, neural networks or support vector machines. While this solution does not need any configuration knowledge, it does, however, require examples (configurations) to train machine learning algorithms. The idea is to *label* each configuration with a class (*i.e.*, valid or invalid) based on the execution and observation of the configuration (i.e., does the configuration lead to defects in resulting 3D models?). Unfortunately, in 3D printing, the number of known configurations can be insufficient given the huge space of parameter values (according to [1], some parametric models have more than 10^{30} configurations). Therefore, it is likely that the configuration process in 3D printers

* This work was done when Maxime Cordy worked at the University of Namur, and was supported by the Fonds de la Recherche Scientifique - FNRS under Grant(s) O05518F-RG03.
This work benefited from the support of the project ANR-17-CE25-0010-01 VaryVary.

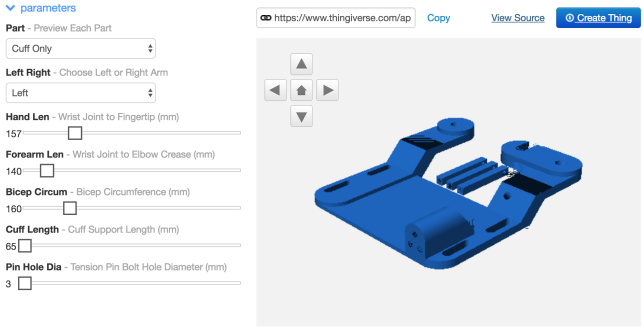


Figure 1: Thingiverse's customizer

```
/* [Parameters] */
// - Choose Left or Right Arm
LeftRight = "Left"; // [Left,Right]
// - Wrist Joint to Fingertip (mm)
HandLen = 135; //[135:230]

/* [Hidden] */
ArmVersion = "V2.1/";
HandPerc = round((HandLen / 135) * 100);
WristWidth = 0.72 * HandPerc;
```

Figure 2: A code excerpt of the Thing shown in Figure 1

would greatly benefit from *both* explicit configuration knowledge and classification algorithms.

This paper constitutes a first endeavour towards joining the forces of knowledge-based configuration and learning-aided configuration. Our contribution is to study whether common classification algorithms are appropriate to facilitate the configuration process in 3D printing *without* the support of any configuration knowledge. To achieve this, we developed a toolchain that can (i) extract the parameters of a 3D object model and their respective domain of values; (ii) sample a subset of all possible parameterizations; (iii) invoke an oracle to determine the validity of any parameterization of a given sample; and (iv) train different classifiers on the sample to predict the validity of parameterizations that were not part of the sample. Using our tools, we study the topology of 31 models (in terms of their parameters) and evaluate the prediction capability of 17 classification algorithms. Our experiments reveal that in most of the cases, the classifiers manage to achieve high scores. However, an in-depth look into specific models shows that the classifiers struggle to provide correct decisions for boundary configurations and one specific model, thereby motivating the need for supporting the classifiers with configuration knowledge.

The paper is structured as follows. Section 2 presents the context of 3D objects and the motivation for supporting their configuration process. Section 3 describes our approach and toolchain, while Section 4 explains the experimental results. We discuss our perspectives in Section 5 and present the related work in Section 6.

2 THINGIVERSE AND 3D OBJECT PRINTING

Thingiverse [39] is a community website in which members share and download 3D object models – called *Things*. Two formats are supported: Stereo Lithography (*.stl*) and OpenSCAD (*.scad*). Among the OpenSCAD models, some are specified as parametric and can be configured on the website to produce 3D objects of different shapes, sizes, etc. Thingiverse's configurator – named *Customizer* – can read any model in the OpenSCAD format and subsequently generate a form-like configuration interface to allow for setting parameter values; see Figure 1. Once parameters are set, the Customizer displays a preview of the object to be printed and allows users to download the parameterized model in the *.stl* format, which is the de-facto standard supported by all printing software.

OpenSCAD models are text files that contain the programming instruction to execute in order to produce the 3D object. Additionally, one can declare constants in the model that can then be used in the programming instructions. Although OpenSCAD does not allow the declaration of parameters per se, Customizer relies on a convention according to which parameters are constants whose domain of values is written in comments, between brackets, either as an enumerated list of values or as an interval of real numbers. Only these parametric constants are shown in the Customizer interface; the others remain hidden. For example, Figure 2 shows an excerpt of the code behind the Thing shown in Figure 1. Here, the constant `LeftRight` is considered as a parameter because it has an associated comment specifying the two values that this constant may take. Another parameter is `HandLen`; in this case, the authorized values are specified as an interval. All the other constants are not parametric and their value is given either explicitly or as a function of the other constants/parameters.

Although this ad-hoc syntax allows one to restrict the parameter values to a controlled set, not all allowed configurations are valid – that is, lead to an object printed correctly. There are multiple reasons why an object is badly printed. Some of them are generic, e.g. the parameterized model does not define a closed shape (more precisely, is not a two-dimensional manifold), or the thickness of the object is too low. Others, however, are due to incompatibilities with a specific 3D printer. In this work, we focus on the generic reasons and disregard the compatibility issues.

Currently, the invalidity of a given configuration can only be detected either after the printing of the object, or by the software responsible for transforming *.stl* files into low-level instructions for a particular 3D printer. This software tool, named *Slic3r*, is capable to provide some anticipated information on the yet-to-be-print 3D object, such as its volume and dimensions, as well as to predict some defects that may occur. Its reasoning scope, however, is limited to that of one configuration at a time, which motivates the need to increase the capability of the *Customizer* (or any equivalent configurator) to provide feedback to users as they change the configuration of their 3D objects.

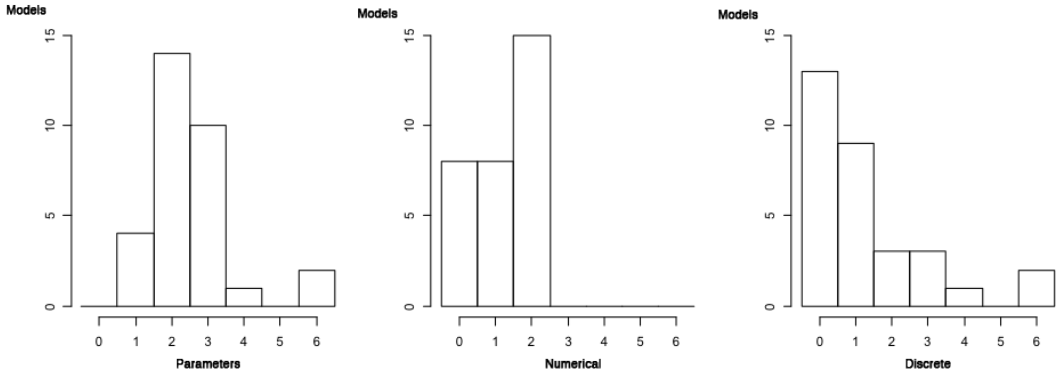


Figure 3: Number of parameters in the 31 studied models

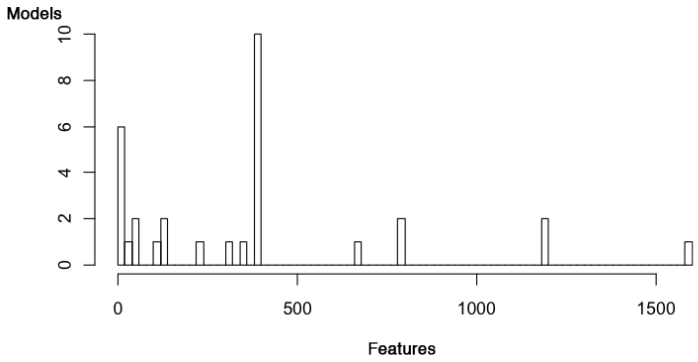


Figure 4: Sample size for the 31 studied models

3 SAMPLING, TESTING, LEARNING

Our objective is to assess the capability of different classification algorithms to predict whether a configuration will lead to a correctly printed 3D object. To achieve this, we developed a toolchain that allows us to (i) extract, from an OpenSCAD file, the set of its parameters and their domain of values, (ii) sample a number of configurations, (iii) determine the validity of a given configuration based on some oracle, and (iv) train classification algorithms based on the sampled configurations and their validity label.

To extract the parameters, we implemented an OpenSCAD parser in Java using ANTLR. Given that there exists no up-to-date specification of the language, our implementation is based on the source code of its reference implementation, the official Wiki, and our own experiments. In addition to retrieving the parameter lists, we associate any parameter with the corresponding comment that specifies its domain of value, following Thingiverse's conventions.

Classification algorithms exploit statistical relationships between parameters' values and their associated labels (or class). Doing that, a separation can be built between classes in the parameters' value space to predict the class of every configuration (including untested configurations). From the parameters and their value domain, we generate a sample of configurations that will act as the training set for the classifier. This sample is important as it needs to cover as well as possible the parameters' space.

In OpenSCAD, we observe two properties: (1) there are no known cross constraints among parameters; (2) there are real-valued domains, which leads to an uncountable number of configurations. It motivates the use of the following sampling technique to build a training set. When the value domain of a parameter is an enumerated list, we consider every possible value in the list. As for domains specified as an interval of real numbers, we consider 20 equidistant values between the minimum and maximum authorized value. In fine, the sample of configurations is the Cartesian product of the retained value for all parameters. Although other approaches exist to sample configurations effectively [19, 26, 42], it is out of our scope to try and compare their applicability in our context.

To build our oracle, we use the diagnosis capabilities of *Slic3r* as this tool can systematically detect the most common issues in 3D object models and has often a short response time; though we noted that in rare cases, the validation time of a configuration could reach multiple hours because of the time needed to transform the parametric *.scad* into a parameterized *.stl* model.

To perform our experiments, we gathered a total of 5057 OpenSCAD parametric models. Given the inability to predict the complexity of a model and the validation time of its configuration, we

adopted an empirical approach where we successively apply two filters. The first filter removes those models for which the sampling procedure would return more than 10^{12} configurations. Although this filter is redundant with the second, it requires small time to apply. Then, the second filter removes those models that required more than one hour to generate the sampled configurations and validate them. The first filter left us with 1580 models, while the second reduced this number to 201. For 35 of them, our sampling returned only invalid configurations, while for 135 others only valid configurations were sampled. We deemed these cases uninteresting for our experiments and thus discarded them too.

In the end, 31 models remained whose topology are presented in Figure 3. We notice that those models have one to six parameters, including at most two whose domain of value is an interval of real numbers. When such numeric parameters are present, the total number of parameters never exceeds three. This apparently small number, however, does not reflect the variability of the models, which mainly originates from the extent of the interval of domain values. Figure 4 depicts the number of configuration returned by our sampling procedure. Given that only 15 models have two numeric parameters, each of which is set to 20 different values by our sampling procedure, we observe that 16 of the 31 models thus yield a training-set size of at least 400, up to 1,600.

To generate a test set that is different from the training set, we reuse the same sampling procedure except that, for numeric parameters, we consider 21 equidistant values instead of the 20 used for the training set. The test set thus contains $\frac{21 \times p}{20}$ more instances than the training set, where p is the number of numeric parameters in the model. Since 20 and 21 are coprime integers, it guarantees that none of the value is common to both sets, i.e., no test instance is part of the training set. This property, however, does not hold for models that consist of enumerated parameters only.

Finally, we used well-known classification algorithms implemented in the open-source Java software WEKA [15] (see Table 1). Since our objective is a preliminary assessment of the feasibility and the potential of applying machine learning to support configuration processes, we did not favour one algorithm over another regardless of interesting properties like interpretability. As for hyperparameters, either we opted for values that are commonly used or we found appropriate values experimentally. To evaluate a classifier, we use common metrics: accuracy, precision, recall and F1-score.

4 RESULTS

Our experiments consist of two parts. The first part is a quantitative evaluation of the performance of each classifier trained and evaluated on each model separately. The second part is a detailed analysis of four particular models which provide various representative properties as well as interesting insights.

4.1 Overall Results

Table 1 presents, for each classifier, the average accuracy, precision, recall and F1-score computed over all 31 models.

Traditional machine learning performance evaluations rely on the computation of those measures. In our case, since we know the oracle's decision (given by Slic3r), performance evaluation consists in assessing how much the machine learning's predictions agree with the oracle's decision. Only two decisions can be taken: valid if the configuration can be printed or invalid if it cannot. Comparison results are usually represented as a confusion matrix which reports on one dimension labels from the oracle and on the other dimension labels from the machine learning algorithm. On the main diagonal of the matrix, agreements between the two entities are reported. Configurations that are classified by both the classifier and the oracle as valid are True Positives while configurations that are considered as not valid from either points of view are True Negatives. On the other diagonal, we found configurations for which both decisions disagree. If the classifier states that a configuration is valid while the oracle disagrees, the configuration is said to be a False Positive. On the contrary, if a configuration is classified as invalid by the classifier but not by the oracle, then the configuration is a False Negative. Precision, Recall, Accuracy and F1-score are a combination of those four elements that help to quantify the degree of agreement between the oracle and the classifier which, in the end, assesses the performances of the classifier.

Results show that overall, the classifiers manage to capture correctly the configuration knowledge from the training sample. They all achieve high scores; in particular, the average F1-score never falls below 0.93. However, a look at Figure 5 reveals that the F1-score occasionally falls dramatically for specific models, and this fall occurs in all used classifiers. In particular, a single model is responsible for the lowest F1-score in all classifiers. A closer look at this model reveals that it has a single parameter whose value ranges from 0 to 1,000,000. This value is actually used as a seed for randomizing the generation of the 3D object, which means that there is no a priori correlation between neighbouring values. This model being peculiar, we included it as part of our detailed studies.

4.2 Detailed Analysis

Hopper Upgrade Extension.[1] Our first model represents a cylinder (see Figure 6). It has a single, numeric parameter named *coupling inner dimension* whose domain of values is the interval [30, 40]. This parameter encodes the outer diameter of the cylinder, while the inner diameter is fixed to 32. Therefore, every assignment of the parameter equal to or below 32 yields an invalid 3D object. Among the 20 instances of our sampled training set, the invalid instance with the highest parameter value sets it to 31.58, while the valid instance with the lowest parameter value sets it to 32.10. The border value 32 is thus not part of the training set. The 21-instance test set, however, consists of every round and .5 values from 30 to 40, including 32. The results were identical for all classifiers: 16 true positives, 4 true negatives, and 1 false positive, leading to an F1-score of 0,97. Unsurprisingly the false-positive instance is the border value 32, which is wrongly evaluated by all classifiers. In particular, a closer look at the `C4.5` and `Ripper` classifiers revealed that both consider the value 31.58 as the highest invalid instance, de facto acting as the delimiter to separate invalid and valid classes. This simple way of drawing boundaries between classes is certainly a source of many mistakes in models with continuous parameters, which questions the capability of classifiers to discover the actual boundary values without additional support.

[1] https://www.thingiverse.com/apps/customizer/run?thing_id=111850. MakerBot account required (signing up is free).

Classifier	Accuracy	Precision	Recall	F1-Score
Hoeffding Tree	0.9384	0.9313	0.9874	0.9567
Naive Bayes	0.9507	0.9473	0.9852	0.9642
C4.5 (J48)	0.9562	0.9531	0.9824	0.9665
Support Vector Machine (Linear)	0.9477	0.9565	0.9722	0.9523
Logistic Model Tree	0.9633	0.9593	0.9843	0.9708
Logistic Regression	0.9623	0.9568	0.9902	0.9715
K*	0.9584	0.9527	0.9898	0.9696
REP Tree	0.9612	0.9566	0.9826	0.9688
PART Decision List	0.9614	0.9595	0.9810	0.9692
RIPPER (JRip)	0.9628	0.9578	0.9835	0.9696
Random Committee	0.9680	0.9692	0.9793	0.9735
Multilayer Perceptron	0.9684	0.9627	0.9902	0.9753
Support Vector Machine (PUK with $\omega = 1, \sigma = 0.1$)	0.9688	0.9635	0.9890	0.9749
Random Forest	0.9673	0.9692	0.9780	0.9729
Support Vector Machine (PK with $d = 3, \gamma = 2, c_0 = 0.5$)	0.9368	0.9555	0.9639	0.9543
K-Nearest Neighbours (IBK)	0.9673	0.9692	0.9780	0.9729
Support Vector Machine (RBF with $\gamma = 5$)	0.9506	0.9715	0.9412	0.9304

Table 1: Average scores over all models for each classifier.

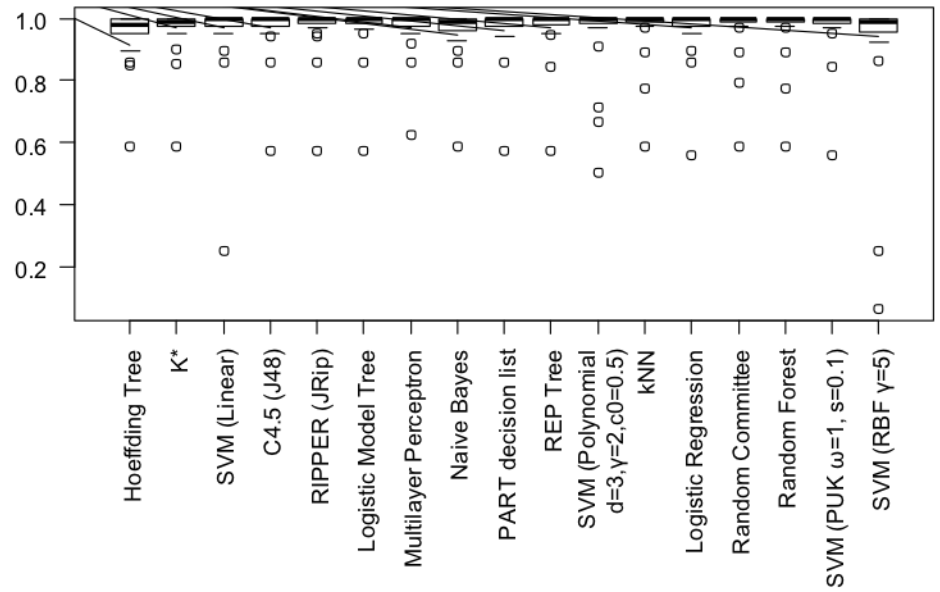


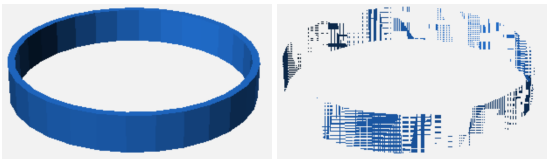
Figure 5: Box plots showing the distributions of the F1-scores per classifier, across all models.

Holesphere.[2] This model represents a sphere with three cylindric holes inside. It includes two parameters: *sphere radius*, which accepts values in $[2, 100]$ and *hole radius*, whose domain is $[1, 100]$. As illustrated in Figure 7, problems occur when the hole radius is too big w.r.t. the sphere radius. Figure 8 depicts the division of the configuration space in two half-planes, one containing all the valid configurations (in blue) and the other including the invalid ones (in red). It also represents as squares the erroneous predictions made by the C4.5 algorithm. More generally, the classifiers sometimes make erroneous predictions in borderline cases.

BLTouch mounting adapter for remixed Micromake effector.[3] Our third example is a spare part of some 3D printer. It consists of a cylinder with two holes including one in the centre. The model has four parameters controlling the position and the shape of the cylinder and the holes: *horizontal offset* (with value in $[0, 5]$), *vertical offset* ($[0, 50]$), *base height* ($\{1, 2, 3\}$) and *hole diameter* ([0,3]).

[2]https://www.thingiverse.com/apps/customizer/run?thing_id=74150

[3]https://www.thingiverse.com/apps/customizer/run?thing_id=1657673



(a) set to 34 (b) set to 32

Figure 6: Rendering of the Hopper Upgrade Extension with different values set to coupling inner dimension.

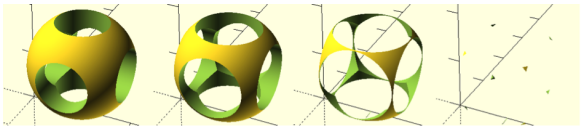


Figure 7: Rendering of the Holesphere with sphere radius set to 10 and hole radius set to 5, 6, 7 and 8 (from left to right).

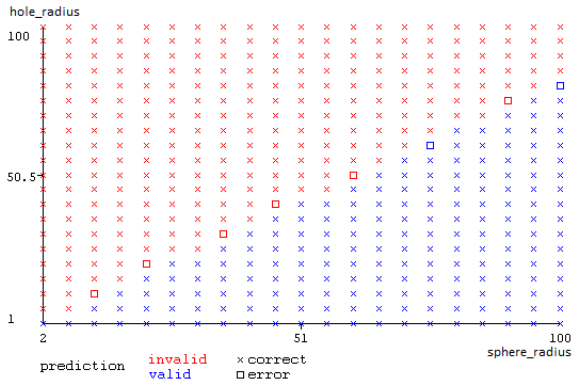


Figure 8: Configuration space of the Holesphere model. Invalid and valid configurations are all found in their respective half-plane. Crosses and squares show the predictions of the C4.5 classifier w.r.t. the validity of any configuration.

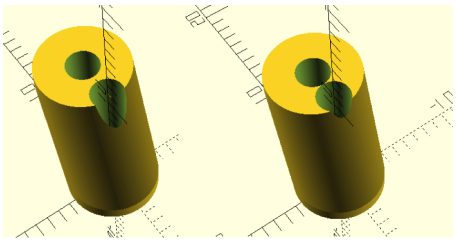


Figure 9: Rendering of the BLTouch mounting adapter with a horizontal offset of 4 (left) and 2.5 (right).

Among all the configurations, only those that have a *horizontal offset* of 2.5 are invalid, as this makes the two holes tangent and therefore lead to a non-manifold object (see Figure 9). This particular value was sampled in the test set but not in the training set. Therefore, the training set contained only valid configurations and all classifiers were unable to detect the invalid instances. More precisely, all classifiers considered that all configurations were valid. This yielded 1260 true positives and 63 false positives.

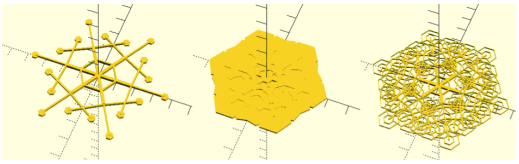


Figure 10: Rendering of the Snowflakes with the seed parameter set to 1 (left), $1+2\times10^{-16}$ (centre) and $1+4\times10^{-16}$ (right).

Blizzard of Unique Snowflakes.[4] This model is peculiar because it includes only one parameter, named *seed*, which is used to initialize a random number generator (RNG) that will impact the result of the printing. It has a value domain of $[0, 10^6]$ and the slightest variation in the seed value yields very different results (see Figure 10). This model is the one where all classifiers give their worst scores. Indeed, the F1-score of any classifier does not exceed 0.625 and even reaches 0.25 for two SVM-based classifiers. These poor results naturally come from the fact that the outcome of the printing is determined randomly.

4.3 Concluding Remarks

Although the classifiers obtain good scores overall, our detailed analysis of the four cases reveals weaknesses that learning-based configuration seems unable to overcome. In the Hopper Upgrade and BLTouch adapter examples, a single parameter value delimits the valid configuration from the invalid ones. If this value is not part of the training set, all classifiers are unable to accurately extrapolate it from the nearby, known instances. Clearly, machine learning would benefit from expert knowledge that would help to identify the exact delimiting value. The Holesphere case exhibits a similar problem, where the validity of a configuration depends on the relative values of two parameters. In such cases, the classifiers manage to draw the edge between the two half-planes; the accuracy of this edge increases as the size of the training set gets higher. Alternatively, knowing the mathematical equation defining this edge would yield exact results. The Blizzard example is special, as there is no explicit mathematical relation between the parameter value (the RNG seed) and the validity of the resulting configuration. It is thus unsurprising and unavoidable that machine learning performs poorly. In such cases, the only remedy is to gain explicit knowledge of which seed values lead to correct results, or to extract constraints over the model variables impacted by the RNG. More generally, all these imperfections suggest that an ideal solution would combine knowledge learned from machine learning (e.g. interpreted in the form of constraints) with explicit expert knowledge. The interpretation of knowledge from learning models and the combination with apriori knowledge is thus an important direction that we will follow in future work. This problem is not trivial, notably because

[4] https://www.thingiverse.com/apps/customizer/run?thing_id=37525

Classifier	Precision	Recall	F1-Score
Support Vector Machine (PK with $d = 3, \gamma = 2, c_0 = 0.5$)	0.353	0.857	0.5
Random Forest	0.5	0.714	0.588
Random Committee	0.5	0.714	0.588
REP Tree	0.571	0.571	0.571
Logistic Model Tree	0.571	0.571	0.571
Multilayer Perceptron	0.556	0.714	0.625
Logistic Regression	0.455	0.714	0.556
C4.5 (J48)	0.571	0.571	0.571
Naive Bayes	0.5	0.714	0.588
Support Vector Machine (RBF with $\gamma = 5$)	1	0.143	0.25
Support Vector Machine (Linear)	1	0.143	0.25
Support Vector Machine (PUK with $\omega = 1, \sigma = 0.1$)	0.455	0.714	0.556
Hoeffding Tree	0.5	0.714	0.588
kNN (IBk)	0.5	0.714	0.588
K*	0.5	0.714	0.588
RIPPER (JRip)	0.571	0.571	0.571
PART Decision List	0.571	0.571	0.571

Table 2: Average scores over all models for each classifier.

conflicts may arise between different sources of knowledge (e.g. the expert or the oracle might be wrong).

Threats to validity. Our results have likely been influenced by multiple factors. The first threat is the set of 3D models we selected to analyze. In particular, we discarded models requiring substantial computation resources (due to a large number of configurations or to a high analysis time), as our goal was only to gain first insights. However, this made us ignore models with large configuration space, which are likely to be more challenging for the classifiers. A second threat is our sampling method that arbitrarily selects equidistant parameters values. This makes the assumption that diversity is uniformly distributed over the space of parameter values, which is likely untrue in many models. Moreover, for 3D models with few to no numeric parameters, the training set and the test will not be disjoint. A third threat lies in the oracle we used, i.e. Slic3R. This validation program is sound but not complete; therefore, the classifiers may learn that a given configuration is valid although it is not. In practical settings, the benefits of using Slic3R are reduced by the fact that, in many cases, this program runs sufficiently fast to be invoked at runtime (i.e. while configuring). Nevertheless, our objective was a preliminary assessment of the capability of classification algorithms to predict defects in 3D objects. Relying on Slic3R allowed us to achieve this objective, although we are aware that a practical solution would have to rely on other sources to get labelled configurations. In the same vein, other alternatives to Slic3R exist and we could have used any of them either as alternatives or complementarily. Finally, the classification algorithms and their hyperparameters have been chosen arbitrarily while experimenting. Changing those would likely affect the results. However, we are confident that our conclusions, and in particular those related to the four cases analysed in detail, would remain valid.

5 PERSPECTIVES

Testing oracle. As part of our approach, one needs an automated testing procedure (*an oracle*) capable of identifying possible defects of a configured 3D model and thus of labeling configurations. In general, there are two important requirements. First, the procedure should be fast and not costly. Otherwise, we will typically collect a low number of configurations to populate the training set. Second, the oracle should be complete and sound, i.e., not missing defects or reporting false defects. Retrospectively looking at our attempt (see Section 3): (1) *Slic3r* has the merit of being quite fast; (2) the diagnosis information of *Slic3r* may be incomplete and we may wrongly classify 3D models as defect-free. To reinforce the completeness of the oracle, it would certainly require the use of more costly procedures. A radical strategy is to try printing some 3D models, for real. Overall, there is a trade-off between the cost and effectiveness of the testing procedure.

Involvement of experts as part of the learning process. Right now, our approach is agnostic and is independent of users' knowledge. Yet, some makers do have an extensive knowledge or prior experience with parametric 3D models; we could exploit it. A general and open problem is how to combine expert knowledge with our learning-based process. In particular, such knowledge can guide the way we sample and test configurations, typically for specifically focusing on some problematic parameters (instead of equally exploring all parameters). An example is the case of *Hopper Upgrade Extension* (previous section): an advanced user could state that the 32 value of the numerical parameter *coupling inner dimension* deserves a special attention. We could also use our learning approach to *verify* some intuitions and knowledge of makers.

Mining knowledge out of forums. In Thingiverse, each Thing has a dedicated forum where makers exchange information about parametric 3D objects. Discussions include complaints and advice on how to 3D print an object, or how to set proper values. We aim to mine and formalize such configuration knowledge since it can

prevent misconfigurations. It can also guide our sampling strategy in order to focus on more difficult configuration cases. Overall, the mining process is complementary to our proposal.

Integration into Thingiverse. An open issue is how to integrate learning-aided configuration as part of makers' community such as Thingiverse. Since the exploratory study of Acher et al. [1], there is still no support for cross-tree constraints[5]. The consequence is that learned constraints cannot be made explicit for further reuse and enforcing configuration knowledge. As future work, we aim to initiate a dialogue with stakeholders of the 3D printing domain. We also plan to re-engineer some existing tools (e.g., configurators) to showcase the added value of our specialization approach.

6 RELATED WORK

6.1 Customization and 3D Printing

Oehlberg et al. [29] conducted an analysis of over 175,000 digital designs from Thingiverse to better understand how online communities customize 3D models. The underlying motivation is that makers frequently combine and adapt designs for physical objects. So-called remixes are predominantly generated designs from Customizer, a built-in web app for adjusting parametric designs (see Section 2). Alock et al. [3] investigated users' activities on Thingiverse and their conversations. The key findings are that various barriers exist when using, customizing, and printing 3D designs. Our work precisely aims at raising existing limitations and is a first step toward learning-aided configuration of 3D designs.

Acher et al. [1] conducted a field study of Thingiverse to explore how variability is modelled and implemented in the 3D printing domain. A noticeable result is that cross-tree constraints cannot be expressed in OpenSCAD programs despite their intrinsic presence. Building upon this identified problem and vision paper, our work goes several steps further: we automate the reverse engineering of variability; we automate the derivation and testing of 3D models; we apply learning techniques to capture constraints.

6.2 Machine Learning and Configuration

The prediction of the performance of individual configurations is subject to intensive research [17, 18, 21, 24, 25, 30, 34, 35, 37, 41, 43, 44]. Statistical machine learning is typically used to address a regression problem and predict quantitative values of configurations out of a sample of measurements.

Guo et al. pioneered the use of regression techniques in the context of configurable systems [17]; they relied on regression trees (i.e., CART) to predict the performance of untested, new configurations. Siegmund et al. combined machine-learning and sampling heuristics to compute the individual influences of configuration options and their interactions [34]. So-called performance-influence models are meant to ease understanding and debugging of configurable systems. Nair et al. [28] proposed to find optimal configurations using a learning to rank approach.

In our case, we are interested in predicting the defects (if any) of 3D models' configurations. The problem boils down to a *classification* problem (as opposed to regression problems considered in prior work). Closest is the approach of Temple et al. [38] that tries to learn constraints out of a sample of configurations labeled as valid or invalid. The *specialization* approach has been applied to video synthesizers, computer vision processing chains, Web servers, or video encoders [37]. Acher et al. propose to apply similar learning techniques for generating paper variants that are acceptable (e.g., with regards to page limits) [2]. 3D printing is an interesting application domain with similar yet specific problems. First, one needs cost-effective, automated procedures and testing oracles to execute, observe and finally label configurations of 3D printing models. Second, one needs a sampling strategy to learn out of a sample. Several techniques have been proposed for testing or learning variability [20, 26, 30, 33, 34, 42]. The challenge of the Thingiverse dataset is that OpenSCAD programs exhibit numerical options that are typically difficult to discretize. Third, one needs more evidence about the applicability of learning techniques. This paper provides preliminary results that suggest the positive potential of the approach, but more research work is needed.

6.3 Testing and Configurable Systems

Numerous techniques have been developed to verify product lines and configurable systems either based on testing, type checking, model checking, or theorem proving [40]. For instance, SPLat is a dynamic analysis technique for pruning irrelevant configurations. The goal is to reduce the combinatorial number of variants (e.g., Java programs) to examine [23]. SPLif aims to detect bugs of software product lines with incomplete feature models [36]. It helps to prioritize failing tests and configurations. Black-box or white-box software testing techniques have been developed. The idea is to generate random inputs and resulting outputs are observed for detecting faults in single programs. Whitebox fuzzing, such as SAGE, consists of executing the program and gathering constraints on inputs using search techniques and heuristics, and to exploits constraints to guide the test generation [16].

7 CONCLUSION

We conducted a feasibility study for automatically identifying and learning defects of configurable objects to 3D print. We analyzed the variability of dozens of parameterized SCAD programs used to derive 3D objects coming from Thingiverse, a popular place for makers. The key findings are as follows:

- defects in parametric 3D models do exist and a testing oracle, based on Slic3r, can automatically detect some configurations that lead to defects;
- constraints among parameters could prevent misconfigurations and avoid users to configure 3D models with defects;
- a learning-based approach can find constraints with only a reasonable number of tests over a sampling of configurations;
- constraints cannot be formally expressed in a scad specification, despite evidence of their importance.

As future work, we have discussed some open problems (e.g., the involvement of 3D printings experts, integration of our approach into Thingiverse). 3D printing has shown to be a challenging application domain that can benefit from variability modeling techniques.

[5]Last access 07 November 2018: https://customizer.makerbot.com/docs

REFERENCES

[1] Mathieu Acher, Benoit Baudry, Olivier Barais, and Jean-Marc Jézéquel. 2014. Customization and 3D Printing: A Challenging Playground for Software Product Lines. In *18th International Software Product Line Conference*. Florence, Italy. https://hal.inria.fr/hal-01018937

[2] Mathieu Acher, Paul Temple, Jean-Marc Jézéquel, José A. Galindo, Jabier Martinez, and Tewfik Ziadi. 2018. VaryLATEX: Learning Paper Variants That Meet Constraints. In *Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems, VAMOS 2018, Madrid, Spain, February 7-9, 2018*. 83–88. DOI: http://dx.doi.org/10.1145/3168365.3168372

[3] Celena Alcock, Nathaniel Hudson, and Parmit K Chilana. 2016. Barriers to using, customizing, and Printing 3D designs on thingiverse. In *Proceedings of the 19th International Conference on Supporting Group Work*. ACM, 195–199.

[4] Liliana Ardissono, Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, Ralph Schäfer, and Markus Zanker. 2002. A Framework for Rapid Development of Advanced Web-based Configurator Applications. In *ECAI'02*. IOS Press, Amsterdam, The Netherlands, The Netherlands, 618–622.

[5] Timo Asikainen, Tomi MÃďnnistÃű, and Timo Soininen. 2004. Using a configurator for modelling and configuring software product lines based on feature models. In *In Workshop on Software Variability Management for Product Derivation, Software Product Line Conference*. 24–35.

[6] Don S. Batory. 2005. Feature Models, Grammars, and Propositional Formulas.. In *SPLC'05 (LNCS)*, Vol. 3714. Springer, 7–20.

[7] M. Ter Beek, A. Legay, A. Lluch Lafuente, and A. Vandin. 2018. A framework for quantitative modeling and analysis of highly (re)configurable systems. *IEEE Transactions on Software Engineering* (2018).

[8] David Benavides, Sergio Segura, Pablo Trinidad, and Antonio Ruiz Cortés. 2007. FAMA: Tooling a Framework for the Automated Analysis of Feature Models. In *VaMoS'07*. 129–134.

[9] Danilo Beuche. 2008. Modeling and Building Software Product Lines with Pure: :Variants. In *SPLC'08*. 358.

[10] Andreas Classen, Quentin Boucher, and Patrick Heymans. 2011. A text-based approach to feature modelling: Syntax and semantics of TVL. *SCP* 76 (December 2011), 1130–1143. Issue 12.

[11] Maxime Cordy and Patrick Heymans. 2018. Engineering Configurators for the Retail Industry: Experience Report and Challenges Ahead. In *ACM SAC '18*. 2050–2057.

[12] Xuegong Ding. 2008. Product Configuration on the Semantic Web Using Multi-Agent. In *ICNSC '08*. 304–309.

[13] Alexander Felfernig, Lothar Hotz, Claire Bagley, and Juha Tiihonen. 2014. *Knowledge-based Configuration: From Research to Business Cases* (1 ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[14] Gerhard Fleischanderl, Gerhard E. Friedrich, Alois Haselböck, Herwig Schreiner, and Markus Stumptner. 1998. Configuring Large Systems Using Generative Constraint Satisfaction. *IEEE Intelligent Systems* 13, 4 (July 1998), 59–68.

[15] Eibe Frank, Mark A. Hall, and Ian H. Witten. 2016. The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", Morgan Kaufmann, Fourth Edition. (2016).

[16] Patrice Godefroid, Michael Y. Levin, and David Molnar. 2012. SAGE: Whitebox Fuzzing for Security Testing. *Queue*, Article 20 (2012), 8 pages. DOI: http://dx.doi.org/10.1145/2090147.2094081

[17] Jianmei Guo, Krzysztof Czarnecki, Sven Apel, Norbert Siegmund, and Andrzej Wasowski. 2013. Variability-aware performance prediction: A statistical learning approach. In *ASE*.

[18] Jianmei Guo, Dingyu Yang, Norbert Siegmund, Sven Apel, Atrisha Sarkar, Pavel Valov, Krzysztof Czarnecki, Andrzej Wasowski, and Huiqun Yu. 2018. Data-efficient performance learning for configurable systems. *Empirical Software Engineering* 23, 3 (2018), 1826–1867. DOI: http://dx.doi.org/10.1007/s10664-017-9573-6

[19] Axel Halin, Alexandre Nuttinck, Mathieu Acher, Xavier Devroey, Gilles Perrouin, and Benoit Baudry. 2018. Test them all, is it worth it? Assessing configuration sampling on the JHipster Web development stack. *Empirical Software Engineering* (17 Jul 2018). DOI: http://dx.doi.org/10.1007/s10664-018-9635-4

[20] Axel Halin, Alexandre Nuttinck, Mathieu Acher, Xavier Devroey, Gilles Perrouin, and Benoit Baudry. 2018. Test them all, is it worth it? Assessing configuration sampling on the JHipster Web development stack. *Empirical Software Engineering* (July 2018). DOI: http://dx.doi.org/10.07980 Empirical Software Engineering journal.

[21] Pooyan Jamshidi, Miguel Velez, Christian Kästner, Norbert Siegmund, and Prasad Kawthekar. 2017. Transfer Learning for Improving Model Predictions in Highly Configurable Software. In *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE Computer Society, Los Alamitos, CA, 31–41. DOI: http://dx.doi.org/10.1109/SEAMS.2017.11

[22] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. 1990. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report CMU/SEI-90-TR-21.

[23] Chang Hwan Peter Kim, Darko Marinov, Sarfraz Khurshid, Don Batory, Sabrina Souto, Paulo Barros, and Marcelo D.Amorim. 2013. SPLat: Lightweight Dynamic Analysis for Reducing Combinatorics in Testing Configurable Systems. In *ESEC/FSE 2013*.

[24] Martin PfannemÃijller Michael MatthÃĪ Andy SchÃijrr Markus Weckesser, Roland Kluge and Christian Becker. 2018. Optimal Reconfiguration of Dynamic Software Product Lines Based on Performance-Influence Models. In *Software Product Lines Conference (SPLC'18)*.

[25] Jabier Martinez, Jean-Sébastien Sottet, Alfonso García Frey, Tegawendé F. Bissyandé, Tewfik Ziadi, Jacques Klein, Paul Temple, Mathieu Acher, and Yves Le Traon. 2018. Towards Estimating and Predicting User Perception on Software Product Variants. In *New Opportunities for Software Reuse - 17th International Conference, ICSR 2018, Madrid, Spain, May 21-23, 2018, Proceedings*. 23–40. DOI: http://dx.doi.org/10.1007/978-3-319-90421-4_2

[26] Flávio Medeiros, Christian Kästner, Márcio Ribeiro, Rohit Gheyi, and Sven Apel. 2016. A Comparison of 10 Sampling Algorithms for Configurable Systems. In *ICSE'16*.

[27] V. Myllarniemi, M. Raatikainen, and T. Mannisto. 2007. Using a Configurator for Predictable Component Composition. In *33rd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO 2007)*. 47–58.

[28] Vivek Nair, Zhe Yu, Tim Menzies, Norbert Siegmund, , and Sven Apel. 2018. Finding Faster Configurations using FLASH.

[29] Lora Oehlberg, Wesley Willett, and Wendy E. Mackay. 2015. Patterns of Physical Design Remixing in Online Maker Communities. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI 2015, Seoul, Republic of Korea, April 18-23, 2015*. 639–648. DOI: http://dx.doi.org/10.1145/2702123.2702175

[30] Jeho Oh, Don S. Batory, Margaret Myers, and Norbert Siegmund. 2017. Finding near-optimal configurations in product lines by random sampling. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*. 61–71. DOI: http://dx.doi.org/10.1145/3106237.3106273

[31] D. Sabin and R. Weigel. 1998. Product configuration frameworks-a survey. *IEEE Intelligent Systems and their Applications* 13, 4 (Jul 1998), 42–49. DOI: http://dx.doi.org/10.1109/5254.708432

[32] Mark Santolucito, Ennan Zhai, Rahul Dhodapkar, Aaron Shim, and Ruzica Piskac. 2017. Synthesizing configuration file specifications with association rule learning. *Proceedings of the ACM on Programming Languages* 1, OOPSLA (2017), 64.

[33] A. Sarkar, Jianmei Guo, N. Siegmund, S. Apel, and K. Czarnecki. 2015. Cost-Efficient Sampling for Performance Prediction of Configurable Systems (T). In *ASE'15*.

[34] Norbert Siegmund, Alexander Grebhahn, Christian Kästner, and Sven Apel. 2015. Performance-Influence Models for Highly Configurable Systems. In *ESEC/FSE'15*.

[35] Norbert Siegmund, Marko RosenmüLler, Christian KäStner, Paolo G. Giarrusso, Sven Apel, and Sergiy S. Kolesnikov. 2013. Scalable Prediction of Non-functional Properties in Software Product Lines: Footprint and Memory Consumption. *Inf. Softw. Technol.* (2013).

[36] Sabrina Souto, Divya Gopinath, Marcelo d'Amorim, Darko Marinov, Sarfraz Khurshid, and Don Batory. 2015. Faster Bug Detection for Software Product Lines with Incomplete Feature Models. In *SPLC '15*.

[37] Paul Temple, Mathieu Acher, Jean-Marc Jézéquel, and Olivier Barais. 2017. Learning Contextual-Variability Models. *IEEE Software* 34, 6 (2017), 64–70. DOI: http://dx.doi.org/10.1109/MS.2017.4121211

[38] Paul Temple, José Angel Galindo Duarte, Mathieu Acher, and Jean-Marc Jézéquel. 2016. Using Machine Learning to Infer Constraints for Product Lines. In *Software Product Line Conference (SPLC'16)*. Beijing, China. DOI: http://dx.doi.org/10.1145/2934466.2934472

[39] Thingiverse. 2019. MakerBot Industries. (2019). Retrieved October 11, 2019 from http://www.thingiverse.com

[40] Thomas Thüm, Sven Apel, Christian Kästner, Ina Schaefer, and Gunter Saake. 2014. A Classification and Survey of Analysis Strategies for Software Product Lines. *Comput. Surveys* (2014).

[41] Pavel Valov, Jean-Christophe Petkovich, Jianmei Guo, Sebastian Fischmeister, and Krzysztof Czarnecki. 2017. Transferring Performance Prediction Models Across Different Hardware Platforms. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, ICPE 2017, L'Aquila, Italy, April 22-26, 2017*. 39–50. DOI: http://dx.doi.org/10.1145/3030207.3030216

[42] 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 *Proceeedings of the 22nd International Conference on Systems and Software Product Line - Volume 1, SPLC 2018, Gothenburg, Sweden, September 10-14, 2018*. 1–13. DOI: http://dx.doi.org/10.1145/3233027.3233035

[43] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2008. SATzilla: Portfolio-based Algorithm Selection for SAT. *J. Artif. Intell. Res.* 32 (2008), 565–606. DOI: http://dx.doi.org/10.1613/jair.2490

[44] Yi Zhang, Jianmei Guo, Eric Blais, and Krzysztof Czarnecki. 2015. Performance Prediction of Configurable Software Systems by Fourier Learning (T). In *ASE'15*.