The Journal of Systems & Software 182 (2021) 111044

Contents lists available at ScienceDirect

The Journal of Systems & Software

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

Learning software configuration spaces: A systematic literature review☆

Juliana Alves Pereira [a,b,*], Mathieu Acher [a], Hugo Martin [a], Jean-Marc Jézéquel [a], Goetz Botterweck [c], Anthony Ventresque [d]

[a] Univ Rennes, Inria, CNRS, IRISA, France
[b] Pontifical Catholic University of Rio de Janeiro, Brazil
[c] Trinity College Dublin, Lero–The Irish Software Research Centre, Ireland
[d] University College Dublin, Ireland

ARTICLE INFO

Article history:
Received 7 June 2019
Received in revised form 15 April 2021
Accepted 12 July 2021
Available online 20 August 2021

Keywords:
Systematic literature review
Software product lines
Machine learning
Configurable systems

ABSTRACT

Most modern software systems (operating systems like Linux or Android, Web browsers like Firefox or Chrome, video encoders like ffmpeg, x264 or VLC, mobile and cloud applications, etc.) are highly configurable. Hundreds of configuration options, features, or plugins can be combined, each potentially with distinct functionality and effects on execution time, security, energy consumption, etc. Due to the combinatorial explosion and the cost of executing software, it is quickly impossible to exhaustively explore the whole configuration space. Hence, numerous works have investigated the idea of learning it from a small sample of configurations' measurements. The pattern "sampling, measuring, learning" has emerged in the literature, with several practical interests for both software developers and end-users of configurable systems. In this systematic literature review, we report on the different application objectives (*e.g.*, performance prediction, configuration optimization, constraint mining), use-cases, targeted software systems, and application domains. We review the various strategies employed to gather a representative and cost-effective sample. We describe automated software techniques used to measure functional and non-functional properties of configurations. We classify machine learning algorithms and how they relate to the pursued application. Finally, we also describe how researchers evaluate the quality of the learning process. The findings from this systematic review show that the potential application objective is important; there are a vast number of case studies reported in the literature related to particular domains or software systems. Yet, the huge variant space of configurable systems is still challenging and calls to further investigate the synergies between artificial intelligence and software engineering.

1. Introduction

End-users, system administrators, software engineers, and scientists have at their disposal thousands of options (a.k.a. features or parameters) to configure various kinds of software systems in order to fit their functional and non-functional needs (execution time, output quality, security, energy consumption, etc.). It is now ubiquitous that software comes in many variants and is highly configurable through conditional compilations, command-line options, runtime parameters, configuration files, or plugins. Software product lines (SPLs), software generators, dynamic system, self-adaptive systems, variability-intensive systems are well studied in the literature and enter in this class of configurable software systems (Svahnberg et al., 2005; Pohl et al., 2005; Apel et al., 2013; Sayagh et al., 2018; Benavides et al., 2010; Cashman et al., 2018; Hallsteinsen et al., 2008; Morin et al., 2009).

From an abstract point of view, a software configuration is simply a combination of options' values. Though customization is highly desirable, it introduces an enormous complexity due to the combinatorial explosion of possible variants. For example, the Linux kernel has 15,000+ options and most of them can have 3 values: "yes", "no", or "module". Without considering the presence of constraints to avoid some combinations of options, there may be $3^{15,000}$ possible variants of Linux — the estimated number of atoms in the universe is 10^{80} and is already reached with 300 Boolean options. Though Linux is an extreme case, many software systems or projects exhibit a very large configuration space; this might bring several challenges.

On the one hand, developers struggle to maintain, understand, and test configuration spaces since they can hardly analyze or execute all possible variants. According to several studies (Halin

☆ Editor: Raffaela Mirandola.
* Corresponding author.
E-mail address: jpereira@inf.puc-rio.br (J.A. Pereira).

https://doi.org/10.1016/j.jss.2021.111044
0164-1212/

et al., 2019; Sayagh et al., 2018), the flexibility brought by variability is expensive as configuration failures represent one of the most common types of software failures. Configuration failures is an "undesired effect observed in the system's delivered service" that may occur due to a specific combination of options' values (configurations) (Mathur, 2008). On the other hand, end-users fear software variability and stick to default configurations (Xu et al., 2015; Zheng et al., 2007) that may be sub-optimal (*e.g.*, the software system will run very slowly) or simply inadequate (*e.g.*, the quality of the output will be unsuitable).

Since it is hardly possible to fully explore all software configurations, the use of machine learning techniques is a quite natural and appealing approach. The basic idea is to learn out of a *sample* of configurations' observations and hopefully generalize to the whole configuration space. There are several applications ranging from performance prediction, configuration optimization, software understanding to constraint mining (*i.e.*, extraction of variability rules) – we will give a more exhaustive list in this literature review. For instance, end-users of x264 (a configurable video encoder) can estimate in advance the execution time of the command-line `X264 --no_cabac --no_fast_pskip --rc_lookahead 60 --ref 5 -o vid.264 vid.y4m` (see Fig. 1), since a machine learning model has been crafted to predict the performance of configurations. End-users may want to use the fastest configuration or know all configurations that meet an objective (*e.g.*, encoding time should be less than 10 s). Developers of x264 can be interested in understanding the effects of some options and how options interact.

For all these use-cases, a pattern has emerged in the scientific literature: *"sampling, measuring, learning"*. The basic principle is that a procedure is able to learn out of a sample of configurations' measurements (see Fig. 1). Specifically, many software configuration problems can actually be framed as statistical machine learning problems under the condition a sample of configurations' observations is available. For example, the prediction of the performance of individual configurations can be formulated as a regression problem; appropriate learning algorithms (*e.g.*, CART) can then be used to predict the performance of untested, new configurations. In this respect, it is worth noticing the dual use of the term *feature* in the software or machine learning fields: features either refer to software features (*a.k.a.* configuration options) or to variables a regressor aims to relate. A way to reconcile and visualize both is to consider a configuration matrix as depicted in Fig. 1. In a configuration matrix, each row describes a configuration together with observations/values of each feature and performance property. In the example of Fig. 1, the first configuration has the feature `no_cabac` set to False value and the feature `ref` set to 9 value while the encoding time performance value is 3.1876 s. We can use a sample of configurations (*i.e.*, a set of measured configurations) to train a machine learning model (a regressor) with predictive variables being command-line parameters of x264. Unmeasured configurations could then be predicted. Even for large-scale systems like the Linux Kernel, the same process of "sampling, measuring, learning" can be followed (see, e.g. Acher et al. (2019a,b)). Some additional steps are worth exploring (like feature engineering prior to learning) while techniques for sampling and learning should be adapted to scale at its complexity, but the general process remains applicable.

Learning software configuration spaces is, however, not a pure machine learning problem and there are a number of specific challenges to address at the intersection of software engineering and artificial intelligence. For instance, the sampling phase involves a number of difficult activities: (1) picking configurations that are valid and conform to constraints among options – one needs to resolve a satisfiability problem; (2) instrumenting the executions and observations of software for a variety of configurations – it can have an important computational cost and is hard to engineer especially when measuring non-functional aspects of software; (3) meanwhile, we expect that the sample is representative to the whole population of valid configurations otherwise the learning algorithm may hardly generalize to the whole configuration space. The general problem is to find the right strategy to decrease the cost of labeling software configurations while minimizing prediction errors. From an empirical perspective, one can also wonder to what extent learning approaches are effective for real-world software systems present in numerous domains.

While several studies have covered different aspects of configurable systems over the last years, there has been no secondary study (such as systematic literature reviews) that identifies and catalogs individual contributions for machine learning configuration spaces. Thus, there is no clear consensus on what techniques are used to support the process, including which quantitative and qualitative properties are considered and how they can be measured and evaluated, as well as how to select a significant sample of configurations and what is an ideal sample size. This stresses the need for a secondary study to build knowledge from combining findings from different approaches and present a complete overview of the progress made in this field. To achieve this aim, we conduct a *Systematic Literature Review* (SLR) (Kitchenham and Charters, 2007) to identify, analyze and interpret all available important research in this domain. We systematically review research papers in which the process of sampling, measuring, and learning configuration spaces occurs — more details about our research methodology are given in Section 2. Specifically, we aim of synthesizing evidence to answer the following four research questions:

- *RQ1. What are the concrete applications of learning software configuration spaces?*
- *RQ2. Which sampling methods and learning techniques are adopted when learning software configuration spaces?*
- *RQ3. Which techniques are used to gather measurements of functional and non-functional properties of configurations?*
- *RQ4. How are learning-based techniques validated?*

To address *RQ1*, we analyze the application objective of the study (*i.e.*, why they apply learning-based techniques). It would allow us to assess whether the proposed approaches are applicable. With respect to *RQ2*, we systematically investigate which sampling methods and learning techniques are used in the literature for exploring the SPL configuration space. With respect to *RQ3*, we give an in-depth view of how each study measures a sample of configurations. In addition, *RQ4* follows identifying which sampling design and evaluation metrics are used for evaluation.

By answering these questions, we make the following five contributions:

1. We identified six main different application areas: *pure prediction*, *interpretability*, *optimization*, *dynamic configuration*, *evolution*, and *mining constraints*.
2. We provide a framework classification of four main stages used for learning: *Sampling*, *Measuring*, *Learning*, *Validation*.
3. We describe 23 high-level sampling methods, 5 measurement strategies, 64 learning techniques, and 50 evaluation metrics used in the literature. As case studies, we identify 95 configurable systems targeting several domains, and functional and non-functional properties. We relate and discuss the learning and validation techniques with regard to their application objective.
4. We identify a set of open challenges faced by the current approaches, in order to guide researchers and practitioners to use and build appropriate solutions.

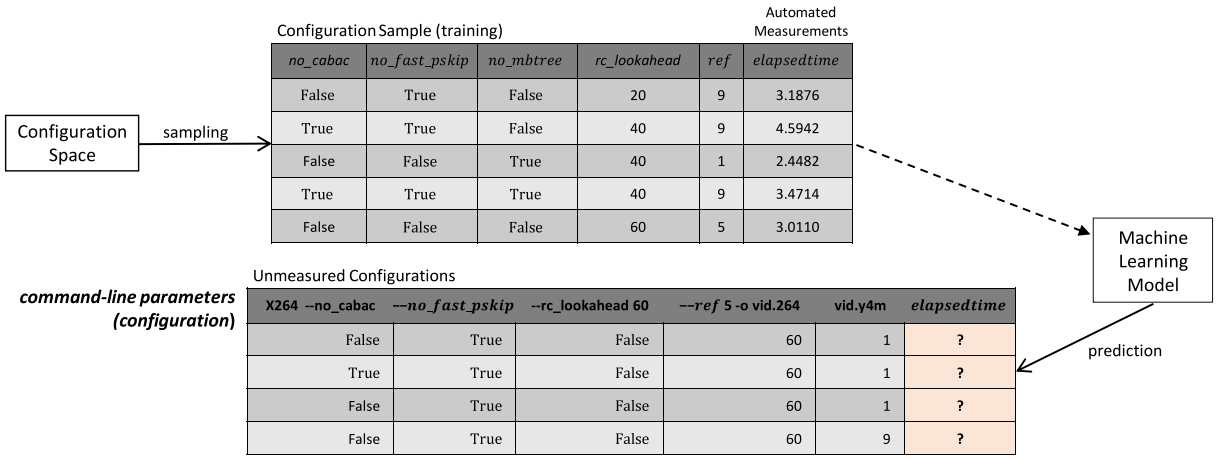

Fig. 1. Features, configurations, sample, measurements, and learning.

5. We build a Web repository (Pereira et al., 2019) to make our SLR results publicly available for the purpose of reproducibility and extension.

Overall, the findings of this SLR reveal that there is a significant body of work specialized in learning software configurable systems with an important application in terms of software technologies, application domains, or goals. There is a wide variety in the considered sampling or learning algorithms as well as in the evaluation process, mainly due to the considered subject systems and application objectives. Practitioners and researchers can benefit from the findings reported in this SLR as a reference when they select a learning technique for their own settings. To this end, this review provides a classification and catalog of specialized techniques in this field.

The rest of the paper is structured as follows. In Section 2, we describe the research protocol used to conduct the SLR. In Section 3, we categorize a sequence of key learning stages used by the ML state-of-the-art literature to explore highly configurable systems. In Section 4, we discuss the research questions. In Section 5, we discuss the current research themes in this field and present the open challenges that need attention in the future. In Section 6, we discuss the threats to the validity of our SLR. In Section 7, we describe similar secondary studies and indicate how our literature review differs from them. Finally, in Section 8, we present the conclusions of our work.

2. The review methodology

We followed the SLR guidelines by Kitchenham and Charters (2007) to systematically investigate the use of learning techniques for exploring the SPL configuration space. In this section, we present the SLR methodology that covers two main phases: *planning the review* and *conducting the review*. The paper selection process is shown in Fig. 2. Next, we report the details about each phase so that readers can assess their rigor and completeness, and reproduce our findings.

2.1. Planning the review

For identifying the candidate primary studies of Fig. 2, we first defined our SLR scope (i.e., identification of the need for a review and specification of the research questions). Then, we developed a review protocol.

The need for a systematic review. The main goal of this SLR is to systematically investigate and summarize the state-of-the-art of the research concerning learning techniques in the context of software configurable systems. The purpose of conducting this SLR has partially been addressed in the introduction and was motivated by the lack of a systematic study carried on this topic. According to Kitchenham and Charters (2007), an SLR should provide a valuable overview of the status of the field to the community through the summarization of existing empirical evidence supported by current scientific studies. The outcomes of such an overview can identify whether, or under what conditions, the proposed learning approaches can support various use-cases around configurable systems and be practically adopted (*e.g.*, for which context a specific learning technique is much suitable). By mean of this outcome, we can detect the limitations in current approaches to properly suggest areas for further investigation.

The research questions. The goal of this SLR is to answer the following main research question: *What studies have been reported in the literature on learning software configuration spaces since the introduction of Software Product Lines in the early 1990s (Kang et al., 1990) to date (2019)?* However, this question is too broad, so we derived the four sub-questions defined in Section 1. *RQ1* classifies the papers with regards to their application objective, i.e., for which particular task the approach is suited and useful. We can group studies into similar categories for comparison. It is also of interest to identify the practical motivations behind learning approaches. We verified whether the authors indicated a specific application for their approach; otherwise, we classified the approach as pure prediction. *RQ2–RQ4* seek to understand key steps of the learning process. *RQ2* reviews the set of sampling methods and learning-based techniques used in the literature. *RQ3* describes which subject software systems, application domains, and functional and non-functional properties of configurations are measured and how the measurement process is conducted. *RQ4* aims to characterize the evaluation process used by the researchers, including the sample design and supported evaluation metric(s). Finally, addressing these questions will allow us to identify trends and challenges in the current state-of-the-art approaches, as well as analysis their maturity to summarize our findings and propose future works.

The review protocol. We searched for all relevant papers published up to May 31st, 2019. The search process involved the use of 5 scientific digital libraries[1]: IEEE Xplore Digital Library,[2]

[1] We decided not to use Google Scholar due to search engine limitations, such as the very strict size of the search string.

[2] http://ieee.org/ieeexplore.