Self-Adaptive Learning for Configurable Software Systems

Mathieu Acher[1], Clément Quinton[2], and Tewfik Ziadi[3]

[1] University of Rennes, IRISA, Rennes, France
[2] University of Lille, Inria Lille-Nord Europe, Lille, France
[3] Sorbonne University, Paris, France

1 Context

Any software-intensive system is configurable and subject to variations to fit functional, performance, and security requirements. The ability to vary (variability) is thus crucial and must be handled from design to runtime. Software systems usually offer a myriad of configuration options – a very general term to refer to command line parameters, features, conditional compilation otpions, feature toggles, configuration files, plugins, etc. Though variability is a long-standing property of software systems, variable software is challenging developers due to the combinatorial explosion of possible configurations. Final users have also difficulties to fine-tune options. The fundamental reason is that developers and users can hardly know all properties of all configurations: it is practically not feasible to execute and measure all configurations in all possible conditions. As a result, manually selecting the best or a good-enough software configuration *w.r.t.* a performance criteria is known to be a hard task. Modern highly configurable software systems such as operating systems, IoT/cloud environments, robot embedded software, etc. provide thousands of options (15,000+ options for the recent version of the Linux kernel [2]). At this scale, a large configuration space is complex to be maintained, tested and configured by the developer/user.

Since it is not possible to entirely explore the whole configuration space and relate each configuration to the proper non-functional and functional requirements, the idea of applying machine learning techniques is appealing and more and more explored [3]. The basic principle is to train a learning model out of observations of several software configurations (a sample). Machine learning can be used to predict failures/performance of certain configurations or anticipating a configuration change depending on a previously-encountered context [8, 4]. It is thus a way to *e.g.,* predict the properties of any configuration, identify influential options, ensure non-regression or select the "best" configuration.

However, learning a model of a configurable system may be very costly. The learning process is trained on a configuration sample whose metrics are measured (*e.g.,* a subset of the whole configuration space) and is then generalized to other configurations [4, 9, 5]. The fact is that each configuration measure requires a significant amount of time and resources while the measure itself is somehow uncertain. Overall, learning a configuration space is a trade-off between cost and accuracy. Hopefully, the investments realized to learn a software configuration space pay off and generalize to all situations in which the software is deployed and executed.

2 The Multiple Dimensions of Configuration Spaces

Unfortunately, it might not be the case: other factors may break the learned model and make it inaccurate, e.g., for performance prediction.

A first threat to learning generalization is *software evolution* (or variability in time). Software systems, and highly configurable ones in particular, are meant to evolve over time for multiple reasons: after a bug fix, to add a new functionality, to support a new version, etc. In consequence, the configuration space of the software evolves accordingly and the variability differs over time. Therefore, the results obtained after performing a learning process over a certain configuration space is more likely to not be valid anymore after evolving the software.That is, one must learn multiple times from an evolving configuration space to ensure a proper result (*i.e.* to ensure that the resulting map metrics/configuration is correct). To comply with the evolving nature of configurations spaces, one thus need to update the learning models accordingly on a regular basis.

Another threat is external factors of a configurable system, such as the operating system or the hardware used to execute the software. This context influences the way the software varies, and affects both design time and run time configuration spaces [9]. At design time, the configuration space varies whenever the hardware supposed to host the deployed configuration changes. At run time, the configuration space evolves simultaneously with its technical environment, *e.g.,* adding or removing a device in the network, unexpected workload or server failure. The configuration space thus evolves for multiple reasons, in various contexts and in different periods. **A general challenge is to update the learned model of a configurable system in a cost-effective way**.

Ideally, one does not need to measure all configurations for a new version or for a new context: one can partly transfer configuration knowledge that has been already synthesized. Our observation is that there are opportunities for the learning process to adapt depending on *when* and *where* the configuration space has evolved and *what* has evolved [7]. We observe that there exist different configurations of the learning process for supporting this adaptation: one can change the learning algorithm (e.g, linear regression vs decision tree vs neural network), its hyperparameters; one can change the sampling strategy and the software options that are measured, etc. Overall, **the challenge is that not only the software is configurable, but also the learning process that should self-adapt to update learned software configuration models**.

3 Self-Adaptive Learning of Configuration Spaces

Similarly to learning the configuration space of a software system and since the learning process is itself configurable, it is possible to learn the configuration space of the learning process. That is, one could learn what is the best configuration of a learning process for a given software configuration space depending on how this configuration space evolves. To address this concern, one could think of a continuous, self-adaptive learning process that automatically updates the

learning model when needs be in a cost-effective way. This process would reconfigure the learning mechanism (with the right algorithm, the proper parameters, etc.) for efficiently realizing the expected task.

But similarly to learning the best *system configuration*, learning the best *learning configuration* is not a trivial task. As stated in previous sections, learning has *(i)* a high cost and *(ii)* can hardly generalize to any anticipated context or to a new software evolution. One thus needs to tailor the learning process to avoid restarting it from scratch. To put in practice such an abstract learning layer, one has to define a solution which answers the following questions:

C_1: **When** to learn the configuration space and update the prediction model?
C_2: **What** configuration options and (interpretable) information should be learned?

Proposing such a solution is even more challenging when considering the whole system life-cycle. Indeed, at both design and run time, something can change that forces the learning process to re-learn the model to *e.g.,* get the best or optimal configuration.

Continuous Learning at Design Time. Learning variability model at design time have been intensively studied during the last years (*e.g.,* [6, 1]). The general idea in this context is to analyse the configuration space at design time in terms of existing artefacts provided by the developers to infer variability models. As mentioned above, the artefacts can evolve at each commit/release and the learned models should be revisited to consider these new design evolutions. Such changes are typically tracked within a continuous integration system (*e.g.,* in a devops context). Most of existing research works propose re-starting the learning process to consider this situation. In this work we defend a radically new vision where the learning process is self-configurable to follow the different evolutions of the system at design time. Developers can thus continuously evolving the artefacts while keeping a clear view on the variability in the considered configurable system.

Autonomous Learning at Run Time. The configuration space of the software can evolve at run time (*e.g.,* addition/removal of a device, evolving workload, connectivity loss, etc.) and may imply to update the learning model. But updating a model is costly, even more at run time with limited resources and time. There is a need for a self-configurable learning process that would learn from previous system evolution if and what *region* of the model has to be updated. One thus need mechanisms to decide (at the learning level) if the system evolution is relevant or significant enough to update the learning model (*e.g.,* based on historical data of past evolution) and if so, optimize this update by only changing parts or regions of the model impacted by the system's changes. This would result in a self-adaptive learning process in charge of properly configuring the learning process of the evolving system's configuration space. To the best of our knowledge, there is no approach that suggests what, when and how to learn from an evolving configuration space. By addressing these challenges, we believe

that the next generation of learning processes will be self-adaptive, either in a completely autonomous manner or with possible supervisions of users/experts.

4 Conclusion and Research Directions

Learning a model for a highly configurable software system is a time and energy consuming process that most systems cannot afford to repeat over time during their life-cycle. But since the configuration space of the system is likely to evolve over time and over external factors, the model will have to be learned again anyway to stay consistent with the system and its functional and performance properties. To address this issue, we consider the learning process as a configurable system, which can thus be fine-tuned with respect to what can be learned, how it can be learned and when it can be learned. We would like to investigate and propose a series of innovative tools and approaches to manage such a configurable learning, both at design time in a continuous manner and at run time in an autonomous manner.

References

1. Acher, M., Cleve, A., Collet, P., Merle, P., Duchien, L., Lahire, P.: Extraction and evolution of architectural variability models in plugin-based systems. Software and Systems Modeling **13**(4), 1367–1394 (2014). https://doi.org/10.1007/s10270-013-0364-2, https://doi.org/10.1007/s10270-013-0364-2
2. Acher, M., Martin, H., Pereira, J.A., Blouin, A., Jézéquel, J.M., Khelladi, D.E., Lesoil, L., Barais, O.: Learning Very Large Configuration Spaces: What Matters for Linux Kernel Sizes. Research report (Oct 2019), https://hal.inria.fr/hal-02314830
3. Alves Pereira, J., Martin, H., Acher, M., Jézéquel, J.M., Botterweck, G., Ventresque, A.: Learning Software Configuration Spaces: A Systematic Literature Review. Tech. rep. (2019)
4. Guo, J., Czarnecki, K., Apel, S., Siegmund, N., Wasowski, A.: Variability-aware performance prediction: A statistical learning approach. In: 28th International Conference on Automated Software Engineering (ASE). pp. 301–311 (2013)
5. Jamshidi, P., Velez, M., Kästner, C., Siegmund, N.: Learning to sample: Exploiting similarities across environments to learn performance models for configurable systems. In: 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 71–82 (2018)
6. Martinez, J., Ziadi, T., Bissyandé, T.F., Klein, J., Traon, Y.L.: Bottom-up adoption of software product lines: a generic and extensible approach. In: Proceedings of the 19th International Conference on Software Product Line. pp. 101–110 (2015)
7. Metzger, A., Quinton, C., Mann, Z.Á., Baresi, L., Pohl, K.: Feature-model-guided online learning for self-adaptive systems. CoRR **abs/1907.09158** (2019)
8. Sarkar, A., Guo, J., Siegmund, N., Apel, S., Czarnecki, K.: Cost-efficient sampling for performance prediction of configurable systems. In: 30th IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 342–352 (2015)
9. Temple, P., Acher, M., Jézéquel, J., Barais, O.: Learning contextual-variability models. IEEE Software **34**(6), 64–70 (2017)