ICO: A Platform for Optimizing Highly Configurable Systems

Edouard Guégain, Amir Taherkordi, Clément Quinton

ICO: A Platform for Optimizing Highly Configurable Systems

Edouard Guégain*, Amir Taherkordi†, Clément Quinton*
*Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
†University of Oslo, Norway
*{edouard.guegain;clement.quinton}@univ-lille.fr; †amirhost@ifi.uio.no

***Abstract*—Dealing with large configuration spaces is a complex task for developers, especially when manually searching for the configuration that best suits both their functional and performance requirements. Indeed, a well-performing configuration may not fit developers' needs because of conflicting functional requirements, or *vice-versa*. In this paper, we propose ICO, a lightweight, domain-agnostic platform that supports multi-objective optimization for configurable software. The purpose of ICO is to provide the developer with the best-performing configuration by altering as little as possible the initial one, in order to remain as close as possible to the developer's functional requirements. We explain the foundations of ICO, describe its architecture, and explain how it can be used either through a command-line client or an Eclipse plugin. Finally, we assess ICO by evaluating its execution time and the time saved to users compared to a manual optimization.**

***Index Terms*—tools, software, variability, optimization, performance**

I. Introduction

Our society relies on numerous software systems that exhibit a large set of functionalities, features, and parameters. This variability aims at addressing various needs of our daily lives and satisfies as many requirements as possible. That is, modern software systems can exhibit millions of configurations [1], *i.e.*, a large configuration space. For developers, managing such highly configurable systems comes with the challenges of *(i)* dealing with the huge number of configurations and *(ii)* providing the best possible software to end-users.

Although the configuration satisfying the developer's requirements (*i.e.*, containing all the desired features) may be valid, it is not possible to ensure that this configuration is optimal regarding given performance indicators. That is, it is not possible to manually ensure that, among millions of other valid configurations, the current configuration also satisfies performance requirements in terms of, *e.g.*, memory footprint, response time, or energy consumption.

In this paper, we thus propose the *Iterative Configuration Optimizer* (ICO) platform. ICO provides developers with automated support for finding configurations that meet their functional requirements (*i.e.*, features that have to be included or excluded) while also efficiently complying with specified performance indicators.

II. Background and Motivation

Managing software variability and performance is a difficult task: configuration spaces can be large and manual inspection of the performance of each configuration is not a viable option. A practical way to encode software variability is by means of feature models [2]. A feature model, such as the one in Figure 1, is a tree of hierarchically organized features that describes – with the help of cross-tree constraints expressing inter-feature dependencies – the possible and allowed feature combinations, *i.e.*, the software configurations. Automated analysis of feature models provides support for checking the validity of a configuration, in particular regarding the set of features that developers decide to include or exclude from such configurations [3].

There has been an effort from the software variability community to tackle the issue of creating high-performing and valid configurations, by providing various tool supports [4]–[7]. However such tools usually exhibit one of the two following drawbacks:

- Ad-hoc mechanisms and algorithms, requiring an upfront development investment to be used in different contexts or with different performance indicators [4]–[6];
- Large all-in-one tools, designed to handle a wide range of projects but requiring users to go through a steep learning curve and to deal with numerous technical requirements [7].

Both industrial and academic practitioners and researchers are thus missing a simple, turnkey solution to manage and optimize their software configurations. To address the limitations previously described, we thus propose a generic, easy-to-use, and lightweight platform, ICO.

III. Related Work

Numerous researchers have proposed performance prediction approaches with the objective of creating estimated performance models for system features [8]–[10]. These techniques rely on machine learning to infer performance data from configuration samples, aiming *(i)* to detect feature interactions that significantly impact system performance and *(ii)* provide more accurate predictions compared to models that neglect such interactions. In particular, Siegmund *et al.* predict configuration performance by summing up the impact of each feature [11], [12].

In the context of performance optimization for configurable systems, various methods have been suggested to identify optimal or near-optimal configurations based on specific performance indicators. Deterministic approaches have been ex-

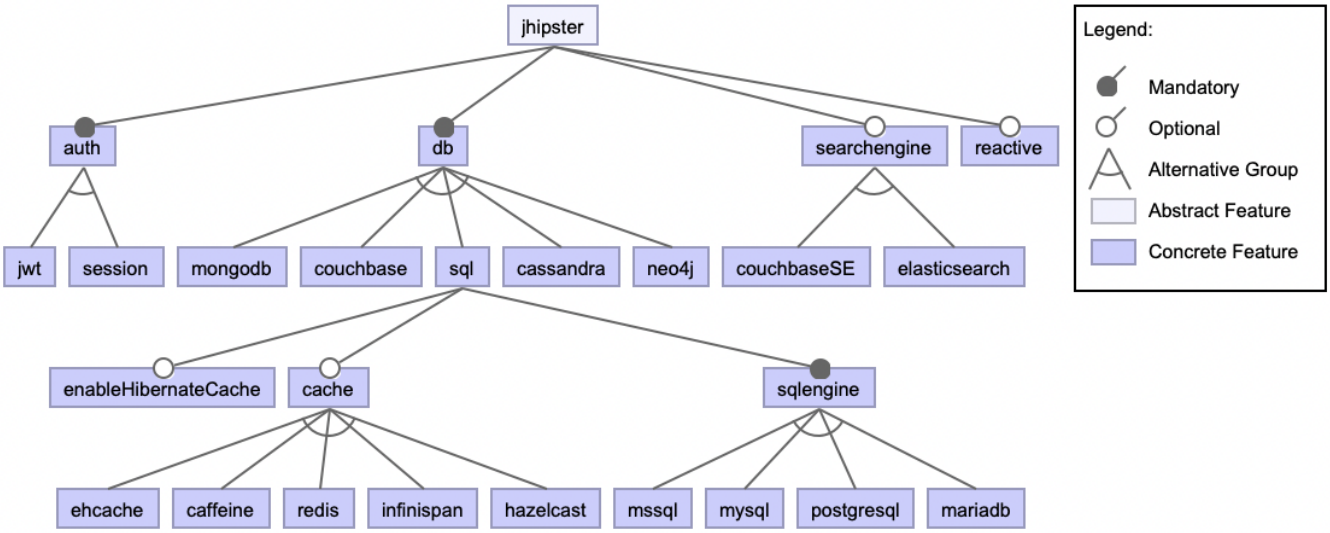

Fig. 1: An excerpt of a feature model.

plored [13]–[15], as well as genetic algorithms, such as the one used by Hierons *et al.* [16], to minimize the number of measurements required for optimization. Other authors leverage performance prediction techniques as mentioned earlier [7].

However, these approaches do not consider an initial configuration. For example, Nair *et al.* initiate their optimization process from a random configuration [15]. In contrast, the approach presented in this paper focuses on optimizing a set of performance indicators while staying as close as possible to the user-defined initial configuration. This optimization objective is similar to that of Soltani *et al.*, who consider user preferences to optimize a configuration, but their approach does not support the optimization of pre-existing configurations [17].

IV. The ICO Platform

ICO is built upon the approach we presented in [18]. This approach is designed to compute the performance of individual features and aims at optimizing the performance of configurations containing such features. Specifically, our approach aims at maximizing performance gains while minimizing changes to the current configuration, as described in [19], contrary to existing optimization methods that strive to identify the best possible configuration. This minimization of changes ensures applicability in critical environments, where any configuration change is a liability. With ICO, we thus provide the implantation of such an algorithm as well as the related automated support.

In particular, the scope of ICO is to optimize a configurable system regarding any functional or non-functional performance indicator as long as its configuration space is encoded as a feature model. Moreover, it is admitted in the community that interactions between features can impact performances [20], [21], and that it is possible to improve performance through interactions between pairs of features [18]. Thus, ICO also supports the optimization of performances through pairwise interactions. Furthermore, ICO supports multi-objective optimization, where performance is evaluated based on multiple indicators *e.g.*, both the power usage and memory footprint. In the remainder of the paper, *performance indicators* is used as a placeholder referring to either a unique indicator or a composition of indicators. Finally, while the approach described in [18] is tested against one use case, ICO is domain agnostic and can be used for any configurable software whose variability is represented by a feature model. Such a model can be defined at design time, during the specification of the software, or built a posteriori, relying on domain knowledge and an analysis of the documentation and source code. Recent works allow for partial automation of this process [22], [23].

V. Implementation

To address the limitations described in Section II, we thus propose a generic, easy-to-use, and lightweight tool, ICO, as an implementation of the previously proposed approach. The ICO platform[1] is a set of software components interacting together to help developers optimize the configuration – *w.r.t.* performance indicators – of the software being developed. The ICO platform is composed of three tools:

- ICOLIB, a library that performs the optimizations;
- ICOCLI, a command-line tool to interact with ICOLIB;
- ICOPLUGIN, an Eclipse plugin to interact with ICOLIB.

Figure 2 presents the architecture of ICO. ICO executes the user's instructions regarding *(i)* the configuration to optimize, *(ii)* the feature model encoding the configuration space of the software, and *(iii)* its related performance data files.

[1]The source code is available at https://gitlab.inria.fr/ico

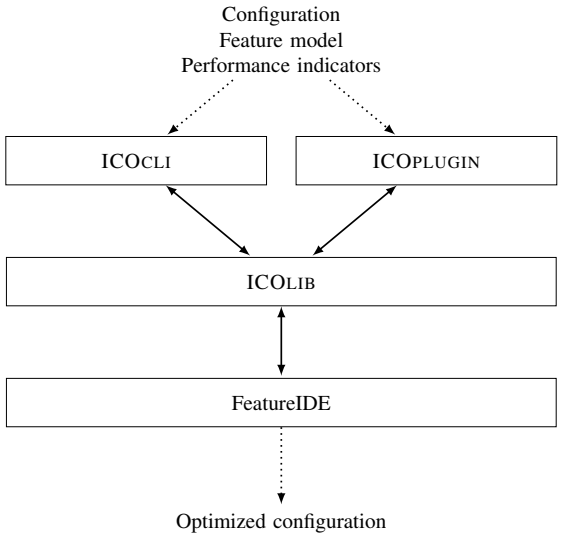

Fig. 2: The architecture of the ICO platform.

The architecture of the platform is flexible enough to be extended by any front-end components interacting with ICOLIB. These components take as input performance data as CSV files. Performance files describe the performance of individual features as well as the performance of pairs of features, in order to take feature interactions into account [20], [21]. Such data can be a direct assessment of the feature's performances, *e.g.*, the number of lines of code, or an evaluation of their impact on configurations' performances, *e.g.*, time or energy. Through either ICOCLI or ICOPLUGIN, the user's instructions are sent to ICOLIB which in turn relies on the FeatureIDE library [24] to perform an automated analysis of the configurations. In particular, the library checks the validity of the resulting optimized configurations returned by ICOLIB. Relying on this library also makes ICO more versatile, as it provides support for a wide range of feature models and configuration file formats, such as CNF and DIMACS for feature models or XML and Equation for configurations.

Based on the user's inputs (*e.g.*, features that have to be included or excluded for functional reasons), ICOLIB provides suggestions to improve the current configuration by maximizing its performance indicators. Precisely, ICO suggests either an addition or removal of a feature, or a substitution of a feature with another one. Such suggestions are presented to the user either after completing the configuration or in real-time during the configuration process, *e.g.*, by indicating which feature should be added next to make the configuration both valid and more efficient. As such, ICO can thus be considered both an optimizer and a recommender system [25], [26].

A. ICOlib

ICOLIB is the central component of the platform. It provides a facade exposing the API handling all operations that can be performed with ICO: `load` a project, `display` current performances, `manage` constraints (*i.e.*, the lists of features that have to be included or excluded from the configuration), `list` or `apply` improvement suggestions and `save` the new configuration. While managing constraints and listing/applying suggestions are core functionalities of ICO, the processes of loading, validating, and saving configuration are delegated to the FeatureIDE library. Relying on the strategy pattern, ICOLIB is provided with two optimization approaches, implementing respectively the feature-wise and pairwise approaches from [18]. Thanks to this architectural design, new optimization algorithms can seamlessly be integrated to ICO. The time required to generate suggestions is highly dependent on the structure of the feature model, especially on the number of features and cross-tree constraints (and their complexity). However, as suggestions are independent of each other, their generation can be parallelized. ICOLIB current Java implementation relies on `Stream.parallelStream()` for such a task.

The ICO platform comes with two ICOLIB clients: the first one as an Eclipse[2] Plugin – ICOPLUGIN; the second one as a command line tool – ICOCLI.

B. ICOPlugin

ICOPLUGIN is an Eclipse plugin developed to interact with ICOLIB and implemented as an Eclipse view. Whenever a configuration file is selected by the user, the ICOPLUGIN view displays the five following tabs:

- Features, to list and apply feature-based suggestions;
- Interactions, to list and apply interaction-based suggestions;
- Constraints, to manage the exclusion and inclusion of features;
- Details, to monitor the current configuration performances;
- Logs, to monitor ICOLIB execution logs.

Figure 3 shows the Interactions suggestion tab of ICOPLUGIN. Both Features and Interactions tabs give access to the Apply all suggestions and Find suggestions functionalities of their respective mode. When listed, each suggestion exhibits an Apply button to let the developer perform the proposed suggestion. The optimized configuration belongs to the configuration space encoded by the feature model presented in Figure 1. The suggestions are ordered according to their improvement rate, *i.e.*, applying the first suggestion will have the most effective impact regarding the considered performance indicators. While each suggestion can be applied individually by clicking on its related Apply button, the Apply all suggestions functionality automates the process of applying all the best suggestions at once. Precisely, ICO computes new suggestions every time the configuration is improved and applies the best suggestion recursively until no further improvements can be made. Suggestions must be recomputed after each iteration, as the changes applied to the configuration may turn other recommendations obsolete.

[2]https://www.eclipse.org/

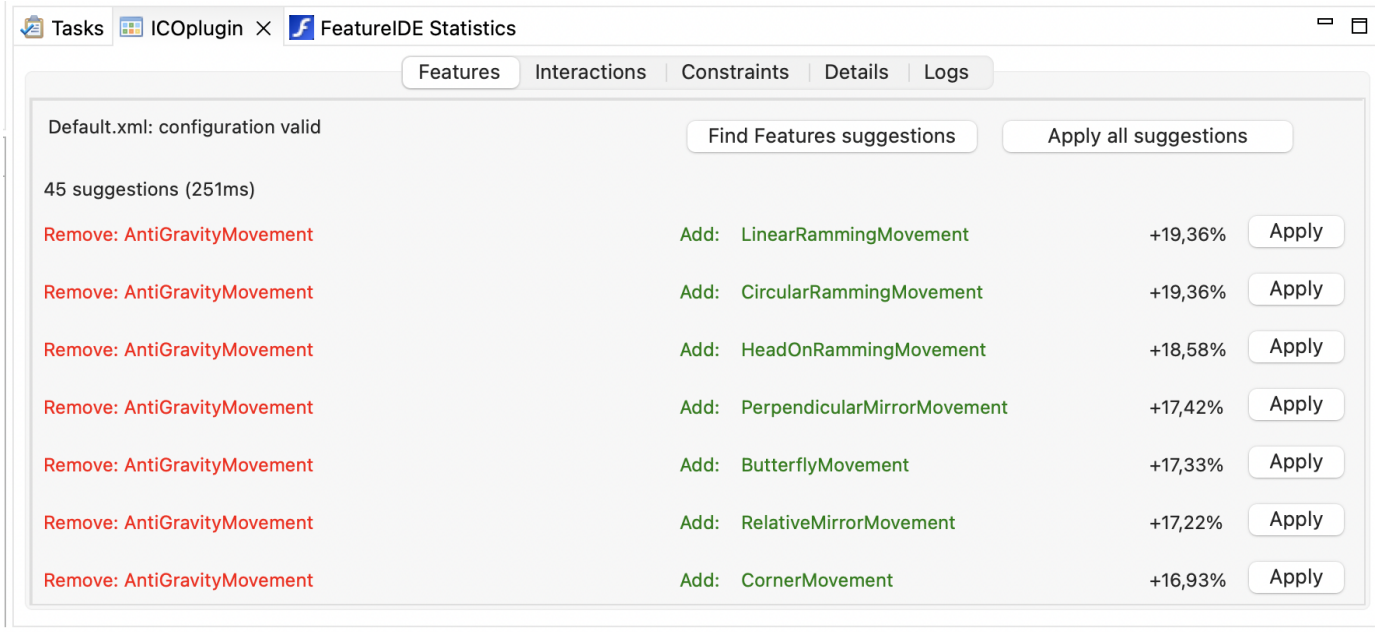

Fig. 3: A list of suggestions in ICOPLUGIN.

Listing 1: Single-line mode

```
./ICOcli -P ./project -C config.xml -e feat1,feat2 -i
    ↪ feat3 -F -A -S -N
```

Listing 2: Interactive mode

```
./ICOcli
> load -P ./project -C config.xml
> exclude --add feat1,feat2
> include --add feat3
> apply -F -A
> save
> exit
```

Fig. 4: The two modes of ICOCLI.

C. ICOCLI

Another means to interact with ICOLIB is by using ICOCLI, a lightweight command line interface program. ICOCLI proposes two modes: the single-line mode, where instructions are given as parameters, and the interactive mode where instructions are given sequentially by the developer in a shell. It is also possible to mix the two modes by giving some instructions as parameters (*e.g.*, the project path), and then running the remaining ones in the shell. The single-line mode is relevant for automation (*e.g.*, for CI/CD or in an automated process) or to perform a single task, whereas the interactive mode provides a more human-friendly interaction with the tool, enabling an in-depth exploration of the variability of the software and its performances, *e.g.*, comparing the performance of features and interactions or the impact of constraints on suggestions.

Figure 4 presents the same set of tasks computed in the two different modes. The developer loads the software project `./project` and the configuration file `./project/config.xml` respectively with parameters `-P` and `-C`. Then, features `feat1` and `feat2` are added to the exclusion list and feature `feat3` is added to the inclusion list using parameters `-e` and `-i` in single-line mode and the `exclude`/`include` commands in interactive mode. All the best feature-related suggestions are then applied with parameters `-F -A` in single-line mode or by using command `apply -F -A` in interactive mode. Finally the configuration `./default.xml` is overwritten by `-S` in single-line mode or `save` in an interactive mode. The developer quits the program with `exit` in interactive and mixed modes, while in single-line mode the `-N` parameter prevents the shell from opening.

Finally, ICOCLI offers the ability to perform arbitrary edits to the configuration (*e.g.*, removing a feature), a functionality also offered by ICOPLUGIN through the FeatureIDE API.

VI. EVALUATION

In our previous work [18], [19], we conducted empirical evaluations to assess the accuracy and effectiveness of our optimization algorithms when optimizing the highly configurable systems RobocodeSPL and GPL-FH-java. In this paper, we complete these evaluations with an assessment of the implementation of the ICO platform, especially regarding its time efficiency and usability.

A. Internal Evaluation

The time efficiency of ICO is assessed through several configurable software of different sizes. In particular, we monitored the time that ICO needs to return a list of suggestions over the configurable software GPL-FH-java, JHipster, PPU, TightVNC, RobocodeSPL, and axTLS. For JHipster and GPL-FH-java, all the valid configurations were assessed by ICO. For larger systems PPU, TightVNC, Robocode, and axTLS,

50 configurations were randomly[3] sampled and evaluated. The considered performance indicators are the boot time for JHipster, the size and execution time for GPL-JH-java [19], and the energy consumption for Robocode [18], whose related performance data were monitored during our previous work. Regarding PPU, TightVNC, and axTLS, the performance data were randomly generated. All the measures were performed on an Intel i5 CPU @ 2GHz with 16GB of RAM. Table I summarizes the average response time of ICO for each of the assessed configurable systems.

	Features	Products	median time (s)
JHipster	26	118	0.060
GPL-FH-java	38	156	0.093
PPU	52	28800	0.053
TightVNC	28	295201	0.209
Robocode	72	$1,3e+6$	0.263
axTLS	96	$8.3e+11$	55.679

TABLE I: The response time of ICO.

In the smallest feature models JHipster, GPL-FH-java, and PPU, the execution time is trivial. The median generation time for TightVNC and Robocode is higher but remains limited for a human user. However, larger feature models are more challenging to tackle, as the parallelization of the implementation reaches the limits of the machine. Therefore, in future work, we plan to refine the implementation of parallelization, for instance by leveraging GPU resources. We also plan to implement heuristics to reduce the number of invalid candidates that are generated and then dismissed.

B. External validation

The purpose of the ICO platform is to automate the optimization of configurations, in order to save time and to reduce the error rate of this process. We conducted a survey to determine how developers could benefit from using ICO when trying to find the best suitable configuration in a large configuration space, with respect to their functional requirements (*i.e.*, the options to include or exclude) and non-functional requirements (*i.e.*, the performance of the system). Precisely, 15 software developers, with 5 to 20 years of professional experience, were asked to perform the same experiment with and without ICO.

We provided the participants with a configuration file and the following question: *"How to improve the configuration of this software?"*. The configurable system is Robocode, and the performance indicator to optimize is the energy consumption of the system. The energy consumption of the system was assessed in [18], and the performance data of each option was provided to the user.

The experiment consists in comparing the time saved by optimizing a configuration with ICO instead of a tool-less approach. Specifically, the users were given two tasks: improving a specific configuration a first time *w.r.t.* feature performances and a second time *w.r.t.* interaction performances. They were asked to perform these two tasks first by relying on a spreadsheet containing the performance data, and then by using ICOPLUGIN.

The participants received an explanation of the usage of the spreadsheet and ICOPLUGIN, and they performed the tasks first on a toy configuration before the actual evaluation.

Task \ Avg. time	Without ICO	With ICO	Gain
Feature-wise	131,9s	14,3s	89%
Interaction-wise	164,7s	10,1s	94%

TABLE II: The time to optimize a configuration by the users.

Table II presents the results of this evaluation. Regarding the feature-wise optimization, the average time to perform the experiment was 131.9 seconds without ICO and 14,3 seconds with ICO, resulting in an 89% time reduction. As of the interaction-wise optimization, the average time to perform the experiment was 164,7 seconds without ICO and 10,1 seconds with ICO, *i.e.*, a 94% time reduction. The average time to perform the experiment without ICO increased between the feature-wise optimization and the interaction-wise one, as the latter is more complicated. Contrarily, it remained consistent when using ICO as both optimizations take the form of a similar task in ICOPLUGIN.

ICO thus offers substantial time savings on the optimization process, while requiring a very minimal learning step. Such results tend to confirm the relevance of ICO as cost-effective and easily accessible optimization tool.

VII. CONCLUSION

In this paper, we presented ICO, a platform designed to improve the performance of a configuration. Our approach extends our previous work [18] and provides developers with suggestions to improve the performances of their configurations. We conducted a preliminary evaluation assessing the time performance of ICO and how it can be used in practice by developers to save time during the configuration and optimization of their software. In future work, we plan to develop a plugin to be integrated with IntelliJ IDEA to address a wider range of developers. We are also considering improving the parallelization process (used for generating suggestions) by leveraging GPU with the CUDA framework [27], in order to tackle larger feature models.

ACKNOWLEDGEMENTS

The research leading to these results received funding from French Research Agency through the ANR-19-CE25-0003 KOALA project and from the Norwegian Research Council through the DILUTE project (Grant No. 262854/F20).

[3]Random according to FeatureIDE product generator.

REFERENCES

[1] H. Martin, M. Acher, L. Lesoil, J. M. Jezequel, D. E. Khelladi, J. A. Pereira, Transfer learning across variants and versions : The case of linux kernel size, IEEE Transactions on Software Engineering (2021).

[2] A. Metzger, C. Quinton, Z. Á. Mann, L. Baresi, K. Pohl, Feature model-guided online reinforcement learning for self-adaptive services, in: E. Kafeza, B. Benatallah, F. Martinelli, H. Hacid, A. Bouguettaya, H. Motahari (Eds.), Service-Oriented Computing, Springer International Publishing, Cham, 2020, pp. 269–286.

[3] D. Benavides, S. Segura, A. Ruiz-Cortés, Automated analysis of feature models 20 years later: A literature review, Information Systems 35 (2010) 615–636.

[4] J. Rodas-Silva, J. A. Galindo, J. Garcia-Gutierrez, D. Benavides, Selection of software product line implementation components using recommender systems: An application to wordpress, IEEE Access 7 (2019) 69226–69245.

[5] J. Guo, J. White, G. Wang, J. Li, Y. Wang, A genetic algorithm for optimized feature selection with resource constraints in software product lines, Journal of Systems and Software 84 (2011).

[6] X. Lian, L. Zhang, J. Jiang, W. Goss, An approach for optimized feature selection in large-scale software product lines, Journal of Systems and Software 137 (2018) 636–651.

[7] N. Siegmund, M. Rosenmüller, M. Kuhlemann, C. Kästner, S. Apel, G. Saake, Spl conqueror: Toward optimization of non-functional properties in software product lines, Software Quality Journal (2012) 487–517.

[8] C. Kaltenecker, A. Grebhahn, N. Siegmund, S. Apel, The interplay of sampling and machine learning for software performance prediction, IEEE Software 37 (4) (2020) 58–66.

[9] J. Guo, K. Czarnecki, S. Apel, N. Siegmund, A. Wasowski, Variability-aware performance prediction: A statistical learning approach, in: 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2013, pp. 301–311.

[10] N. Siegmund, A. Grebhahn, S. Apel, C. Kästner, Performance-influence models for highly configurable systems, in: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, 2015, p. 284–294.

[11] N. Siegmund, S. S. Kolesnikov, C. Kästner, S. Apel, D. Batory, M. Rosenmüller, G. Saake, Predicting performance via automated feature-interaction detection, in: Proceedings of the 34th International Conference on Software Engineering, ICSE '12, Zurich, Switzerland, 2012, p. 167–177.

[12] N. Siegmund, M. Rosenmüller, C. Kästner, P. G. Giarrusso, S. Apel, S. S. Kolesnikov, Scalable prediction of non-functional properties in software product lines: Footprint and memory consumption, Information and Software Technology 55 (3) (2013) 491–507.

[13] I. Švogor, I. Crnković, N. Vrček, An extensible framework for software configuration optimization on heterogeneous computing systems: Time and energy case study, Information and Software Technology 105 (2019) 30–42.

[14] R. Olaechea, S. Stewart, K. Czarnecki, D. Rayside, Modelling and multi-objective optimization of quality attributes in variability-rich software, in: Proceedings of the fourth international workshop on nonfunctional system properties in domain specific modeling languages, 2012, pp. 1–6.

[15] V. Nair, Z. Yu, T. Menzies, N. Siegmund, S. Apel, Finding faster configurations using flash, IEEE Transactions on Software Engineering 46 (7) (2020) 794–811.

[16] R. M. Hierons, M. Li, X. Liu, S. Segura, W. Zheng, Sip: Optimal product selection from feature models using many-objective evolutionary optimization, ACM Trans. Softw. Eng. Methodol. 25 (4 2016).

[17] S. Soltani, M. Asadi, D. Gašević, M. Hatala, E. Bagheri, Automated planning for feature model configuration based on functional and non-functional requirements, in: Proceedings of the 16th International Software Product Line Conference-Volume 1, 2012, pp. 56–65.

[18] E. Guégain, C. Quinton, R. Rouvoy, On reducing the energy consumption of software product lines, Proceedings of the 25th ACM International Systems and Software Product Line Conference - Volume A (2021) 89–99.

[19] E. Guégain, A. Taherkordi, C. Quinton, Configuration optimization with limited functional impact, in: International Conference on Advanced Information Systems Engineering, Springer, 2023, pp. 53–68.

[20] N. Siegmund, S. S. Kolesnikov, C. Kästner, S. Apel, D. Batory, M. Rosenmüller, G. Saake, Predicting performance via automated feature-interaction detection, Proceedings of the 34th International Conference on Software Engineering (2012) 167–177.

[21] M. Calder, A. Miller, Feature interaction detection by pairwise analysis of ltl properties—a case study, Formal Methods in System Design 28 (2006) 213–261.

[22] J. Martinez, D. Wolfart, W. K. Assunção, E. Figueiredo, Insights on software product line extraction processes: argouml to argouml-spl revisited, in: Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A-Volume A, 2020, pp. 1–6.

[23] G. K. Michelon, L. Linsbauer, W. K. Assunção, S. Fischer, A. Egyed, A hybrid feature location technique for re-engineeringsingle systems into software product lines, in: Proceedings of the 15th International Working Conference on Variability Modelling of Software-Intensive Systems, 2021, pp. 1–9.

[24] T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake, T. Leich, Featureide: An extensible framework for feature-oriented software development, Science of Computer Programming 79 (2014) 70–85.

[25] J. A. Pereira, P. Matuszyk, S. Krieter, M. Spiliopoulou, G. Saake, A feature-based personalized recommender system for product-line configuration, Proceedings of the 2016 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (2016).

[26] J. A. Pereira, S. Schulze, S. Krieter, M. Ribeiro, G. Saake, A context-aware recommender system for extended software product line configurations, Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems (2018).

[27] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, J. C. Phillips, Gpu computing, Proceedings of the IEEE 96 (2008) 879–899.