The ICO Tool Suite: Optimizing Highly Configurable Systems

Edouard Guégain, Amirhosein Taherkordi, Clément Quinton

The ICO Tool Suite: 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 some given requirements. In this paper, we present the ICO tool suite, a set of software components designed to help developers optimize their configuration *w.r.t* a set of performance indicators. This tool suite can be used through a command-line client or an Eclipse plugin, while the loose coupling with its core library make it flexible-enough to be integrated into existing tools. ICO is assessed by evaluating the time it saves to users over a manual optimization.**

I. Introduction

Our society relies on numerous software systems that exhibit a large set of functionalities, options and parameters. This variability aims at addressing various needs of our daily lives and satisfy as many requirements as possible. That is, modern software systems can exhibit millions of configurations [1], *i.e.*, a large configuration space. As a developer, 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.

A practical way to encode large configuration spaces is by means of feature models [2]. A feature model, such as the one in Figure 2a, is a tree of hierarchically organized features which 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 developers decide to include or exclude from such configurations [3].

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

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

II. Why ICO?

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. There has been an effort from the software variability community to address these issues 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 indicators [4]–[6];
- Large all-in-one tools, built with the intent of dealing with any project, but requiring *(i)* to go through a steep learning curve and *(ii)* to deal with a lot of 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 tool suite, ICO.

ICO is built upon the approach presented in [8]. This approach was designed to compute the energy consumption of features and aims at optimizing the energy consumption of configurations containing such features. Specifically, our approach aims at maximizing performance gains while minimizing changes to the existing configuration [9], contrary to existing optimization approaches that tend to search for the best possible configuration.

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, ICO supports multi-objective optimization with performances composed of several indicators, *e.g.*, energy consumption, 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 [8] is tested against one use case, ICO is domain agnostic and can be used with any configurable software represented by a feature model.

III. The ICO Tool Suite

The ICO tool suite[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 tool suite 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.

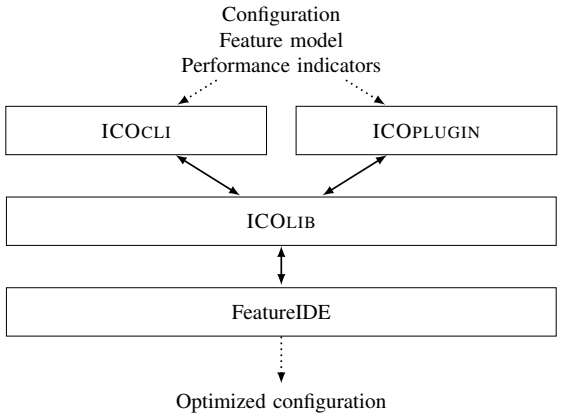


Fig. 1: The architecture of the ICO tool suite.

Figure 1 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.

The architecture of the tool suite 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 can describe the performance of individual features as well as the performance of pairs of features, in order to take feature interactions into account [10], [11]. 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 [12] 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 model 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. These suggestions can be the addition or removal of a feature, or a substitution of a feature with another one. They are provided both for a completed configuration, or dynamically *while* configuring by suggesting 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 [13], [14].

A. ICOlib

ICOLIB is the central component of the tool suite. 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 [8]. 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.

Two ICOLIB clients are provided: the first one as an Eclipse Plugin (ICOPLUGIN), and the second one as a command line tool (ICOCLI).

B. ICOPlugin

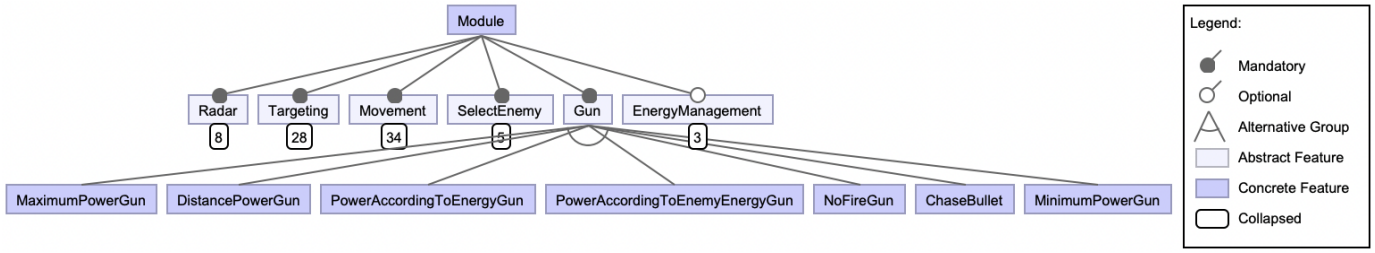
ICOPLUGIN is an Eclipse[2] plugin developed to interact with ICOLIB and implemented as an Eclipse view. Whenever a configuration file is selected by the user, the ICOPLUGIN view displays the five following tabs:

- 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.

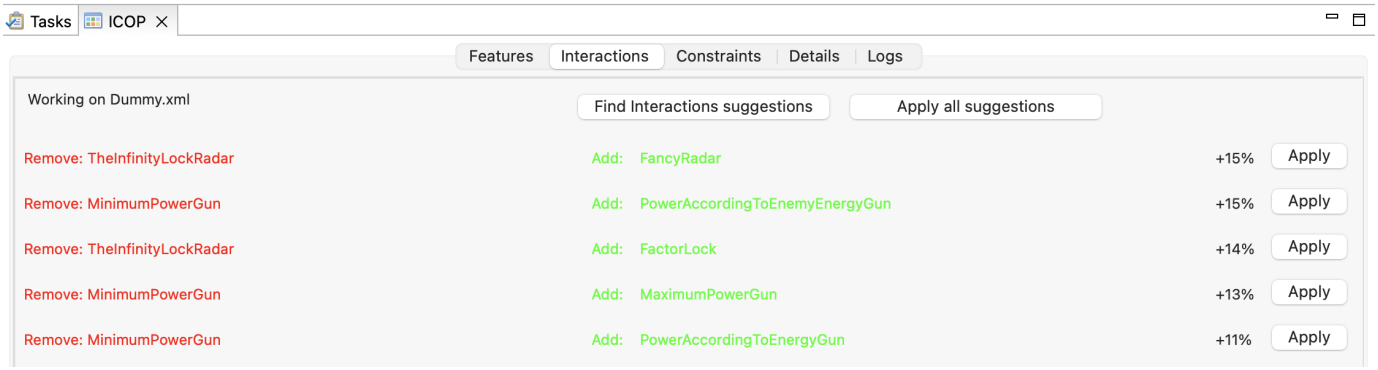
Both Features and Interactions suggestion tabs give access to the Apply all suggestions and Find suggestions functionalities of their respective mode. When listed, each suggestion exhibits an Apply button to let the developer perform the proposed suggestion. Figure 2 shows the Interactions tab of ICOPLUGIN. The configuration to optimize belongs to the configuration space encoded by the feature model presented in Figure 2a. Figure 2b is a screenshot of the Interactions tab of ICOPLUGIN providing the user with a list of suggestions. The

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

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



(a) An excerpt of a feature model.



(b) The interaction suggestion tab of ICOPLUGINwhile optimizing a configuration from the feature model.

Fig. 2: Usage of ICOPLUGIN

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. Each suggestion can be applied individually by clicking on its related Apply button. The Apply all suggestions button is used to apply all the best suggestions in an automated way, by recomputing new suggestions for the improved configuration and applying the best suggestion recursively[3].

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 the remaining instructions 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 suite, 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 3 presents the same set of tasks computed in the two different modes. The developer loads the software project `./project` and the configuration file `./project/default.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 the `apply -F -A` command in interactive mode. Finally the configuration `./default.xml` is overwritten by `-S` in single-line mode or `save` in 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.

ICOCLI also offers the ability to perform arbitrary modifications to the configurations, which could otherwise be done through FeatureIDE in ICOPLUGIN.

Listing 1: Single-line mode

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

Listing 2: Interactive mode

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

Fig. 3: The two modes of ICOCLI.

[3]Suggestions must be recomputed to avoid overwriting improvements *e.g.*, with suggestions "replace A by B (+15%)" and "replace A by C (+10%)", applying all suggestions without recomputing them would replace A by B and then by C, which is suboptimal.

IV. Evaluation

In our previous work [8], we conducted an empirical evaluation to assess the accuracy and effectiveness of our optimization algorithms when optimizing the energy consumption of the highly configurable system RobocodeSPL. In this paper, to assess ICO, which implements this algorithm, we extend that previous evaluation with additional experiments on the same configurable system, Robocode. This system is composed of 92 features which be derived into 1.3×10^6 configurations. The observed performance indicator is the energy consumption of the configurations.

We conducted a survey to determine how developers could benefit from using ICO in their everyday work when dealing with a large configuration space and trying to find the best suitable configuration with respect to their functional (*i.e.*, features to be included or excluded) and non-functional (*i.e.*, performance indicators) requirements. 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 experiment consists in comparing the time saved by optimizing a configuration with ICO instead of manually. 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 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 a first time on a toy configuration before the actual evaluation.

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

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

Table I 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 an easily accessible optimization tool.

V. Conclusion

In this paper, we presented ICO, a tool suite designed to improve the performance of a configuration. Our approach extends our previous work [8] and provides developers with suggestions to improve the performances of their configurations. We conducted a preliminary evaluation showing how ICO can be used in practice by developers to save time during the configuration and optimization of their software. As future work, we plan to develop a plugin to be integrated with IntelliJ IDEA to address a wider range of developers and we are considering improving the parallelization process (used for generating suggestions) by leveraging GPU with the CUDA framework [15], in order to tackle larger feature models.

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] M. Lienhardt, F. Damiani, E. B. Johnsen, J. Mauro, Lazy product discovery in huge configuration spaces, in: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, ICSE '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 1509–1521.

[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] Édouard 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.

[9] E. Guégain, A. Taherkordi, C. Quinton, Configuration optimization with limited functional impact, in: Proceedings of the 35th International Conference on Advanced Information Systems Engineering, CAiSE 2023, Lecture Notes in Computer Science.

[10] 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.

[11] 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.

[12] 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.

[13] 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).

[14] 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).

[15] 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.