Performance Prediction in the Presence of Feature Interactions
– Extended Abstract –

Norbert Siegmund,[1] Sergiy Kolesnikov,[1] Christian Kästner,[2] Sven Apel,[1] Don Batory,[3] Marko Rosenmüller,[4] and Gunter Saake[4]
[1]University of Passau, Germany, [2]Carnegie Mellon University, USA
[3]University of Texas at Austin, USA, [4]University of Magdeburg, Germany

1 Introduction. Customizable programs and program families provide user-selectable *features* allowing users to tailor the programs to the application scenario or platform. Beside functional requirements, users are often interested in non-functional requirements, such as a binary-size limit, a minimized energy consumption, and a maximum response time. To tailor a program to non-functional requirements, we have to know in advance which feature selection, that is, configuration, affects which non-functional properties. Due to the combinatorial explosion of possible feature selections, a direct measurement of all of them is infeasible.

In our work, we aim at *predicting* a configuration's non-functional properties for a specific workload based on the user-selected features [SRK$^+$11, SRK$^+$13]. To this end, we quantify the influence of each selected feature on a non-functional property to compute the properties of a specific configuration. Here, we concentrate on performance predictions only. Unfortunately, the accuracy of performance predictions may be low when considering features only in isolation, because many factors influence performance. Usually, a property, such as performance, is program-wide: it emerges from the presence and interplay of multiple features. For example, database performance depends on whether a search index or encryption is used and how both features interplay. If we knew how the combined presence of two features influences performance, we could predict a configuration's performance more accurately. Two features *interact* if their simultaneous presence in a configuration leads to an unexpected behavior, whereas their individual presences do not. We call feature interactions that affect performance *performance feature interactions*.

We improve the accuracy of predictions in two steps: (i) We detect which features interact and (ii) we measure to what extent they interact. In our approach, we aim at finding the sweet spot between prediction accuracy, measurement effort, and generality in terms of beeing independent of the application domain and the implementation technique. The distinguishing property of our approach is that we neither require domain knowledge, source code, nor complex program-analysis methods, and our approach is not limited to special implementation techniques, programming languages, or domains.

Our evaluation is based on six real-world case studies from varying domains (e.g., databases, encoding libraries, and web servers) using different configuration techniques. Our experiments show an average prediction accuracy of 95 percent, which is a 15 percent improvement over an approach that takes no interactions into account [SKK$^+$12].

2 Approach. We detect feature interactions in two steps: (a) We identify which features interact and (b) with heuristics, we search for the combination of these *interacting features* to pin down the actual feature interactions. Next, we give an overview of both steps.

Detecting Interacting Features. To identify which features interact, we quantify the performance contribution of each feature. Our idea to detect interacting features is as follows: First, we determine a feature's performance contribution in isolation (i.e., how a feature influences a program's performance when no other feature is present) – called minimal delta. Second, we determine a feature's contribution when combined with all other features – called maximum delta. Finally, we compare for each feature its minimal and maximal delta. Our assumption is, if the deltas differ from each other, then there must be, at least, one other feature that is responsible for this change. After applying this approach to all features, we know which features interact (but not in which specific combinations). The remaining task is to determine which combinations of these interacting features cause an actual feature interaction.

Heuristics to Detect Feature Interactions. To pin down performance feature interactions, we developed three heuristics based on our experience with product lines and previous experiments. We identify a feature interaction by predicting the performance of a certain feature combination and comparing the prediction against the actually measured performance. If the difference exceeds a certain threshold (e.g., to compensate for measurement bias), we found a feature interaction. Next, we shortly describe these heuristics.

- **Pair-Wise Interactions (PW)** – We assume that pair-wise interactions are the most common form of non-functional feature interactions. Hence, we measure all pair-wise combinations of interacting features (i.e., *not* all features) and compare them with our predictions to detect interactions.
- **Higher-Order Interactions (HO)** – We assume that second-order feature interactions (i.e., interactions among three features) can be predicted by analyzing already detected pair-wise interactions. The rationale is, if three features interact pair wise in any combination, they likely participate also in a triple-wise (second-order) interaction.
- **Hot-Spot Features (HS)** – We assume the existence of *hot-spot* features. In previous experiments, we identified that there are usually few features that interact with many features and there are many features that interact only with few features. Using this heuristic, we perform additional measurements to locate interactions of hot-spot features with other interacting features.

Using these heuristics, we performed a series of experiments with the six real-world case studies Berkeley DB Java, Berkeley DB C, SQLite, Apache web server, LLVM compiler infrastructure, and x264 video encoder. We found that applying these heuristics improves measurement accuracy from 80 %, on average, to 95 %, on average, which is within the measurement error.

References

[SKK+12] Norbert Siegmund, Sergiy Kolesnikov, Christian Kästner, Sven Apel, Don Batory, Marko Rosenmüller, and Gunter Saake. Predicting Performance via Automated Feature-Interaction Detection. In *Proc. ICSE*, pages 167–177. IEEE, 2012.

[SRK+11] Norbert Siegmund, Marko Rosenmüller, Christian Kästner, Paolo Giarrusso, Sven Apel, and Sergiy Kolesnikov. Scalable Prediction of Non-functional Properties in Software Product Lines. In *Proc. SPLC*, pages 160–169. IEEE, 2011.

[SRK+13] Norbert Siegmund, Marko Rosenmüller, Christian Kästner, Paolo Giarrusso, Sven Apel, and Sergiy Kolesnikov. Scalable Prediction of Non-functional Properties in Software Product Lines: Footprint and Memory Consumption. *Information and Software Technology*, 55(3):491–507, 2013.