Collaborative Recommendation of Search Heuristics For Constraint Solvers*

Damian Garber[1,*], Tamim Burgstaller[1], Alexander Felfernig[1], Viet-Man Le[1], Sebastian Lubos[1], Trang Tran[1] and Seda Polat-Erdeniz[1]

[1]Graz University of Technology, Inffeldgasse 16b, Graz, 8010, Austria

Abstract

Feature models (FM) support the management of variability properties of software, products, and services. To enable feature model configuration, these models have to be translated into a corresponding formal representation (e.g., a satisfiability or constraint satisfaction representation). Specifically in interactive configuration, efficient response times are crucial. In this paper, we show how to improve the performance of constraint solvers (supporting FM configuration) on the basis of exploiting the concepts of collaborative filtering for recommending solver search heuristics (variable orderings and value orderings). As a basis for our recommendation approach, we used data (configurations) synthesized from real-world feature models using different state-of-the-art synthesis approaches. A performance analysis shows that, with heuristics recommendation, significant improvements of solver runtime performance compared to standard solver heuristics can be achieved.

Keywords

Feature models, configuration, constraint solving, search heuristics, performance optimization, collaborative filtering

1. Introduction

Feature models (FMs) are in wide-spread use for modeling variability properties [1, 2]. These properties can be translated into a formal representation [3] to support various types of reasoning tasks, for example, in the context of feature model analysis and feature model configuration. In this paper, we focus on the aspect of *feature model configuration* where users of a configuration system define their preferences (e.g., in terms of intended feature inclusions) and the feature model configurator then tries to find a corresponding complete configuration which defines inclusion or exclusion for each feature.

Feature model configuration needs to be efficient which can become challenging specifically with large and complex configuration knowledge bases. The major means of improving the performance of solvers (specifically SAT and constraint solvers) is to employ different *search heuristics* which can help to cut down the search space as fast as possible. Following the idea of integrating machine learning (ML) with constraint solving [4], we propose to apply recommender systems [5], more specifically, collaborative filtering [6], to recommend solver search heuristics for a new FM configuration task.

A major precondition for implementing such a machine learning approach is the availability of training data that help to identify relevant heuristics. An issue in this context is the availability of datasets – this cannot be guaranteed specifically at the very beginning when an FM configurator has not been used that often. An alternative to datasets collected from real-world user interactions is data synthesis [7, 8]. In this paper, we focus on developing and comparing different configuration data synthesis strategies (see, e.g., Pereira et al. [8]) to gain deeper insights regarding the impact of the used strategies on the quality of the heuristics determined by our collaborative filtering approach. As discussed in Pereira et al. [8] (the focus of their work is performance prediction, for example, in video encoding scenarios), performance prediction is feasible, however, synthesizing high-quality and small sample data is a challenging task (see also [9]).

There exist a couple of approaches to integrate machine learning with constraint solving focusing on the aspect of identifying relevant heuristics on the basis of given datasets – see, for example, Erdeniz et al. [10] and Uta et al. [11]. These approaches focus on *predicting* relevant attribute values (features) for a user and – at the same time - *improving* constraint solver performance by choosing appropriate *variable value ordering heuristics*. In contrast to related work, we extend the recommendation scope by also supporting the identification of efficient *variable orderings*. At the same time, we analyze the impact of different data synthesis strategies on the quality of the recommended heuristics (which we regard as a new contribution to the fields of feature modeling and knowledge-based configuration).

ConfWS'23: 25th International Workshop on Configuration, Sep 6–7, 2023, Málaga, Spain
*Corresponding author.
dgarber@ist.tugraz.at (D. Garber);
tamim.burgstaller@ist.tugraz.at (T. Burgstaller);
alexander.felfernig@ist.tugraz.at (A. Felfernig);
vietman.le@ist.tugraz.at (V. Le); slubos@ist.tugraz.at (S. Lubos);
ttrang@ist.tugraz.at (T. Tran); spolater@ist.tugraz.at
(S. Polat-Erdeniz)
0009-0005-0993-0911 (D. Garber)

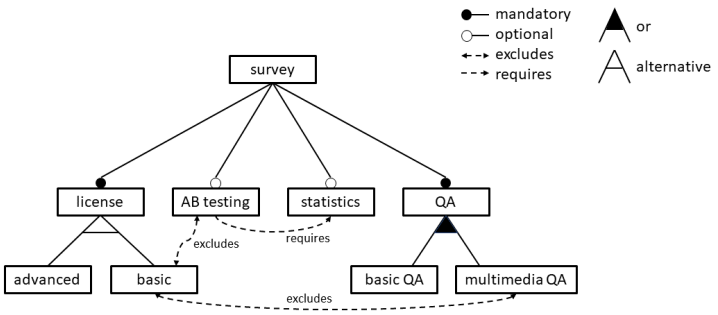

Figure 1: Example feature model of a *survey software* (based on Le et al. [12]).

The major contributions of this paper are the following. (1) we show how to apply configuration space learning concepts in FM configurator performance optimization. (2) our recommendation approach takes into account both, the recommendation of variable orderings and variable value orderings. (3) we compare different data synthesis strategies with regard to their applicability in search heuristics selection. (4) our evaluation results on the basis of real-world configuration (feature) models [13, 14] indicate significant performance improvements.

The remainder of this paper is organized as follows. In Section 2, we provide an example feature model with the related constraint-based representation. In Section 3, we introduce an approach to collaborative filtering based recommendation of constraint solver search heuristics. In Section 4, we provide an overview of the synthesis approaches we have used in our recommendation settings. Performance evaluation results are summarized in Section 5. Threats to validity are discussed in Section 6. The paper is concluded with a discussion of future research issues in Section 7.

2. Example Configuration Task

In the following, we introduce an example feature model representing the variability properties of a *survey software* (see Figure 1). Each configured *survey* software must have included a corresponding *license* model (which can be either *advanced* or *basic*). The features *statistics* and *ABtesting* are optional ones, i.e., must not be part of every configuration. Finally, each survey software configuration must include a selected interaction mode (in terms of the type of questions (feature *QA*) supported) which consists of at least one out of question answering (feature *basicQA*) and multimedia based question answering (feature *multimediaQA*).

The feature model in Figure 1 includes different relationships and cross-tree constraints $c_i \in C$. First, each survey software configuration must include the root feature (c_1 : $survey = true$) and either an advanced or basic license (c_2 : $survey \leftrightarrow license$ and c_3 : $license \leftrightarrow advanced \veebar basic$).[1] Furthermore, ABtesting and statistics are optional (c_4 : $ABtesting \rightarrow survey$ and c_5 : $statistics \rightarrow survey$). Each selected question mode must include at least one out of basic and multimedia (c_6 : $QA \leftrightarrow survey$ and c_7 : $QA \leftrightarrow basicQA \vee multimediaQA$). Finally, the FM includes a set of cross-tree constraints: basic licenses are incompatible with ABtesting (c_8 : $\neg(basic \wedge ABtesting)$), the inclusion of ABtesting requires the inclusion of statistics (c_9 : $ABtesting \rightarrow statistics$), and a basic license must not be combined with a multimedia answering mode (c_{10} : $\neg(basic \wedge multimediaQA)$).

Summarizing, our example feature model includes the list of (Boolean-valued) features (variables v_i) $F = \{v_1$: $survey$, v_2 : $license, v_3$: $ABtesting$, v_4 : $statistics, v_5$: QA, v_6 : $basic$, v_7 : $advanced, v_8$: $basicQA, v_9$: $multimediaQA\}$. Furthermore, the model includes the set of constraints $C = \{c_1..c_{10}\}$. These are the two major elements of an *FM configuration task* defined in terms of a constraint satisfaction problem (CSP) (see Definition 1).

Definition 1 (FM Configuration Task). An FM configuration task (V, C, R) can be defined as a CSP, where V is a set of (Boolean-valued) variables $V = \{v_1, ..., v_n\}$ and $C = \{c_1..c_m\}$ is a set of feature model constraints. Finally, $R = \{r_1..r_k\}$ is a set of user requirements also represented in terms of constraints (mostly variable assignments).

With a configuration task definition[2], we can introduce the concept of an *FM configuration* (Definition 2).

Definition 2 (FM Configuration). An FM configuration for an FM configuration task (V, C, R) is an assign-

[1] $\veebar$ denotes a logical *xor*.
[2] Without loss of generality, we focus on feature models and corresponding Boolean variable domains.

Table 1
Solver search heuristics for solving a new configuration task (*task*) can be determined by reusing the search heuristics already applied to create the nearest neighbor (*NN*) configuration(s) (*id* = configuration identifier). In this example, the selected nearest neighbor is configuration 3 – the corresponding search heuristics can be applied to solve the new FM configuration task. In this context, H=*highest value first* and L=*lowest value first*.

id	Configuration									Variable Ordering									Value Ordering									Runtime [ms]
	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	
$conf_1$	1	1	0	1	1	1	0	1	0	9	3	6	1	8	4	7	5	2	L	L	H	L	L	H	H	H	L	229.133
$conf_2$	1	1	0	1	1	1	0	1	0	5	2	3	9	7	1	8	6	4	H	L	L	L	L	H	L	L	H	218.384
$conf_3$	1	1	0	1	1	0	1	1	0	4	2	7	5	8	6	1	3	9	H	H	H	H	L	H	L	L	L	191.296
$conf_4$	1	1	0	0	1	0	1	1	0	8	2	7	9	6	5	4	1	3	H	L	H	H	L	H	L	L	H	116.995
task	?	1	?	1	?	?	?	?	?	? →	4	2	7	5	8	6	1	3	9	? →	H	H	H	H	L	H	L	L	L	?

ment $conf = \{v_1 = va_1 \wedge .. \wedge v_n = va_n\}$ where $conf \cup C \cup R$ is consistent and every variable in V has an assignment, i.e., we assume assignment completeness.

Assuming a set of defined user requirements $R = \{r_1 : advanced = true, r_2 : ABtesting = true, r_3 : basicQA = true\}$ (i.e., users do not need to define their preferences with regard to all features) could result in the following complete configuration $conf = \{survey = true, license = true, advanced = true, basic = false, ABtesting = true, statistics = true, QA = true, basicQA = true, multimediaQA = false\}$.

Having introduced the concepts of a configuration task and a corresponding configuration, we are now able to discuss our collaborative filtering based constraint solver search heuristics recommendation approach in more detail.

3. Collaborative Search Heuristics Recommendation

Our basic idea is to apply different types of nearest neighbor based collaborative filtering [6] for the purpose of recommending relevant solver search heuristics (including both, *variable orderings* (the order in which the solvers tries to instantiate variables) and *variable value orderings* also denoted as *variable domain strategies* (the order in which the solver instantiates variable values) for a new configuration task (see Definition 1).

Our used variable value orderings (i.e., variable domain strategies) are *highest first* (H) and *lowest first* (L), i.e., the constraint solver starts with trying to instantiating the highest or the lowest variable value first.[3] For the purposes of our experiments, we use different data synthesis approaches [9] (see Section 4). Each entry of a synthesized dataset represents a complete configuration consistent with the constraints in C and R (see Definition 2). In addition to the feature settings (inclusion or exclusion), each entry also includes information about (1) the used variable ordering, (2) variable value ordering, and (3) runtime (in *ms*) to find the corresponding *configuration* (see Table 1). We use such entries to identify (reuse) search heuristics for completing new configuration tasks.

k-nearest neighbors (k=1). Table 1 shows a simplified example of how to apply kNN (k nearest neighbor) based approaches for recommending search heuristics for a new configuration task (in this example, we assume $k = 1$). The table contains four (in our case synthesized) entries of complete configurations including further information on the used solver search heuristics, i.e, variable and variable value orderings. Finally, for each configuration we have information about the corresponding solver runtime (in *ms*).

In this example, the new configuration *task* needs to be solved. The idea is to identify nearest neighbor configurations $conf_i$ on the basis of the similarity between the new configuration task and the available (complete) configuration entries ($conf_1$ – $conf_4$ in Table 1). Following Formula 1 for determining the similarity between the new configuration task and each $conf_i$, configurations $conf_1$–$conf_3$ have the same similarity, i.e., $sim(task, conf_1) = sim(task, conf_2) = sim(task, conf_3) = 1.0$.[4] In this context, V' denotes variables with associated specified user requirements, i.e., those variables of V which have a corresponding initial value assignment in the configuration *task* definition (e.g., $\{v_2, v_4\}$ in Table 1).

$$sim(task, conf) = \frac{|\{v_i \in V' : val(v_i, task) = val(v_i, conf)\}|}{|\{v_i \in V\}|} \quad (1)$$

In this context, *task* (the initial user requirements) represents a partial configuration, since not every variable needs to have an assigned value – assigned values are assumed to represent user requirements $r_i \in R$.

k-nearest neighbors (k>1). The *k-nearest neighbor* based approach identifies the k most similar configurations $conf_i$ compared to the current configuration *task* (see Formula 1) and then chooses the configuration with

[3] In our example, strategy *H* first tries to include feature v_i, i.e., $v_i = true$.

[4] $val(v_{i,})$ denotes the value of variable v_i.

the best solver runtime performance (which is then the so-called nearest neighbor). Since $conf_3$ has the lowest (best) runtime among the identified nearest neighbors, we can reuse the solver search heuristics used to determine $conf_3$. If $k = 1$, only one nearest neighbor is identified and the corresponding solver search heuristics are applied to the current configuration task. The major difference between $k = 1$ and $k > 1$ is that in $k > 1$ settings it could be the case that a configuration with lower similarity (see Formula 1) is selected due to a better corresponding runtime performance.

k-nearest neighbors (k>1, weighted). The previously discussed k-nearest neighbor approach takes into account the similarity between the current configuration (task) and already existing configurations $conf_i$. In our experiments, we were also interested in the impact of taking into account tradeoffs between configuration similarity and solver runtimes (see Formula 2).

$$sim_t(task, conf) = \frac{max(runtime) - runtime(conf)}{1 - sim(task, conf) + \lambda} \quad (2)$$

Similar to the *k-nearest neighbor* based approach ($k > 1$), this weighted approach (see Formula 2) as well identifies configurations similar to the current configuration (task), but then uses similarity as a weighting factor, i.e., not just selects the nearest neighbor with the best runtime performance. This way, we determine those nearest neighbors with a good runtime which are at the same as similar as possible to the given configuration *task*.

In Formula 2, we have introduced a small constant λ to avoid division by 0 which could happen in situations where the requirements in the given configuration *task* are equivalent with the corresponding variable settings in *conf*, i.e., the similarity is 1.0. Finally, we want to mention that *max(runtime)* denotes the highest (global) runtime value used to represent the worst (highest) runtime observed in the (FM-specific) synthesized data.

4. Used Data Synthesis Approaches

Overall Synthesis Approach. The methods we chose for synthesizing configuration data are based on those discussed in Pereira et al. [9]. We have applied these synthesis approaches in our constraint solver search heuristics recommendation scenario for the purpose of generating complete and consistent configurations, i.e., each variable has a corresponding assignment and all assignments are consistent with the constraints in C. As solver input, we have generated a set of user requirements $r_i \in R$ representing around 10% of the features contained in the corresponding feature model. For each variable, a corresponding variable value ordering heuristics has been chosen randomly. Finally, we also chose a variable ordering for variables not contained in the generated set of user requirements R. On the basis of this initial input (randomly generated search heuristics and user requirements), a solver has been activated with a repetition factor of 5 to determine the average runtime needed to solve the defined setting (see also Tables 2 and 3). Following the overview of Pereira et al. [9], we have used and evaluated the following data synthesis approaches.

Random Sampling. Random Sampling is one of the most widely used methods to synthesize data for configuration problems [15, 16, 17]. There are several variations [9, 13] that can be differentiated with regard to the number generated samples. One of these variations generates a fixed (pre-defined) number of configurations, regardless of the properties of the underlying feature model. The *fixed number* approaches we have tested in the context of our evaluation are: *100*, *200*, *500*, *1000*, *2000* and *10000*.

In addition to this rather static approach, we have applied other approaches which take into account feature model sizes. *Random N* focuses on generating N random configurations where N equals the number of features in the feature model. There also exist some variations of this approach where the number of generated configurations equals $2N$(Random $2N$) or $3N$(Random $3N$). Furthermore, the number of configurations can also be systematically reduced by introducing the synthesis variants *Random* $\frac{1}{4}N$, *Random* $\frac{1}{2}N$ and *Random* $\frac{3}{4}N$.

Heuristics Based Sampling. Other approaches are based on heuristics for configuration generation.

Feature Frequency Heuristic (FFH) The Feature Frequency Heuristic (FFH) [18] generates configurations following the strategy of ensuring that each feature occurs at least a predefined times in the resulting configurations (corresponding thresholds can also be defined per feature). In our evaluation, we have applied the (global) feature-wise threshold values of 5, 10, and 20.

Feature Coverage Heuristic (FCH). FCH [19, 20, 18] tries to ensure the presence of every feature combination of *size t* in the generated data. This approach could be applied in different ways, for example, by generating all possible *t-way* feature combinations. We used the ACTS tool[5] provided by [21], which generates so-called covering arrays [22]. This way, we generated data sets with 2-way and with 3-way coverage. Higher coverage comes at very high computational costs for larger models which forced us to omit corresponding synthetizations with our used ACTS tool [21].

[5] https://csrc.nist.gov/projects/automated-combinatorial-testing-for-software.

Table 2
Constraint solver performance with k-nearest neighbor based (following the nearest neighbor selection approach of Formula 1) search heuristics recommendations. In this context, the constraint solver performance of different feature models has been evaluated. Compared to standard solver runtimes (without heuristics recommendations – see Table 4)) , we can observe significant corresponding runtime improvements. Values in bold indicate the best configuration synthesis strategy, values with a grey background the best corresponding k value.

	Linux				UClinux-distribution				Busybox				WeaFQAS				REAL-FM-11				MobileMedia			
	k = 1	k = 5	k = 10	k = 20	k = 1	k = 5	k = 10	k = 20	k = 1	k = 5	k = 10	k = 20	k = 1	k = 5	k = 10	k = 20	k = 1	k = 5	k = 10	k = 20	k = 1	k = 5	k = 10	k = 20
N = 100	**366.39**	**371.05**	**371.82**	**370.39**	30.20	29.31	29.21	29.23	6.17	5.93	5.92	5.93	0.49	0.36	0.36	0.35	0.17	0.10	0.10	0.09	0.13	0.08	0.07	0.07
N = 200	371.71	374.69	374.33	372.28	30.13	30.72	30.08	30.08	6.02	6.07	6.21	6.30	0.37	0.36	0.37	0.38	0.07	0.07	0.07	0.07	0.06	0.05	0.05	0.05
N = 500	378.97	375.05	379.77	381.11	30.56	30.45	30.57	30.98	6.54	6.52	6.56	6.59	0.41	0.41	0.42	0.42	0.09	0.12	0.12	0.12	0.06	0.06	0.06	0.06
N = 1000	384.09	378.06	377.27	379.48	32.38	32.15	32.38	32.15	7.54	7.79	7.67	7.67	0.50	0.52	0.51	0.52	0.11	0.11	0.12	0.12	0.08	0.08	0.08	0.08
N = 2000	394.45	393.54	392.77	390.67	36.60	36.64	36.75	36.96	9.24	8.73	8.87	9.03	0.67	0.66	0.68	0.67	0.17	0.17	0.17	0.17	0.12	0.12	0.12	0.13
N = 10000	500.68	510.78	484.87	510.41	69.44	70.38	69.36	70.01	19.76	19.86	21.31	21.54	3.51	3.38	3.57	3.60	0.73	0.75	0.74	0.76	0.53	0.53	0.52	0.53
Random $\frac{1}{4}$N	406.34	387.72	386.92	389.04	31.55	31.79	31.52	31.46	6.17	6.14	6.19	6.12	**0.34**	**0.34**	**0.34**	**0.34**	0.06	0.06	0.06	0.06	0.04	**0.04**	**0.03**	0.04
Random $\frac{2}{4}$N	419.47	401.34	399.21	405.95	32.17	31.61	32.07	31.77	6.46	6.46	6.46	6.44	0.35	0.35	0.35	0.35	0.06	0.06	0.06	0.06	0.04	0.04	0.04	0.04
Random $\frac{3}{4}$N	422.74	424.84	426.05	450.48	33.25	33.40	33.32	33.15	6.79	6.73	6.75	6.73	0.35	0.36	0.36	0.36	0.06	0.06	0.06	0.06	0.04	0.04	0.04	0.04
Random N	444.35	440.96	438.66	438.72	34.62	34.45	34.79	34.50	6.99	7.03	7.04	7.04	0.36	0.37	0.37	0.37	0.06	0.06	0.06	0.06	0.04	0.04	0.04	0.04
Random 2N	525.98	537.85	534.80	513.28	42.60	41.66	40.95	41.43	8.12	8.37	8.48	8.47	0.39	0.39	0.40	0.40	0.07	0.07	0.07	0.07	0.04	0.04	0.04	0.04
Random 3N	632.08	643.38	615.72	635.11	48.20	47.78	48.31	48.44	9.73	9.77	9.90	9.76	0.42	0.42	0.42	0.43	0.07	0.07	0.07	0.07	0.04	0.04	0.04	0.05
FCH (2-Way)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**0.06**	**0.06**	**0.06**	**0.05**	**0.04**	0.04	0.04	**0.03**
FCH (3-Way)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.06	0.06	0.06	0.06	0.04	0.04	0.04	0.04
FFH (5)	386.46	374.25	373.67	374.49	**28.68**	**28.74**	**28.70**	**28.69**	**6.11**	**5.88**	**5.92**	**5.89**	0.35	0.35	0.35	0.35	0.06	0.06	0.06	0.06	0.04	0.04	0.04	0.04
FFH (10)	372.84	371.83	374.77	378.29	29.14	29.21	29.20	29.14	6.00	5.96	5.94	5.95	0.35	0.36	0.36	0.36	0.06	0.06	0.06	0.06	0.04	0.04	0.04	0.04
FFH (20)	377.96	379.57	376.61	380.04	30.17	30.17	30.15	30.20	6.06	6.05	6.05	6.07	0.37	0.38	0.38	0.38	0.06	0.06	0.06	0.07	0.04	0.04	0.04	0.04

Table 3
Constraint solver performance with k-nearest neighbor based search heuristics recommendations (following the nearest neighbor selection approach of Formula 2). Again, the constraint solver performance of different feature models has been evaluated. Using this approach, we can observe further solver runtime improvements compared to the basic k-nearest neighbor based approach. Note that for $k = 1$ the performance values are the same as in Table 2 due to the fact that the same heuristics are selected in this case.

	Linux				UClinux-distribution				Busybox				WeaFQAS				REAL-FM-11				MobileMedia			
	k = 1	k = 5	k = 10	k = 20	k = 1	k = 5	k = 10	k = 20	k = 1	k = 5	k = 10	k = 20	k = 1	k = 5	k = 10	k = 20	k = 1	k = 5	k = 10	k = 20	k = 1	k = 5	k = 10	k = 20
N = 100	**366.39**	**371.52**	385.70	373.99	30.20	29.03	29.10	29.16	6.17	5.96	**5.97**	5.98	0.49	0.35	0.36	0.36	0.17	0.08	0.07	0.07	0.13	0.06	0.05	0.06
N = 200	371.71	374.70	376.70	377.71	30.13	30.14	30.10	30.87	6.02	6.06	6.04	6.14	0.37	0.37	0.37	0.38	0.07	0.07	0.07	0.07	0.06	0.05	0.05	0.05
N = 500	378.97	381.00	377.46	379.25	65.91	30.26	31.76	33.02	6.54	6.50	6.54	6.55	0.41	0.42	0.43	0.42	0.09	0.09	0.09	0.09	0.06	0.06	0.06	0.06
N = 1000	384.09	381.19	380.19	381.54	32.38	32.37	32.46	32.67	7.54	7.67	7.62	7.69	0.50	0.56	0.50	0.50	0.11	0.11	0.12	0.12	0.08	0.08	0.08	0.09
N = 2000	394.45	392.21	394.54	388.43	36.60	36.66	36.62	36.57	9.24	8.95	8.74	8.93	0.67	0.66	0.68	0.67	0.17	0.18	0.17	0.18	0.12	0.12	0.12	0.13
N = 10000	500.68	505.10	502.58	514.55	69.44	70.19	69.13	69.67	19.76	20.44	20.10	19.83	3.51	3.53	3.66	3.72	0.73	0.74	0.75	0.74	0.53	0.53	0.53	0.53
Random $\frac{1}{4}$N	406.34	389.73	391.04	391.56	31.55	32.02	31.82	32.04	6.17	6.10	6.11	6.13	**0.34**	**0.34**	**0.34**	**0.35**	0.06	0.06	**0.06**	**0.06**	0.04	**0.04**	**0.04**	**0.04**
Random $\frac{2}{4}$N	419.47	405.14	402.24	402.53	32.17	31.87	31.89	31.71	6.46	6.49	6.47	6.49	0.35	0.35	0.35	0.36	0.06	0.06	0.06	0.06	0.04	0.04	0.04	0.04
Random $\frac{3}{4}$N	422.74	423.54	426.13	421.33	33.25	33.64	33.39	33.82	6.79	6.75	6.74	6.77	0.35	0.36	0.36	0.37	0.06	0.06	0.06	0.06	0.04	0.04	0.04	0.04
Random N	444.35	442.24	441.59	440.81	34.62	34.34	34.60	35.09	6.99	7.00	7.00	7.01	0.36	0.37	0.37	0.38	0.06	0.06	0.06	0.07	0.04	0.04	0.04	0.04
Random 2N	525.98	528.67	546.50	515.19	42.60	41.43	41.10	42.04	8.12	8.41	8.52	8.60	0.39	0.39	0.40	0.40	0.07	0.07	0.07	0.07	0.04	0.04	0.04	0.05
Random 3N	632.08	641.79	630.38	636.14	48.20	47.70	48.23	47.99	9.73	9.39	9.47	9.51	0.42	0.43	0.43	0.44	0.07	0.07	0.07	0.07	0.04	0.04	0.04	0.05
FCH (2-Way)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**0.06**	**0.06**	0.06	0.06	**0.04**	0.04	0.04	0.04
FCH (3-Way)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.06	0.06	0.06	0.06	0.04	0.04	0.04	0.04
FFH (5)	386.46	374.04	373.87	**371.93**	**28.68**	**28.69**	**28.76**	**28.82**	6.11	**5.95**	6.03	**5.98**	0.35	0.35	0.35	0.35	0.06	0.06	0.06	0.06	0.04	0.04	0.04	0.04
FFH (10)	372.84	376.19	**373.66**	374.67	29.14	29.15	29.16	29.28	**6.00**	5.98	5.99	6.00	0.35	0.36	0.36	0.37	0.06	0.06	0.06	0.06	0.04	0.04	0.04	0.04
FFH (20)	377.98	377.64	379.47	375.97	30.17	29.88	29.95	29.95	6.06	6.04	6.05	6.09	0.37	0.37	0.38	0.38	0.06	0.06	0.06	0.07	0.04	0.04	0.04	0.05

5. Evaluation

Overall Evaluation Approach. On the basis of the discussed data synthesis and recommendations, we now present the results of a performance evaluation. We have compared solver runtimes with default solver settings with our recommendation based approaches. For the standard setting, we have measured the time needed by the solver to find a solution (see Formula 3).

$$runtime = time(solver) \tag{3}$$

For settings including heuristics recommendation, the runtime calculation also needs to take into account nearest neighbor (NN) identification and heuristics recommendation (see Formula 4).

$$runtime = time(NN) + time(recommendation) + time(solver) \tag{4}$$

Overall, we have compared six different feature models[6] which significantly differ in terms of the number of variables and the number of corresponding feature model constraints. Tables 2–3 provide an overview of the runtime performance of the used constraint solver Choco[7] in scenarios where 10% of the user requirements have been specified (randomly assigned) for the new configuration task. To avoid evaluation biases, for each setting (synthesizer strategy × feature model), we have generated 150 test configuration tasks. In addition, to avoid biases in the training data set, we have generated 5 training datasets for the mentioned settings. With this, we have measured the average runtime needed for solving the configuration task (see also Formulae 3 – 4).

Comparison of Synthesis Strategies. Independent of the used synthesis strategy, solver performance can be

[6] See github.com/diverso-lab/benchmarking [13] and the S.P.L.O.T. repository [14].
[7] See choco-solver.org.

Table 4
Feature models used for evaluating the different heuristics recommendation approaches (also including the standard solver runtime (in *milliseconds*) needed for calculating a solution for a configuration task).

Model	$\|V\|$	$\|C\|$	solver runtime [ms]
Linux	6467	13972	771.93
uClinux-distribution	1580	1793	65.91
busybox	854	905	13.96
WeaFQAS	179	100	1.30
REAL-FM-11	67	64	0.49
mobilemedia	43	32	0.07

significantly improved with search heuristics recommendation, for example, in the context of the Linux feature model (see Table 4), runtimes can be reduced by half. The analysis of the different data synthesis approaches showed that approaches generating smaller datasets in general perform best which can partially explained by the fact that the effort of determining nearest neighbors is reduced. The best performing synthesis approach for small models (*REAL-FM-11 Model* and *MobileMedia Model*) is Feature Coverage Heuristic (2-way) – due to computational overheads, evaluations for larger models have been omitted. For the remaining settings, in the majority of the cases the best performing synthesis approach is the Feature Frequency Heuristic (FFH) with threshold $t = 5$.

Comparison of K-Nearest Neighbor Approaches. When comparing *k-nearest neighbor* and *weighted k-nearest neighbor* based heuristics recommendation, we can observe that both approaches result in a similar solver performance, however, *weighted k-nearest neighbor* appears to be the more stable approach which is less susceptible to outliers (in terms of low solver performance).

Comparison of k-Values. When comparing different k-values, we can observe a tendency that more complex feature models (and corresponding constraint satisfaction problems) tend to perform better with increasing k-sizes. This can be partially explained by the fact that larger models (with larger configuration/solution spaces) need a higher k-value for achieving a certain coverage of the search space. On the other hand, the additional efforts to be taken into account for increasing k-sizes (e.g., in terms of additional efforts in nearest neighbor determination) can to some extent be compensated by higher-quality variable (value) ordering heuristics.

6. Threats to Validity

The major objective of the presented work is constraint solver optimization, however, the presented approach could also be applied in other application domains such as operating system optimization and the optimization of production schedules. We regard corresponding evaluations as a major focus of our future work. We are aware of the variety of FM knowledge representations – not all of these representations will directly profit from the concepts presented in this paper (since we focused on specific constraint solver heuristics). On the one hand, we regard related developments, i.e., learning other types of search heuristics, as a major focus of future research. On the other hand, we want to emphasize that the presented collaborative recommendation approaches can be applied as such in other settings with a focus on the reuse of reasoning knowledge. We want to emphasize that we intentionally focused on comparing (memory-based) collaborative recommendation approaches. Future work will include evaluations with model-based (e.g., neural networks) collaborative recommendation approaches. Finally, we are also aware of different types of parallelization approaches helping the improve search efficiency (see, for example, [23, 24]). We regard a direct comparison with such approaches a major task for future work.

7. Conclusions

In this paper, we have presented an approach to recommend constraint solver search heuristics (variable orderings as well as variable values orderings) which help to improve the performance of constraint solver based feature model configuration. We have applied and combined different types of data synthesis strategies and corresponding collaborative recommendation approaches which have been used as a basis for recommending search heuristics for new feature model configuration tasks. The results of our performance evaluation show that an approach to the recommendation of search heuristics combined with a well-fitted data synthesis approach can lead to significant performance improvements in feature configuration (in our evaluation settings, we could observe significant performance improvements of around 50% (and more) compared to standard solver runtimes). Major issues for future work are the evaluation of our approach in further domains (e.g., operating systems optimization) and the development/inclusion of model-based recommendation approaches.

References

[1] D. Benavides, A. Felfernig, J. Galindo, F. Reinfrank, Automated Analysis in Feature Modelling and Product Configuration, in: ICSR'13, number 7925 in LNCS, Springer, Pisa, Italy, 2013, pp. 160–175.

[2] K. Kang, S. Cohen, J. Hess, W. Novak, S. Peterson, Feature-oriented Domain Analysis (FODA) – Feasi-

bility Study, Technical Report, SEI, Carnegie Mellon University, Pittsburgh, PA, USA, 1990.
[3] D. Benavides, P. Trinidad, A. Ruiz-Cortés, Automated reasoning on feature models, in: CAiSE'05, Springer-Verlag, Berlin, Heidelberg, 2005, p. 491–503.
[4] A. Popescu, S. Polat-Erdeniz, A. Felfernig, M. Uta, , M. Atas, V. Le, K. Pilsl, M. Enzelsberger, T. Tran, An Overview of Machine Learning Techniques in Constraint Solving, Journal of Intelligent Information Systems 58 (2022) 91–118.
[5] A. Falkner, A. Felfernig, A. Haag, Recommendation Technologies for Configurable Products, AI Magazine 32 (2011) 99–108.
[6] M. Ekstrand, J. Riedl, J. Konstan, Collaborative filtering recommender systems, Found. Trends Hum.-Comput. Interact. 4 (2011) 81–173.
[7] K. Meel, Counting, Sampling, and Synthesis: The Quest for Scalability, in: IJCAI-22, 2022, pp. 5816–5820.
[8] J. A. Pereira, M. Acher, H. Martin, J. Jézéquel, Sampling effect on performance prediction of configurable systems: A case study, in: ACM/SPEC International Conference on Performance Engineering, ICPE '20, ACM, 2020, p. 277–288.
[9] J. A. Pereira, M. Acher, H. Martin, J. Jézéquel, G. Botterweck, A. Ventresque, Learning software configuration spaces: A systematic literature review, Journal of Systems and Software 182 (2021) 111044.
[10] S. Polat-Erdeniz, A. Felfernig, R. Samer, M. Atas, Matrix factorization based heuristics for constraint-based recommenders, in: 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19), ACM, Limassol, Cyprus, 2019, pp. 1655–1662.
[11] M. Uta, A. Felfernig, D. Helic, V. Le, Accuracy- and consistency-aware recommendation of configurations, in: ACM International Systems and Software Product Line Conference (SPLC'22), ACM, Graz, Austria, 2022, pp. 79–84.
[12] V. Le, A. Felfernig, M. Uta, T. Tran, C. Vidal, WipeOutR: Automated Redundancy Detection for Feature Models, in: 26th ACM International Systems and Software Product Line Conference, ACM, 2022, pp. 164–169.
[13] R. Heradio, D. Fernandez-Amoros, J. J. Galindo, D. Benavides, D. Batory, Uniform and scalable sampling of highly configurable systems, Empirical Software Engineering 27 (2022) 1–34.
[14] M. Mendonca, M. Branco, D. Cowan, S.P.L.O.T.: Software Product Lines Online Tools, in: 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications, OOPSLA '09, ACM, 2009, pp. 761–762.
[15] M. Acher, P. Temple, J.-M. Jézéquel, J. Galindo, J. Martinez, T. Ziadi, VaryLATEX: Learning Paper Variants That Meet Constraints, in: 12th International Workshop on Variability Modelling of Software-Intensive Systems, ACM, 2018, p. 83–88.
[16] A. Grebhahn, C. Rodrigo, N. Siegmund, F. Gaspar, S. Apel, Performance-influence models of multigrid methods: A case study on triangular grids, Concurrency and Computation: Practice and Experience 29 (2017).
[17] J. Guo, K. Czarnecki, S. Apel, N. Siegmund, Wąsowski, Variability-aware performance prediction: A statistical learning approach, in: 28th IEEE/ACM International Conference on Automated Software Engineering, ASE '13, IEEE Press, Silicon Valley, CA, USA, 2013, pp. 301–311.
[18] A. Sarkar, J. Guo, N. Siegmund, S. Apel, K. Czarnecki, Cost-efficient sampling for performance prediction of configurable systems, in: 30th IEEE/ACM International Conference on Automated Software Engineering, ASE '15, IEEE Press, Lincoln, Nebraska, 2015, pp. 342–352.
[19] C. Kaltenecker, A. Grebhahn, N. Siegmund, J. Guo, S. Apel, Distance-based sampling of software configuration spaces, in: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), IEEE, Montrea, Quebec, Canada, 2019, pp. 1084–1094.
[20] M. Lillack, J. Müller, U. Eisenecker, Improved prediction of non-functional properties in software product lines with domain context, in: S. Kowalewski, B. Rumpe (Eds.), Software Engineering 2013, Gesellschaft für Informatik e.V., Bonn, 2013, pp. 185–198.
[21] L. Yu, Y. Lei, R. Kacker, D. Kuhn, Acts: A combinatorial test generation tool, in: 6th IEEE International Conference on Software Testing, Verification and Validation, IEEE, Luxembourg, 2013, pp. 370–375.
[22] C. Colbourn, Combinatorial Aspects of Covering Arrays, Le Matematiche 59 (2004) 125–172.
[23] L. Bordeaux, Y. Hamadi, H. Samulowitz, Experiments with Massively Parallel Constraint Solving, in: IJCAI'09, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2009, pp. 443–448.
[24] V. Le and C. Vidal Silva and A. Felfernig and D. Benavides and J. Galindo and T.N.T. Tran, FastDiagP: An Algorithm for Parallelized Direct Diagnosis, in: AAAI'23, 2023, pp. 6442–6449.