Is Seeding a Good Strategy in Multi-objective Feature Selection When Feature Models Evolve?

Takfarinas Saber[a,*], David Brevet[b], Goetz Botterweck[c], Anthony Ventresque[a,*]

[a] *Lero@UCD, School of Computer Science, University College Dublin, Dublin 4, Ireland*
[b] *Institut Supérieur d'Informatique, de Modélisation et de leurs Applications, Clermont-Ferrand, France*
[c] *Lero@UL, University of Limerick, Ireland*

Abstract

Context: When software architects or engineers are given a list of all the features and their interactions (i.e., a Feature Model or FM) together with stakeholders' preferences – their task is to find a set of potential products to suggest the decision makers. *Software Product Lines Engineering* (SPLE) consists in optimising those large and highly constrained search spaces according to multiple objectives reflecting the preference of the different stakeholders. SPLE is known to be extremely skill- and labour-intensive and it has been a popular topic of research in the past years.
Objective: This paper presents the first thorough description and evaluation of the related problem of *evolving* software product lines. While change and evolution of software systems is the common case in the industry, to the best of our knowledge this element has been overlooked in the literature. In particular, we evaluate whether seeding previous solutions to genetic algorithms (that work well on the general problem) would help them to find better/faster solutions.
Method: We describe in this paper a benchmark of large scale evolving FMs, consisting of 5 popular FMs and their evolutions – synthetically generated following an experimental study of FM evolution. We then study the performance of a state-of-the-art algorithm for multi-objective FM selection (SATIBEA) when seeded with former solutions.
Results: Our experiments show that we can improve both the execution time and the quality of SATIBEA by feeding it with previous configurations. In particular, SATIBEA with seeds proves to converge an order of magnitude faster than SATIBEA alone.
Conclusion: We show in this paper that evolution of FMs is not a trivial task and that seeding previous solutions can be used as a first step in the optimisation - unless the difference between former and current FMs is high, where seeding has a limited impact.

Keywords: Software Product Lines, Multi-objective, Genetic Algorithm, Evolution

1. Introduction

Software Product Lines (SPL) is a branch of Software Engineering that aims at designing software products based on a composition of pre-defined software artefacts, increasing the reusability and personalisation of software products [3, 42]. Software architects, when they design new products or adapt existing products, navigate a set of features in a Feature Model (FM). Each of these features represents an element of a software artefact that is of importance to some stakeholders. Through its structure and additional constraints, each FM describes all possible products as combinations of features. One of the issues with FMs is that they can be very large – for instance in our study we work with FMs composed of more than 13,000 features and of nearly 300,000 constraints. *Optimising FM Configurations*, i.e., *selecting the set of features* that could lead to potential real products, is then a difficult problem [31]. This problem is also called SPL configuration as it consists in configuring products from the FMs.

In theory, software architects use SPL engineering to find one product – the product that matches their needs the most and does not violate any of the Feature Model's constraints. But in practice, the notion of the 'best' product is controversial, as there are different perspectives on what is a good product. For instance, some stakeholders may consider that energy consumption of the products is the most important objective to optimise, while for others it can be the cost of licensing the features; or some stakeholders see the reliability as the key element (for instance if they run critical applications), while other stakeholders have a strict performance policy and they need assurance that the selected features follow some guidelines. Figure 1 shows an example of SPL configuration according to two dimensions: number of known defects and cost (the lower the better for both dimensions). Possible products, found by an SPL optimisation algorithm, are represented as coloured circles. The good products, i.e., those that are better than any other one in a combination of objectives are represented by black circles. Product f, for instance, is not considered a good one, as product b is better than f in both dimensions. Similarly, product a, while worse than f in terms of cost, is better than all the

*Corresponding authors
Email addresses: takfarinas.saber@ucdconnect.ie (Takfarinas Saber), david.brevet@isima.fr (David Brevet), goetz.botterweck@lero.ie (Goetz Botterweck), anthony.ventresque@ucd.ie (Anthony Ventresque)

others in terms of known defects - it is then considered a good product. Those good products form a set, called *Pareto set* or *Pareto front*.

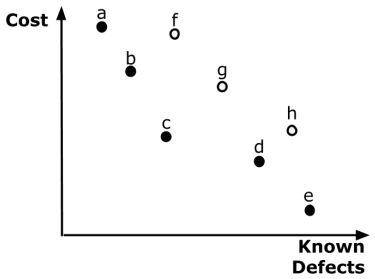

Figure 1: Possible products in 2-dimensions. In black are the non-dominated solutions (the good ones) and in white the dominated solutions (the bad ones).

Since it is unlikely in practice that only one dimension would be considered when optimising the feature configuration, the SPL engineering problem can be interpreted as a *multi-objective optimisation problem* [33, 48]. In fact, software architects tend to favour tools that allow them to manipulate good products, i.e., possible products that are better than every other possible product on a particular combination of objectives.

Another related problem that has only been addressed in the literature recently [13], is *feature selection in a multi-objective context when the FMs evolve*. Software requirements and artefacts evolve constantly; customers and other stakeholders change their opinions about what an application should do and how it should achieve that. Such changes can be reflected in Feature Models [44]: for instance, we have seen in our study that a large FM (such as the one behind the Linux kernel) evolves regularly and substantially (every few months a new version is released with up to 7% difference from the previous one). In this context, it seems odd to generate random bootstrapping populations for the state-of-the-art genetic algorithms, such as SATIBEA. It is tempting on the contrary to use the fact that FMs have evolved and that the SPL configurations generated previously, while not totally applicable, are close and can be adapted. This is a strategy called *seeding* and our intuition is that this could prove helpful in the context of Feature Model selection - especially since SATIBEA (and the other evolutionary algorithms) is very dependent on the initial population.

Seeding for search-based software engineering is not a novel idea as such (e.g., see papers by Fraser and Arcuri [26] and Alshahwan and Harman [2]). However, our approach is novel for various reasons:

- usually seeding is done by taking a few good/previous solutions that are inserted in the initial population - while in this paper, we take all the previous solutions that we adapt to create a starting population.
- the data sets we use for our experiments in this paper are large and very constrained, which is not always the case in search-based software engineering contexts for which seeding is known to work. This, of course, calls for a proper evaluation that we report here.
- we also studied the performance of seeding against a large variety of data sets, of different size and demographics (varying in their numbers and ratios of features and constraints). This gives us some more assurance that our conclusions are correct.

From a more general perspective, our contributions in this paper are the following:

- We propose a benchmark[1] for the analysis of evolving SPL; this data set has been generated following a study of the demographics and evolution of a large SPL (Linux kernel). This data set is important to provide a good evaluation of the different algorithms under different evolution scenarios;
- We propose *eSATIBEA* which is a modification of the state-of-the-art SATIBEA [33] for evolving SPL. eSATIBEA adapts previous solutions to new FMs to improve and speed-up the results of SATIBEA;
- We evaluate SATIBEA and eSATIBEA on the evolving SPL problem and show that eSATIBEA improves both the execution time and the quality of SATIBEA. In particular, eSATIBEA converges an order of magnitude faster than SATIBEA alone.

The rest of this paper is organised as follows: Section 2 defines the problem of multi-objective features selection when Feature Models evolve. Section 3 presents a large study of the evolution of a Feature Model: 20 versions of the Linux kernel (up to 13,000+ features and nearly 300,000 constraints). This study of the demographics of the Feature Model helps us to create the synthetic evolutions of 5 large and popular FMs. In particular, we are able to create evolved data sets using two parameters representing the evolution in terms of features and constraints. Section 4 describes SATIBEA, the state-of-the-art resolution algorithm for multi-objective SPL problems and the seeding mechanism for SPL configuration. In particular, we present a modification of SATIBEA that we call eSATIBEA – for SATIBEA in the context of Evolution. Section 5 describes the hardware set-up and presents the various metrics we use to compare algorithms. Those metrics are standard in the community and are classified as quality and diversity metrics. Section 6 evaluates SATIBEA and eSATIBEA against the 5 evolved data sets – and with different degrees of evolution. We show that eSATIBEA performs better in terms of quality and converges faster than SATIBEA (an order of magnitude faster). Section 7 presents threats to the validity of the results. Section 8 describes the related work. Section 9 concludes our study and proposes some future directions that we would like to explore.

Note that the study that we report in this paper follows a previous work [13] published at SSBSE 2016, the symposium dedicated to Search Based Software Engineering. In the SSBSE paper, we introduced the problem of optimisation of evolving

[1] Available here: http://hibernia.ucd.ie/EvolvingFMs/ upon acceptance of this paper.

Feature Models and provided some preliminary results using one data set and one metric. In the current paper, we extend the study to 5 data sets and 5 metrics, and we describe the data sets and the techniques in depth.

2. Problem Definition

In this section, we present the three elements that define the problem in our paper.

- Software Product Line Engineering, in particular how to describe variations of software applications as configurations of a Feature Model.
- Multi-objective optimisation; picking features can lead to many products for which the quality can be seen from different perspectives. MOO gives a framework to address this sort of problems.
- Evolution of Software Product Lines: Software applications, requirements, implementation, etc., change constantly and the Feature Models need to be updated to reflect these evolutions.

2.1. Software Product Line Engineering

Software engineers often need to adapt software artefacts to the needs of a particular customer [19]. Software Product Line Engineering is a software paradigm that aims at managing those variations in a systematic fashion. For instance, all software artefacts (and their variations) can be interpreted as a set of features [57] which can be selected and combined to obtain a particular product.

Feature Models can be represented as a set of features and relations (constraints) between them. Figure 2 shows a simple FM with 10 features linked by several relations. For instance, each 'Screen' has to be of exactly one of three types, i.e., 'Basic', 'Colour' or 'High Resolution'. When deriving a product from the product line, we have to select a subset of features $\mathcal{S} \subseteq \mathcal{F}$ that satisfies the FM $\mathcal{F}$ – and the requirements of the stakeholder/customer. This configuration can be described as a satisfiability problem (SAT), i.e., finding an assignment to variables (here, features) in the $\{True, False\}$ space. Let $f_i \in \{True, False\}$ be a decision variable set to '$True$' if the feature $F_i \in \mathcal{F}$ is selected to be part of $\mathcal{S}$ and '$False$' otherwise. An FM is equivalent to a conjunction of disjunctive clauses, forming a conjunctive normal form (CNF). Finding a product in the SPL is then equivalent to assigning a value in $\{True, False\}$ to every feature. For instance, in Figure 2 the FM would have the following clauses, among others: $(Basic \vee Colour \vee High\ resolution) \wedge (\neg Basic \vee \neg Colour) \wedge (\neg Basic \vee \neg High\ resolution) \wedge (\neg Colour \vee \neg High\ resolution)$, which describe the alternative between the three features.

2.2. Multi-objective Optimisation

Now, software designers, when configuring an SPL, do not only look for possible products (satisfying the FM) but for products optimising multiple criteria. This is why the problem of SPL configuration has been described as multi-objective.

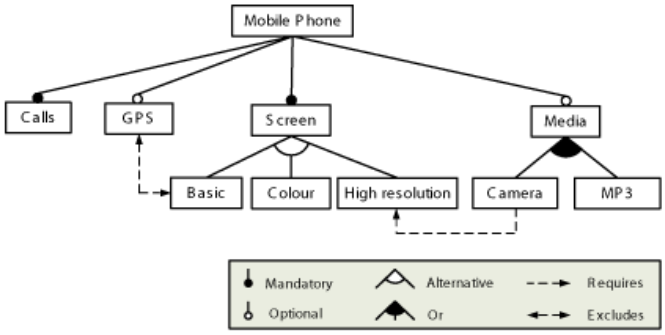

Figure 2: Sample of a Feature Model.

Multi-objective Optimisation (MOO) involves the simultaneous optimisation of more than one objective function. Given that the value of software artefacts can be seen from different angles (such as cost of development, importance for customers, reliability and so on), feature selection in SPL is a good candidate for MOO [33, 48].

Let us consider the minimisation case here – solutions of a MOO problem represent the set of non-dominated solutions defined as follows: Let S be the set of all feasible solutions for a given FM. Then $\forall s \in S$, $O = [O_1(s), ..., O_k(s)]$ represents a vector containing values of the k objectives for a given solution s. We say that a solution s_1 dominates s_2, written as $s_1 \succ s_2$, if and only if $\forall i \in \{1, ..., k\}$, $O_i(s_1) \leq O_i(s_2)$ and $\exists i \in \{1, ..., k\}$ such that $O_i(s_1) < O_i(s_2)$. In other words, in all criteria s_1 is as least as good as s_2, while in at least one criterion it is clearly better (recall we are looking at the minimisation case here, so better means smaller).

All these non-dominated solutions represent a set called a Pareto front: in this set, it is impossible to make any solution better in all objectives without making at least one solution dominated. The Pareto front given in Figure 1 contains solutions a, b, c, d and e because they are not dominated by any other, while f is dominated by b, g is dominated by c, and h is dominated by d. Hence, f, g and h are not in the Pareto front.

Here, following other classical approaches [33, 48] we use 5 objectives:

1. *Correctness* – minimise of the number of violated constraints, proposed by Sayyad et al. [49].
2. *Richness of features* – maximise the number of selected features (have products with more functionality).
3. *Features used before* – minimise the number of selected features that were not used before.
4. *Known defects* – minimise the number of known defects in selected features.
5. *Cost* – minimise the cost of the selected features.

In a different given application context, these objectives could be augmented or replaced with other particular criteria, e.g., consumption of resources or various costs.

2.3. Evolution in SPL

Evolution of SPLs and the corresponding FMs is known to be an important challenge since product lines represent long-term

investments [45]. For instance, in Section 3 we present a study of a large-scale FM, the Linux kernel, and we show that every few months a new FM is released with up to 7% modifications among the features (features added or removed). To the best of our knowledge, the FM/SPL evolution perspective has not been addressed in the multi-objective feature selection literature.

In this paper, we show a potential approach for this optimisation problem which utilises the evolution from one FM to another. The relationship between two versions of a Feature Model is expressed as a mapping between features. Let us assume an FM FM_1 evolved into another FM FM_2. Some of the features $f_i^1 \in FM_1$ are mapped on to features $f_i^2 \in FM_2$ – they are the same or considered the same, while some of the features $f_i^1 \in FM_1$ are not mapped onto any features in FM_2 (f_i^1 has been removed) and features $f_i^2 \in FM_2$ have no corresponding features in FM_1 (f_i^2 has been added). The same can be applied to constraints (removed from FM_1 or added to FM_2). The problem we address concerns adapting the solutions found previously for FM_1 to FM_2.

3. Benchmark for Feature-Model Evolution

Evolution of large software artefacts and applications, typical of what software product lines engineering usually addresses, is a known fact [44]. In this paper we want to study the performance of a classical algorithm for software product line configuration (SATIBEA [33]) in the context of evolution - and in particular we want to evaluate whether seeding previous configurations improves the performance of this algorithm.

A simple option for our study would be to use Feature Models for which we can find various versions. For instance, we present below a large Feature Model and its evolutions. However, our objective is to study the behaviour of two algorithms - and we would like to have a good control of the various evolution parameters: number of features removed/added, number of constraints added/removed etc.

This is why we have created a benchmark for Feature Model evolution: in order to be able to take any Feature Model and synthetically (but realistically) evolve it depending on the parameters required by specific experiments (e.g., small evolution of features but large evolution of constraints, etc.).

3.1. Study of the Linux Kernel

We studied the largest open source Feature Model we could find [1]: the Linux kernel [51] containing a maximum of 13,322 features and 277,521 constraints in its version 2.6.32 (see Table 1). We evaluated the demographics (features, constraints) and evolution of 21 versions of the kernel from version 2.6.12 to version 2.6.32 that are publicly available in the Linux Variability Analysis Tools (LVAT) repository [52].

Feature Models in the LVAT repository are not directly ready to use for our approach (i.e., they are not in the form of an instance of the SAT problem) as they are based on Kconfig model extracts (.exconfig). Therefore, they have to be translated into SAT instances first [10]. To achieve that, we use a tool called VM2BOOL[2] which converts every Feature Model (originally in a Kconfig file) into propositional formulas (stored on a .dimacs file). This transformation allows the optimisation algorithms (e.g., SATIBEA and eSATIBEA) to process the Feature Models.

Using the VM2BOOL tool, however, introduces additional variables into the problem in addition to those representing the features. This allows VM2BOOL to translate the most complicated relationships in the FM and also to avoid an explosion in the size of the propositions. We show in Table 1 the characteristics of the different evolutions in terms of the number of features (the real ones) and also show the size of their respective SAT problem as number of variables and number of clauses (constraints).

We observe that on average there was only 4.6% difference in terms of features between a version and the next. See Table 2 and Table 3 for a complete description of the number of features in common between any two versions – and the corresponding percentage values. Out of this 4.6% modified features, 21.22% were removed features and 78.78% were added features, on average.

Version	Release Date	Difference with previous release (days)	#Features	#Variables	#Clauses
2.6.12	17/06/2005	-	6,756	28,816	117,502
2.6.13	29/08/2005	73	6,952	29,814	120,599
2.6.14	27/10/2005	59	7,162	30,736	123,142
2.6.15	03/01/2006	68	7,324	31,340	125,179
2.6.16	20/03/2006	76	7,480	32,040	129,794
2.6.17	17/06/2006	89	7,616	32,736	132,780
2.6.18	20/09/2006	95	7,974	34,436	140,134
2.6.19	29/11/2006	70	8,326	36,220	152,888
2.6.20	05/02/2007	68	8,452	36,874	156,635
2.6.21	26/04/2007	80	8,664	37,858	162,058
2.6.22	08/07/2007	73	9,012	39,498	185,595
2.6.23	09/10/2007	93	9,304	40,922	181,881
2.6.24	24/01/2008	107	9,882	43,758	197,826
2.6.25	17/04/2008	84	10,260	45,612	206,816
2.6.26	13/07/2008	87	10,594	47,302	217,139
2.6.27	09/10/2008	88	10,908	48,868	218,960
2.6.28	25/12/2008	77	11,400	51,338	229,794
2.6.29	24/03/2009	89	11,948	53,868	248,347
2.6.30	09/06/2009	77	12,352	55,806	256,440
2.6.31	09/09/2009	92	12,804	57,828	266,416
2.6.32	03/12/2009	85	13,322	60,072	277,521

Table 1: Version number (2.6.*), release date, number of days between previous and current releases and number of features of the different versions of the Linux kernel considered in our study. We also show the number of Boolean variables and the number of clauses (constraints) in the SAT representation of the FM of each version.

To summarise, Table 4 shows the percentages of evolution for both features and constraints between two consecutive FMs.

We also evaluated the size of the clauses/constraints in the problem, as we need to know how the constraints we add in the problem should look like. We found that a large proportion of the FMs' constraints have 6 features (39%), 5 features (16%), 18 features (14%) or 19 features (14%). See Figure 3 for a more detailed report of the constraints sizes. Now we will be able to create randomly new constraints (or delete existing constraints

[2] `https://bitbucket.org/tberger/vm2bool`

	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
13	6,676																			
14	6,622	6,896																		
15	6,570	6,844	7,108																	
16	6,466	6,734	6,980	7,196																
17	6,328	6,594	6,832	7,040	7,316															
18	6,296	6,560	6,794	7,000	7,270	7,558														
19	6,234	6,496	6,726	6,906	7,168	7,448	7,862													
20	6,136	6,398	6,626	6,806	7,070	7,342	7,748	8,210												
21	6,078	6,336	6,562	6,742	7,006	7,278	7,682	8,140	8,380											
22	5,982	6,234	6,446	6,626	6,886	7,140	7,534	7,982	8,208	8,480										
23	5,878	6,126	6,338	6,516	6,774	7,028	7,420	7,862	8,088	8,352	8,878									
24	5,842	6,088	6,300	6,474	6,728	6,982	7,370	7,806	8,032	8,294	8,812	9,232								
25	5,752	5,996	6,208	6,380	6,632	6,884	7,262	7,700	7,922	8,184	8,694	9,096	9,704							
26	5,730	5,976	6,186	6,356	6,606	6,838	7,216	7,648	7,866	8,126	8,628	9,022	9,600	10,120						
27	5,600	5,848	6,058	6,196	6,444	6,676	7,050	7,480	7,692	7,952	8,438	8,826	9,394	9,896	10,354					
28	5,492	5,740	5,950	6,088	6,332	6,560	6,930	7,350	7,558	7,808	8,288	8,678	9,236	9,730	10,172	10,718				
29	5,468	5,716	5,916	6,052	6,296	6,518	6,884	7,302	7,510	7,756	8,232	8,620	9,176	9,670	10,112	10,648	11,312			
30	5,426	5,674	5,872	6,008	6,250	6,470	6,834	7,244	7,452	7,698	8,172	8,558	9,112	9,600	10,034	10,560	11,210	11,844		
31	5,418	5,664	5,862	5,998	6,240	6,454	6,818	7,228	7,436	7,682	8,148	8,534	9,084	9,568	10,000	10,526	11,170	11,792	12,290	
32	5,396	5,640	5,838	5,974	6,216	6,430	6,792	7,202	7,410	7,654	8,120	8,506	9,048	9,520	9,946	10,472	11,108	11,700	12,188	12,688

Table 2: Number of features in common between any two versions of of the Linux kernel 2.6.*

	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
13	96.02																			
14	92.46	96.28																		
15	89.7	93.44	97.05																	
16	86.44	90.02	93.31	96.2																
17	83.08	86.58	89.7	92.43	96.06															
18	78.95	82.26	85.2	87.78	91.17	94.78														
19	74.87	78.02	80.78	82.94	86.09	89.45	94.42													
20	72.59	75.69	78.39	80.52	83.64	86.86	91.67	97.13												
21	70.15	73.13	75.73	77.81	80.86	84	88.66	93.95	96.72											
22	66.37	69.17	71.52	73.52	76.4	79.22	83.59	88.57	91.07	94.09										
23	63.17	65.84	68.12	70.03	72.8	75.53	79.75	84.5	86.93	89.76	95.42									
24	59.11	61.6	63.75	65.51	68.08	70.65	74.58	78.99	81.27	83.93	89.17	93.42								
25	56.06	58.44	60.5	62.18	64.63	67.09	70.77	75.04	77.21	79.76	84.73	88.65	94.58							
26	54.08	56.4	58.39	59.99	62.35	64.54	68.11	72.19	74.24	76.7	81.44	85.16	90.61	95.52						
27	51.33	53.61	55.53	56.8	59.07	61.2	64.63	68.57	70.51	72.9	77.35	80.91	86.12	90.72	94.92					
28	48.17	50.35	52.19	53.4	55.54	57.54	60.78	64.47	66.29	68.49	72.7	76.12	81.01	85.35	89.22	94.01				
29	45.76	47.84	49.51	50.65	52.69	54.55	57.61	61.11	62.85	64.91	68.89	72.14	76.79	80.93	84.63	89.11	94.67			
30	43.92	45.93	47.53	48.63	50.59	52.38	55.32	58.64	60.33	62.32	66.15	69.28	73.76	77.72	81.23	85.49	90.75	95.88		
31	42.31	44.23	45.78	46.84	48.73	50.4	53.24	56.45	58.07	59.99	63.63	66.65	70.94	74.72	78.1	82.2	87.23	92.09	95.98	
32	40.5	42.33	43.82	44.84	46.65	48.26	50.98	54.06	55.62	57.45	60.95	63.84	67.91	71.46	74.65	78.6	83.38	87.82	91.48	95.24

Table 3: Percentage of features in common between any two versions of of the Linux kernel 2.6.*

		Features	Constraints
Difference		4.57%	11.73%
among which	Removed	21.23%	38.43%
	Added	78.77%	61.57%

Table 4: Average evolution of features/constraints between two consecutive FMs.

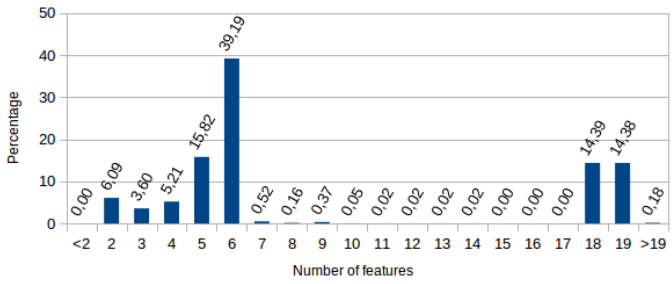

Figure 3: Distribution of the constraints' sizes (i.e., how many features they contain).

if need be) according to the proportions we find on this big data set.

From this study, we generated a synthetic benchmark of FM evolution based on the real evolution of the Linux kernel – hence a realistic benchmark but with more variability than in a real one, allowing us also to get several synthetic data sets corresponding to these characteristics. Our FM generator uses two parameters representing the percentage of feature modifications (added/removed) and the percentage of constraint modifications (added/removed). The higher those percentages are, the more different the new FM will be from the original one. Our FM generator uses the proportions we observed in the 20 FMs to generate new features/remove old ones, and to generate new constraints of a particular length. We use the following values to generate new FMs: from 5% of modified features and 1% of modified constraints (FM 5_1) to 20% of modified features and 10% of modified constraints (FM 20_10). In our evaluations,

we generate 10 synthetic FMs for each values of the parameters.

You will see later in this paper (e.g., Figures 6) the results of different SPLE algorithms on data sets generated synthetically by our generator (10 different combinations of parameters, and 10 data sets per combination).

In addition to the Linux kernel, we also include four other major data sets:

- eCos [8] (Embedded Configurable Operating System) is a real-time operating system designed for embedded systems and applications with a single process. It is also free and open-source.
- Fiasco [9] is a microkernel that can be used to construct flexible systems such as Unix-like operating systems.
- FreeBSD [53] is a free and open-source operating system similar to Linux.
- μClinux [9] is a microcontroller that is derived from the Linux kernel.

These data sets are widely used in the literature [1], cover large real-world Feature Models and are available in the LVAT repository [52]. For each of these data sets and for each evolution target, we generate 10 different versions. For instance, given the FM of FreeBSD and an evolution target of 10_5, we generate 10 evolutions of the FM with 10% of modifications of the features and 5% of modifications of the constraints.

Table 5 describes the size of the initial versions that are considered for each of the data sets, with their number of features, the size of the SAT problem representing their respective FMs on both variables and clauses. The execution time of 1,200 seconds used on the Linux kernel comes from [33] (the first presentation of SATIBEA). For the other data sets, we use smaller execution times based on the convergence time of our algorithms. Note that anyway our experiments show the impact of time as we present the evolution and not only the final results – beside, this paper is more interested in showing the importance of seeding to reach good results in short execution times.

data set	Version	#Features	#Variables	#Clauses	Time (s)
Linux kernel	2.6.12	6,756	28,816	117,502	1,200
eCos	20100825	1,244	1,244	3,146	50
Fiasco	2011081207	300	1,638	5,228	200
FreeBSD	8.0.0	1,396	1,396	62,183	200
μClinux	3.0	616	1,850	2,468	100

Table 5: Characteristics of the initial version of the different data sets used in our experiments in terms of number of features and size of the SAT problem representing their FM (number of variables and number of clauses). This table also shows the execution time that is allowed on each of them - this parameter is a realistic running time for the underlying FM of each of those data sets.

4. Algorithms

This section describes the algorithms we study in this paper: the state-of-the-art algorithm for multi-objective features selection (i.e., SATIBEA) and our extension to SATIBEA for evolved FMs (i.e., eSATIBEA). It also defines the parameters that are used by each of these algorithms.

4.1. State-of-the-art, SATIBEA

SATIBEA [33] is an improved version of the Indicator-Based Evolutionary Algorithm (IBEA) proposed by Zitzler and Künzli [65] that guides the search by a quality indicator given by the user. Previous to SATIBEA several techniques have been tried to solve the Multi-objective Optimisation for SPL. As most of the random techniques and genetic algorithms tend to generate invalid solutions (given the large and constrained search space, any random, mutation or crossover operation is tricky) setting the number of violated constraints as a minimisation objective has been proposed by Sayyad et al. [49]. It is obviously not the best possible decision and is acceptable only because of the size of the problem, which is otherwise tractable only for small FMs and exact algorithms [43, 38].

SATIBEA has been introduced to help IBEA finding valid products using a SAT solver. The purpose is to change the mutation process of IBEA's genetic algorithm: when an individual is mutated, three different exclusive mutations can be applied:

1. The standard bit-flip mutation proposed by IBEA.
2. Replacing the individual by another one generated by the SAT solver that does not violate any constraints.
3. Transforming the individual into a valid one using the SAT solver (repair).

With this new mutation approach, SATIBEA improves the quality of the solutions found by IBEA: it is capable of finding valid optimised products, but gives also better values in quality metrics (e.g., hypervolume).

4.2. Using Seeds in Evolved FMs, eSATIBEA

When an FM evolves, more or less modifications appear in features and constraints, depending on how far the new model is from the original one. We propose to take advantage of previous FM configurations (when they exist) to feed SATIBEA with solutions of the original model. Let's suppose two FMs: F_1 and F_2 with F_2 being an evolution of F_1 (i.e., features/constraints added and removed). We consider that we already found a set of solutions S_1 by applying a multi-objectives optimisation algorithm (SATIBEA in our case) on F_1. Instead of leaving SATIBEA with an initial random population for F_2, we adapt S_1 to F_2 such that for each individual, we remove bits representing removed features and add bits with random values for each new feature, then we compute the objective functions and give these new individuals as an initial population to SATIBEA that will run normally on F_2. Our hope is that these initial individuals will be of good quality or at least better than random solutions.

4.3. Parameters

We use in our experiment the same values as in [33] for all the parameters in both SATIBEA and eSATIBEA:

- Population size: 300 individuals
- Offspring population size: 300 individuals
- Crossover rate: 0.8. This represents the probability for a couple of individuals in the population to purchase a crossover, i.e., a mix of their characteristics.

– Mutation rate: 0.001. this represents the probability for each bit (true if a feature is selected, 0 otherwise) of an individual to be flipped.

– Solver mutation rate: 0.02. This represents the probability of using the SAT solver to correct a solution during the mutation process.

5. Experimental Set-up

This section presents the hardware configuration, the metrics we use to evaluate the performance of our algorithms and the two tests we use to validate the significance of our results. All our algorithms are implemented in Java and the tests are performed on a machine running Ubuntu 12.4 LTS 64bits with 62GB of RAM and 12 core Intel(R) Xeon(R) 2.20GHz CPU (our algorithms use only one core). We evaluate the performance of our techniques based on two types of metrics, following Wang et al. [61] practical guide for selecting quality indicators: (i) quality metrics: how good is the obtained Pareto front? and (ii) diversity metrics: how large/representative are its solutions?

5.1. Quality Metrics

We use three quality metrics to assess the quality of Pareto front solutions produced by the different algorithms.

5.1.1. Hypervolume (HV)

The intuition behind the hypervolume [66, 67] is that it gives the volume (defined in the k dimensions of the search space) dominated by the Pareto front. The hypervolume is the area between the solutions and the reference point. The reference point represents the worst possible value for each objective. The obtained measure represents the area covered by our approximate Pareto front: the higher the better.

In a more formal way, the hypervolume is defined in [33, 14] as follows:

Let A be the set of points on the Pareto front, then the hypervolume of A is represented by:

$$HV(A) = \lambda(\bigcup_{s\in A}[O_1(s), r_1] \times ... \times [O_k(s), r_k]) \quad (1)$$

where: λ is the Lebesgue measure [32], k is the number of objectives, $[r_1, ..., r_k]$ is a reference point taken far from it and $[O_1(s), r_1] \times ... \times [O_k(s), r_k]$ is the k-dimensional hyper cuboid consisting of all points that are weakly dominated by the point s but not weakly dominated by the reference point.

5.1.2. Epsilon (ϵ)

This metric measures the shortest distance that is required to transform every solution in a Pareto front A to dominate the reference front R [68]. Given the hardness of finding the optimal Pareto front and as it is commonly done in practice [25], we define the reference front as the set of all non-dominated solutions obtained by the different algorithms throughout all the iterations. The lower the ϵ value the better is the Pareto front. The ϵ metric finds the smallest multiplier ϵ such that every solution R is dominated by at least one solution in A, and is defined as:

$$\epsilon(A, R) = min(\epsilon) \mid \forall s \in R,\ \exists s' \in A,\ s \geq \epsilon.s' \quad (2)$$

5.1.3. Inverted Generation Distance (IGD)

This metric is the average of distances $d(s, A)$ for every solution s in the reference front R and its closest solution in the Pareto front A [35]. This metric is complementary of the ϵ metric in its way to evaluate the distance between the Pareto and the reference fronts, and the lower the IGD the better the Pareto front.

$$IGD(A, R) = \frac{\sum_{s\in R} d(s, A)}{|A|} \quad (3)$$

5.2. Diversity Metrics

We use in our evaluation two metrics that ensure that the set of solutions we present to the decision maker in order to choose from is diverse enough.

5.2.1. Pareto Front Size (PFS)

This metric corresponds to the quantity of solutions that are not dominated in a set of solutions A. In our case, we count the number of non-dominated solutions in the population of every generation. The higher this number the better as it means that we provide more choices to the decision makers to navigate and to select amongst them.

$$PFS(A) = |R| \quad (4)$$

5.2.2. Spread (S)

This metric measures the solutions extent spread in the Pareto front and evaluates their distribution [22]. The higher the spread the more diverse if the Pareto front (i.e., the better). For a set of solutions A, consider that for every two consecutive solutions $(s_a, s_b) \in A^2$ d_a is the distance between the solutions s_a and s_b, and $s_i \in A$ and $s_j \in A$ are the two furthest solutions in A:

$$S(A) = \frac{d_i + d_j + \sum_{a\in\{1..|A|-1\}}(d_a - d_{avg})}{d_i + d_j + d_{avg}.(|A| - 1)} \quad (5)$$

with $d_{avg} = \frac{\sum_{a\in\{1..|A|\}} d_i}{|A|}$

5.3. Statistical Analysis and Tests

In order to validate the significance of our comparison, we perform a statistical test using a non-parametric test: the two-tailed Mann-Whitney U test (MWU). On every target, MWU takes in the different performance values obtained by both eSATiBEA and SATIBEA on a given metric from each run (in our case 30). MWU returns the p-value that one of the algorithms obtains different values than the other. We consider tests significant when they are below a significance level of 0.05. Moreover, given the small number of runs in our experiment, and in order to lower the risk of having incorrect rejection of true null hypothesis, we use a conservative but safe adjustment

(i.e., the standard Bonferroni adjustment [4]) which reduces the chances of their erroneous rejection. Furthermore, following the advice in the practical guide proposed by Arcuri and Briand [4], we also use the non-parametric $\hat{A}_{12}$ [59] effect size measure which evaluates the ratio of runs from the first algorithm that outperform the second one. It is considered in the literature that when $\hat{A}_{12}$ is above 0.71, differences between the algorithms are large.

6. Evaluation

We evaluate in this section two algorithms: SATIBEA, known in the literature as the best algorithm for multi-objective configuration of SPLs, and our contribution: eSATIBEA.

First, we evaluate their performance on the Linux kernel benchmark described earlier in this paper, both in terms of quality and diversity of solutions while varying the evolution targets. We show the average results of 30 runs for each evolution target and each algorithm. We perform first this thorough and detailed study of the Linux kernel in order to give the reader a good sense of what is happening with the two algorithms. We also evaluate the significance of the results obtained by eSATIBEA over those of SATIBEA at four key optimisation snapshots (i.e., the initial, first, middle and last generations of the genetic algorithms).

Then, we extend the comparison to the 4 other data sets and study the gain that eSATIBEA brings over SATIBEA according to the evolution targets on every single metric. This broader evaluation will give us a sense of how stable are the algorithms (as they will be used against data sets of different varieties) and how robust our conclusions are.

6.1. Quality Performance of SATIBEA and eSATIBEA on the Linux Kernel

We compare the quality of the solutions in every generation of eSATIBEA and SATIBEA when run on the different Linux kernel instances with various evolution targets.

Figure 4 shows the evolution of SATIBEA's and eSATIBEA's performance on the different evolutions of the Linux kernel in terms of HV, ϵ and IGD.

First, we see that eSATIBEA starts with a high HV from its start (389.15% better than SATIBEA's HV on average) and maintains this HV quality in the following generations; whereas SATIBEA starts with a low HV that it keeps improving over time to eventually reach eSATIBEA's HV on some targets. SATIBEA does not always reach eSATIBEA's HV on all targets though (eSATIBEA outperforms SATIBEA on HV on almost all evolutions except 10_1 and has an average improvement over SATIBEA of 8.37% on average at the end of the execution time), particularly on those with large evolution targets (e.g., on the evolution 20_10 eSATIBEA gets a 37.37% better HV than SATIBEA on average at the end of their execution).

We also see a similar behaviour for eSATIBEA on the ϵ metric as it achieves good results since the beginning of its execution time (eSATIBEA gets $\sim$ 31.6 times better ϵ value than SATIBEA at its start), but unlike HV, eSATIBEA improves on its initial ϵ metric performance (eSATIBEA achieves $\sim$ 2.6 times better ϵ value at the end of its execution time in comparison to its beginning on average). SATIBEA also starts with poor ϵ values but they improve quickly. However, SATIBEA never catches up with the ϵ values of eSATIBEA (eSATIBEA finishes with $\sim$ 10.9 times better ϵ values than SATIBEA on average).

We notice that eSATIBEA also starts with a large advantage in terms of IGD (eSATIBEA starts with an IGD $\sim$ 12.7 times better than SATIBEA on average), but unlike HV and ϵ it does not improve it. Even worse, eSATIBEA worsens slightly its IGD (notice the exponent on the IGD scale). SATIBEA has the same behaviour with IGD it has with HV. SATIBEA starts with a poor IGD that it quickly improves to reach similar results than those of eSATIBEA on most targets, but without ever outperforming it (eSATIBEA gets 21% better IGD than SATIBEA on average).

Overall, we see that eSATIBEA starts with an initial population with a good quality (far better than SATIBEA's). We also see that eSATIBEA maintains the quality of its population in terms of ϵ value while at the same time maintaining its HV, and only worsening its IGD by at most 10^{-3}. In any case and for every quality metric, eSATIBEA always achieves a better quality of population than SATIBEA within the 4+ initial generations. eSATIBEA also converges to a similar or a better quality of solutions than SATIBEA in all cases and on all metrics. We notice that eSATIBEA converges to a better quality of population on instances with the largest evolution distance (e.g., 20_10) which clearly indicates that SATIBEA's initial population is not representative enough to allow the exploration of the entire search space, and that using solutions that are not feasible w.r.t. the evolved FM as an initial population allows eSATIBEA to explore a larger search space.

6.2. Diversity Performance of SATIBEA and eSATIBEA on the Linux Kernel

We compare the evolution in terms of diversity for both eSATIBEA and SATIBEA against our Linux kernel data set and various evolution targets.

Figures 5 show the evolution of SATIBEA and eSATIBEA's performance on the different evolutions of the Linux kernel in terms of PFS and Spread.

We see that eSATIBEA achieves a good PFS of $\sim$ 290 from the start (28% better than SATIBEA's PFS on average) and oscillates within an interval of +/-10 around this value. SATIBEA starts with a lesser PFS at $\sim$ 230 but it improves quickly (within the first few generations of the genetic improvement) to reach eSATIBEA's results and even slightly outperforming them sometimes. At the end of the execution time, eSATIBEA outperforms SATIBEA on PFS on almost all evolutions (except 5_1) with an average improvement of 1.58%.

We also see that eSATIBEA starts its optimisation with an initial population that is of a decent but not the best Spread (despite being $\sim$ 2.33 times better than the spread of SATIBEA's initial population), that it improves over time. SATIBEA's initial population has a small Spread. SATIBEA improves it quickly to outperform eSATIBEA's Spread, but this improvement is not stable and looks more like an 'N-shaped' one. At

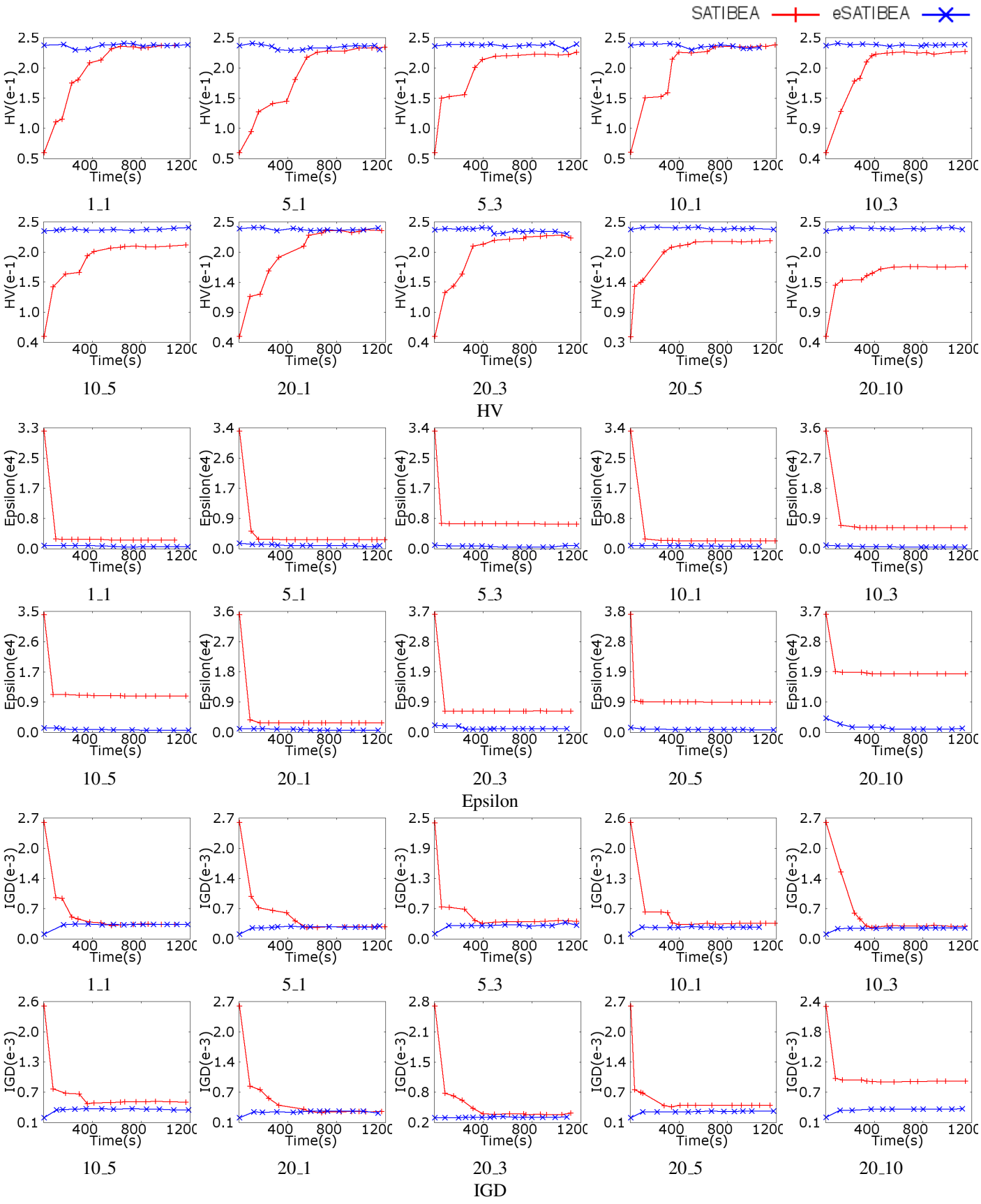

Figure 4: Evolution of the quality of SATIBEA's and eSATIBEA's solutions for the Linux kernel.

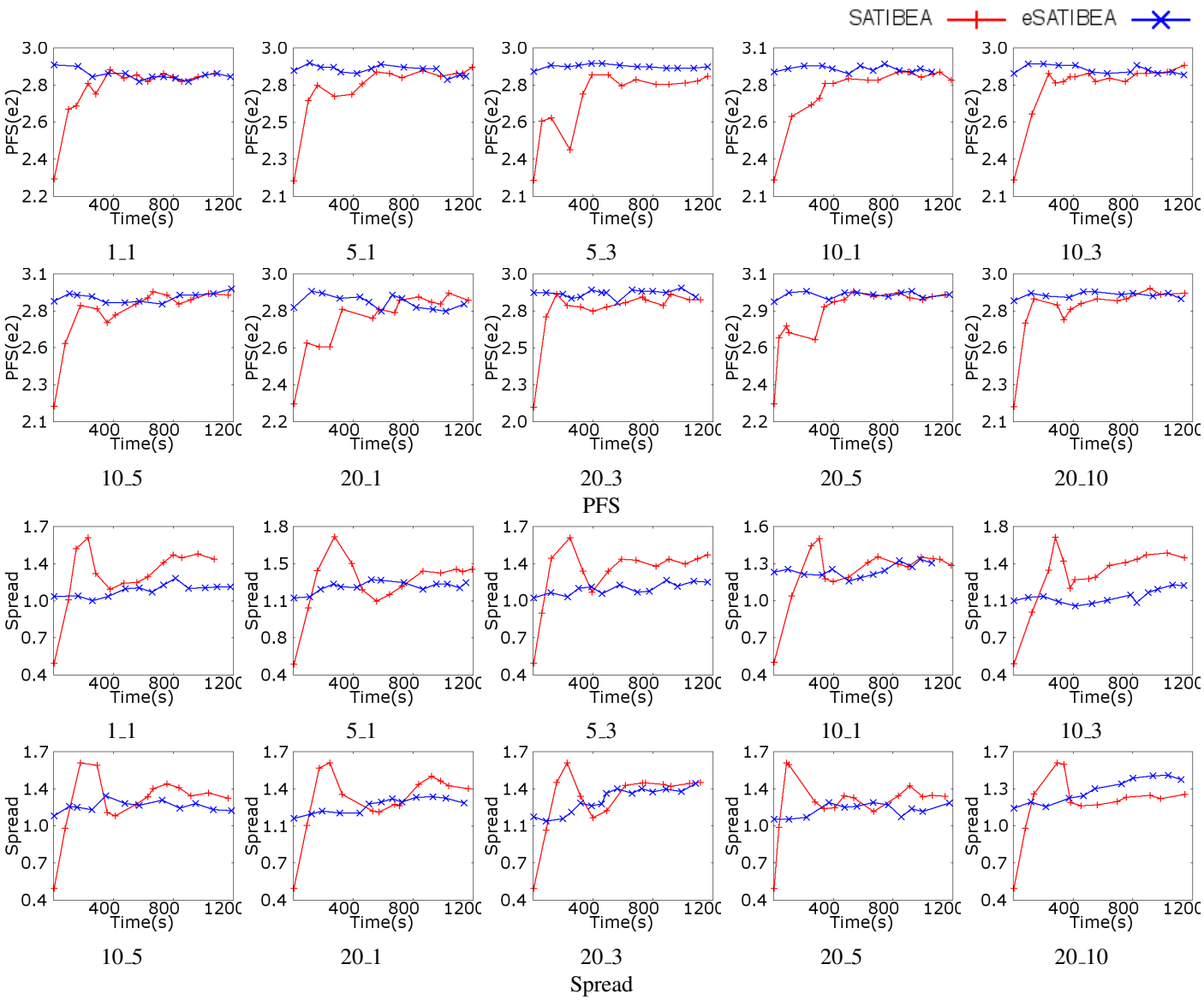

Figure 5: Evolution of the diversity of SATIBEA and eSATIBEA's solutions for the Linux kernel.

the end of the execution time, SATIBEA usually finishes with a population that has a slightly better Spread than eSATIBEA (SATIBEA finishes with a better Spread than eSATIBEA on 7 cases out of 10, with a Spread improvement of 7.87% on average).

Overall, eSATIBEA starts with a population with a good PFS but with a not-so-good Spread and successfully improves its Spread without negatively impacting the PFS. SATIBEA, however, starts with a population that has both a poor Spread and a small PFS. SATIBEA quickly improves the quality of its population to catch up eSATIBEA in terms of PFS and to outperform it in terms of Spread.

Globally, we can say that eSATIBEA gets an initial population that has an almost optimal quality w.r.t. all the quality metrics considered in our experiments and has an almost optimal PFS. However, eSATIBEA's Spread is not-so-good, and eSATIBEA continues improving its ϵ measure and Spread, without negatively impacting other metrics (i.e., HV, IGD and Spread). On the other hand, SATIBEA starts with an initial population that is poor on all the metrics, but SATIBEA improves it quickly and reaches results that are oftentimes similar to eSATIBEA and even slightly better on Spread.

However, Spread is not a significant metric on its own as you can have solutions that are well spread that have a poor quality (eSATIBEA usually achieves better quality metrics that SATIBEA). Results also suggest that the further the evolution, the larger the gap between eSATIBEA's and SATIBEA's results and that SATIBEA axes more on performance than on exploration of the search space and corroborate the poor diversity results that were observed in Henard et al. [33] paper.

6.3. Significance of eSATIBEA's results over SATIBEA on the Linux Kernel

We analyse in Table 6 the results obtained by both SATIBEA and eSATIBEA on the Linux kernel on four particular snapshots of their optimisation (i.e., initial, first, middle and last generations of their genetic algorithm). We also evaluate the significance of eSATIBEA's results over those of SATIBEA at each of these snapshots. We report the non-parametric WMU significance test that is corrected using the standard Bonferroni adjustment. We also evaluate the ratio of runs of eSATIBEA that outperform SATIBEA using the non-parametric $\hat{A}_{12}$ effect size measure. Notice that these two non-parametric tests (i.e., MWU and $\hat{A}_{12}$) show the significance of the first algorithm getting larger results than the second, and that some metrics are to be maximised while others are to be minimised. Therefore, in order to show the benefit of using eSATIBEA, we run both tests with eSATIBEA over SATIBEA for metrics to maximise, and SATIBEA over eSATIBEA for metrics to minimise.

We see in Table 6 that the initial generation of eSATIBEA is of a better quality than SATIBEA's initial generation on all targets of the Linux kernel. The initial generation of eSATIBEA is 346.85%, 95.44% and 93.36% better on average than SATIBEA's initial generation respectively on HV, ϵ and IGD. The initial generation of eSATIBEA is also of a better diversity than SATIBEA's initial generation as it is 30.31% better on PFS and 126.96% better on Spread on average. We also notice that all comparisons at this snapshot are statistically significant with at most an MWU p-value in the order of 1E-11. In addition, we also see that all the $\hat{A}_{12}$ measures are at 1 regardless of the metric, which clearly indicates that the initial generation of eSATIBEA is better than the one from SATIBEA in all runs.

We also see in Table 6 that the first generation created by eSATIBEA is of a better quality than SATIBEA's first generation on all targets of the Linux kernel despite a reduction in performance in comparison to what has been seen in the initial generation. The first generation of eSATIBEA is 89.92%, 79.60% and 63.84% better on average than SATIBEA's first generation respectively on HV, ϵ and IGD. The first generation of eSATIBEA is also of a better diversity than SATIBEA's first generation as it is 10.16% better on PFS and 13.41% better on Spread on average. We also notice that all comparisons at this snapshot are statistically significant with at most an MWU p-value in the order of 1E-8. In addition, we also see that all the $\hat{A}_{12}$ measures are at 1 for quality metrics, which clearly indicates that the first generation of eSATIBEA is still qualitatively better than the one from SATIBEA in all runs. But, we see that this is not totally the case for diversity metrics as $\hat{A}_{12}$ is less than 1 on few targets. However, despite this drop in $\hat{A}_{12}$, eSATIBEA's diversity is still significantly better as $\hat{A}_{12}$ is always larger than 0.9.

We notice from Table 6 that when eSATIBEA and SATIBEA reach their middle generation the results are not as sharp as in the two previous snapshot generations (i.e., initial and first). However, eSATIBEA maintains a small lead over SATIBEA as eSATIBEA outperforms SATIBEA on all targets and gets 11.46%, 83.62% better results on HV and ϵ on average. eSATIBEA also outperforms SATIBEA on IGD on most targets (i.e., 8 out of 10) with an average improvement of 18.29%. Qualitative results are also significance in the middle generation and are at most in the order of 1E-07. The $\hat{A}_{12}$ measure, however, is always in the advantage of eSATIBEA on HV and ϵ with an $\hat{A}_{12}$ at least larger than 0.87, but not always on IGD (especially with targets of a small modified constraint ratios i.e., 1.1., 5.1. and 20.1). Regarding diversity metrics we see a small advantage for eSATIBEA over SATIBEA on average: eSATIBEA is 1.40% and 0.46% better than SATIBEA on average on respectively PFS and Spread. However, eSATIBEA's is not always better than SATIBEA. Furthermore, although results are always significant on both MWU and $\hat{A}_{12}$ on PFS, they are not on Spread.

We see from Table 6 that the last generations of eSATIBEA and SATIBEA are close to each other in terms of results. However, eSATIBEA is still qualitatively better than SATIBEA on average with almost always significant results (except on target 20.1). eSATIBEA gets an improvement in HV on 8 targets out of 10 with an average of 7.48% over SATIBEA. eSATIBEA outperforms SATIBEA on all targets in terms of ϵ with an average improvement of 83.63%. eSATIBEA also outperforms SATIBEA in terms of IGD on 9 targets out of 10 with an average improvement of 19.87%. When it comes to diversity metrics, we see the opposite as SATIBEA is outperforming eSATIBEA on average on both PFS and Spread. However, results are mostly not significant with MWU p-values beyond 0.05 and $\hat{A}_{12}$ measures below 0.71 in many targets.

Overall, we see that eSATIBEA outperforms SATIBEA on all targets during their initial and first generations, but tend to converge to similar results in their middle and last generations. However, we see that the more distant is the target, the more eSATIBEA maintains its performance over SATIBEA (especially on 20.3, 20.5 and 20.10).

6.4. Gain of eSATIBEA over SATIBEA

The previous subsections described in depth the results obtained by SATIBEA and eSATIBEA on the evolutions of the Linux kernel. The current subsection is showing aggregate results for the 5 data sets (Linux kernel, eCos, Fiasco, FreeBSD, and μLinux). This will give us a sense of the trends observed on each of those data sets and whether the conclusions made on the Linux kernel can be generalised.

Figures 6 show the evolution of the gain achieved by eSATIBEA over SATIBEA on each metric and for every evolution target when run on the five different data sets. Notice that the gain is obtained from the subtraction of SATIBEA's results from eSATIBEA's at any given time for metrics that are to be maximised (i.e., HV, PFS and Spread), and by subtracting eSATIBEA's results from SATIBEA's for the metrics that are to be minimised (i.e., ϵ and IGD). Therefore, a positive gain is always better for eSATIBEA than a negative one.

We see that regarding the HV, the gain is positive on all data sets, for all evolutions and during the whole experiment. This gives a good advantage to eSATIBEA in comparison to SATIBEA in terms of quality of solutions. We see that this gain is larger during the first quarter of the execution time and that it decreases quickly to almost become null. However, we also

Table 6: Evaluation of the performance of eSATIBEA vs. SATIBEA on the different targets of the Linux kernel in four optimisation snapshots (i.e., initial, first, middle and last generations) using five different metrics (i.e., HV, ϵ, IGD, PFS and Spread). We also report the significance of eSATIBEA's results in comparison to those of SATIBEA at each of the four snapshots by means of MWU and $\hat{A}_{12}$.

Target	Metric	SATIBEA				eSATIBEA				Sig Init		Sig First		Sig Middle		Sig Last	
		Init	First	Middle	Last	Init	First	Middle	Last	MWU	$\hat{A}_{12}$	MWU	$\hat{A}_{12}$	MWU	$\hat{A}_{12}$	MWU	$\hat{A}_{12}$
	HV (e-2)	5.809	10.893	21.172	23.538	23.607	23.775	23.914	23.695	1.51E-11	1.00	1.51E-11	1.00	1.42E-08	0.92	3.52E-07	0.87
	ϵ (e2)	323.041	27.290	26.162	23.500	8.427	8.427	4.181	5.260	6.06E-13	1.00	9.74E-12	1.00	1.51E-11	1.00	1.51E-11	1.00
1.1	IGD (e-4)	25.782	9.340	3.887	3.501	1.373	3.441	3.471	3.493	1.51E-11	1.00	1.51E-11	1.00	4.97E-01	0.50	2.94E-04	0.76
	PFS (e2)	2.270	2.680	2.860	2.890	2.940	2.930	2.870	2.870	6.02E-13	1.00	1.30E-11	1.00	7.86E-08	0.89	1.43E-02	0.66
	S (e-1)	5.061	10.518	11.985	14.038	10.826	10.873	11.196	11.626	1.51E-11	1.00	6.27E-08	0.90	9.25E-09	0.08	1.51E-11	0.00
	HV (e-2)	5.899	9.351	21.676	23.397	23.640	23.996	22.924	22.995	1.51E-11	1.00	1.51E-11	1.00	4.42E-07	0.87	3.49E-03	0.70
	ϵ (e2)	330.902	48.830	25.035	24.817	15.470	12.100	9.020	8.230	6.06E-13	1.00	1.43E-11	1.00	1.51E-11	1.00	1.51E-11	1.00
5.1	IGD (e-4)	25.940	9.695	2.916	3.043	1.441	2.809	3.124	3.282	1.51E-11	1.00	1.51E-11	1.00	1.39E-01	0.42	8.18E-06	0.82
	PFS (e2)	2.160	2.690	2.880	2.910	2.890	2.940	2.900	2.850	5.99E-13	1.00	2.93E-11	0.99	7.59E-04	0.74	1.41E-01	0.58
	S (e-1)	5.128	10.483	11.101	14.142	11.450	11.512	13.161	12.908	1.51E-11	1.00	1.51E-11	1.00	1.26E-01	0.59	1.68E-08	0.08
	HV (e-2)	5.581	14.754	21.862	22.499	23.565	23.858	23.484	23.903	1.51E-11	1.00	1.51E-11	1.00	1.51E-11	1.00	1.44E-10	0.97
	ϵ (e2)	327.501	70.491	69.071	68.469	9.310	7.680	4.870	8.220	6.06E-13	1.00	1.06E-11	1.00	1.51E-11	1.00	1.51E-11	1.00
5.3	IGD (e-4)	23.829	6.870	3.854	3.964	1.451	3.114	3.264	3.165	1.51E-11	1.00	1.51E-11	1.00	4.88E-10	0.96	2.75E-11	0.99
	PFS (e2)	2.220	2.590	2.880	2.870	2.900	2.940	2.940	2.930	1.56E-12	1.00	1.23E-11	1.00	1.97E-05	0.81	6.27E-04	0.74
	S (e-1)	5.000	9.311	12.931	14.328	10.654	11.106	11.767	11.981	1.51E-11	1.00	9.78E-11	0.98	6.11E-03	0.31	9.28E-10	0.05
	HV (e-2)	5.980	15.002	22.720	23.812	23.712	23.901	23.465	23.354	1.51E-11	1.00	1.51E-11	1.00	2.34E-08	0.91	6.51E-04	0.74
	ϵ (e2)	327.668	27.461	21.806	21.806	8.790	8.570	7.490	6.080	6.06E-13	1.00	1.48E-11	1.00	1.48E-11	1.00	1.51E-11	1.00
10.1	IGD (e-4)	25.188	6.304	3.892	3.902	1.637	3.073	3.081	3.124	1.51E-11	1.00	1.51E-11	1.00	1.51E-11	1.00	1.51E-11	1.00
	PFS (e2)	2.250	2.640	2.860	2.860	2.910	2.930	2.950	2.910	2.02E-12	1.00	1.23E-11	1.00	1.37E-05	0.81	2.91E-02	0.64
	S (e-1)	4.758	10.138	12.850	12.614	12.101	12.320	11.636	12.834	1.51E-11	1.00	1.51E-11	1.00	2.28E-01	0.56	2.77E-08	0.09
	HV (e)	5.169	12.337	22.410	22.676	23.680	24.108	23.808	23.872	1.51E-11	1.00	1.51E-11	1.00	3.69E-11	0.99	1.51E-11	1.00
	ϵ (e)	345.886	68.823	61.886	61.679	9.900	8.310	4.570	4.830	6.06E-13	1.00	1.39E-11	1.00	1.51E-11	1.00	1.50E-11	1.00
10.3	IGD (e)	25.885	15.077	3.192	3.144	1.419	2.624	2.785	2.784	1.51E-11	1.00	1.51E-11	1.00	3.03E-11	0.99	1.51E-11	1.00
	PFS (e)	2.230	2.640	2.890	2.940	2.890	2.950	2.890	2.880	2.00E-12	1.00	1.27E-11	1.00	2.38E-07	0.88	1.40E-03	0.72
	S (e)	5.007	9.746	12.849	14.743	10.808	11.130	10.854	12.204	1.51E-11	1.00	1.51E-11	1.00	1.01E-07	0.11	3.69E-11	0.01
	HV (e-2)	5.398	13.911	20.593	21.107	23.506	23.620	23.607	24.092	1.51E-11	1.00	1.51E-11	1.00	1.51E-11	1.00	1.51E-11	1.00
	ϵ (e2)	337.497	108.028	104.636	104.109	12.510	12.510	7.500	6.110	6.06E-13	1.00	1.44E-11	1.00	1.50E-11	1.00	1.51E-11	1.00
10.5	IGD (e-4)	25.109	7.606	4.737	4.843	1.561	3.209	3.403	3.157	1.51E-11	1.00	1.51E-11	1.00	1.51E-11	1.00	1.51E-11	1.00
	PFS (e)	2.200	2.610	2.870	2.930	2.890	2.940	2.880	2.970	1.17E-12	1.00	1.29E-11	1.00	3.77E-09	0.93	2.31E-01	0.56
	S (e2)	4.857	10.204	12.333	12.911	11.342	12.164	12.447	11.792	1.51E-11	1.00	1.51E-11	1.00	7.15E-06	0.83	1.34E-04	0.23
	HV (e-2)	4.808	11.845	22.632	23.460	23.806	23.982	23.445	23.913	1.51E-11	1.00	1.51E-11	1.00	1.60E-09	0.95	3.28E-02	0.64
	ϵ (e2)	354.897	37.528	28.256	28.256	10.720	10.220	5.990	5.610	6.06E-13	1.00	1.51E-11	1.00	1.51E-11	1.00	1.50E-11	1.00
20.1	IGD (e-4)	25.997	8.637	3.099	3.095	1.775	3.056	3.229	2.829	1.51E-11	1.00	1.51E-11	1.00	2.10E-01	0.44	6.64E-03	0.31
	PFS (e2)	2.320	2.660	2.850	2.900	2.860	2.950	2.840	2.880	2.56E-12	1.00	1.88E-10	0.97	3.49E-05	0.80	5.66E-02	0.62
	S (e-1)	5.052	10.610	11.758	13.824	11.232	11.624	12.675	12.576	1.51E-11	1.00	1.67E-11	1.00	1.96E-02	0.66	4.33E-05	0.20
	HV (e-2)	5.278	12.864	21.988	22.346	23.784	23.996	23.117	23.084	1.51E-11	1.00	1.51E-11	1.00	1.58E-10	0.97	1.51E-11	1.00
	ϵ (e3)	36.243	6.554	6.581	6.528	2.247	1.965	1.169	1.169	6.06E-13	1.00	1.42E-11	1.00	1.51E-11	1.00	1.51E-11	1.00
20.3	IGD (e-4)	26.632	7.980	3.443	3.668	2.701	2.737	2.933	2.832	1.51E-11	1.00	1.51E-11	1.00	4.08E-11	0.99	1.51E-11	1.00
	PFS (e2)	2.120	2.740	2.810	2.860	2.910	2.910	2.910	2.880	3.20E-12	1.00	4.77E-11	0.99	3.89E-04	0.75	3.69E-01	0.47
	S (e-1)	5.082	10.286	12.007	14.528	11.461	11.096	13.539	14.417	1.51E-11	1.00	1.51E-11	1.00	3.81E-03	0.70	1.35E-01	0.42
	HV (e-2)	4.249	13.347	21.005	21.676	23.746	24.067	24.113	23.734	1.51E-11	1.00	1.51E-11	1.00	1.51E-11	1.00	1.51E-11	1.00
	ϵ (e2)	368.767	99.553	95.211	94.012	14.440	9.940	8.550	7.602	6.06E-13	1.00	1.34E-11	1.00	1.51E-11	1.00	1.51E-11	1.00
20.5	IGD (e-4)	25.953	7.789	4.478	4.425	1.731	3.019	3.121	3.193	1.51E-11	1.00	1.51E-11	1.00	1.51E-11	1.00	1.51E-11	1.00
	PFS (e2)	2.320	2.700	2.920	2.950	2.910	2.960	2.960	2.950	1.56E-12	1.00	1.28E-11	1.00	1.94E-10	0.97	4.88E-01	0.50
	S (e-1)	4.900	10.339	13.109	13.091	11.032	11.063	12.183	12.466	1.51E-11	1.00	1.67E-11	1.00	1.42E-04	0.77	1.23E-01	0.41
	HV (e-2)	5.257	14.065	16.808	17.194	23.368	23.733	23.673	23.629	1.51E-11	1.00	1.51E-11	1.00	1.51E-11	1.00	1.51E-11	1.00
	ϵ (e3)	36.364	18.903	18.200	18.192	4.766	2.905	1.442	1.588	6.06E-13	1.00	1.19E-11	1.00	1.51E-11	1.00	1.51E-11	1.00
20.1	IGD (e-4)	23.014	9.230	8.507	8.662	1.754	3.107	3.289	3.402	1.51E-11	1.00	1.51E-11	1.00	1.51E-11	1.00	1.51E-11	1.00
	PFS (e2)	2.160	2.720	2.850	2.920	2.870	2.920	2.930	2.880	2.58E-12	1.00	1.35E-11	1.00	1.37E-07	0.89	2.42E-01	0.55
	S (e-1)	4.804	9.947	11.868	12.841	11.656	12.165	13.308	14.094	1.51E-11	1.00	1.51E-11	1.00	3.35E-11	0.99	2.09E-09	0.94

see that the gain on large evolution targets does not decrease to zero, but rather converges to higher values. This is an interesting point to notice as it means that when the evolution is important (large difference between an FM and the new version) eSATIBEA shows more robustness and leverages the previous populations.

We also see the same type of behaviour when looking at the gain on ϵ. However, this gain stays positive (good for eSATIBEA) for a larger number of evolution targets.

When it comes to the gain on IGD, we notice two different trends: (i) on Linux kernel and FreeBSD data sets where we have the same behaviour as with the gain on HV, and (ii) on eCos, Fiasco and μClinux where we start with a positive gain which drops sharply to even turning negative (meaning eSATIBEA is better for these values) for a short duration, before getting back to the positive side and stabilising around zero for the rest of the execution. Notice that despite these two trends, the difference is not significant enough given the small size of the results.

The gains on the variety metrics look more erratic as they oscillate a lot between positive and negative values. The gain on FPS is mostly positive on the Linux kernel as it starts with large values which decrease over the execution time. However, we see a totally different trend on other data sets, as gains start with negative values before narrowing the interval of their oscillation (usually within [-30, 30]). The gains on the Spread are also varying a lot between positive and negative values. The gain starts with positive values on all targets but quickly drops below zero. This is then followed by an increase to reach null values and oscillates around it.

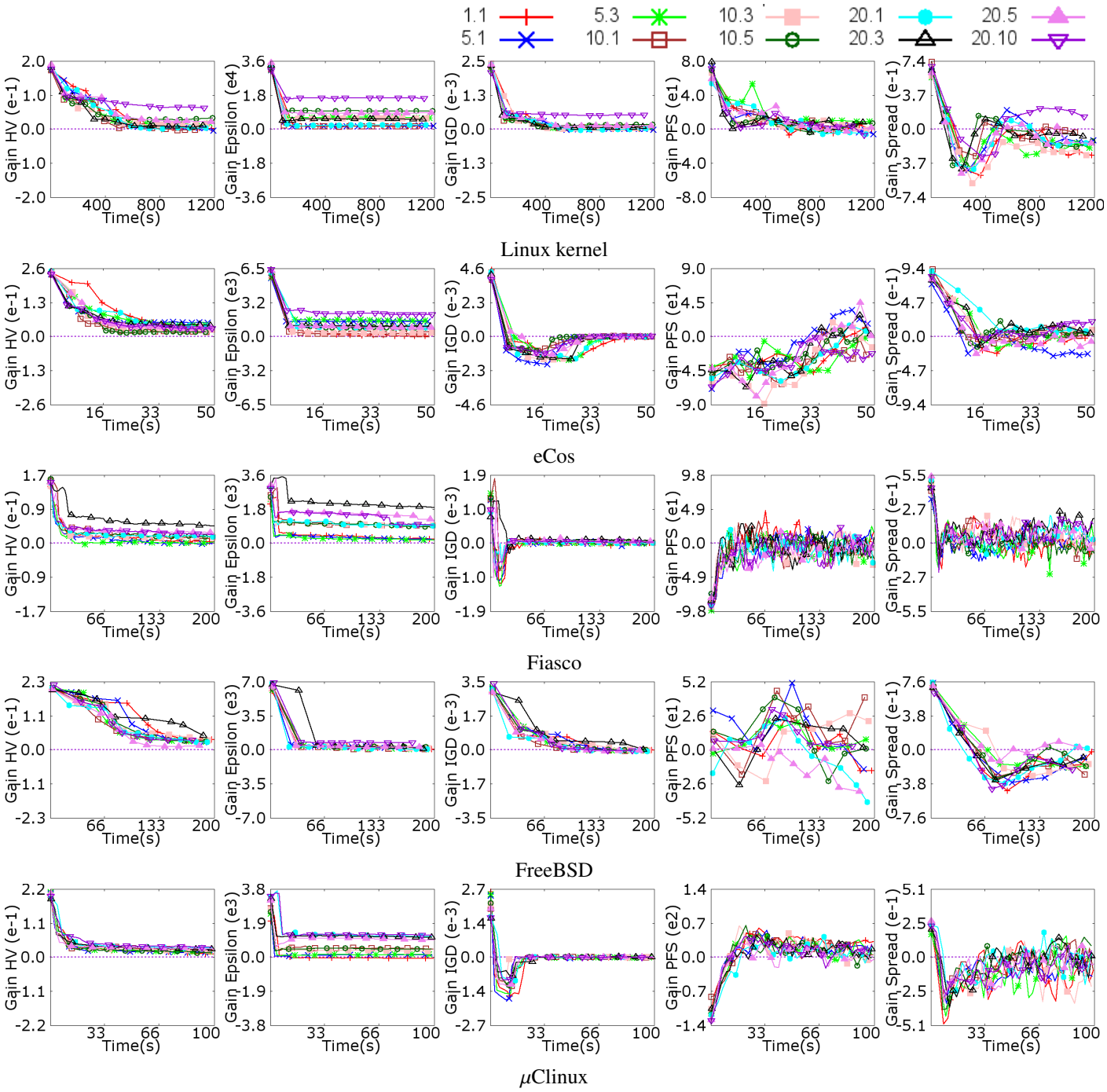

Figure 6: Gain of eSATIBEA on SATIBEA for all data sets on all metrics. Note that while the gain is computed as result(eSATIBEA) - result(SATIBEA) for metrics that are to be maximised, we inverse that for those that are to be minimised.

Overall, Figure 6 confirms what has been seen in Figure 4: eSATIBEA does not only find solutions of better quality than SATIBEA only on the Linux kernel, but also on the other data sets. Similarly, they also show that the further the evolution is from the original FM, the more eSATIBEA converges to qualitatively better results than SATIBEA. However, Figure 6 seems to indicate that eSATIBEA only outperforms SATIBEA in terms of variety of solutions at the beginning of the experiments, but they usually end up taking on each other for the rest of the experiment without any of them having a clear lead.

7. Threats to Validity

We have identified some threats to the validity of our evaluation. The first threat to validity in our work is in the choice of data sets. Our work is based on five data sets (i.e., Linux kernel, eCos, Fiasco, FreeBSD and μClinux). Therefore, results might not generalise to other data sets. In order to lower this risk, we

have selected data sets of different sizes, which are, moreover, the same as those used by Henard et al. [33].

The second threat to validity is in the creation of the benchmark and the choice of target distances. We generate in our work 10 different targets with a difference of features between 1 and 20% and a difference of number of constraints between 1 and 10% from the initial version. Next, given that we could only find the different versions of the Linux kernel data set to study their evolution patterns, we use the same patterns on all the data sets which might not be true in reality. Last, we generate the target instances with some random parameters which might lead to a lack of coverage of specific cases. In our work, we generate 10 target instances for each target in order to lower this risk.

The last threat to valid is in the experiments themselves with errors in the code, disturbances from the machine used for the experiments, and the effect of randomness in both within SATIBEA and during the generation of seeds. To lower this risk, we use the code of SATIBEA made available by Henard et al. [33] that we extend using the same language and libraries. We also run the experiments on the same machine in total isolation (no parallel processes with the exception of those proper to the OS itself). Furthermore, we run our experiments for 30 iterations (each run of eSATIBEA given a different set of seeds as initial population, that is found by SATIBEA on the original instance) and check the significance of the results using two different tests.

8. Related Work

This section describes the relevant related work that we have used for our research. In particular, we have studied extensively the optimisation approaches for SPLE, both in the mono- (only one dimension) and multi-objective contexts. We have also examined the concept of evolution in SPLE, a relatively overlooked but important concept in Software Engineering and SPLE. For more complete reviews of optimisation problems and techniques for SPLE, see Harman et al. [31] and Lopez-Herrejon et al. [41].

Obviously, any studies of SPLE, in particular, when optimisation algorithms are used to identify and improve potential products, rely on understanding the effect/impact (e.g., performance, cost) of features in products. See the works of Siegmund et al. [56, 55, 54], Valov et al. [58], and Zhang et al. [64] for a complete description of this topic.

8.1. Mono- and Multi-objective Optimisation in SPLE

As we have described it before, the goal of feature selection in SPLE is to find and improve products in a large search space composed of a large number of features and constraints. This can be formalised as an optimisation problem for which many techniques have been described in the operations research and optimisation literature. A lot of research has been done in SPLE community in this context; in particular, authors have described the problem as mono-objective or multi-objective, depending on whether they consider the different dimensions of the optimisation as independent or not.

8.1.1. Mono-objective Optimisation

Benavides et al. [5] provide an overview of concepts for an automated reasoning on feature models, including 'optimum products' based on a single objective function. They later introduce FAMA [7], an extensible framework for automated analysis of feature models, and provide a literature review on the topic [6].

White et al. [62] suggest 'Filtered Cartesian Flattening' (FCF) an approximation technique that selects highly optimal feature sets while considering resource constraints.

Guo et al. [29] suggest GAFES, a genetic algorithm for product configuration, where they use repair to transform invalid selections, occurring after crossover operations, into valid ones. They report that their approach outperforms FCF [62] for large generated feature models.

As we suggest in this paper, feature selection in SPLE is better suited for multi-objective optimisation – so while some of the former works describe well the feature selection and so on, we focus on a multi-objective definition of the feature selection problem.

8.1.2. Multi-objective Optimisation in SPLE

Karimpour and Ruhe [36] frame the scoping of a product line (i.e., the decision on which features to include) as a bi-criteria optimisation problem with a heuristic considering *profit* and *stability*.

Dos Santos Neto et al. [21] apply multi-objective optimisation to find product portfolios (that minimise *cost* and maximise *relevancy*) and provide a discussion of work on the related Next Release Problem.

Colanzi and Vergilio [15, 16, 17, 18] interpret the design of a product line architecture (PLA) as a multi-objective optimisation problem. They extend a genetic algorithm (NSGA-II) with a feature-driven crossover operator that aims to improve feature modularisation.

Guizzo et al. [27] extend the work by using a mutation operator that is based on design patterns.

Henard et al. [34] use a multiple-objective genetic algorithm combined with constraint solving techniques to generate tests for SPLs.

Other works on test optimisation for product lines are provided, for instance, by Wang et al. [60] and Lopez-Herrejon et al. [39, 40]

Sayyad et al. [49] define a five-objective optimisation problem based on feature configuration, aiming to minimise total costs, known defects, and rule violations, and maximise the total number of products offered by products as well as features reused from previous products. They report that IBEA (Indicator-Based Evolutionary Algorithm) has an advantage over the previously often used NSGA-II due to the way it exploits user preference. In subsequent work, they investigate the effect of parameter tuning and the effects of slowing down the rates of crossover and mutation [46, 47]. Finally, they suggest a heuristic to 'seed' the IBEA used earlier with a pre-computed solution [48].

Olaechea et al. [43] provide a comparison of exact techniques (Guided Improvement Algorithm, GIA) and approximate tech-

niques (IBEA) for multi-objective optimisation applied to feature configuration.

Guo et al. [30] aim to speed up exact techniques with parallel extensions, comparing the speed-up when using collaborative communication, divide-and-conquer, or a combination of both. They report that one algorithm called FS-GIA (Feature Split GIA) outperforms all other proposed algorithms and scales well up to 64 cores.

Henard et al. [33] introduce SATIBEA where they combine the genetic algorithm (IBEA) with a SAT solver to repair invalid configurations. The work presented in this paper is based on an extension of SATIBEA to evolving Feature Models, which we call eSATIBEA.

We follow the work of Henard et al. [33] in our problem definition and modelling, and we were inspired by some of the work by Sayyad et al. [49]. The important difference with all the previous work though is the evaluation of FM evolution on the algorithms, in particular, whether seeding previous solutions improve the results.

8.2. Evolution of Software Product Lines

The other important element in our study is the evolution of SPLE, that has been somewhat studied in the past – mostly the tooling/engineering aspects of it, not the optimisation as in our own study.

Botterweck et al. [12, 45] suggest EvoFM, a model-based approach for planning and tracking product line evolution at a feature level. Guo et al. [28] suggest approaches for maintaining the consistency in evolving feature models.

Czarnecki et al. [20] provide a fundamental model of feature model configuration called 'staged configuration', where a feature model is configured in multiple phases, leading to a step-wise specialisation. White et al. [63] consider the configuration of a feature model as a multi-step process. However, they interpret the process as a constraint satisfaction problem where, for instance, each configuration step may have costs and constraints on these costs need to be observed.

Dhungana et al. [24, 23] aim to ease product line evolution by organising variability models of large product lines as a set of interrelated fragments. They provide semi-automatic tool support to merge fragments into complete product line models and consider the coevolution of variability models and their corresponding meta-models. Schmid and Verlage [50] discuss the economic impact of product line adoption and evolution. Laguno and Crespo [37] provide a systematic mapping study on product line evolution. Botterweck and Pleuss [11] give an overview of concepts in software product line evolution.

9. Conclusion

This paper has presented a new problem: the configuration of Software Product Lines when the Feature Models they are based on evolve.

To study this problem, we have proposed a benchmark based on a study of the evolution of a large Feature Model (up to 13,000+ features and nearly 300,000 constraints, 20 different versions over 4+ years). Our empirical study gave us the characteristics and demographics behind the evolution of that Feature Model - and we have replicated it to generate synthetic versions of 5 different data sets (Feature Models).

We have compared SATIBEA, the leading algorithm in the literature, and our contribution eSATIBEA (which takes an adaptation of the previous solutions as initial population) in this evolving context. Our experiments show that eSATIBEA improves both the execution time and the quality of SATIBEA by feeding it with previous configurations. In particular, eSATIBEA proves to converge an order of magnitude faster than SATIBEA alone.

Acknowledgement

This work was supported with the financial support of the Science Foundation Ireland grant 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero - the Irish Software Research Centre (`www.lero.ie`).

References

[1] Abbas, A., Siddiqui, I. F., Lee, S. U.-J., 2016. Multi-objective optimization of feature model in software product line: Perspectives and challenges. IJST.

[2] Alshahwan, N., Harman, M., 2011. Automated web application testing using search based software engineering. In: ASE. pp. 3–12.

[3] Apel, S., Batory, D. S., Kästner, C., Saake, G., 2013. Feature-Oriented Software Product Lines - Concepts and Implementation. Springer.

[4] Arcuri, A., Briand, L., 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: ICSE. pp. 1–10.

[5] Benavides, D., Martín-Arroyo, P. T., Cortés, A. R., 2005. Automated reasoning on feature models. In: CAiSE. pp. 491–503.

[6] Benavides, D., Segura, S., Cortés, A. R., 2010. Automated analysis of feature models 20 years later: A literature review. Inf. Syst. 35 (6), 615–636.

[7] Benavides, D., Segura, S., Trinidad, P., Cortés, A. R., 2007. FAMA: tooling a framework for the automated analysis of feature models. In: VaMoS. pp. 129–134.

[8] Berger, T., She, S., Lotufo, R., Wasowski, A., Czarnecki, K., 2010. Variability modeling in the real: a perspective from the operating systems domain. In: ASE. pp. 73–82.

[9] Berger, T., She, S., Lotufo, R., Wasowski, A., Czarnecki, K., 2012. Variability modeling in the systems software domain. Generative Software Development Laboratory, University of Waterloo, Technical Report.

[10] Berger, T., She, S., Lotufo, R., Wasowski, A., Czarnecki, K., 2013. A study of variability models and languages in the systems software domain. IEEE Transactions on Software Engineering, 1611–1640.

[11] Botterweck, G., Pleuss, A., 2014. Evolution of software product lines. In: Mens, T., Serebrenik, A., Cleve, A. (Eds.), Evolving Software Systems. Springer, pp. 265–295.

[12] Botterweck, G., Pleuss, A., Dhungana, D., Polzer, A., Kowalewski, S., 2010. Evofm: feature-driven planning of product-line evolution. In: PLEASE@ICSE. pp. 24–31.

[13] Brevet, D., Saber, T., Botterweck, G., Ventresque, A., 2016. Preliminary study of multi-objective features selection for evolving software product lines. In: SSBSE. pp. 274–280.

[14] Brockhoff, D., Friedrich, T., Neumann, F., 2008. Analyzing hypervolume indicator based algorithms. In: PPSN. pp. 651–660.

[15] Colanzi, T. E., 2012. Search based design of software product lines architectures. In: ICSE. pp. 1507–1510.

[16] Colanzi, T. E., Vergilio, S. R., 2012. Applying search based optimization to software product line architectures: Lessons learned. In: SSBSE. pp. 259–266.

[17] Colanzi, T. E., Vergilio, S. R., 2013. Representation of software product line architectures for search-based design. In: CMSBSE@ICSE. pp. 28–33.

[18] Colanzi, T. E., Vergilio, S. R., 2016. A feature-driven crossover operator for multi-objective and evolutionary optimization of product line architectures. Journal of Systems and Software 121, 126–143.

[19] Coplien, J., Hoffman, D., Weiss, D., 1998. Commonality and variability in software engineering. IEEE Software 15 (6), 37–45.

[20] Czarnecki, K., Helsen, S., Eisenecker, U. W., 2004. Staged configuration using feature models. In: SPLC. pp. 266–283.

[21] de Alcântara dos Santos Neto, P., Britto, R., de Andrade Lira Rabelo, R., Cruz, J., Ayala, W., 2016. A hybrid approach to suggest software product line portfolios. Appl. Soft Comput. 49, 1243–1255.

[22] Deb, K., Mohan, M., Mishra, S., 2003. Towards a quick computation of well-spread pareto-optimal solutions. In: EMO. pp. 222–236.

[23] Dhungana, D., Grünbacher, P., Rabiser, R., Neumayer, T., 2010. Structuring the modeling space and supporting evolution in software product line engineering. Journal of Systems and Software 83 (7), 1108–1122.

[24] Dhungana, D., Neumayer, T., Grünbacher, P., Rabiser, R., 2008. Supporting evolution in model-based product line engineering. In: SPLC. pp. 319–328.

[25] Fonseca, C. M., Knowles, J. D., Thiele, L., Zitzler, E., 2005. A tutorial on the performance assessment of stochastic multiobjective optimizers. In: EMO 2005. p. 240.

[26] Fraser, G., Arcuri, A., 2012. The seed is strong: Seeding strategies in search-based software testing. In: ICST. pp. 121–130.

[27] Guizzo, G., Colanzi, T. E., Vergilio, S. R., 2014. A pattern-driven mutation operator for search-based product line architecture design. In: SSBSE. pp. 77–91.

[28] Guo, J., Wang, Y., Trinidad, P., Benavides, D., 2012. Consistency maintenance for evolving feature models. Expert Syst. Appl. 39 (5), 4987–4998.

[29] Guo, J., White, J., Wang, G., Li, J., Wang, Y., 2011. A genetic algorithm for optimized feature selection with resource constraints in software product lines. Journal of Systems and Software 84 (12), 2208–2221.

[30] Guo, J., Zulkoski, E., Olaechea, R., Rayside, D., Czarnecki, K., Apel, S., Atlee, J. M., 2014. Scaling exact multi-objective combinatorial optimization by parallelization. In: ASE. pp. 409–420.

[31] Harman, M., Jia, Y., Krinke, J., Langdon, W. B., Petke, J., Zhang, Y., 2014. Search based software engineering for software product line engineering: A survey and directions for future work. In: SPLC. pp. 5–18.

[32] Hawkins, T., 2001. Lebesgue's theory of integration: its origins and development. Vol. 282. American Mathematical Soc.

[33] Henard, C., Papadakis, M., Harman, M., Le Traon, Y., 2015. Combining multi-objective search and constraint solving for configuring large software product lines. In: ICSE. pp. 517–528.

[34] Henard, C., Papadakis, M., Perrouin, G., Klein, J., Traon, Y. L., 2013. Multi-objective test generation for software product lines. In: SPLC. pp. 62–71.

[35] Ishibuchi, H., Masuda, H., Tanigaki, Y., Nojima, Y., 2015. Modified distance calculation in generational distance and inverted generational distance. In: EMO. pp. 110–125.

[36] Karimpour, R., Ruhe, G., 2015. A search based approach towards robust optimization in software product line scoping. In: GECCO. pp. 1415–1416.

[37] Laguna, M. A., Crespo, Y., 2013. A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring. Sci. Comput. Program. 78 (8), 1010–1034.

[38] Loesch, F., Ploedereder, E., 2007. Optimization of variability in software product lines. In: SPLC. pp. 151–162.

[39] Lopez-Herrejon, R. E., Chicano, F., Ferrer, J., Egyed, A., Alba, E., 2013. Multi-objective optimal test suite computation for software product line pairwise testing. In: ICSM. pp. 404–407.

[40] Lopez-Herrejon, R. E., Ferrer, J., Chicano, F., Egyed, A., Alba, E., 2014. Comparative analysis of classical multi-objective evolutionary algorithms and seeding strategies for pairwise testing of software product lines. In: CEC. pp. 387–396.

[41] Lopez-Herrejon, R. E., Linsbauer, L., Egyed, A., 2015. A systematic mapping study of search-based software engineering for software product lines. Information & Software Technology 61, 33–51.

[42] Metzger, A., Pohl, K., 2014. Software product line engineering and variability management: achievements and challenges. In: FSE. pp. 70–84.

[43] Olaechea, R., Rayside, D., Guo, J., Czarnecki, K., 2014. Comparison of exact and approximate multi-objective optimization for software product lines. In: SPLC. pp. 92–101.

[44] Passos, L., Czarnecki, K., Apel, S., Wasowski, A., Kästner, C., Guo, J., 2013. Feature-oriented software evolution. In: VaMoS. p. 17.

[45] Pleuss, A., Botterweck, G., Dhungana, D., Polzer, A., Kowalewski, S., 2012. Model-driven support for product line evolution on feature level. Journal of Systems and Software 85 (10), 2261–2274.

[46] Sayyad, A. S., Goseva-Popstojanova, K., Menzies, T., Ammar, H., 2013. On parameter tuning in search based software engineering: A replicated empirical study. In: RESER. pp. 84–90.

[47] Sayyad, A. S., Ingram, J., Menzies, T., Ammar, H., 2013. Optimum feature selection in software product lines: Let your model and values guide your search. In: CMSBSE@ICSE. pp. 22–27.

[48] Sayyad, A. S., Ingram, J., Menzies, T., Ammar, H., 2013. Scalable product line configuration: A straw to break the camel's back. In: ASE. pp. 465–474.

[49] Sayyad, A. S., Menzies, T., Ammar, H., 2013. On the value of user preferences in search-based software engineering: a case study in software product lines. In: ICSE. pp. 492–501.

[50] Schmid, K., Verlage, M., 2002. The economic impact of product line adoption and evolution. IEEE Software 19 (4), 50–57.

[51] She, S., 2013. Feature model synthesis. Ph.D. thesis, University of Waterloo.

[52] She, S., 2013. Lvat: Linux variability analysis tools. http://code.google.com/p/linux-variability-analysis-tools.

[53] She, S., Lotufo, R., Berger, T., Wasowski, A., Czarnecki, K., 2011. Reverse engineering feature models. In: ICSE. pp. 461–470.

[54] Siegmund, N., Grebhahn, A., Apel, S., Kästner, C., 2015. Performance-influence models for highly configurable systems. In: ESEC/FSE. pp. 284–294.

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

[56] Siegmund, N., Rosenmüller, M., Kuhlemann, M., Kästner, C., Apel, S., Saake, G., 2012. SPL conqueror: Toward optimization of non-functional properties in software product lines. Software Quality Journal 20 (3-4), 487–517.

[57] Thüm, T., Batory, D., Kästner, C., 2009. Reasoning about edits to feature models. In: ICSE. pp. 254–264.

[58] Valov, P., Guo, J., Czarnecki, K., 2015. Empirical comparison of regression methods for variability-aware performance prediction. In: SPLC. pp. 186–190.

[59] Vargha, A., Delaney, H. D., 2000. A critique and improvement of the cl common language effect size statistics of mcgraw and wong. Journal of Educational and Behavioral Statistics, 101–132.

[60] Wang, S., Ali, S., Gotlieb, A., 2013. Minimizing test suites in software product lines using weight-based genetic algorithms. In: GECCO. pp. 1493–1500.

[61] Wang, S., Ali, S., Yue, T., Li, Y., Liaaen, M., 2016. A practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software engineering. In: ICSE. pp. 631–642.

[62] White, J., Dougherty, B., Schmidt, D. C., 2009. Selecting highly optimal architectural feature sets with filtered cartesian flattening. Journal of Systems and Software 82 (8), 1268–1284.

[63] White, J., Galindo, J. A., Saxena, T., Dougherty, B., Benavides, D., Schmidt, D. C., 2014. Evolving feature model configurations in software product lines. Journal of Systems and Software 87, 119–136.

[64] Zhang, Y., Guo, J., Blais, E., Czarnecki, K., Yu, H., 2016. A mathematical model of performance-relevant feature interactions. In: SPLC. pp. 25–34.

[65] Zitzler, E., Künzli, S., 2004. Indicator-based selection in multiobjective search. In: PPSN. pp. 832–842.

[66] Zitzler, E., Thiele, L., 1998. Multiobjective optimization using evolutionary algorithms—a comparative case study. In: PPSN. pp. 292–301.

[67] Zitzler, E., Thiele, L., 1999. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. IEEE Transactions on Evolutionary Computation 3 (4), 257–271.

[68] Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., Da Fonseca, V. G., 2003. Performance assessment of multiobjective optimizers: An analysis and review. IEEE Transactions on evolutionary computation, 117–132.