MILPIBEA: Algorithm for Multi-objective Features Selection in (Evolving) Software Product Lines

Takfarinas Saber[1](✉), David Brevet[2], Goetz Botterweck[3], and Anthony Ventresque[1]

[1] School of Computer Science, University College Dublin, Dublin, Ireland
{takfarinas.saber,anthony.ventresque}@ucd.ie

[2] Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes, Aubiere, France
david.brevet@uca.fr

[3] University of Limerick, Limerick, Ireland
goetz.botterweck@lero.ie

Abstract. *Software Product Lines Engineering* (SPLE) proposes techniques to model, create and improve groups of related software systems in a systematic way, with different alternatives formally expressed, e.g., as *Feature Models*. Selecting the 'best' software system(s) turns into a problem of improving the quality of selected subsets of software features (components) from feature models, or as it is widely known, *Feature Configuration*. When there are different independent dimensions to assess how good a software product is, the problem becomes even more challenging – it is then a multi-objective optimisation problem. Another big issue for software systems is evolution where software components change. This is common in the industry but, as far as we know, there is no algorithm designed to the particular case of multi-objective optimisation of evolving software product lines. In this paper we present MILPIBEA, a novel hybrid algorithm which combines the scalability of a genetic algorithm (IBEA) with the accuracy of a mixed-integer linear programming solver (IBM ILOG CPLEX). We also study the behaviour of our solution (MILPIBEA) in contrast with SATIBEA, a state-of-the-art algorithm in static software product lines. We demonstrate that MILPIBEA outperforms SATIBEA on average, especially for the most challenging problem instances, and that MILPIBEA is the one that continues to improve the quality of the solutions when SATIBEA stagnates (in the evolving context).

Keywords: Software product line · Feature selection · Multi-objective optimisation · Evolutionary algorithm · Mixed-integer linear programming

© Springer Nature Switzerland AG 2020
L. Paquete and C. Zarges (Eds.): EvoCOP 2020, LNCS 12102, pp. 164–179, 2020.
https://doi.org/10.1007/978-3-030-43680-3_11

1 Introduction

Software Engineering combines various domains [1]. Software Product Lines (SPL) is one of these domains that deal with groups of related software systems as an ensemble, instead of handling each of them independently [2]. SPL is getting more attention by the software industry as it simplifies software reuse [3] and enables better reliability and important reduction in cost [4].

A common way to represent a product line, all available products and their essential characteristics, is a Feature Model (FM). Every feature corresponds to an element of a software system/product that is of interest to a particular company. Each FM describes the available configuration choices, and consequently the set of all possible products as combinations of features. These FMs can grow to become very large (e.g., in this paper, we use FMs with ∼7k features and ∼350k constraints).

When deriving a particular product from the product line, we have to perform a feature selection. To find the best possible product, we *optimise the feature selection*, i.e., pick the set of features which gives us the 'best' product [5]. Since in practice various characteristics often have to be considered simultaneously (e.g., cost, technical feasibility, or reliability) finding the 'best' feature selection is an instance of a *multi-objective optimisation problem* [6].

A similar problem that is not fully studied in the literature is *the multi-objective feature selection* when FMs evolve. There is a continual evolution of software libraries and a constant change in customers' preferences regarding the requirements of software applications. These evolutions appear as an adaptation of the FM from a version to another. For instance, Saber et al. [7] have shown in their study that the large FM representing the Linux kernel evolves continuously. They have also shown that a new version of the kernel is released every few months with a successive difference that can go up to 7%.

In this paper, we propose to leverage the evolution context when performing optimisations of feature configurations. It seems odd to generate random bootstrapping populations for SATIBEA in the presence of well-performing solutions for similar problem instances. At the same time, it might be beneficial to exploit the fact that the FM has evolved and that configurations generated previously are close enough and can be adapted.

This paper presents our approach, MILPIBEA, which was initially designed to address the problem of feature selection in a multi-objective context when the FMs evolve, but proved to be better than SATIBEA both when the FMs evolve and when they do not. MILPIBEA is a hybrid algorithm that uses a genetic algorithm (IBEA) and a mixed-integer linear programming (MILP) solver (IBM ILOG CPLEX).

SATIBEA [6] (also a hybrid algorithm) faces a difficult challenge: the search space is so large and constrained that mutation and crossover operations generate a large number of infeasible solutions. SATIBEA uses a SAT solver to fix these infeasible solutions and obtains (close to) viable individuals at each generation of the genetic algorithm. However, this process has two major issues: (i) it is time-consuming – an empirical study of SATIBEA showed that the vast majority of

the execution time consists in fixing the faulty individuals; (ii) it modifies the individuals, often substantially, which defies the idea behind genetic algorithms – where you expect to inherit properties from previous generations and modify the individuals only marginally. MILPIBEA's correction of individuals is both more efficient and more effective, making sure the corrected individuals are closer to the ones generated by IBEA's mutation and crossover.

This paper makes the following contributions:

- We propose MILPIBEA, a hybrid algorithm that outperforms the SATIBEA [6] both in terms of execution time and quality of the solutions.
- We thoroughly evaluate SATIBEA and MILPIBEA on evolving and non-evolving SPL problems and show that MILPIBEA is 42% better than SATIBEA in hypervolume on average, especially for the most challenging problem instances, and that MILPIBEA is the one that continues to improve the quality of solutions when SATIBEA stagnates (in the evolving context) and does improve the quality of solutions.

Combining a solver with a multi-objective evolutionary algorithm has already been proposed to address the particular problem of multi-objective feature selection in SPL [6,8,9] and problems from various other problem domains (e.g., cloud computing [10–12]). However, this is the first work that proposes using a MILP solver for the multi-objective feature selection in SPL.

The remainder of this paper is organised as follows: Sect. 2 describes the context of our study. Section 3 provides the overall set-up and the benchmark for evolving SPL. We then discuss potential improvements in three steps: Sect. 4 motivates seeding previous solutions when dealing with evolving FMs. Section 5 compares the correction mechanisms of SATIBEA and MILPIBEA. Section 6 discusses how MILPIBEA performs in comparison to SATIBEA in terms of achieved hypervolume and required time. Finally, Sect. 7 concludes the paper.

2 Background

In this section, we present four elements that form our research's background:

- Software Product Line Engineering, in particular how to describe variations of software applications as configurations of a feature model.
- Multi-objective optimisation (MOO); 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, and implementations change constantly. Therefore, feature models need to be updated to reflect these evolutions [7].
- SATIBEA, a state-of-the-art algorithm to address the MOO for feature selection in feature models [6] and the same when FMs evolve [7].

2.1 Software Product Line Engineering

Software engineers often need to adapt software artefacts to the needs of a particular customer. Software Product Line Engineering (SPLE) 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 which can be selected and combined to obtain a particular product.

Feature Models can be represented as a set of features and connecting relationships (constraints). Figure 1 shows a toy FM which has ten features connected by several relationships. For instance, each 'Screen' has to be have exactly one of three types, i.e., 'Basic', 'Colour' or 'High Resolution'. When deriving a software product from the software 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., instantiating variables (in our case, features) with the values true or false in a way that satisfies all the constraints. Let $f_i \in \{\text{true, false}\}$ which is 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 represented in a conjunctive normal form (CNF). Finding a product in the SPL is then equivalent to assigning a value in $\{\text{true, false}\}$ to every feature. For instance, in Fig. 1 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 screen features. When configuring a SPL, software designers do not limit themselves to finding possible products (satisfying the FM) but also attempt to discover products optimising multiple criteria. For this reason, SPL configuration is modelled as multi-objective problem.

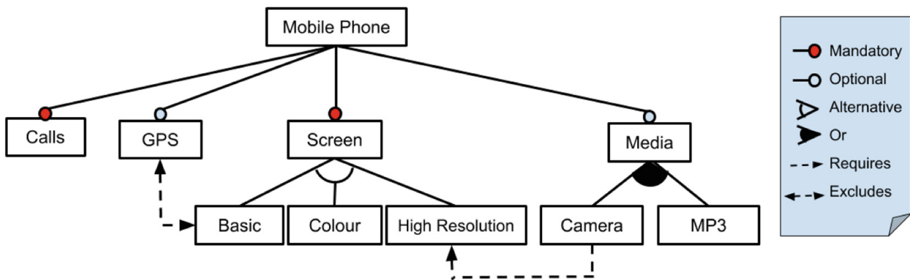

Fig. 1. Example of a feature model

2.2 Multi-objective Optimisation

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 (e.g., cost, importance, reliability), feature selection in SPL is a good candidate for MOO [6].

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 $\forall x \in S$, $F = [O_1(x), ..., O_k(x)]$ represents a vector containing values of the k objective functions for a given solution x. We say that a solution x_1 dominates x_2, written as $x_1 \succ x_2$, if and only if $\forall i \in \{1, ..., k\}$, $O_i(x_1) \leq O_i(x_2)$ and $\exists i \in \{1, ..., k\}$ such that $O_i(x_1) < O_i(x_2)$. We also say that x_i is a non-dominated solution if there is no other solution x_j in the Pareto front s.t. x_j dominates x_i.

All the non-dominated solutions represent a set called a Pareto front: in this set, it is impossible to find any solution better in all objectives than the other solutions in the set. The Pareto front given in Fig. 2 contains solutions x_1, x_2, x_4, x_6, x_7 because they are not dominated by any other, while, for instance, x_8 is dominated by x_1. Hence, x_8 is not in the Pareto front.

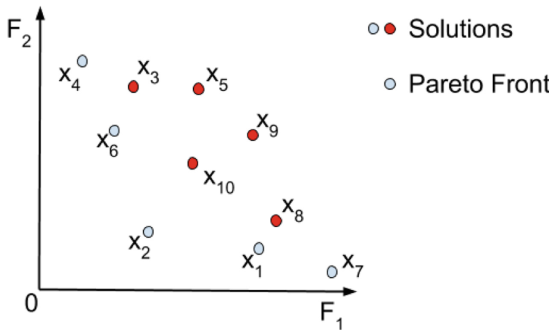

Fig. 2. Example of a Pareto front with two minimisation objectives.

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 [13]. For instance, in Sect. 3 we describe a study of a large-scale FM, the Linux kernel by Saber et al. [7] which shows that every few months a new FM is released with up to 7% modifications among the features (features added or removed).

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 $f_i^2 \in FM_2$ (treated as the same), whereas other 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.

2.4 SATIBEA

SATIBEA [6] is an extension of the Indicator-Based Evolutionary Algorithm (IBEA) which guides the search by a quality indicator given by the user. Previously to SATIBEA, several techniques have been tried to solve the multi-objective feature selection in 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. [14] and has since been widely used in the literature [6–8]. It is not the best possible decision and is acceptable only for small problems (only small FMs are solvable with exact algorithms [15]).

SATIBEA has been introduced to help IBEA find valid products using a SAT solver. SATIBEA changes the mutation process of IBEA: 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).

Using this novel mutation approach, SATIBEA finds better solutions than IBEA: it finds valid optimised products, but also gives better values in quality metrics.

In this paper, we propose MILPIBEA; a novel technique that addresses some of SATIBEA's limitations (i.e., slow and stagnating performance improvements).

3 System Set-Up

This section presents the different elements that we have used in our implementation: the data set, the objectives we use for our multi-objective optimisation problem, the metric we use (i.e., hypervolume), the parameters we use for the genetic algorithm (i.e., IBEA) and the hardware configuration.

3.1 Benchmark for Evolving FMs

Our work is based on the largest open-source FM we could find in the literature: the Linux kernel version 2.6.28 containing 6,888 features and 343,944 constraints.

Saber et al. [7,16] studied the demographics (features/constraints) and evolution pattern of 21 successive versions of the Linux kernel (going from 2.6.12 to 2.6.32). They observed that on average there was only 4.6% difference in terms of features between a version and the next (out of those changes, 21.22% were removed features and 78.78% were added features). They 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 and found that a large proportion of the FMs' constraints have 6 features (39%), 5 features (16%), 18 features (14%) or 19 features (14%). Saber et al. [7] put at our disposal a generator of synthetic FM evolutions 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.

The FM generator provided by Saber et al. 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 is from its original. The FM generator uses the proportions 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 evolved 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 parameter values. Data is available at https://github.com/aventresque/EvolvingFMs.

3.2 Optimisation Objectives

We use a set of optimisation objectives from the literature [6]:

1. *Correctness* – minimise the number of violated constraints, proposed by Sayyad et al. [14].
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 (we use random integer values between 0 and 10).
5. *Cost* – minimise the cost of the selected features (we use random real values between 5.0 and 15.0).

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

3.3 Hypervolume Indicator

We evaluate the quality of our solutions cost using the hypervolume metric [17]. Intuition behind the hypervolume is that it gives the volume (in the k dimensions of the search space) dominated by a set of non-dominated solutions. Hypervolume is the region between the solutions and the reference point (the higher the better). The reference point is set with the worst value for each of the objectives.

3.4 System and Algorithms Set-Up

We use the source code provided by SATIBEA's authors and make MILPIBEA publicly available at https://github.com/takfarinassaber/MILPIBEA. The tests are performed on a machine with 62 GB of RAM and 12 core Intel(R) Xeon(R) 2.20 GHz CPU. We use the following parameters for our genetic algorithm:

- Population size: 300 individuals.
- Offspring population size: 300 individuals.
- Crossover rate: 0.8. Represents the probability of two individuals in the population to perform a crossover (an exchange of their selected features).
- Mutation rate: 0.001. 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. Represents the probability of using the solver to correct a solution during the mutation process.

We also use one heuristic in our algorithm: we do not do any bit flip for mandatory or dead features as this always lead to invalid products. We use the engine of the MILP solver *IBM ILOG CPLEX*. We use the hypervolume metric proposed by Fonseca et al. [17]. We ran all our algorithm instances for 20 min and determined the average over 10 runs (for each randomly generated instance).

4 Using Seeds in Evolving FM

In this section, we explore how to use seeds (including previously found solutions in the initial population of a new evolution) to take advantage of the fact that the feature model evolved.

When a FM evolves, the modifications of features and constraints depend on how different the two models are (new and original models). 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-objective optimisation algorithm (SATIBEA in our case) on F_1. Instead of leaving SATIBEA with an initial random population for F_2 (similar to what is proposed in [7]), we adapt S_1 to F_2. Therefore, for each individual, we remove bits representing removed features and add bits with random values for each new feature. Then, we compute their objective functions. We give the new resulting individuals as an initial population to SATIBEA that will run normally on F_2. Our hope is that initial individuals will be better than random solutions.

We tested this approach on all the modified versions of the Linux Kernel and all of the results are equivalent: as expected, when supplied with an initial seed SATIBEA converges within a short time (i.e., less than 100 s) whereas the classical SATIBEA needs 700 s to reach the same hypervolume. This approach also has some limits: with a modified version of 20% features and 10% constraints, classical SATIBEA reaches a slightly better hypervolume than the one with seed.

When we give an initial population to SATIBEA, it converges very fast. Still, it is also blocked very fast, i.e., after 100 s on all models, it stagnates and is unable to improve results further. That is why we decided to investigate a better substitution for SATIBEA, starting from its repair technique based on a SAT solver. We describe this approach in the next section.

5 Correcting Individuals

In this section, we present two ways of correcting non-feasible individuals, i.e., a situation that happens very often during the execution of the genetic algorithm for our problem. Indeed, both mutation and crossover, the basic operations of (SAT)IBEA, generate quite a large ratio of infeasible individuals, given the size of the search space and the number of constraints that can be violated.

The first solution we present is the one proposed in the definition of SATIBEA [6]. The second one is our own improved solution using the MILP solver. Eventually, we propose an evaluation of the two techniques with an example.

5.1 How SATIBEA Corrects Solutions

SATIBEA's correction method occurs in the mutation phase of the genetic algorithm. IBEA takes an individual that violates one or several constraints out of the population and corrects it, using a SAT solver. This leads to the individual being now valid (no longer violating constraints). Figure 3 shows an example of SATIBEA's repair technique on a FM with 5 features (f_1 to f_5) and 3 constraints (c_1 to c_3). The constraints are shown on the left-hand side of Fig. 3, with c_2 marked as violated.

(1) First, an individual with assignment {*1 1 1 0 0*} is selected for repair due to the violation of constraints c_2 (which causes the individual to be invalid). This is shown in row (1) in the table on the right-hand side of Fig. 3.

(2a) Second, SATIBEA unsets (this is represented by '_' in the example) all the bits that belong to a violated constraint. Here, constraint c_2 is violated, so f_4 and f_5 are unset. This is in row $(2a)$ of the table.

(2b) Third, SATIBEA unsets all the bits that are evaluated as 'false' in every constraint. Each of these can either be a feature without a negation sign in the constraint (i.e., f) that is set to false or a feature with a negation (i.e., $\overline{f}$) that is set to true. All of these are unset. In our example, f_2 is assigned to true and is evaluated at false in the constraint c_1 ($\overline{f}_2$). Therefore, SATIBEA unsets f_2. This is shown in row $(2b)$ of the table.

(3) Eventually, the resulting partial assignment is given to the SAT solver to complete the unset values while satisfying the constraints of the FM. SATIBEA's correction always obtains a valid solution if it exists. In our case, SATIBEA results in a new individual (i.e., {*1 0 1 1 0*}). This is shown on line 3 of Fig. 3. Note that this procedure cannot guarantee to always return a valid individual as the problem may be unsatisfiable.

c_1 $(f_1 \vee \bar{f}_2 \vee f_3) \wedge$
c_2 (Violated) $(f_4 \vee f_5) \wedge$
c_3 $(\bar{f}_2 \vee f_3 \vee \bar{f}_5)$

	f_1	f_2	f_3	f_4	f_5
1)	1	1	1	0	0
2a)	1	1	1	_	_
2b)	1	_	1	_	_
3)	1	0	1	1	0

Fig. 3. Correction of an individual in SATIBEA. The original individual, violating constraint 2, is shown on line 1 and the different steps of SATIBEA's correction are shown on lines $2a$, $2b$ and 3.

Although this correction technique is fast and improves the classical IBEA algorithm, the number of flipped bits is large. This often creates new individuals that are far from the original ones (before the correction). This issue is that those individuals were obtained by mutation in IBEA and modifying them too much is against the idea behind genetic algorithms (i.e., inheriting and preserving good characters). For instance, from the individual {*1 1 1 0 0*} (line 1 of Fig. 3) that violates the constraints, it would be better to obtain individual {*1 1 1 1 0*} that does not violate the constraints (instead of {*1 0 1 1 0*}). The next subsection describes our MILP-based correction technique that overcomes this problem.

5.2 How Our MILP Technique Corrects Solutions

Our new method corrects individuals and avoids the problem described in previous section (i.e., a large number of flipped bits between the initial individuals and the corrected ones). This method corrects the faulty individuals and minimises the number of flipped bits which are not part of any violated constraint.

Applied to the example in Fig. 3, only features f_4 and f_5 are unset. CPLEX solves the problem of finding a valid individual by assigning values to f_4 and f_5 while at the same time, minimising the total bit flips on the rest of the features (i.e., f_1,f_2 and f_3). One possible output is {*1 1 1 1 0*} which does not modify any fixed bit, unlike SATIBEA's one (i.e., {*1 0 1 1 0*} which has one modification on the feature f_2).

Using our method, CPLEX is guaranteed to find a valid individual. Moreover, it returns an individual that is as close as possible to the original one. In our method, we use the model defined by Eqs. 1a, 1b, and 1c.

$$\textit{Minimise} \qquad \sum_{x \in T}(1-x) + \sum_{x \in F} x \tag{1a}$$

$$\text{Subject to} \qquad \sum_{x \in P_i} x + \sum_{x \in N_i}(1-x) \geq 1, \quad \forall i \in \{1, .. n\} \tag{1b}$$

$$x \in \{0,1\}, \quad \forall x \in X \tag{1c}$$

With n number of clauses, X set of features, $T \subset X$ set a features fixed at true, $F \subset X$ set of features fixed at false, $P_i \subset X$ set of features without negation in clause i, and $N_i \subset X$ set of features with negation in clause i.

In the MILP model above, we aim to minimise the number of flipped features that were not part of violated constraints in the original individual: if the feature was originally at True (i.e., '1'), then we count it as a modification if and only if it changes to False (i.e., '0'). Similarly, when the feature was originally at False and is changed to True, we also count it as a modification. As in the technique using the SAT model, each clause is represented by a linear constraint. Every feature without a negation is considered as '1' when selected, and every feature that is negated is considered as '1' when unselected. The sum of every feature within a clause has to be larger or equal to 1 to validate it.

Table 1. Comparison of SATIBEA and MILP corrections. Higher values of hypervolume (HV) are better. Lower values of both time and number of modifications (#mod) are better. Best values for each instance in bold.

Instance	No correction	SATIBEA correction			MILP correction		
	HV	HV	Time (ms)	#mod	HV	Time (ms)	#mod
1_1	1.09	1.12	8,895	2,696	**1.18**	**6,801**	**141**
5_1	2.15	2.27	8,474	2,660	**2.35**	**2,337**	**353**
5_3	1.03	1.14	9,005	2,747	**1.25**	**5,675**	**297**
10_1	1.00	1.04	9,273	2,615	**1.16**	**2,192**	**798**
10_3	1.08	1.19	10,255	2,732	**1.33**	**4,044**	**99**
10_5	1.02	1.18	10,339	2,784	**1.33**	**4,082**	**110**
20_1	0.90	0.96	9,762	2,528	**1.03**	**2,143**	**412**
20_3	2.16	2.28	9,699	2,691	**2.47**	**2,891**	**148**
20_5	1.06	1.20	10,222	2,752	**1.36**	**2,962**	**124**
20_10	0.72	0.73	10,877	3,008	**0.75**	**6,719**	**92**

5.3 Comparison with Respect to the Correction Process

In Table 1, we compare our correction method against SATIBEA's correction. Each instance corresponds to an evolved FM and is represented by a couple (x_y) where x is the percentage of features modified and y the percentage of constraints modified. We took the 300 individuals given by SATIBEA on the original FM as seeds for the evolved versions. SATIBEA found 62 solutions that do not violate constraints in the original FM. Obviously, these solutions violate some constraints in each of the evolved FMs. We compared both SATIBEA's and MILP's correction methods applied on the 62 individuals. We measured the hypervolume (HV), the average execution time for each individual (in milliseconds) and the average number of modified features from the original individual

to the corrected one (#mod). We also added the hypervolume of non-corrected solutions (NC) as a baseline.

We can see that when applying a correction, both algorithms improve the hypervolume composed to non-corrected individuals. However, MILP's corrections outperforms SATIBEA's corrections. Correction using a MILP solver improved hypervolume of SATIBEA's correction by 147% on average, while only requiring 48% of its execution time. Moreover, we notice that the number of modified features per individual using the MILP correction is one order of magnitude lower than when using SATIBEA's correction (on average SATIBEA's correction requires 2,721 feature modifications whereas MILP's correction only requires 257). As our correction method needs less modifications of individuals to transform them into valid ones, it could be more interesting to use it instead of SATIBEA's one in a genetic algorithm: indeed, less modifications imply a better conservation of the accumulated knowledge during the generations. An implementation of our genetic algorithm with this type of correction method in the mutation part is described in the next section.

6 Performance of MILPIBEA vs. SATIBEA

We now report on how MILPIBEA and SATIBEA perform on the multi-objective features selection problem – in particular with respect to the achieved hypervolume and the required time for that. We initially discuss the general feature selection problem and then the case of evolved FMs.

6.1 On the Multi-objective Feature Selection Problem

Figure 4 show the evolution using SATIBEA or MILPIBEA in terms of hypervolume when applied on our 10 generated models: each of them is a modification of the 2.6.28 version of Linux kernel represented by a couple (x_y) where x is the percentage of features modified and y the percentage of constraints modified. This hypervolume is measured based only on the individuals of the current population. We are not seeding the initial population: the problem studied in these result is the multi-objective feature selection problem, without the notion of evolution. The initial population is generated randomly for both algorithms.

Our results indicate that MILPIBEA outperforms SATIBEA with an improvement of 41.2% hypervolume on average. Figure 4 also indicates that MILPIBEA is more efficient on the most constrained problems (i.e., with constraint modifications $\geq 5\%$). MILPIBEA reaches a good hypervolume after 100 s, then increases slowly. We can see that SATIBEA's hypervolume increases with a slower pace than MILPIBEA's; then its hypervolume stays stable (within a small interval).

6.2 With Evolved Feature Models

We now compare MILPIBEA and SATIBEA in the case of the multi-objective feature selection problem in evolving FMs. As described in Sect. 2, the notion of

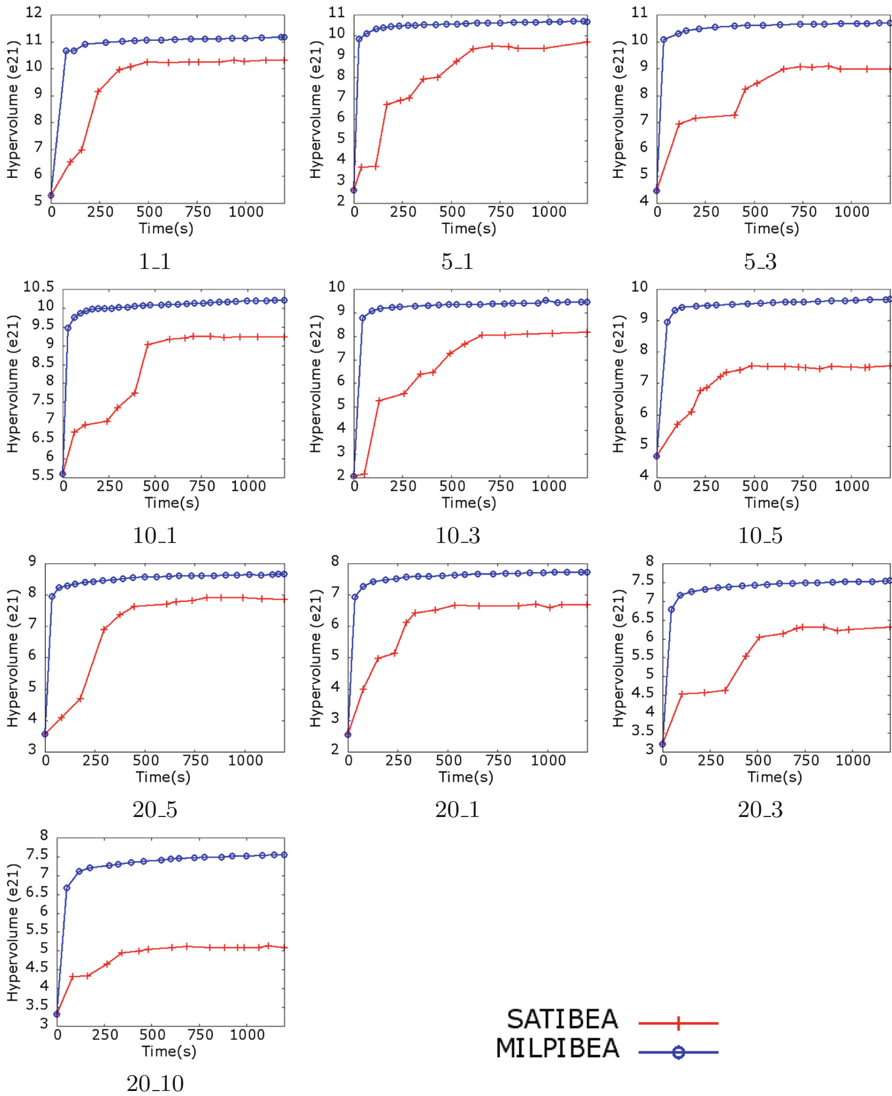

Fig. 4. Comparison of MILPIBEA and SATIBEA on various evolved FMs. The higher the better for the hypervolume.

evolution is represented by features/constraints modifications in the FM. In our case, the Linux kernel 2.6.28 is the original FM, and we generated 10 modified versions. Because of evolution, the original FM has been optimised, and its solutions as given as initial population to SATIBEA and MILPIBEA: the purpose is to improve the quality of results on modified FMs as fast as possible.

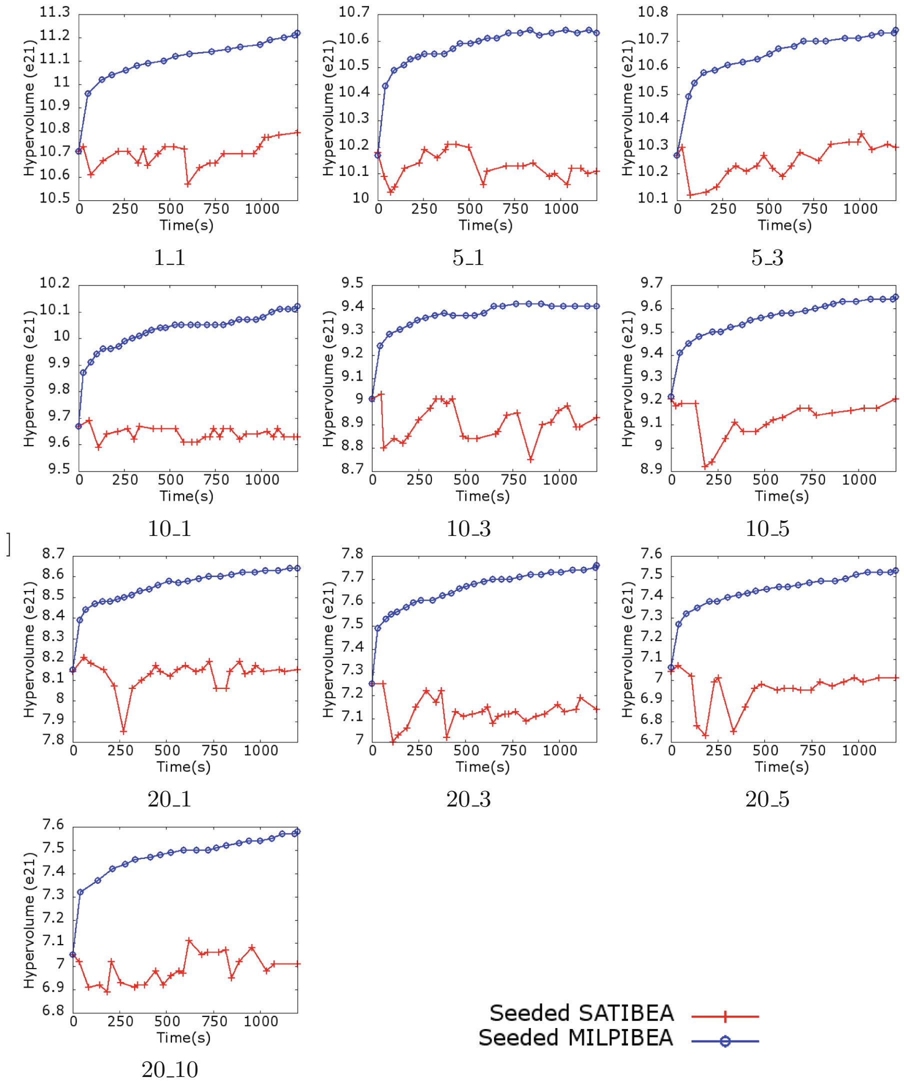

Fig. 5. Comparison of the hypervolume achieved by seeded MILPIBEA and seeded SATIBEA on various evolved FMs (higher values are better).

Figure 5 show the hypervolume of the individuals at every new generation for both SATIBEA and MILPIBEA when given the solutions of the original FM as initial population. We see that both algorithms start from a relatively good hypervolume, which shows the quality of the initial population.

We also see that MILPIBEA successfully improves the hypervolume, whereas SATIBEA struggles when seeded. This is mainly because MILPIBEA has a correction method that allows it to take advantage of the initial population's good

characteristics by not changing a lot of features in individuals that are obtained from the crossover. However, SATIBEA requires to modify several features, making the individuals obtained by the repair almost random.

Moreover, we can observe that unlike SATIBEA, MILPIBEA's hypervolume continues improving slowly even after the limit (i.e., 1200 s). A larger allowed time would lead to better solutions. MILPIBEA stagnates after 40 min beyond which we might consider adding a local search phase [18–21].

When comparing MILPIBEA without seeds and MILPIBEA with seeds: after the first generation, MILPIBEA with seeds is 10.5% better in hypervolume than without seeds. It also reaches 97.28% of MILPIBEA's final hypervolume (computed in 1200 s) after only one generation (42.29 s on average). This shows us that a good initial population improves the time needed to reach good solutions.

7 Conclusion and Future Work

In this paper, we have presented the importance of the evolution in SPL by introducing the multi-objective features selection in evolving SPL problem. To solve this problem, we proposed a method based on a combination of a genetic algorithm (IBEA) with a MILP solver (i.e., CPLEX). We observed that this method not only outperforms SATIBEA on the multi-objective features selection but also achieves faster better results in the context of evolving SPL. Our thorough evaluation shows the importance of using a MILP solver to reduce the number of modifications when correcting an individual.

Our future work will investigate the performance with respect to other multi-objective performance metrics and the utility of a local search when the genetic algorithm stagnates.

Acknowledgment. This work was supported by Science Foundation Ireland grant 13/RC/2094.

References

1. Ramirez, A., Romero, J.R., Ventura, S.: A survey of many-objective optimisation in search-based software engineering. J. Syst. Softw. **149**, 382–395 (2019)
2. Metzger, A., Pohl, K.: Software product line engineering and variability management: achievements and challenges. In: FSE, pp. 70–84 (2014)
3. Neto, J.C., da Silva, C.H., Colanzi, T.E., Amaral, A.M.M.M.: Are mas profitable to search-based PLA design? IET Softw. **13**(6), 587–599 (2019)
4. Nair, V., et al.: Data-driven search-based software engineering. In: MSR, pp. 341–352 (2018)
5. Harman, M., Jia, Y., Krinke, J., Langdon, W.B., Petke, J., Zhang, Y.: Search based software engineering for software product line engineering: a survey and directions for future work. In: SPLC, pp. 5–18 (2014)
6. Henard, C., Papadakis, M., Harman, M., Le Traon, Y.: Combining multi-objective search and constraint solving for configuring large software product lines. In: ICSE, pp. 517–528 (2015)

7. Saber, T., Brevet, D., Botterweck, G., Ventresque, A.: Is seeding a good strategy in multi-objective feature selection when feature models evolve? Inf. Softw. Technol. **95**, 266–280 (2018)
8. Guo, J., et al.: Smtibea: a hybrid multi-objective optimization algorithm for configuring large constrained software product lines. Softw. Syst. Model. **18**(2), 1447–1466 (2019)
9. Yu, H., Shi, K., Guo, J., Fan, G., Yang, X., Chen, L.: Combining constraint solving with different MOEAs for configuring large software product lines: a case study. In: COMPSAC, vol. 1, pp. 54–63 (2018)
10. Saber, T., Marques-Silva, J., Thorburn, J., Ventresque, A.: Exact and hybrid solutions for the multi-objective VM reassignment problem. IJAIT **26**(01), 1760004 (2017)
11. Saber, T., Ventresque, A., Marques-Silva, J., Thorburn, J., Murphy, L.: Milp for the multi-objective VM reassignment problem. ICTA **I**, 41–48 (2015)
12. Saber, T., Gandibleux, X., O'Neill, M., Murphy, L., Ventresque, A.: A comparative study of multi-objective machine reassignment algorithms for data centres. J. Heuristics **26**(1), 119–150 (2019). https://doi.org/10.1007/s10732-019-09427-8
13. Pleuss, A., Botterweck, G., Dhungana, D., Polzer, A., Kowalewski, S.: Model-driven support for product line evolution on feature level. J. Syst. Softw. **85**(10), 2261–2274 (2012)
14. Sayyad, A.S., Menzies, T., Ammar, H.: On the value of user preferences in search-based software engineering: a case study in software product lines. In: ICSE, pp. 492–501 (2013)
15. Xue, Y., Li, Y.F.: Multi-objective integer programming approaches for solving optimal feature selection problem: a new perspective on multi-objective optimization problems in SBSE. In: ICSE, pp. 1231–1242 (2018)
16. Brevet, D., Saber, T., Botterweck, G., Ventresque, A.: Preliminary study of multi-objective features selection for evolving software product lines. In: Sarro, F., Deb, K. (eds.) SSBSE 2016. LNCS, vol. 9962, pp. 274–280. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47106-8_23
17. Fonseca, C.M., Paquete, L., López-Ibánez, M.: An improved dimension-sweep algorithm for the hypervolume indicator. In: CEC, pp. 1157–1163 (2006)
18. Shi, K., et al.: Mutation with local searching and elite inheritance mechanism in multi-objective optimization algorithm: a case study in software product line. Int. J. Softw. Eng. Knowl. Eng. **29**(09), 1347–1378 (2019)
19. Saber, T., Delavernhe, F., Papadakis, M., O'Neill, M., Ventresque, A.: A hybrid algorithm for multi-objective test case selection. In: CEC, pp. 1–8 (2018)
20. Saber, T., Ventresque, A., Brandic, I., Thorburn, J., Murphy, L.: Towards a multi-objective VM reassignment for large decentralised data centres. In: UCC, pp. 65–74 (2015)
21. Saber, T., Ventresque, A., Gandibleux, X., Murphy, L.: GenNePi: a multi-objective machine reassignment algorithm for data centres. In: HM, pp. 115–129 (2014)