The Journal of Systems and Software 103 (2015) 392–411

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Applying multiobjective evolutionary algorithms to dynamic software product lines for reconfiguring mobile applications

Gustavo G. Pascual[a,*], Roberto E. Lopez-Herrejon[b], Mónica Pinto[a], Lidia Fuentes[a], Alexander Egyed[b]

[a] Department of Languages and Computer Science, University of Málaga, 29071 Málaga, Spain
[b] Institute for Systems Engineering and Automation, Johannes Kepler University Linz, Austria

ARTICLE INFO

Article history:
Received 1 December 2013
Revised 11 December 2014
Accepted 17 December 2014
Available online 3 January 2015

Keywords:
DSPL
Dynamic reconfiguration
Evolutionary algorithms

ABSTRACT

Mobile applications require dynamic reconfiguration services (DRS) to self-adapt their behavior to the context changes (e.g., scarcity of resources). Dynamic Software Product Lines (DSPL) are a well-accepted approach to manage runtime variability, by means of late binding the variation points at runtime. During the system's execution, the DRS deploys different configurations to satisfy the changing requirements according to a multiobjective criterion (e.g., insufficient battery level, requested quality of service). Search-based software engineering and, in particular, multiobjective evolutionary algorithms (MOEAs), can generate valid configurations of a DSPL at runtime. Several approaches use MOEAs to generate optimum configurations of a Software Product Line, but none of them consider DSPLs for mobile devices. In this paper, we explore the use of MOEAs to generate at runtime optimum configurations of the DSPL according to different criteria. The optimization problem is formalized in terms of a Feature Model (FM), a variability model. We evaluate six existing MOEAs by applying them to 12 different FMs, optimizing three different objectives (usability, battery consumption and memory footprint). The results are discussed according to the particular requirements of a DRS for mobile applications, showing that PAES and NSGA-II are the most suitable algorithms for mobile environments.

1. Introduction

Mobile applications demand runtime reconfiguration services that make it possible for them to self-adapt their behavior to the continual contextual changes that occur in their environment (e.g., scarcity of available resources) (Brataas et al., 2011; Capilla et al., 2014; Floch et al., 2013; Mizouni et al., 2014). In some applications, the reconfiguration can be made to maintain a certain quality of service (QoS) the user requires, in others the reconfiguration can be made to offer a personalized service to the user such as location-based services, and even to provide the user's personal suggestions based on the recognized activity and context (Mizouni et al., 2014). For instance, the battery level of the mobile device may be a critical decision parameter in changing the behavior of an application if the goal of this reconfiguration is to extend the lifespan of the battery and hence the device connectivity (Mizouni et al., 2014).

One accepted approach to manage the runtime variability of applications is the Dynamic Software Product Line (DSPL) approach. DSPLs produce software capable of adapting to changes, by means of binding the variation points at runtime (Hallsteinsen et al., 2008). This requires to model the elements that could be adapted dynamically as dynamic variation points and to generate, at runtime, the different variants of the DSPL.

A *runtime configuration* is the set of values assigned to the dynamic variation points, defining a member of the dynamic SPL. If a change in the execution context is detected, then a reconfiguration service should generate a new runtime configuration adapted to the new context. Therefore, the reconfiguration service should be continuously generating optimum *runtime configurations* adapted to the changing context. But, which and how many objectives should be considered in the generation of optimum configurations? In the case of mobile applications, multiple objectives should be taken into account like, loss of network connectivity, drastic increase or reduction of the available resources (e.g., battery, memory, CPU) or the user preferences about quality of service (QoS). For instance, if a user wants to save battery life in a mobile phone application, the reconfiguration service should generate a configuration with a low battery consumption while trying to keep the quality of service as high as possible.

Harman et al. (2014) show how Search-Based Software Engineering (SBSE) has been successfully applied by different approaches to SPLs. In this paper, we demonstrate that SBSE, and in particular

* Corresponding author. Tel.: +34665372568.
E-mail addresses: gustavo@lcc.uma.es (G.G. Pascual), roberto.lopez@jku.at (R.E. Lopez-Herrejon), pinto@lcc.uma.es (M. Pinto), lff@lcc.uma.es (L. Fuentes), alexander.egyed@jku.at (A. Egyed).

http://dx.doi.org/10.1016/j.jss.2014.12.041
0164-1212/© 2015 Elsevier Inc. All rights reserved.

multiobjective evolutionary algorithms (MOEAs), can be used to solve the problem of generating a valid configuration of a DSPL at runtime. Runtime configurations of a DSPL are generated from a variability model, which specifies the common and the variable elements of the dynamic product line. Most of DSPL approaches use Feature Models (FMs) as the de facto standard to specify the commonalities and variabilities of the product line in terms of *features* and *constraints* between them (Cetina et al., 2008; Dinkelaker et al., 2010; Rosenmüller et al., 2011; Trinidad et al., 2007; White et al., 2007). In this case, runtime configurations are defined in terms of features, and are known as dynamic feature model configurations. This means that the set of valid configurations that can be deployed during the execution of the application is then determined by the FM. Therefore, the MOEA needs the FM that models the dynamic variation points. With an FM available at runtime, an MOEA can generate valid variants of the application adapted to the context changes.

In this paper, we explore the use of MOEAs to generate at runtime the variants of the DSPL that fit the current execution context with regard to several optimization criteria such as battery consumption or usability. However, in order to be suitable for our reconfiguration service, the employed algorithms should satisfy several requirements such as:

1. *Fast enough execution time*. Reconfiguring the application should not harm excessively the user experience. Furthermore, since we are focusing on mobile devices, the response time of the optimization algorithm should be of few seconds.
2. *Generate only valid configurations*. While in different kinds of approaches (Sayyad et al., 2013) configurations which do not satisfy all the constraints can be useful, it is not appropriate for a reconfiguration service to deploy invalid configurations of the application. Therefore, the optimization algorithm should only return valid configurations (i.e., configurations which satisfy all the constraints).
3. *Multiobjective optimization*. Generally, it is necessary to generate configurations which are optimal regarding several criteria (e.g., battery consumption, usability).
4. *Support for DSPLs in mobile applications*. In DSPLs the number of variation points that need to be managed is usually much lower than in SPLs at design time. The reason is that in DSPLs only the variations points that can change at runtime need to be considered. Therefore, unlike design time SPLs, in DSPLs only a subset of the FM variation points are considered; the rest of them are fixed at design time. For instance, the variability model of the Linux kernel (Lotufo et al., 2010) contains more than 6000 features, but although some of these variation points can be decided at runtime (e.g., I/O scheduler, CPU frequency governor), many points are decided at design time because they depend on the hardware of the target device (e.g., CPU architecture, CPU model, virtualization support, cryptography hardware). Moreover, since our approach focuses on the development of mobile applications, the DSPLs managed in our approach would generally be even smaller than DSPLs for desktop applications.

In this sense, several algorithms have been defined which are able to obtain an optimal configuration of an FM according to a given optimization criteria (Benavides et al., 2010; Guo et al., 2011; Li et al., 2012; Sayyad et al., 2013; Soltani et al., 2012; White et al., 2009a, 2009b). However, none of them are suitable for reconfiguration in mobile devices mainly because they were proposed to optimize the configuration of Software Product Lines (SPLs), being designed to be used only at design time. Furthermore, only Sayyad et al. (2013) support multiobjective optimization (i.e., generating configuration which are optimal regarding different criteria simultaneously), but not for DSPLs.

Our experiments have been performed using six existing MOEAs algorithms, which have been applied to 12 FMs in order to optimize three objectives (usability, battery and memory). We have evaluated these algorithms according to the requirements imposed by a reconfiguration service for mobile applications and the results show that PAES and NSGA-II are the most suitable MOEAs to be used in mobile environments. They have the lowest execution time and, at the same time, they satisfy the rest of requirements.

Following the Introduction, the rest of the paper is organized as follows. The backgrounds to DSPLs and FMs are presented in Section 2. After this, the related work is discussed in Section 3 and the realization of our reconfiguration mechanism is described in Section 4. Then, the experimental setup and the evaluations results are presented in Sections 5 and 6 respectively. Finally, in Section 7 the evaluation results and threats to validity are discussed, while our conclusions and future work are described in Section 8.

2. Background

In this section we show the basics of DSPLs and FMs, which are used in our approach to reconfigure mobile applications at runtime.

2.1. Dynamic software product lines

An SPL is “a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.”[1] DSPLs redefine existing SPL engineering processes by moving them to runtime, with the goal of ensuring that system adaptations lead the system to a valid state. So, in SPLs the engineering processes are able to generate several systems of the same family at design time, but a DSPL is considered a single system able to adapt its behavior at runtime.

The variability model is the central artifact for both SPLs and DSPLs for formally specifying their commonalities and variabilities. The engineering processes of SPLs generate products by selecting concrete values for the variable characteristics specified in the variability model. This means that the SPL engineer binds the variation points at design time considering the requirements of the intended product. In contrast, in DSPLs the variability model describes the potential range of variations that can be produced at runtime for a single product, i.e., *the dynamic variation points* already defined in the introduction. Then, as the set of dynamic variation points drive system adaptation, they must be available to be consulted at runtime by a reconfiguration service. But these dynamic variation points must make reference to the system architectural components. So, in DSPLs the system architecture supports all the possible adaptations defined by the set of dynamic variation points (Hallsteinsen et al., 2008).

So, as part of a DSPL definition the engineer must identify: (i) the range of potential adaptations supported by the system in terms of architectural components; (ii) define an explicit representation of the valid configuration space of the system; (iii) the context changes that may trigger an adaptation, i.e., the criteria (which can have several objectives) to initiate a reconfiguration or Decision Making Process (DMP); and (iv) the set of possible reactions to context changes that should be supported by the system. However, the way these issues are implemented may differ greatly, as will be shown in Section 3.1.

Since for the majority of DSPLs the decision to initiate a reconfiguration is made autonomously by the system (not by a human), they are considered a good technology for developing self-adapting systems such as mobile applications. In this sense, most of DSPL approaches share some common capabilities and goals with the Autonomic Computing (AC) paradigm (IBM, 2005) such as the monitoring of the environment and the generation of successive configurations.

[1] http://www.sei.cmu.edu/productlines/

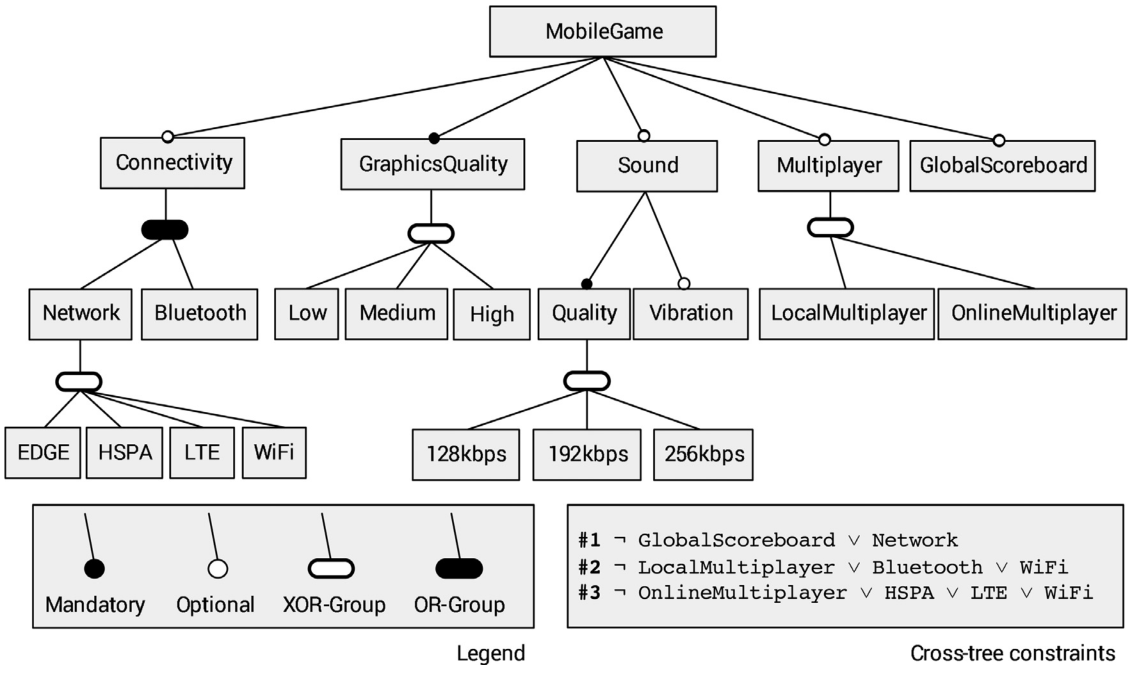


Fig. 1. Feature model example.

2.2. Feature models

As we already stated in the Section 2.1, the variability model is the central artifact of both SPLs and DSPLs. Feature models are widely used for modeling variability in SPLs. Although they are typically used in the requirements specification phase, they can be successfully applied to manage variability in other phases of the software development life cycle (Acher et al., 2011; Morin et al., 2009). FMs are organised in a hierarchical structure (Fig. 1), where each feature is decomposed into children features, which can be connected to their parent individually using optional/mandatory connectors (if the child feature is optional/mandatory) or in groups (an OR group if one or more child features can be selected or an XOR group if only exactly one child feature can be selected). Selecting a feature means that its parent should be selected too.

Fig. 1 shows the FM of a game for mobile devices.The root feature, `MobileGame`, is decomposed in the `Sound`, `Connectivity`, `GraphicsQuality`, `GlobalScoreboard` and `Multiplayer` features. While the `GraphicsQuality` feature is mandatory and thus has to be included in all the generated configurations, the rest of them are optional—i.e., they are variation points. The `Network` and `Bluetooth` features are part of an OR group, meaning that one or both of them can be selected simultaneously, while the `LocalMultiplayer` and the `OnlineMultiplayer` features are in an XOR group, and thus only exactly one of them can be part of a particular configuration.

In addition to the relationships between features shown in the tree (i.e., the tree constraints), it is also possible to specify Cross-Tree Constraints (CTCs) between features. In some cases, these CTCs are specified as `A requires B` or `A excludes B` statements. The first one states that, in the case that feature `A` is selected in a particular configuration of the FM, feature `B` should also be included. The second one states that the features `A` and `B` are mutually exclusive and, therefore, they cannot be selected simultaneously in the same FM configuration.

CTCs can also be defined in Conjunctive-Normal-Form (CNF) notation, which allows to define more complex constraints. In CNF, CTCs are expressed as a conjunction of clauses, where a clause is a disjunction of positive and negative literals (features); otherwise expressed, the set of CTCs is specified as a logical *AND* of *OR*s. For instance, the CTC `#3` in Fig. 1 states that, in the case that the `OnlineMultiplayer` feature is selected, it is necessary to select the `HSPA`, `LTE` or `WiFi` features:

$$\text{OnlineMultiplayer} \implies \text{HSPA} \vee \text{LTE} \vee \text{WiFi} \quad (1)$$

Thanks to the extensive use of FMs, it is possible to take advantage of their wide support (Acher et al., 2010; Benavides et al., 2010; Matinlassi, 2004; White et al., 2009a) and the existing tools (e.g., FAMA,[2] Hydra,[3] SPLOT,[4] FeatureIDE; Kastner et al., 2009). Moreover, FMs are specified using formal languages, as for instance CSP (Constraint Satisfaction Problems) (Tsang, 1993). This means that the visual representations of FMs are only for the purpose of facilitating the writing and understanding of the FMs, but then the existing tools automatically map this graphical representation into a CSP specification. This allows reasoning about variability, as well as other capacities of FMs such as the generation of valid product configurations, the quantification of the number of possible configurations, etc. (Benavides et al., 2010).

3. Related work

In this section we discuss the existing DSPL approaches which are closest to our work, identifying and commenting the differences between them. Moreover, we also present those approaches that use evolutionary algorithms in the domain of SPLs or DSPLs.

3.1. Overview of DSPL approaches

As stated in Section 2.1, there are important differences among the DSPL approaches that are available in the literature. The most relevant differences can be categorized as follows:

1. The moment when the valid configuration space is generated.
2. The decision making process used to trigger a reconfiguration.
3. The optimization criteria used to generate the successive configurations, if any is used.

[2] FaMa Tool Suite, http://www.isa.us.es/fama/
[3] Hydra project website, http://caosd.lcc.uma.es/spl/hydra/
[4] S. P. L. O. T.: Software product lines online tools, http://www.splot-research.org/

Table 1
DSPL approaches.

Approach	Generation of configuration	Decision making	Optimization mechanism	Mobile devices
Dinkelaker et al. (2010)	Design time	Design time plans	None	No
Rouvoy et al. (2009)	Design time	Utility function	Brute force/heuristics	Yes
Shen et al. (2011)	Design time	ECA rules	None	No
Rosenmüller et al. (2011)	Partially runtime	Manual/ECA rules	None	No
White et al. (2007)	Partially runtime	Cost function	CSP solver	Yes
Cetina et al. (2008)	Runtime	ECA rules	None	No
Trinidad et al. (2007)	Runtime	Manual	None	No
Our approach	**Runtime**	**Multiobjective function**	**MOEA**	**Yes**

4. The suitability of the approach for developing applications for mobile devices.

In the rest of this section we discuss how these issues are addressed by the compared DSPL approaches. The results of the study are summarized in Table 1, where there is a column for each discussed issue.

3.1.1. Generation of the valid configuration space

As can be seen in Table 1, some DSPL approaches generate at design time the configurations that are then deployed at runtime (Dinkelaker et al., 2010; Rouvoy et al., 2009; Shen et al., 2011). However, since the potential number of configurations can grow exponentially, some of these approaches consider at runtime only a subset of the valid configurations, which are pre-loaded into the system. This is an important drawback, especially in our case where one of the goals is to generate optimal configurations. It is very difficult to ensure at design time that the list of pre-loaded configurations includes the optimal ones according to the defined objectives and possible context changes.

There are other DSPLs that generate the configurations partially at runtime (Rosenmüller et al., 2011; White et al., 2007) or completely at runtime (Cetina et al., 2008; Trinidad et al., 2007). On the one hand, White et al. (2007) specify, using a domain specific modeling language, the components of the product line architecture for mobile devices, their dependencies and composition rules and the non-functional requirements of each component. These models are then converted into a CSP, allowing the generation of configurations at runtime using a CSP solver. This cannot be considered a completely runtime approach because even though the CSP solver is used to generate at runtime a customized version of the application based on the available resources on the mobile device, this approach does not provide a mechanism to adapt the application installed in the mobile device at runtime. Instead, when changes in the application configuration are needed, the application needs to be stopped and initiated again with the new configuration.

Also the approach of Rosenmüller et al. (2011) is partially at runtime approach. Firstly, part of the variability of the SPL is reduced at design time, generating several DSPLs which are subsets of the complete SPL. Then, an adaptation mechanism is included in each DSPL that is capable of generating different configurations of that DSPL at runtime.

Regarding the runtime approaches, Cetina et al. (2008) propose a model driven approach for modeling pervasive systems, where the variability information is introduced by means of model transformations, enabling the runtime reconfiguration according to the execution context. Then, using a model reasoner they can generate valid configurations at runtime, although only a proof of concept has been proposed for this model reasoner. In the proposal of Trinidad et al. (2007), each feature in the FM is mapped to a component in the software architecture than can be activated or deactivated. Then, using a CSP solver, they perform real-time FM analysis and therefore generate valid configurations at runtime.

Our approach can generate at runtime any of the configurations that are part of the valid FM configuration space. Concretely, the FM configurations are generated on demand using an MOEA, which is executed on the mobile device. The software architecture reconfiguration plan is then calculated as the difference between the current and the new FM configuration generated by the optimization algorithm.

3.1.2. Decision making process

Regarding the decision making process, some DSPL approaches are based on the definition of a set of event-condition-action (ECA) rules (Cetina et al., 2008; Shen et al., 2011). An ECA rule includes the event that triggers a reconfiguration, a condition about the system state that must be evaluated as true, and the reconfiguration plan or actions that should be executed. The main problem with this approach is that the number of rules could become intractable, especially if the number of potential configurations is high. Moreover, during the specification of the ECA rules it is very difficult to identify and list all the possible events and conditions that may later occur in the system at runtime.

Goal-based approaches, such as White et al. (2007) and Rouvoy et al. (2009) and our approach, overcome this problem since they do not need to enumerate all the "context change–product configuration" pairs at design time. Instead, these approaches use a 'function' to calculate the cost or utility of each generated configuration. The main restriction of these approaches, which is crucial for an approach that has to be used at runtime, may be that this is done at the cost of more runtime overhead. In the work of White et al. (2007), a *cost function*, which measures the cost regarding a given criteria (e.g., mobile data consumption) is optimized according to the available resources. Then, at runtime, the variant with the lowest cost and that does not exceed the available resources is generated.

In the MUSIC middleware of Rouvoy et al. (2009), an *utility function*, which typically refers to the user's overall satisfaction, is defined to decide which is the most appropriate configuration from among the set of valid ones.

In the work of Trinidad et al. (2007), a CSP solver is used to reason about the configuration proposed by the user, being able to determine whether it is valid or not, among other operations. However, an automated decision making mechanism is not provided, being the user responsible for manually proposing new configurations.

In our approach, we use a *multiobjective function* that specifies multiple objectives to be optimized, such as the battery consumption or usability. This is done by specifying the "contribution" to these objectives of each feature in the FM.

3.1.3. Optimization mechanism

Those approaches including a goal-based decision making process (Rouvoy et al., 2009; White et al., 2007, and our approach) need a mechanism to find which is the best configuration according to the goal(s) at runtime. Firstly, in the MUSIC middleware (Rouvoy et al., 2009), it is possible to decide, at design time, which optimization algorithm is applied to select the most appropriate configuration for the current execution context. Concretely, it is possible to choose between brute force, in which all the valid configurations are evaluated, and some heuristics which reduce the solution space. However, both options have limitations. On the one hand, the use of the brute force requires the generation of all the valid configurations, which is a very time consuming task. On the other hand, the use of heuristics may leave out of the solution space the best configurations.

Secondly, in White et al. (2007), a variant engine selection, which uses a CSP solver to generate optimal configurations, is executed on a server computer. Then, when the application is being deployed in a particular mobile device, the resources available in the device, such as WiFi capability, CPU, RAM memory or resolution, are sent to the variant selection engine. The CSP solver discards those configurations which are not suitable for that device in particular, generating then the one which optimizes a *cost function*. Finally, the generated configuration is deployed in the mobile device. As seen, the use of the CSP solver to generate an exact solution is too costly as to be run in the mobile and the requirement of using a server computer reduces the usability of the approach.

Finally, in our approach, as stated before, we use an MOEA, which efficiently generates valid configurations that are optimal regarding several criteria. Therefore, our approach does not depend on a network connection to an external server to offload the execution of the optimization algorithm. Instead, our goal is to choose an algorithm efficient enough as to be directly executed on the mobile devices.

The use of a multiobjective optimization mechanism to automate the generation of variants of an application at runtime is not specific to DSPL approaches. In other domains, such as Service-Oriented Architectures (SOAs), multiobjective optimization heuristics are also used for the efficient selection and composition of services based on QoS and other system attributes (Canfora et al., 2008; Mirandola et al., 2014; Wada et al., 2012; Yao and Chen, 2009).

3.1.4. Suitability for mobile devices

As can be seen in Table 1, most of the DSPL approaches are not suitable for mobile devices. This is the case of the work of Dinkelaker et al. (2010), as well as the work of Cetina et al. (2008), which are based on a *models@runtime* approach, the main drawback of which is that the management and transformations of models at runtime is too costly to use in a mobile device.

The proposal of Shen et al. (2011) is based on *JBoss AOP* (Hat, 2010), which is not available in mobile devices. The reason for this limitation is that JBoss AOP relies on *cglib*, a Java library for runtime bytecode generation which is not available in Java virtual machines for mobile devices such as Dalvik, the virtual machine used in the Android operating system. However, although the use of JBoss AOP in mobile devices is not feasible, other AOP languages, such as AspectJ (GoPivotal, 2014), can be used in these devices.

In the approach presented by Trinidad et al. (2007), a CSP solver is used to reason about the variability of the DSPL and, therefore, it is not appropriate for mobile devices. This is also the case of the work of Rosenmüller et al. (2011), which uses a SAT solver to reconfigure the applications at runtime. The use of a CSP or a SAT solver has the advantage of generating exact solutions to the optimization problem but at the cost of requiring more computational time, which is an important limitation in the case of mobile applications.

One approach that is suitable for mobile devices is the MUSIC middleware (Rouvoy et al., 2009). This middleware runs on top of the OSGI platform, which can be executed on mobile devices. Furthermore, a mobile version of the MUSIC middleware for Android devices is available, although it is no longer maintained.

Finally, the approach of White et al. (2007) can generate variants at runtime for mobile devices. However, the variant selection engine needs to be executed in a server and the execution time of the CSP solver is very high (more than 35 s). Therefore, this approach is appropriate for the initial deployment of an application's variant which meets the requirements of a mobile device in particular, but not for optimizing the application according to the execution context of the mobile device.

Table 2
Optimization algorithms for FMs.

Approach	Objectives	Mobile	CTCs
Li et al. (2012)	Single	N/A	Limited
Benavides et al. (2005)	Single	No	No
Djebbi et al. (2007)	Single	No	Limited
Soltani et al. (2012)	Single	No	Yes
White et al. (2009a)	Single	Yes	Yes
Shi et al. (2010)	Single	Yes	Limited
Guo et al. (2011)	Single	Yes	Limited
Sayyad et al. (2013)	Multiple	No	Yes
Our approach	**Multiple**	**Yes**	**Yes**

In our approach, since we specifically focus on mobile devices, we must use an evolutionary algorithm that is efficient enough as to be actually executed on mobile devices. In order to achieve this goal, the execution time is one of the criteria we have used to select the different evolutionary algorithms considered in this paper, as explained in Section 6.

3.2. Optimization algorithms for FMs

Those DSPL approaches which model the variability using FMs need algorithms to generate configurations from FMs according to a given criterion. These algorithms mainly differ in:

1. *Efficiency for mobile device execution*. In order to be suitable for reconfiguring mobile applications at runtime, the optimization algorithm should be very efficient regarding its execution time. Furthermore, its implementation should be executable on a mobile device.
2. *Number of objectives*. We distinguish whether the algorithm can optimize one single objective or multiple objectives simultaneously.
3. *Cross-Tree Constraints support*. We evaluate the support provided by the algorithm to specify CTCs. We distinguish the cases in which no support is provided, a limited support is provided (only for `A requires B` or `A excludes B` constraints), and all CTCs are supported.

Table 2 summarizes those approaches from Section 2.1 that use optimization algorithms, as well as other algorithms available in the literature for generating configurations of FMs.

Li et al. (2012) present an algorithm for transforming the problem of selecting a configuration of an FM into a 0-1 Programming problem, which can be solved using different algorithms. Although it does not support optimization based on an utility function, it can find configurations which do not exceed a certain amount of resources. With regards to CTCs, only basic CTCs are supported. Moreover, since an evaluation of their approach is not provided, it is not possible to assess whether it is suitable for mobile devices or not.

In the work of Benavides et al. (2005), FMs are specified as a CSP problem. Then, a solver is used to analyze the variability of the FM, as well as to find optimal configurations regarding a given criteria. However, we can identify two important drawbacks to this approach: (1) it does not support CTCs, and (2) it optimizes a problem in which the complexity increases exponentially with respect to the number of features, using a CSP solver. Therefore, it is not suitable for runtime reconfiguration because the CSP solver generates an exact solution.

The work of Djebbi et al. (2007) provides support for finding optimal configurations of an FM regarding a given criteria, such as the cost of implementation, allowing in addition to discard those configurations which do not satisfy a given set of requirements. However, only basic CTCs are supported and the solver is implemented in GNU-Prolog (Diaz and Codognet, 2001), which is not available for mobile devices.

Soltani et al. (2012) propose a method for finding configurations of FMs taking into account functional and non-functional requirements. To this end, Hierarchy Task Network Planning is used (Sacerdoti, 1975). Although it provides full support for CTCs, it is not suitable for mobile devices since the reconfiguration time is very high, even when it is executed on a desktop computer.

In the work of White et al. (2009a) Filtered Cartesian Flattening (White et al., 2008) is applied to find configurations of an FM which are optimal regarding a single objective. It is mentioned in their paper that CTCs are supported, but it remains unclear which kind of CTCs are supported as the case study provided does not contain any CTC. Their evaluation results show that their approach provides nearly-optimal configurations and the execution time of the algorithm is very low. Even having a good execution time, as just said the algorithm is not multiobjective. The work of Shi et al. (2010) is a slight variation of the approach presented by White et al. (2009a). Although it is stated that their proposal is faster, the conclusions appear to be contradictory and an evaluation comparing both algorithms is not provided.

Guo et al. (2011) specify a genetic algorithm to find nearly-optimal configurations of an FM taking into account a single objective. In this work, valid configurations are generated using a `fix` operator, but only basic CTCs are supported. A comparison with the work of White et al. (2009a) is provided, and the results show that the execution time of the algorithm of Guo et al. is lower, but at the cost of generating slightly worse configurations. Once again, it is not valid for our purposes because it is a single objective algorithm.

Finally, Sayyad et al. (2013) use several MOEAs that are extensively used to find configurations of FMs, which can be optimized regarding different criterion simultaneously. In this work, all CTCs are supported, and a `fix` operator is also used, but for a different purpose. In our approach, as well as in the approach of Guo et al. (2011), the goal of our `fix` operator is to repair an infeasible configuration, generating a valid one. However, the goal of the `feature fixing` operator of Sayyad et al. is to ensure that a list of features, which are common to all the valid configurations, are going to be present in all the possible configurations generated by the MOEA. As a result, invalid configurations are also generated, which are not useful for our purpose. Finally, as will be shown in Section 5, in the work of Sayyad et al., as well as in our approach, a seeding technique is applied to generate the initial population.

4. Dynamic reconfiguration approach

In this section, we explain our overall DSPL approach, describing those tasks which are performed at design time and those which are performed at runtime. Then, given that MOEAs are a central part of our dynamic reconfiguration service, we explain how we formally specify FMs and how we encode them in order to be used as input for these algorithms. We use MOEAs in our approach because they enable the efficient generation of valid configurations at runtime. However, there are other types of algorithms, such as *anytime* algorithms (Carlin and Zilberstein, 2011; Hansen et al., 1997), that could be used for this purpose. To the best of our knowledge, the application of these kinds of algorithms to the domain of our paper has not been explored yet, so we plan to study them as part of our future work in order to assess whether they yield better results.

Finally, we describe briefly our `fix` operator, which is used to adapt MOEAs and ensure that only valid configurations are generated at runtime.

4.1. Approach overview

Fig. 2 summarizes our approach. Firstly, variability modeling using FMs and the optimization criteria are defined at design time. The

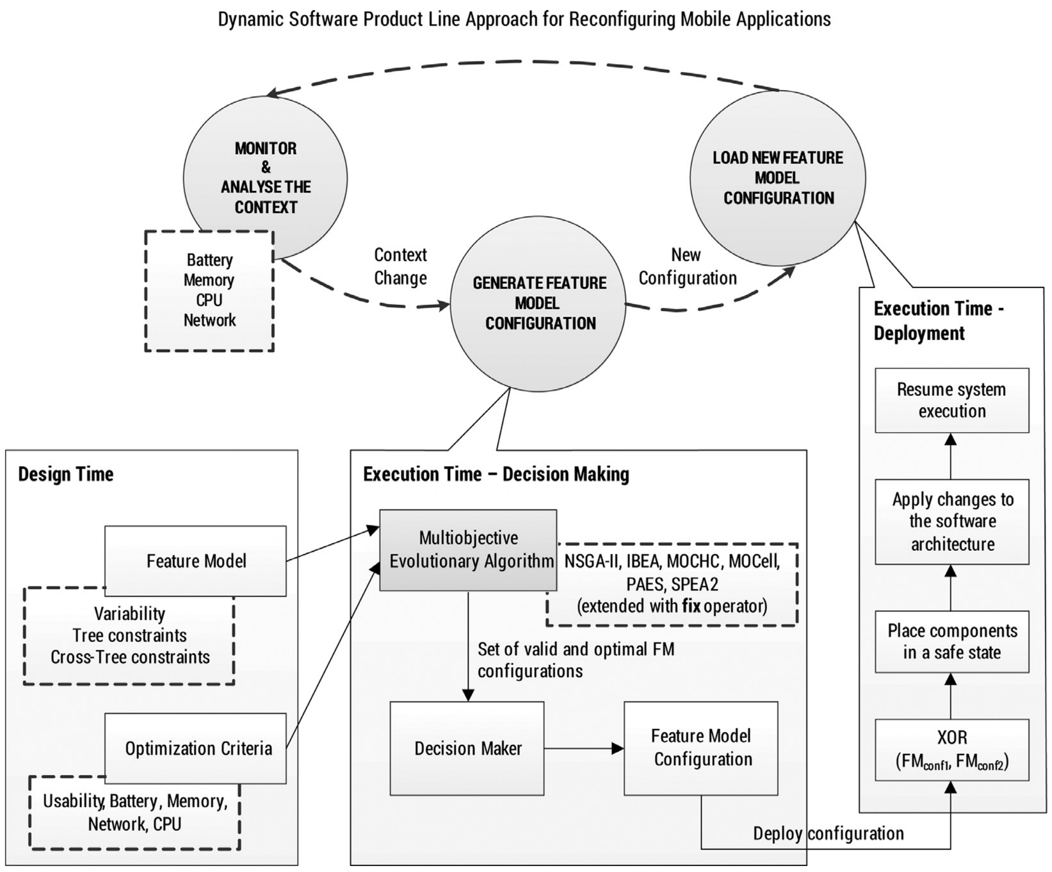
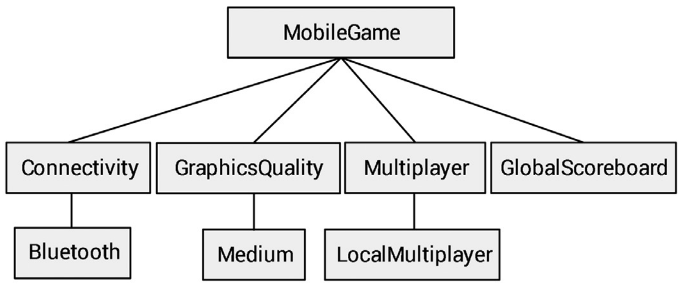

Dynamic Software Product Line Approach for Reconfiguring Mobile Applications
MONITOR & ANALYSE THE CONTEXT
Battery
Memory
CPU
Network
Context Change
GENERATE FEATURE MODEL CONFIGURATION
New Configuration
LOAD NEW FEATURE MODEL CONFIGURATION
Design Time
Feature Model
Variability
Tree constraints
Cross-Tree constraints
Optimization Criteria
Usability, Battery, Memory, Network, CPU
Execution Time – Decision Making
Multiobjective Evolutionary Algorithm
NSGA-II, IBEA, MOCHC, MOCell, PAES, SPEA2 (extended with fix operator)
Set of valid and optimal FM configurations
Decision Maker
Feature Model Configuration
Deploy configuration
Execution Time - Deployment
Resume system execution
Apply changes to the software architecture
Place components in a safe state
XOR (FMconf1, FMconf2)

Fig. 2. Approach overview.



S = {MobileGame, Connectivity, Bluetooth, GraphicsQuality, Medium, Multiplayer, LocalMultiplayer, GlobalScoreboard}
F − S = {Network, EDGE, HSPA, LTE, WiFi, Low, High, Sound, Quality, 128kbps, 192kbps, 256kbps, Vibration, OnlineMultiplayer}

Specification of the configuration

#1 ¬ GlobalScoreboard ∨ Network	C_{1p} = {Network}	C_{1n} = {GlobalScoreboard}
#2 ¬ LocalMultiplayer ∨ Bluetooth ∨ WiFi	C_{2p} = {Bluetooth,WiFi}	C_{2n} = {LocalMultiplayer}
#3 ¬ OnlineMultiplayer ∨ HSPA ∨ LTE ∨ WiFi	C_{3p} = {HSPA,LTE,WiFi}	C_{3n} = {OnlineMultiplayer}

Specification of the CTCs

#1 $C_{1p} \cap S = \emptyset$	$C_{1n} \cap F - S = \emptyset$	**Not satisfied**
#2 $C_{2p} \cap S$ = {Bluetooth}	$C_{2n} \cap F - S = \emptyset$	**Satisfied**
#3 $C_{3p} \cap S = \emptyset$	$C_{3n} \cap F - S$ = {OnlineMultiplayer}	**Satisfied**

Evaluation of the CTCs

Fig. 3. Specification and evaluation of CTCs.

FM determines exactly which configurations can be deployed at runtime, and the optimization criteria provide the information used at runtime to decide which configuration fits best the current execution context. The optimization criteria can comprise both functional and non-functional properties of the application such as usability, battery consumption, memory usage, CPU usage, bandwidth usage, etc.

At runtime, a loop is executed which consists of the following actions:

1. *Monitor and analyse the context*. The relevant context information (e.g., battery level, available memory, network state) is monitored and the gathered data are analyzed in order to determine whether it is necessary or not to reconfigure the mobile application.
2. *Generate feature model configuration*. When the context has changed and a reconfiguration is needed, a multiobjective evolutionary algorithm is executed, which generates a set of configurations of the FM which are optimal with regards to different optimization criteria. Then, the *Decision Maker* decides, from among this set of configurations, which one fits best the current execution context.
3. *Load new feature model configuration*. The configuration of the FM which has been generated in the previous step is deployed, replacing the previous one. To this end, the differences between both configurations(FM_{conf1} and FM_{conf2}) are calculated. As will be shown in Section 4.2, we encode the FM configurations as an array of boolean variables. Therefore, the differences between the previous configuration and the new one can be easily found by applying an XOR operation. Then, in order to ensure that the reconfiguration process is performed flawlessly, all the components are placed in a safe state before they are reconfigured. The changes in the configuration are then applied to the software architecture and the execution is resumed. As a result, the mobile application is now adapted to the current context. More details about the deployment of new configurations in our approach can be found in Pascual et al. (2014).

4.2. Formal specification and encoding of FMs

In order to use evolutionary algorithms for generating configurations of FMs, we need to formally specify FMs, including features, tree constraints and cross-tree constraints. This specification allows an efficient access to all the information about the FM.

The features of the FM are represented as a set $F = \{f_i, \ldots, f_n\}$, where n is the number of features in the FM. Then, a configuration of the FM can be defined as a subset of features $S = \{f_i \mid f_i \in F \wedge 1 \le i \le s \le n\}$, where s is the number of selected features.

The tree constraints are modeled by the following functions and sets:

1. The function Parent(f), which returns the parent of each feature $f \in F$, or *nil* in the case that f is the root feature.
2. The functions $B_{\text{OR}}(f)$ and $B_{\text{XOR}}(f)$ which return, for each feature $f \in F$, the features which are included in an OR or XOR group, respectively, together with f. In the case that f is not in a group, $B_{\text{OR}}(f)$ and $B_{\text{XOR}}(f)$ are empty sets. For instance, in the FM shown in Fig. 1:

$$B_{\text{XOR}}(\text{HSPA}) = \{\text{EDGE}, \text{HSPA}, \text{LTE}, \text{WIFI}\}$$
$$B_{\text{OR}}(\text{Network}) = \{\text{Network}, \text{Bluetooth}\}$$
$$B_{\text{XOR}}(\text{Sound}) = B_{\text{OR}}(\text{Sound}) = \emptyset.$$

3. The set $M = \{f_1, \ldots, f_m\}$, which contains all the mandatory features.

The cross-tree constraints are represented in CNF notation, as shown in Section 2.2. The set of CTCs is modeled by the set $C = \{c_i, \ldots c_k\}$. Each constraint c_i is then a clause $c_i = \{C_{ip}, C_{in}\}$, where $C_{ip} = \{f_p \mid f_p \in F \wedge 0 \le p \le n_p\}$ and $C_{in} = \{f_n \mid f_n \in F \wedge 0 \le n \le n_n\}$, being n_p and n_n the number of positive and negative literals, respectively. Then, according to this notation, a configuration S satisfies a CTC $c_i = \{C_{ip}, C_{in}\}$ if and only if $C_{ip} \cap S \neq \emptyset \vee C_{in} \cap (F - S) \neq \emptyset$.

Fig. 3 shows how the CTCs of the FM in Fig. 1 are specified, and the results of evaluating them for a configuration S (which is invalid)

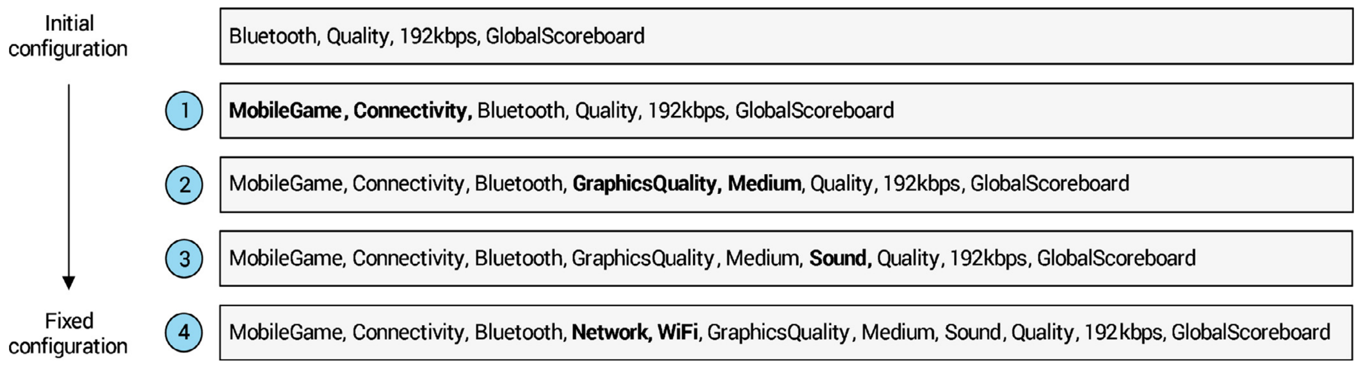

Initial configuration
Bluetooth, Quality, 192kbps, GlobalScoreboard
1
MobileGame, Connectivity, Bluetooth, Quality, 192kbps, GlobalScoreboard
2
MobileGame, Connectivity, Bluetooth, GraphicsQuality, Medium, Quality, 192kbps, GlobalScoreboard
3
MobileGame, Connectivity, Bluetooth, GraphicsQuality, Medium, Sound, Quality, 192kbps, GlobalScoreboard
Fixed configuration
4
MobileGame, Connectivity, Bluetooth, Network, WiFi, GraphicsQuality, Medium, Sound, Quality, 192kbps, GlobalScoreboard

Fig. 4. Example of configuration repaired by the fix operator.

in particular. For each CTC, we check, on the one hand, which positive literals are selected in S and, on the other hand, which negative literals are not selected in the configuration. As can be seen, the set of not selected features is defined as $F - S$. Firstly, in the case of CTC #1, `GlobalScoreboard` is selected in S but not `Network` and, as a result, the configuration S does not satisfy CTC #1. Secondly, CTC #2 is satisfied because `Bluetooth`, which is required by `LocalMultiplayer`, is selected. Lastly, CTC #3 is satisfied because `OnlineMultiplayer` is not selected in S.

Configurations need to be encoded in a way which can be managed by an evolutionary algorithm. To this end, we encode a configuration of an FM as an array of n boolean variables, one for each feature, being *TRUE* in the case that the feature is selected and *FALSE* otherwise.

4.3. The fix operator

During the execution of evolutionary algorithms, solutions are generated randomly by applying the selection, crossover and mutation operators. When a configuration of an FM is generated in this way, there is a high probability that it does not satisfy the tree and cross-tree constraints, rendering it invalid. In our approach, invalid configurations are not useful, thus we need a mechanism to prevent the evolutionary algorithm adding invalid configurations to the population. To this end, we have specified a `fix` operator which takes as input a configuration of an FM, which may be invalid, and applies the necessary transformations, returning as output a configuration which is valid regarding the tree and cross-tree constraints of the FM. The `fix` operator is executed each time the algorithm generates a new configuration, once the mutation operator has been applied.

Algorithm 1 shows the pseudocode of the NSGA-II (Deb et al., 2002) algorithm with the `fix` operator. During the generation of the initial population (Lines 3–9), each time a new solution is randomly generated, the `fix` operator is applied, transforming the invalid configuration of the FM into a valid one. Then, the algorithm can evaluate its fitness and add it to the population. Moreover, during the evolution process (Lines 10–23), each time a new offspring is generated as a result of selecting two solutions from the population, recombining them and applying a mutation operation, it is also repaired using our `fix` operator, its fitness is evaluated and it is added to the offspring population. All the details of the `fix` operator are available online.[5]

In the rest of this section, we describe briefly how the `fix` operator is able to repair an invalid configuration. The operator takes as input a set of features S which represents an invalid configuration. Then, it generates a new valid configuration taking as many features from S as possible. First, it focus on satisfying the tree constraints. To this end, for each feature $f \in S$, the `fix` operator recursively adds to the configuration, as needed, the following features:

Algorithm 1 NSGA-II with the `fix` operator.

```
Require: $P_{size}, p_{crossover}, p_{mutation}, eval_{max}$
Ensure: $PF$ (a set of nondominated solutions)
 1: $P = \emptyset$
 2: $evaluations = 0$
 3: for $i = 1$ to $P_{size}$ do
 4:     $s$ = NewSolution()
 5:     $s$ = Fix($s$)
 6:     EvaluateFitness($s$)
 7:     $evaluations = evaluations + 1$
 8:     $P = P + s$
 9: end for
10: while $evaluations < eval_{max}$ do
11:     $P_O = \emptyset$
12:     for $i = 1$ to $P_{size}/2$ do
13:         $parents$ = Selection(P)
14:         $offspring$ = Crossover($parents, p_{crossover}$)
15:         $offspring$ = Mutation($offspring, p_{mutation}$)
16:         $offspring$ = Fix($offspring$)
17:         EvaluateFitness($offspring$)
18:         $evaluations = evaluations + 1$
19:         $P_O = P_O + offspring$
20:     end for
21:     $P = P \bigcup P_O$
22:     RankingAndCrowdingDistance($P$)
23: end while
24: $PF$ = BestFront($P$)
```

1. The parent of f, Parent(f).
2. The mandatory child features of f, which are those features $f_c \in M \mid \text{Parent}(f_c) = f$.
3. A feature in each OR/XOR group of f, which are the features $f_g \mid P(f_g) = f \wedge B_{OR}(f_g) \bigcup B_{XOR}(f_g) \neq \emptyset$.

Then, the `fix` operator modifies the configuration in order to satisfy the cross-tree constraints. To this end, for each cross-tree constraint $c_i = \{C_{ip}, C_{in}\}$, in the case that it is not satisfied, a feature $f_p \in C_{ip}$ is added to the configuration.

Fig. 4 shows an example of an invalid configuration transformed by the `fix` operator. The initial configuration contains only the `Bluetooth`, `Quality`, `192kbps` and `GlobalScoreBoard` features. Although this configuration is invalid, it is fixed in several steps by the `fix` operator:

1. The feature `Bluetooth` is selected but not its parent. Then, its parent `Connectivity` is selected and, since this operation is executed recursively, the parent of `Connectivity`, which is the feature `MobileGame`, is also selected.

[5] http://caosd.lcc.uma.es/projects/famware/tools.htm

Table 3
Feature model corpus.

	Name	Features	Configurations
1	x264	17	2048
2	Wget	17	8192
3	Berkeley DB Memory	19	3840
4	Sensor Network	27	19152
5	Mobile Game	33	9198
6	Tank War	37	1,741,824
7	Mobile Media	43	2,128,896
8	Mobile Visit Guide	51	33,800,000
9	SPLOT-3CNF-500	500	3.779e15
10	SPLOT-3CNF-1000	1000	1.638e131
11	SPLOT-3CNF-2000	2000	N/A
12	SPLOT-3CNF-5000	5000	N/A

2. When `MobileGame` is selected, the `fix` operator detects that a mandatory child feature, `GraphicsQuality`, is not selected. Therefore, `GraphicsQuality` is added to the configuration and, recursively, a feature from its child XOR group. In this case, the `Medium` feature has been randomly selected.
3. The feature `Quality` is selected but not its parent. Therefore, the `fix` operator adds the feature `Sound` to the configuration.
4. Once these features have been selected, the configuration now satisfies the tree constraints. However, CTC #1 is not satisfied because `GlobalScoreboard` is selected but not `Network`. In order to satisfy the CTC, the `fix` operator adds the `Network` feature to the configuration and, recursively, a random feature from its child XOR group is also selected (`WiFi` in this case).

5. Experimental setup

In this section we describe how our evaluation was carried out: the selection of feature models and its attributes, the selected MOEAs and the evaluation process.

5.1. Feature model corpus and selected attributes

For the evaluation of the search algorithms we have selected 12 FMs, varying in size and complexity and including CTCs. Firstly, FMs 1–8 model the variability of actual SPLs, including some SPLs for mobile devices (FMs 5, 7 and 8). Secondly, FMs 9–12 have been randomly generated and include more features that found in typical SPLs for mobile devices. However, they have been selected to evaluate the scalability of our approach. All these FMs have been made publicly available by their authors at the main SPL-related websites such as SPLOT (see footnote 4, SPL Conqueror (von Guericke University Magdeburg, 2013b), and SPL2go (von Guericke University Magdeburg, 2013a). These FMs have also been described in papers published within the SPL community. Table 3 shows the number of features as well as the number of configurations.[6] However, in the case of FMs 11 and 12, to the best of our knowledge, it remains an open question how to efficiently calculate the number of configurations.

We extended these FMs with the following three attributes:

1. *Usability*. Usability measures, among others, how easy and satisfying to use is the application. It takes real values between 0 and 10, according to a normal distribution.
2. *Battery consumption*. Models the increase in battery consumption, measured in milliamps, introduced by the feature. It takes real values between 10.0 and 20.0 according to a normal distribution.
3. *Memory footprint*. Models the additional memory footprint, in megabytes, introduced by the feature. It takes real values between 0.0 and 10.0 according to a normal distribution.

[6] The values for FMs 9 and 10 were computed with the SPLAR library provided at SPLOT website.

Table 4
Evolutionary algorithms settings.

Algorithm	Parameters
NSGA-II	Population size = 100
IBEA	Population size = 100
	Archive size = 100
MOCHC	Population size = 100
	Initial convergence count = 0.25
	Preserved population = 0.05
	Convergence value = 3
MOCell	Population size = 100
	Archive size = 100
	Feedback = 0.1
PAES	Archive size = 100
	Bisections = 5
SPEA2	Population size = 100
	Archive size = 100

These attributes represent our optimization objectives. Thus the MOEAs we selected should: maximize usability, minimize battery consumption and minimize memory usage.

5.2. Multi-objective evolutionary algorithms

We evaluated the following six MOEAs, available in the jMetal framework (Durillo and Nebro, 2011):

1. Nondominated Sorting Genetic Algorithm II (NSGA-II; Deb et al., 2002).
2. Indicator-based Evolutionary Algorithm (IBEA; Zitzler and Künzli, 2004).
3. Multiobjective Cross generational elitist selection, Heterogeneous recombination, Cataclysmic mutation (MOCHC; Nebro et al., 2007).
4. Cellular Genetic Algorithm for Multiobjective Optimization (MOCell; Nebro et al., 2009).
5. Pareto Archived Evolution Strategy (PAES; Knowles and Corne, 2000).
6. Strength Pareto Evolutionary Algorithm, version 2 (SPEA2; Zitzler et al., 2001).

Our `fix` operator was added to these algorithms, as illustrated in Algorithm 1.

Furthermore, in order to improve the quality of the solutions generated by these MOEAs, we have also applied a *seeding* technique, which has been used previously in the literature (Fraser and Arcuri, 2012; Sayyad et al., 2013). Specifically, the seeding technique used in our approach works as follows:

1. A valid configuration (the *seed*), which includes around 50% of the features in the FM, is pre-computed for each FM.
2. The initial population is filled using the seed:
   (a) A mutation is introduced in the seed.
   (b) The resulting configuration is repaired using the `fix` operator.
   (c) The fixed configuration is added to the initial population.

The crossover operator applied was the *Single-Point Crossover*, with a crossover probability of 0.8. With regards to the mutation, the *Bit Flip Mutation* operator is applied with a probability of 0.1. The number of allowed fitness evaluations is 5000 for all the algorithms. The rest of the settings, which depend on the algorithm in particular, are detailed in Table 4. We have principally chosen values for these settings that are commonly used in the literature (Sayyad et al., 2013) or are default values on the jMetal framework (Durillo and Nebro, 2011). Although an exhaustive optimization of these settings has not been performed, they have proven to provide good results. Furthermore, as suggested by Arcuri and Fraser (2013), and confirmed in Kotelysanskii and Kapfhammer (2014), tuned parameters can improve upon default

Table 5
HV. Median and IQR.

	NSGA-II	IBEA	MOCHC	MOCell	PAES	SPEA2
x264	$0.44_{2.5e-4}$	$0.43_{0.0e+0}$	$0.35_{5.8e-2}$	$0.44_{2.0e-4}$	$0.44_{1.0e-2}$	$0.44_{2.4e-4}$
Wget	$0.49_{3.1e-3}$	$0.48_{0.0e+0}$	$0.36_{8.9e-2}$	$0.49_{2.0e-3}$	$0.47_{1.2e-2}$	$0.49_{9.2e-4}$
Berkeley DB Memory	$0.51_{7.0e-3}$	$0.51_{1.2e-4}$	$0.38_{1.2e-1}$	$0.51_{2.4e-3}$	$0.50_{1.6e-2}$	$0.51_{4.7e-3}$
Sensor Network	$0.34_{3.0e-2}$	$0.38_{3.7e-2}$	$0.21_{3.5e-2}$	$0.35_{3.2e-2}$	$0.25_{3.2e-2}$	$0.33_{3.4e-2}$
Mobile Game	$0.28_{3.8e-2}$	$0.32_{5.9e-2}$	$0.11_{6.2e-2}$	$0.04_{4.5e-2}$	$0.24_{4.2e-2}$	$0.27_{3.8e-2}$
Tank War	$0.49_{1.8e-2}$	$0.53_{1.9e-2}$	$0.38_{7.1e-2}$	$0.49_{1.8e-2}$	$0.42_{5.0e-2}$	$0.49_{1.9e-2}$
Mobile Media	$0.40_{3.6e-2}$	$0.46_{4.0e-2}$	$0.23_{5.5e-2}$	$0.41_{3.6e-2}$	$0.36_{4.3e-2}$	$0.40_{3.8e-2}$
Mobile Guide	$0.43_{1.3e-2}$	$0.45_{1.1e-2}$	$0.33_{3.7e-2}$	$0.42_{1.5e-2}$	$0.36_{5.8e-2}$	$0.43_{1.6e-2}$
SPLOT-3CNF-500	$0.32_{1.6e-2}$	$0.35_{1.8e-2}$	$0.30_{3.8e-2}$	$0.10_{7.0e-2}$	$0.31_{4.6e-2}$	$0.31_{1.6e-2}$
SPLOT-3CNF-1000	$0.19_{1.7e-2}$	$0.19_{1.7e-2}$	$0.34_{4.5e-2}$	$0.13_{1.3e-2}$	$0.21_{4.8e-2}$	$0.18_{1.5e-2}$
SPLOT-3CNF-2000	$0.23_{1.5e-2}$	$0.22_{1.4e-2}$	$0.33_{2.9e-2}$	$0.19_{1.9e-2}$	$0.22_{3.3e-2}$	$0.22_{1.3e-2}$
SPLOT-3CNF-5000	$0.28_{9.8e-3}$	$0.22_{2.3e-2}$	$0.30_{2.1e-2}$	$0.22_{1.3e-2}$	$0.22_{1.9e-2}$	$0.22_{2.9e-2}$

values generally, but they are far from optimal in individual problem instances. Therefore, the objective of this paper is not tuning the values to improve the performance on particular algorithms and FMs, but comparing the MOEAs using default parameter values.

5.3. Evaluation process outline

The evaluation process was divided in two stages. The goal of the first stage was to assess which MOEA yields better results in regard to the quality of the solutions generated and the execution time. To this end, we performed 1000 independent runs for each algorithm and feature model and compared their results using two standard quality indicators. These runs were executed in a desktop computer, with an Intel Core i7-4700K CPU at 3.5 GHz and 16 GB of RAM memory running Ubuntu 13.10. We performed a set of statistical analysis to help us select the best algorithm.

The goal of the second stage was to evaluate the performance of these MOEAs on an actual mobile device. To this end, we developed an Android application, which is available for download,[7] which allowed us to execute the MOEAs shown in Section 5.2 over a set of feature models previously copied to the storage of the mobile device. Using this application it is also possible to configure some parameters of the algorithms and the evaluation process such as the population size, the number of fitness evaluations or the number of independent runs. When the evaluation process is finished, the mobile application generates a set of output files containing the execution time for each independent run of the experiment, as well as the results of the quality indicators. Statistical tests are then applied to these results using a desktop computer. In particular, we executed 100 independent runs for each feature model and for each MOEA on an LG Nexus 5 device running Android 4.4.3. The next section describes the results obtained.

6. Experiment results

In this section we show and analyze the results obtained in the experiment. Firstly, we compare the fronts obtained by each algorithm using two standard quality indicators, *Hypervolume* and *Generational Distance*, with the purpose of identifying which MOEA performs best for our purpose. Secondly, we analyze the execution time for each MOEA and for each FM, in order to know which MOEAs are more efficient and whether their efficiency depends on the size of the FM. Thirdly, we compare the objectives' values obtained in the initial population with the values obtained in the final front. This allows us to quantify easily how the fitness of the population evolves along the execution of the algorithm. Fourthly, we apply the Mann–Whitney[8] U test (Mann and Whitney, 1947) and the $\hat{A}_{12}$ statistic (Vargha and Delaney, 2000) to the values of these metrics to assure that the differences identified are statistically meaningful. Lastly, we execute the MOEAs on a mobile device, evaluating whether they are efficient enough for the dynamic reconfiguration of mobile applications.

6.1. Quality indicators

We have selected two quality indicators that are commonly applied in multiobjective evolutionary algorithms, Hypervolume and Generational Distance. Firstly, these quality indicators and the results obtained are presented in Sections 6.1.1 and 6.1.2. Secondly, the conclusions drawn from the results are shown in Section 6.1.3.

6.1.1. Hypervolume

The Hypervolume (HV) indicator calculates the volume in the objective space covered by the members of a non-dominated set of solutions Q (Zitzler and Thiele, 1999). For each solution $i \in Q$, a hypercube v_i is computed from a reference point and the solution i as the diagonal corners of the hypercube. The reference point can be found by constructing a vector of worst objective function values. The hypervolume is calculated as:

$$\mathrm{HV} = \mathrm{volume}\left(\bigcup_{i=1}^{|Q|} v_i\right)$$

The result of this indicator is affected by the scales of the objectives. Therefore, before calculating its value, it is necessary to perform a normalization procedure. In jMetal, all the objectives values in the Pareto front are normalized before calculating the HV. Higher values for HV are desirable, because a wider set of non-dominated solutions can be obtained.

Table 5 shows the median and the interquartile range (IQR) of the HV values obtained when the multiobjective evolutionary algorithms are applied to our FMs. Each row contains the values obtained for each FM, being the best values highlighted with a dark background.

6.1.2. Generational distance

Sarker and Coello Coello (2002) show that Generational Distance (GD) helps to assess the quality of the results obtained by MOEAs. More concretely, this quality indicator shows how far, on average, the front Q is from the Pareto front (Van Veldhuizen, 1999). It is defined as:

$$\mathrm{GD}(Q) = \frac{1}{n}\sqrt{\sum_{i=1}^{n} {d_i}^2}$$

where n is the number of points in Q and d_i is the Euclidean distance between each point in Q and the nearest solution in the Pareto front.

Lower values for GD are desirable, since this means that the approximated Pareto front Q is closer to the true Pareto front, and a value of 0 indicates that all the points in Q are in the Pareto front. In Table 6, the median and the IQR of the GD values are shown.

[7] http://caosd.lcc.uma.es/projects/famware/tools.htm
[8] This statistical test is equivalent to the Wilcoxon Rank-Sum test.

Table 6
GD. Median and IQR (multiplied by 1e+3).

	NSGA-II	IBEA	MOCHC	MOCell	PAES	SPEA2
x264	$0.00_{0.00}$	$0.00_{0.00}$	$4.02_{6.10}$	$0.00_{0.00}$	$0.74_{1.80}$	$0.00_{0.00}$
Wget	$0.91_{0.27}$	$0.00_{0.00}$	$7.51_{7.60}$	$0.81_{0.21}$	$1.17_{0.59}$	$0.73_{0.26}$
Berkeley DB Memory	$0.22_{0.56}$	$0.00_{0.00}$	$5.66_{9.50}$	$0.00_{0.68}$	$0.54_{0.96}$	$0.00_{0.44}$
Sensor Network	$1.23_{1.90}$	$0.25_{0.68}$	$1.92_{1.10}$	$1.32_{1.80}$	$2.12_{0.87}$	$1.29_{1.40}$
Mobile Game	$2.41_{1.70}$	$1.00_{1.60}$	$3.84_{2.90}$	$36.30_{30.00}$	$2.69_{1.20}$	$2.19_{1.40}$
Tank War	$2.08_{0.67}$	$0.37_{0.31}$	$4.66_{3.20}$	$2.36_{0.68}$	$3.06_{0.88}$	$2.09_{0.63}$
Mobile Media	$2.96_{1.20}$	$1.37_{1.00}$	$7.11_{5.00}$	$3.12_{1.20}$	$3.89_{1.40}$	$2.90_{1.10}$
Mobile Guide	$1.58_{0.38}$	$0.46_{0.23}$	$6.00_{2.80}$	$1.89_{0.45}$	$2.69_{0.79}$	$1.72_{0.37}$
SPLOT-3CNF-500	$13.80_{1.60}$	$11.90_{1.40}$	$9.87_{2.90}$	$154.00_{180.00}$	$10.20_{3.00}$	$14.70_{1.70}$
SPLOT-3CNF-1000	$34.10_{4.10}$	$36.60_{4.70}$	$9.51_{3.40}$	$62.00_{11.00}$	$26.90_{11.00}$	$38.40_{4.80}$
SPLOT-3CNF-2000	$22.90_{2.50}$	$24.90_{2.70}$	$7.11_{2.50}$	$38.60_{5.70}$	$20.30_{7.80}$	$26.10_{2.60}$
SPLOT-3CNF-5000	$12.80_{1.60}$	$20.20_{3.30}$	$4.88_{1.70}$	$21.10_{2.90}$	$16.20_{4.10}$	$20.10_{4.10}$

Table 7
TIME. Median and IQR (in milliseconds).

	NSGA-II	IBEA	MOCHC	MOCell	PAES	SPEA2
x264	$45.5_{2.5}$	$417.0_{13.0}$	$55.6_{8.0}$	$46.6_{7.1}$	$68.2_{6.6}$	$556.0_{21.0}$
Wget	$43.6_{2.0}$	$426.0_{15.0}$	$51.5_{5.4}$	$59.4_{6.2}$	$88.3_{12.0}$	$493.0_{12.0}$
Berkeley DB Memory	$50.9_{4.3}$	$432.0_{15.0}$	$52.6_{2.5}$	$48.3_{1.3}$	$61.1_{3.8}$	$526.0_{23.0}$
Sensor Network	$68.8_{5.1}$	$451.0_{15.0.}$	$90.7_{11.0}$	$84.5_{3.3}$	$109.0_{6.8}$	$611.0_{23.0}$
Mobile Game	$83.0_{3.4}$	$442.0_{12.0}$	$136.0_{34.0}$	$20.7_{1.4}$	$106.0_{6.5}$	$562.0_{16.0}$
Tank War	$93.7_{6.5}$	$481.0_{12.0}$	$136.0_{26.0}$	$96.7_{3.2}$	$96.3_{7.1}$	$454.0_{17.0}$
Mobile Media	$104.0_{9.1}$	$491.0_{15.0}$	$176.0_{36.0}$	$110.0_{6.1}$	$104.0_{6.1}$	$487.0_{22.0}$
Mobile Guide	$114.0_{7.4}$	$499.0_{11.0}$	$146.0_{24.0}$	$122.0_{5.8}$	$116.0_{8.5}$	$472.0_{17.0}$
SPLOT-3CNF-500	$1240.0_{23.0}$	$1490.0_{25.0}$	$2670.0_{250.0}$	$892.0_{28.0}$	$1090.0_{26.0}$	$1450.0_{20.0}$
SPLOT-3CNF-1000	$2620.0_{48.0}$	$2820.0_{32.0}$	$5490.0_{520.0}$	$2200.0_{59.0}$	$2350.0_{37.0}$	$2740.0_{27.0}$
SPLOT-3CNF-2000	$6570.0_{79.0}$	$6930.0_{56.0}$	$14100.0_{1300.0}$	$5870.0_{120.0}$	$6160.0_{110.0}$	$6650.0_{60.0}$
SPLOT-3CNF-5000	$17900.0_{130.0}$	$18100.0_{180.0}$	$43400.0_{3000.0}$	$16400.0_{320.0}$	$17200.0_{230.0}$	$18000.0_{170.0}$

6.1.3. Conclusions

Firstly, we can see the HV results in Table 5:

- In the smallest FMs (`x264`, `Wget` and `Berkeley DB Memory`), all the MOEAs yield similar results, except MOCHC, which covers less volume in the objective space.
- In the following FMs, the size of which is between 27 and 51 features, IBEA covers a higher volume in the objective space than the rest of the algorithms. In this case, MOCHC yields the worst HV values too.
- Unlike the previous cases, MOCHC covers a higher volume in the objective space than the rest of MOEAs in the case of large FMs. In contrast, MOCell yields the worst results in these FMs.

Secondly, regarding GD, it can be seen in Table 6 that, for some algorithms, especially IBEA, some of the GD values are 0. This means that, in all the independent runs of the algorithm for a specific problem, all the solutions included in the final front were part of the Pareto front. Taking into account the rest of values, we can see that:

- IBEA yields the best results in the case of the smaller FMs (17–51 features), which means that the configurations generated by IBEA are closer to the optimal ones. In contrast, MOCHC yields the worst results in the majority of these FMs.
- In the case of large FMs (500–5000 features), the solutions generated by MOCHC are closer to the optimal ones than those generated by the rest of the MOEAs.

Therefore, taking into account both the HV and the GD quality indicators, we can conclude that:

- IBEA generates better solutions in the case of small and medium-sized FMs.
- MOCHC generates better solutions in the case of large FMs.

6.2. Execution time

In our work we focus on providing support for dynamic reconfiguration on mobile devices. Therefore, it is very important to evaluate the time spent by the algorithms in the generation of the Pareto front. Table 7 shows the median and the IQR of the execution time measured for each algorithm and problem.

We can see that:

- IBEA and SPEA2 are significantly slower than the rest of the MOEAs in the case of small and medium-sized FMs. However, they scale very well and their execution time, in the case of large FMs, is similar to the other MOEAs.
- Although the execution time of NSGA-II, MOCHC, MOCell and PAES is similar in the case of small and medium-sized FMs, being NSGA-II slightly faster, we can find significant differences in the case of large FMs:
  - MOCHC does not scale well, and its execution time is very high. However, although it is slower than the rest of the MOEAs, as seen previously, it also generates better configurations.
  - MOCell is faster than the rest of MOEAs but, as seen before, it covers less volume in the objective space and the solutions generated by MOCell are usually further from the Pareto front than those generated by other MOEAs. This is especially noticeable in the case of the `Mobile Game` feature model. In this FM, MOCell is significantly faster than the rest of MOEAs but, as can be seen in Tables 5 and 6, the quality of the solutions generated by MOCell is lower than the quality of the solutions obtained using other algorithms.

Fig. 5 shows the fraction of the execution time that is spent in the `fix` operator. On the one hand, we can see that this time tends to increase according to the complexity of the FM. The reason is that it is harder to repair configurations in complex FMs and it may require several retries before being able to get a valid configuration. However, our approach focuses on mobile DSPLs, and in these cases the `fix` operator yields good results, taking less than 40% of the execution time in the worst case (less than 10% for IBEA, less than 30% for MOCHC and less than 40% for PAES). Moreover, although our fix operator in some cases takes up an important part of the algorithm execution

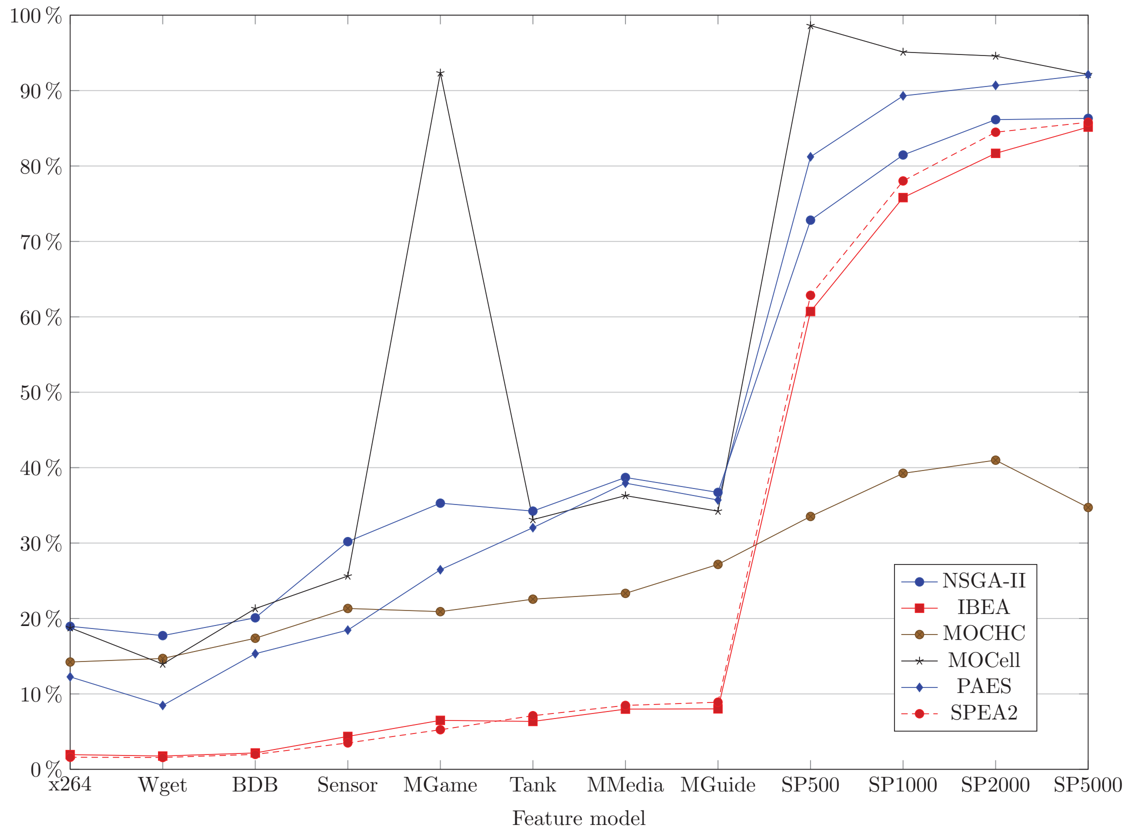


Fig. 5. Fraction of the execution time spent in the fix operator.

time, it does not have a relevant impact on the complete execution times of the algorithms, which are acceptable for mobile applications. Although there is a peak in MOCell in the `MGame` FM, it is due to the very fast execution time of this MOEA for this FM in particular, as can be seen in Table 7, and not due to a higher execution time of the `fix` operator. On the other hand, it can be seen that the fraction of time spent in the fix operator is significantly lower on IBEA and SPEA2 in the case of small and medium-sized FMs, but it is similar to other MOEAs in the case of large FMs. The reason is that, as seen before, IBEA and PAES are slower than the rest of the MOEAs but scale well with respect to the size of the FM. Therefore, although the absolute time spent in the fix operator is similar to the rest of the MOEAs in the smaller FMs, it represents a smaller fraction of the total execution time.

Summarizing, none of the algorithms is the best in all the problems and quality indicators measured. Choosing an MOEA involves a trade-off between the quality of the configurations and the execution time, and it is not straightforward to explain why some MOEAs are more efficient or generate better solutions than others. Instead, our focus is on: (i) demonstrating that MOEAs are efficient enough to reconfigure mobile applications; and (ii) determining which MOEAs are more appropriate for mobile devices (as will be shown in Section 6.4).

6.3. *Statistical analysis*

We apply two statistical tests, Mann–Whitney U test and $\hat{A}_{12}$ statistic, which are commonly applied in randomized algorithms, to assure that the differences among MOEAs identified in the previous section are statistically meaningful (Arcuri and Briand, 2014).

6.3.1. *Mann–Whitney U Test*

We have performed the Mann–Whitney U test in order to check whether the differences in the distributions of values of HV, GD and execution time shown in Section 6.1 are statistically meaningful. The confidence level applied is 95%, meaning that the *p*-value is under 0.05.

Table 8 summarizes the results of the Mann–Whitney U test applied to the HV, GD and execution time values. This table shows, for each pair of algorithms, which one obtains better results for each FM. We can see that the Mann–Whitney U test supports the results shown previously which included the median values of HV, GD and execution time.

In regard to the quality of the generated configurations, measured by the HV and the GD, it confirms that MOCell is usually outperformed by the rest of the algorithms in the case of large FMs, providing nevertheless, very good results on small FMs. In contrast, MOCHC shows the opposite behavior. The Mann–Whitney U test also confirms that the quality of the solutions generated by NSGA-II and IBEA tend to be higher than in the rest of the MOEAs.

Finally, regarding execution time, the Mann–Whitney U test confirms that IBEA and SPEA2 are clearly slower than the rest of algorithms in small and medium-sized FMs, but they scale well in large FMs, outperforming MOCHC in these cases. Regarding the rest of the MOEAs, NSGA-II is usually the fastest algorithm in small FMs, but it is outperformed by PAES and MOCell in larger FMs.

6.3.2. $\hat{A}_{12}$ *statistic*

Applying the $\hat{A}_{12}$ statistic (Vargha and Delaney, 2000), called the *measure of stochastic superiority*, we can assess not only which algorithm yields better results but also how often. In particular, given a performance measure *M*, $\hat{A}_{12}$ measures the probability that running algorithm A yields higher *M* values than running another algorithm B. If the two algorithms are equivalent, then $\hat{A}_{12} = 0.5$. If $\hat{A}_{12} = 0.3$ means that one would obtain higher values for *M* with algorithm A, 30% of the time.

Table 8
Mann–Whitney U Test.

		IBEA						MOCHC						MOCell						PAES						SPEA2					
		x264	Wget	Berkeley DB Memory	Sensor Network	Mobile Game	Tank War	x264	Wget	Berkeley DB Memory	Sensor Network	Mobile Game	Tank War	x264	Wget	Berkeley DB Memory	Sensor Network	Mobile Game	Tank War	x264	Wget	Berkeley DB Memory	Sensor Network	Mobile Game	Tank War	x264	Wget	Berkeley DB Memory	Sensor Network	Mobile Game	Tank War
HV	NSGA-II	▲	▲	▲	▽	▽	▽	▲	▲	▲	▲	▲	▲	▽	▽	▽	▽	▲	▲	▲	▲	▲	▲	▲	▲	▽	▽	▽	▲	▲	–
	IBEA							▲	▲	▲	▲	▲	▲	▽	▽	▽	▲	▲	▲	▽	▲	▲	▲	▲	▲	▽	▽	▽	▲	▲	▲
	MOCHC													▽	▽	▽	▽	▲	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽
	MOCell																			▲	▲	▲	▲	▽	▲	▲	▽	▲	▲	▽	▽
	PAES																									▽	▽	▽	▽	▽	▽
GD	NSGA-II	▲	▽	▽	▽	▽	▽	▲	▲	▲	▲	▲	▲	▲	▽	–	–	▲	▲	▲	▲	▲	▲	▲	▲	▲	▽	▽	–	▽	–
	IBEA							▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲
	MOCHC													▽	▽	▽	▽	▲	▽	▽	▽	▽	▲	▽	▽	▽	▽	▽	▽	▽	▽
	MOCell																			▲	▲	▲	▲	▽	▲	–	▽	▲	▽	▽	▽
	PAES																									▽	▽	▽	▽	▽	▽
TIME	NSGA-II	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▽	▲	▽	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲
	IBEA							▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▲	▲	▲	▲	▲	▽
	MOCHC													▽	▲	▽	▽	▽	▽	▲	▲	▲	▲	▽	▽	▲	▲	▲	▲	▲	▲
	MOCell																			▲	▲	▲	▲	▲	▽	▲	▲	▲	▲	▲	▲
	PAES																									▲	▲	▲	▲	▲	▲

		IBEA						MOCHC						MOCell						PAES						SPEA2					
		Mobile Media	Mobile Guide	SPLOT-3CNF-500	SPLOT-3CNF-1000	SPLOT-3CNF-2000	SPLOT-3CNF-5000	Mobile Media	Mobile Guide	SPLOT-3CNF-500	SPLOT-3CNF-1000	SPLOT-3CNF-2000	SPLOT-3CNF-5000	Mobile Media	Mobile Guide	SPLOT-3CNF-500	SPLOT-3CNF-1000	SPLOT-3CNF-2000	SPLOT-3CNF-5000	Mobile Media	Mobile Guide	SPLOT-3CNF-500	SPLOT-3CNF-1000	SPLOT-3CNF-2000	SPLOT-3CNF-5000	Mobile Media	Mobile Guide	SPLOT-3CNF-500	SPLOT-3CNF-1000	SPLOT-3CNF-2000	SPLOT-3CNF-5000
HV	NSGA-II	▽	▽	▽	▲	▲	▲	▲	▲	▲	▽	▽	▽	▽	▲	▲	▲	▲	▲	▲	▲	▲	▽	▲	▲	–	–	▲	▲	▲	▲
	IBEA							▲	▲	▲	▽	▽	▽	▲	▲	▲	▲	▲	▲	▲	▲	▲	▽	▲	▲	▲	▲	▲	▲	▲	–
	MOCHC													▽	▽	▲	▲	▲	▲	▽	▽	▽	▲	▲	▲	▽	▽	▽	▲	▲	▲
	MOCell																			▲	▲	▽	▽	▽	▽	▲	▽	▽	▽	▽	▽
	PAES																									▽	▽	▽	▲	▲	▽
GD	NSGA-II	▽	▽	▽	▲	▲	▲	▲	▲	▽	▽	▽	▽	▲	▲	▲	▲	▲	▲	▲	▲	▽	▽	▽	▲	–	▲	▲	▲	▲	▲
	IBEA							▲	▲	▽	▽	▽	▽	▲	▲	▲	▲	▲	▲	▲	▲	▽	▽	▽	▽	▲	▲	▲	▲	▲	–
	MOCHC													▽	▽	▲	▲	▲	▲	▽	▽	▲	▲	▲	▲	▽	▽	▲	▲	▲	▲
	MOCell																			▲	▲	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽
	PAES																									▽	▽	▲	▲	▲	▲
TIME	NSGA-II	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	▽	▽	▽	▽	▽	▲	▽	▽	▽	▽	▲	▲	▲	▲	▲	▲
	IBEA							▽	▽	▲	▲	▲	▲	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽
	MOCHC													▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▽	▲	▲	▽	▽	▽	▽
	MOCell																			▽	▽	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲
	PAES																									▲	▲	▲	▲	▲	▲

Table 9
$\hat{A}_{12}$ statistical test results.

	HV					GD					TIME				
	IBEA	MOCHC	MOCell	PAES	SPEA2	IBEA	MOCHC	MOCell	PAES	SPEA2	IBEA	MOCHC	MOCell	PAES	SPEA2
NSGA-II	0.462	0.652	0.570	0.594	0.513	0.612	0.401	0.410	0.439	0.490	0.265	0.395	0.510	0.430	0.268
IBEA		0.700	0.605	0.615	0.552		0.358	0.317	0.353	0.383		0.707	0.736	0.736	0.353
MOCHC			0.438	0.445	0.374			0.521	0.573	0.599			0.608	0.542	0.294
MOCell				0.483	0.427				0.536	0.589				0.416	0.264
PAES					0.433					0.555					0.264

Table 10
TIME. Median and IQR (in milliseconds).

	NSGA-II	IBEA	MOCHC	MOCell	PAES	SPEA2
x264	2870_{620}	27200_{1700}	3420_{1400}	3390_{350}	2260_{450}	14900_{2000}
Wget	2790_{590}	27500_{1900}	3470_{860}	4030_{540}	2710_{680}	13400_{2700}
Berkeley DB Memory	3080_{680}	27600_{2200}	3630_{740}	3330_{310}	2110_{480}	13200_{2600}
Sensor Network	3300_{610}	28400_{1700}	4860_{960}	5210_{510}	3480_{740}	17800_{3000}
Mobile Game	3560_{700}	26900_{1900}	6640_{2400}	655_{220}	3160_{530}	15900_{2600}
Tank War	3870_{770}	29300_{1700}	7490_{1900}	5380_{460}	3240_{520}	10700_{1800}
Mobile Media	4220_{650}	29700_{2000}	9390_{2900}	5840_{460}	3470_{670}	12200_{2100}
Mobile Guide	4570_{590}	30000_{2200}	7550_{1700}	6800_{630}	4070_{740}	11700_{1900}
SPLOT-3CNF-500	31800_{3100}	58600_{3400}	110000_{16000}	16000_{4100}	30500_{2900}	43400_{3300}

Table 9 shows the $\hat{A}_{12}$ values for each algorithm and metric applied in the experiment. In regard to HV, we can see that MOCHC provides lower (i.e., worse) values than the rest of the algorithms the majority of the time. For instance, NSGA-II provides better HV values than MOCHC 65.2% of the cases. In contrast, IBEA yields better HV results most of the time (53.8% in the worst case, when it is compared to NSGA-II, and 70.0% when it is compared with MOCHC).

Regarding GD, we can see that IBEA yields the lowest (i.e., better) results, at least, 61.2% of the time, followed by NSGA-II. In contrast, MOCHC and MOCell yield the worse values, being outperformed by all the MOEAs. The rest of the MOEAs are very close when they are compared with each other because the $\hat{A}_{12}$ values are around 0.5.

The most remarkable differences can be found in the execution time. It can be seen that MOCell and NSGA-II are the fastest algorithms while, on the other hand, IBEA and SPEA2 are clearly the slowest MOEAs.

6.4. Mobile device performance

The most important criterion when choosing an algorithm for dynamically reconfiguring mobile applications is the execution time. After reviewing the current literature, we have not found a standard or widely accepted response time for mobile applications. Moreover, Seow (2008) shows that it is not possible to clearly define a standard response time, because the response time that users would be willing to accept depends on many factors such as the type of current network connection (WiFi or 3G), the available device resources such as memory or the criticality of the information being accessed.

Given that the CPU architecture in mobile devices is usually different than in desktop computers (ARM instead of x86), we cannot assume that algorithms which run faster on a desktop computer will also run faster on mobile devices. Therefore, we have executed all the algorithms in a mobile device, an LG Nexus 5 smartphone running Android 4.4.3, measuring the execution time for each one of them.

Table 10 shows the median and IQR of the time spent in the generation of the Pareto front for all FMs up to `SPLOT-3-CNF-500`. We stopped here because the times obtained for this FM indicate that for mobile applications the FMs managed should not exceed this size if a reasonable response time wants to be obtained.

We can see that, in accordance to the results shown previously in this section, IBEA and SPEA2 are slower than the rest of the algorithms. Given that, in the case of these algorithms, the execution time is very high even when they are applied to small FMs, we can therefore conclude that IBEA and SPEA are not suitable for the dynamic reconfiguration of mobile applications. In the majority of the cases, PAES and NSGA-II are the fastest algorithms. However, as shown previously, the quality of the configurations generated by NSGA-II is higher than the quality of those generated by PAES. Therefore, there is a trade-off between execution time and the quality of the front generated.

Finally, although the execution time is high for large FMs such as `SPLOT-3CNF-500`, as stated, we are focusing on mobile DSPLs, and the results show that our approach is efficient enough to be executed on DSPLs for mobile devices. Furthermore, it is worth noting that these MOEAs can be executed in the background, which can significantly reduce the impact of their execution on the user experience.

7. Discussion

In this section we discuss the results of the experiments shown in Section 6, analyzing whether they fit the requirements for the optimization algorithm described in Section 1. Moreover, we also discuss the main threats to the validity of our approach.

7.1. Requirements satisfaction

In Section 1 we stated that, in order to dynamically adapt mobile applications to the execution context, we need an optimization algorithm that satisfies several requirements. We discuss whether and how we have satisfied these requirements in our approach, taking into account the experimental results.

Generate only valid configurations. We have extended the MOEAs with a `fix` operator that repairs an invalid configuration of the FM, generating a valid one as a result. Therefore, only valid configurations are added to the population during the execution of the MOEAs, which guarantees that the front obtained as the output of the algorithms contains only valid configurations.

Multiobjective optimization support. We have satisfied this requirement by selecting only MOEAs. Moreover, we have demonstrated that the evaluated algorithms can successfully generate configurations of FMs when they are augmented with three different objective functions.

Fast enough execution time. In the case that the time spent in the reconfiguration of the application is too high, the user experience can be harmed considerably, as stated in Section 6.4. In this respect, we have seen that the fastest algorithm is PAES. For instance, in the case of a DSPL of a mobile application with 51 features and 33,800,000 valid configurations, the execution time is about 4 s. However, there is a trade-off between execution time and the quality of the generated configurations. For instance, NSGA-II is slightly slower than PAES but it generates better configurations. Moreover, the use of the jMetal framework also introduces an overhead in the execution times. The reason is that the jMetal framework has been implemented with the objective of being extensible and support very different types of optimization problems, and this flexibility introduces a significant overhead in the implementation of the algorithms. A tailor-made implementation of the MOEAs, especially focused on their efficiency on mobile devices, may reduce the execution time considerably.

Support for DSPLs in mobile applications. As stated before, in DSPLs of mobile applications the variability degree is much lower (around hundreds or thousands of configurations) than in SPLs at design time. We have demonstrated that our approach is suitable for DSPLs of mobile applications by applying it to FMs with up to 33 million configurations. We have obtained good results, showing that our approach generates valid configurations for mobile DSPLs, optimized with respect to several objectives, and with an execution time fast enough to be used in mobile devices.

All the evaluated MOEAs satisfy all the requirements imposed by our approach, excepting the requirement for a fast enough execution time that it is satisfied only by some of them. Concretely, PAES and NSGA-II are the fastest algorithms, although MOCHC and MOCell also present acceptable execution times when they are executed at runtime on a mobile device. Anyway, there is still place for improvement regarding the execution times of these algorithms by generating tailor implementations of them instead of using the jMetal framework.

Based on the results shown in Section 6, we can clearly summarize as conclusion that: *Using MOEAs is an effective and competitive alternative for a dynamic reconfiguration service for mobile applications.*

7.2. Threats to validity

We have followed the guidelines proposed by de Olvieira Barros and Neto (2011) in order to identify the main validity threats to our approach and how they are addressed.

An internal validity threat is the parameter setting. In the experiments we have used standard parameter values and, although the work of Arcuri and Fraser (2013) suggests that default values might be good enough for evaluating some search based techniques, further research is necessary to assess whether their findings can be applied to the SPL context. Furthermore, a sensitivity analysis on some of these parameters would be necessary to ensure the robustness of our approach.

We have also identified two external validity threats. The first validity threat is the selection of the feature models used in the experiments. We have addressed this threat by selecting, on the one hand, FMs which specify the variability of real SPLs, including DSPLs based on real mobile applications. On the other hand, we have also selected some large FMs, which have been randomly generated, to further evaluate the scalability of our approach. In spite of the broad range of FMs that have been considered to evaluate our approach, there could be issues with a particular combination of the FM size, the variability degree and the number of cross-tree constraints for which the results obtained in our experiments may differ. The second validity threat is the selection of the MOEAs. To address this threat, we have included six distinct algorithms, covering diverse techniques of multiobjective optimization. However, since other MOEAs could potentially provide better results, we plan to extend our experiments with different algorithms as part of our future work.

With regards to construct validity threats, the *lack of assessing the validity of cost measures* and the *lack of assessing the validity of effectiveness measures* (de Olvieira Barros and Neto, 2011) have been addressed by using the same number of fitness evaluations in all the algorithms and using standard quality indicators.

Finally, we address conclusion validity threats by executing independent runs of the experiments (1000 independent runs on the computer and 100 on the mobile device) and by applying standard statistical analysis techniques such as Mann–Whitney U test and $\hat{A}_{12}$.

Moreover, all the code and data used to run these experiments are available[9] for replication and further details.

8. Conclusions and future work

In this paper we have presented a novel DSPL approach that provides support for the dynamic reconfiguration of mobile applications, generating optimal configurations with regards to distinct criteria such as quality of service, battery consumption or memory footprint. To this end, the variability of the application is modeled using an FM, and configurations of the FM are generated at runtime using MOEAs. We have adapted commonly used MOEAs by including a `fix` operator, which transforms invalid configurations of an FM into valid ones, in order to generate only valid configurations of an FM at runtime. Then, from among the set of configurations returned by the MOEA, the most appropriate one for the current execution context is selected.

We have evaluated the quality of the Pareto fronts generated by the MOEAs by applying commonly used quality indicators, as well as the time spent in the generation of the Pareto fronts, making sure that the results obtained are statistically meaningful by applying distinct statistical tests.

We have executed the MOEAs on a mobile device, and the results show that the majority of them are efficient enough for the dynamic reconfiguration of mobile applications. With regard to scalability, we have seen that our `fix` operator is a limiting factor in handling very large FMs. However, as stated before, these FMs are not found in typical SPLs for mobile devices and, therefore, mobile devices do not need such scalable methods at this time.

As mentioned before, we will extend the experiments as part of our future work, including:

1. Additional MOEAs (such as those successfully used in the SOA domain).
2. A sensitivity analysis on some of the parameter settings, to assess the robustness of our approach with respect to the choices used in this paper.
3. Additional optimization objectives, to evaluate the scalability of our approach with respect to the number of objectives to optimize.

Moreover, we also plan to implement a tailor-made version of these MOEAs, focusing on efficiency in mobile devices, which may reduce the execution time considerably. Finally, we will incorporate *anytime* algorithms in our evaluation process, in order to assess whether they yield better results than MOEAs.

Acknowledgments

This work is funded by the Spanish Government project TIN2012-34840, by the Andalusian Government projects P09-TIC-5231 and P12-TIC1814, by the International Campus of Excellence Andalucía TECH and by the University of Málaga. It is also partially funded by the Austrian Science Fund (FWF) project P25289-N15 and by the Lise Meitner Fellowship M1421-N15.

[9] http://caosd.lcc.uma.es/projects/famware/tools.htm

Appendix A. Comparison of the initial population with the final front

This appendix shows how the fitness of the population evolves along the execution of the algorithm. To this end, we have measured, for each feature model, the median of the objectives in the first population and in the final front generated by each MOEA.

Figs. A1–A3 show the difference in the objectives values when the final front is compared with the initial population. On the one hand, we can see that in small and medium-sized FMs, the fitness in the final front is clearly higher than the fitness in the initial population. In some FMs (e.g., `x264`, `Wget`, `Sensor Network`, `Mobile Game`, `Mobile Media`), although the usability in the final front has decreased, the decreases in battery consumption and memory usage are much higher, meaning that these configurations are better than those generated in the initial population. For instance, in the case of the `Mobile Game` feature model, PAES has been able to decrease the battery consumption by 17.5% and the memory usage by 22.3% at the cost of decreasing the usability by only 1.75%. There are also FMs (e.g., `Berkeley DB Memory`, `Tank War`, `Mobile Guide`) where the usability has been increased at the

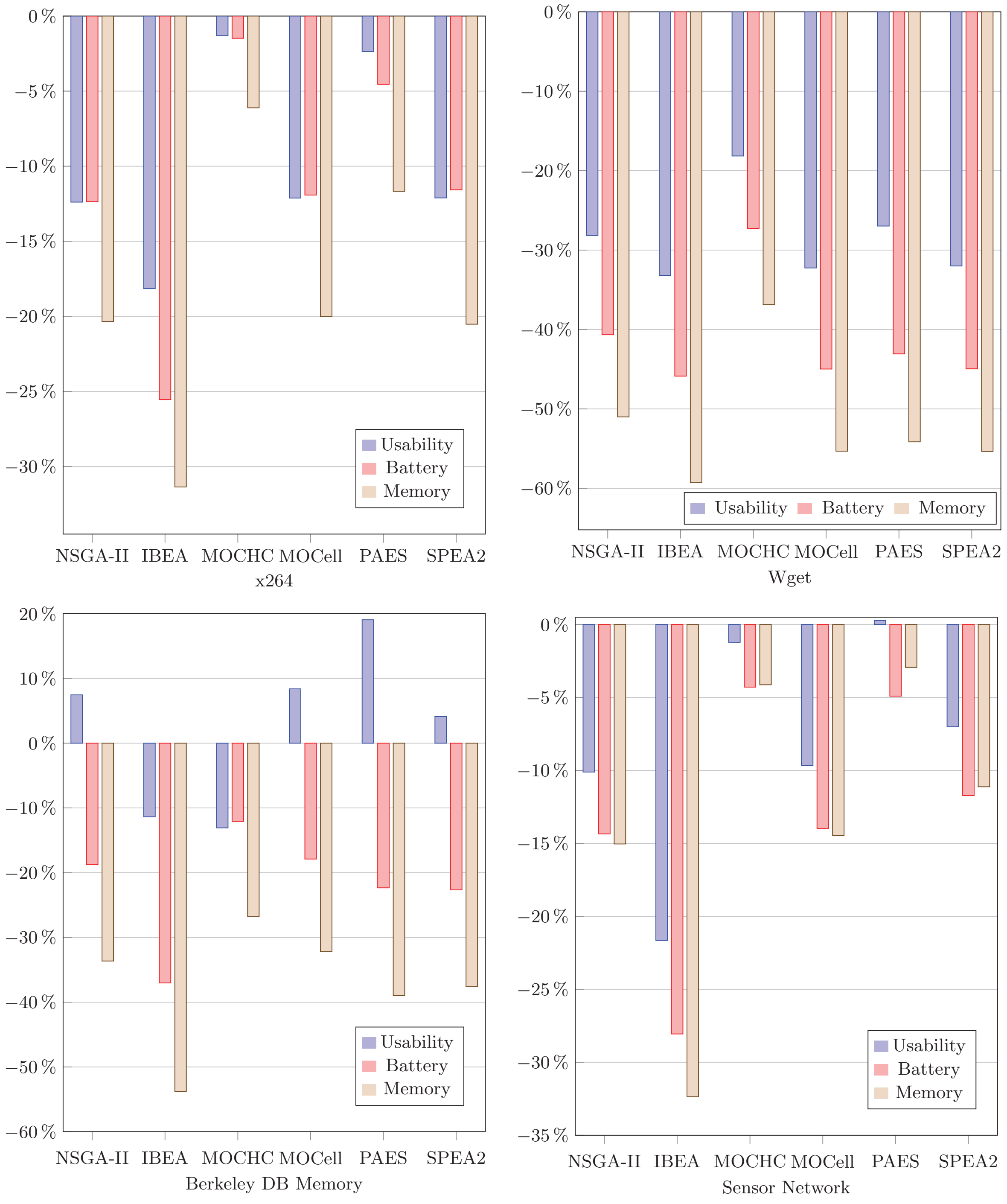


Fig. A1. Mean difference in objective values between the initial population and the final front (1/3).

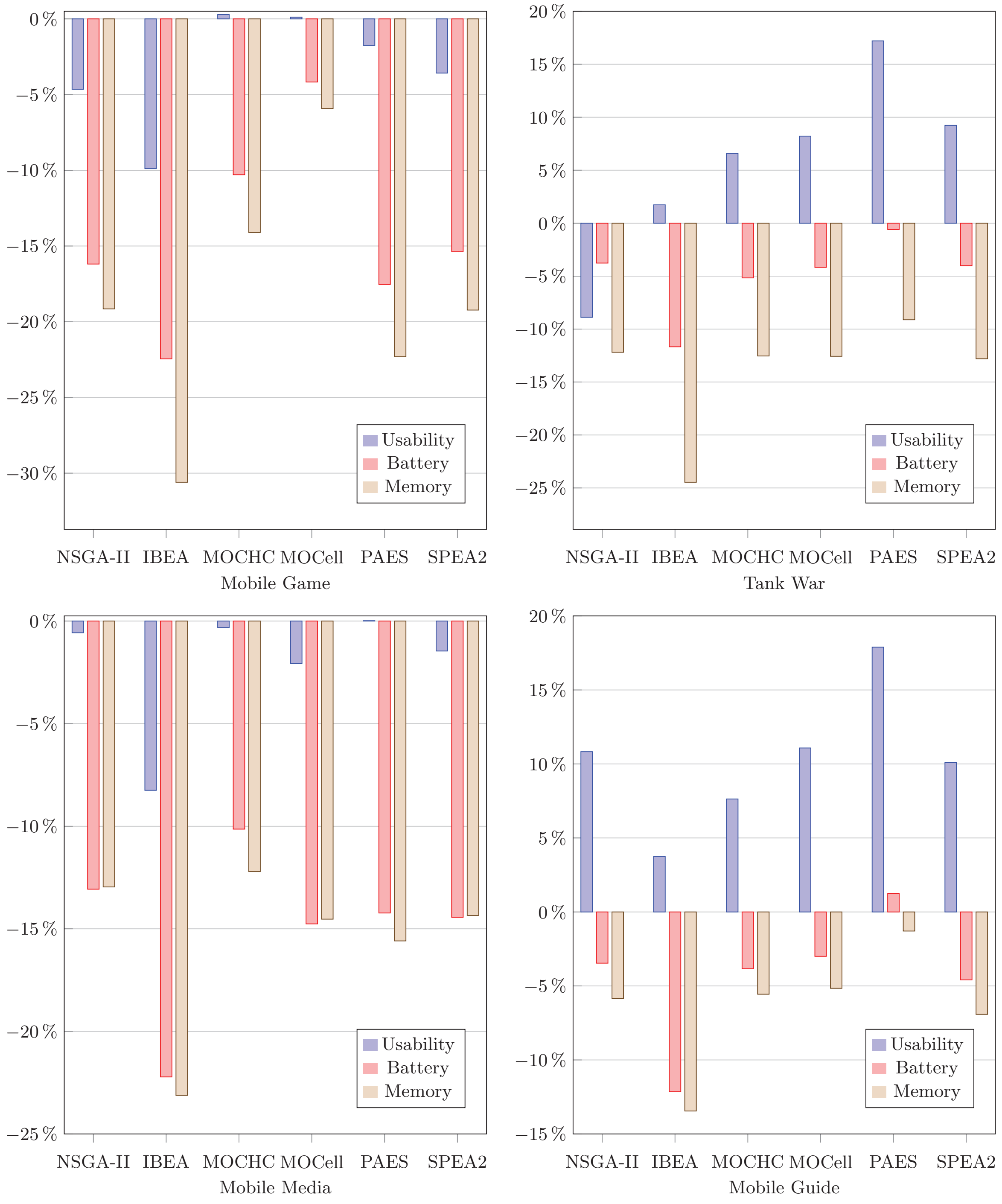


Fig. A2. Mean difference in objective values between the initial population and the final front (2/3).

same time that the battery consumption and the memory usage have decreased. For instance, in `Tank War`, PAES has increased the usability by 17.2%, decreasing the battery consumption and the memory usage by 0.6% and 9.1%, respectively.

On the other hand, in the case of the largest FMs (`SPLOT-3CNF-500`, `SPLOT-3CNF-1000`, `SPLOT-3CNF-2000` and `SPLOT-3CNF-5000`) we can see that the improvement shown in the final front with respect to the initial front is very low (objectives increase or decrease by less than 6%). In these FMs, a significant fraction of the maximum number of evaluations is spent in the generation of the initial population. The reason is that the `fix` operator may require several retries to repair an invalid FM configuration, which has been generated by applying mutations to a valid seed, in order to add it to the initial population. Therefore, these results could be improved upon by improving our `fix` operator. However, these FMs are not the most relevant ones for our approach because their number of configurations is much higher than is usually found in DSPLs for mobile devices.

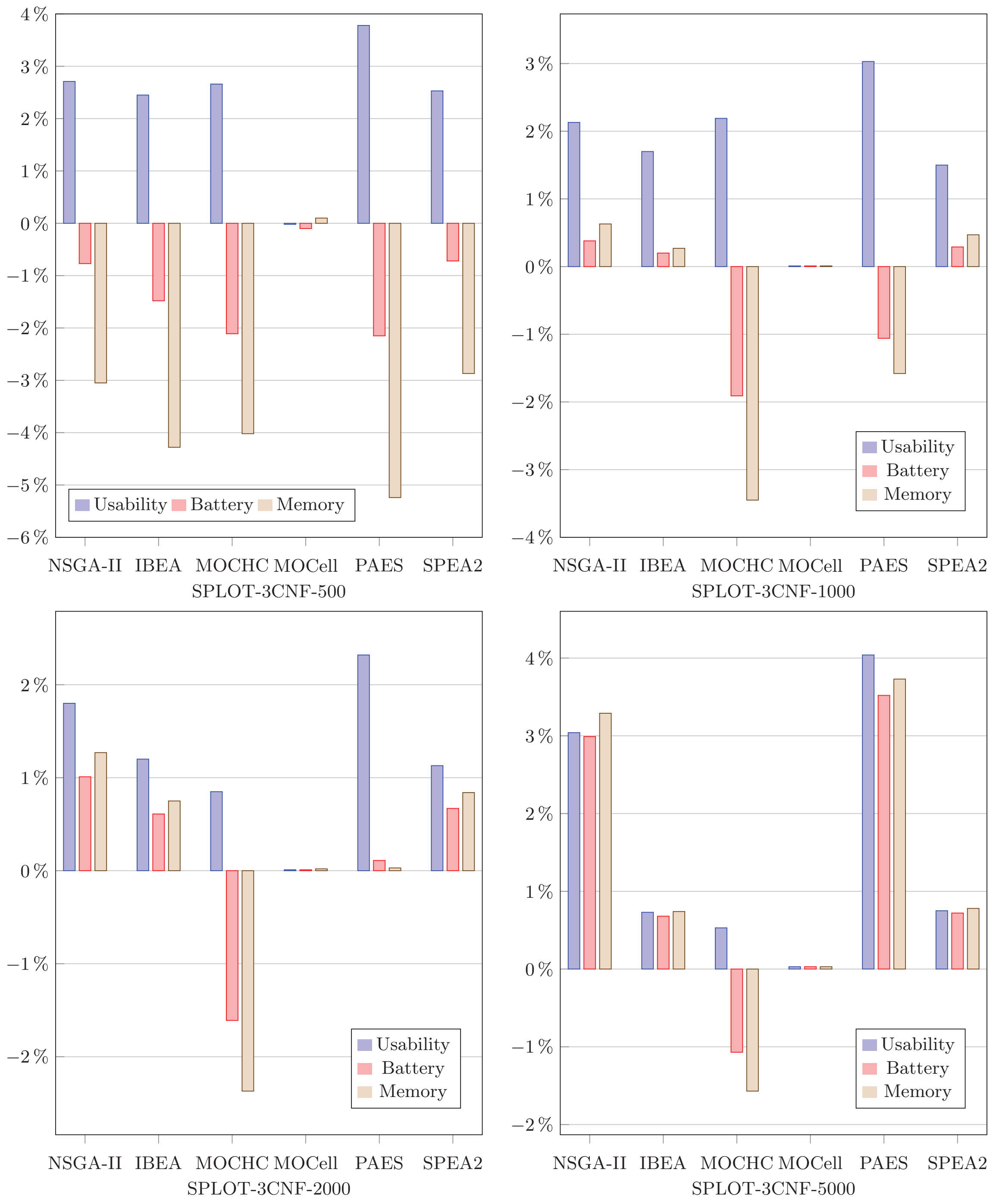


Fig. A3. Mean difference in objective values between the initial population and the final front (3/3).

References

Acher, M., Collet, P., Lahire, P., France, R., 2010. Comparing approaches to implement feature model composition. In: Modelling Foundations and Applications. LNCS, vol. 6138. Springer, Berlin, pp. 3–19.
Acher, M., Cleve, A., Collet, P., Merle, P., Duchien, L., Lahire, P., 2011. Reverse engineering architectural feature models. In: Software Architecture. Lecture Notes in Computer Science, vol. 6903. Springer, Berlin/Heidelberg, pp. 220–235.
Arcuri, A., Briand, L., 2014. A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. Softw. Test. Verif. Reliab. 24 (3), 219–250.
Arcuri, A., Fraser, G., 2013. Parameter tuning or default values? An empirical investigation in search-based software engineering. Emp. Softw. Eng. 18 (3), 594–623.
Benavides, D., Trinidad, P., Ruiz-Cortés, A., 2005. Automated reasoning on feature models. In: Advanced Information Systems Engineering. Springer, pp. 381–390.
Benavides, D., Segura, S., Ruiz-Cortés, A., 2010. Automated analysis of feature models 20 years later: A literature review. Inform. Syst. 35 (6), 615–636.
Brataas, G., Jiang, S., Reichle, R., Geihs, K., 2011. Performance property prediction supporting variability for adaptive mobile systems. In: Proceedings of the 15th International Software Product Line Conference (SPLC'11), vol. 2. ACM, New York, NY, USA, pp. 37:1–37:8
Canfora, G., Penta, M.D., Esposito, R., Villani, M.L., 2008. A framework for qos-aware binding and re-binding of composite web services. J. Syst. Softw. 81 (10), 1754–1769.
Capilla, R., Bosch, J., Trinidad, P., Ruiz-Cortés, A., Hinchey, M., 2014. An overview of dynamic software product line architectures and techniques: Observations from research and industry. J. Syst. Softw. 91, 3–23.
Carlin, A., Zilberstein, S., 2011. Decentralized monitoring of distributed anytime algorithms. In: Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems, Taipei, Taiwan, International Foundation for Autonomous Agents and Multiagents Systems, pp. 157–164.
Cetina, C., Fons, J., Pelechano, V., 2008. Applying software product lines to build autonomic pervasive systems. In: 12th International Software Product Line Conference (SPLC'08). Washington, DC, USA: IEEE, pp. 117–126.

Deb, K., Pratap, A., Agarwal, S., Meyarivan, T., 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. 6 (2), 182–197.
de Olvieira Barros, M., Neto, A., (2011). Threats to Validity in Search-baesd Software Engineering Empirical Studies. Technical Report 0006/2011, Departamento de Informatica Aplicada, Universidade Federal do Estado do Rio de Janeiro.
Diaz, D., Codognet, P., 2001. Design and implementation of the GNU Prolog system. J. Function. Logic Program. 6, 542.
Dinkelaker, T., Mitschke, R., Fetzer, K., Mezini, M., 2010. A dynamic software product line approach using aspect models at runtime. In: First International Workshop on Composition: Objects, Aspects, Components, Services and Product Lines, ACM, pp 11–18.
Djebbi, O., Salinesi, C., Diaz, D., 2007. Deriving product line requirements: The RED-PL guidance approach. In: 14th Asia-Pacific Software Engineering Conference (APSEC'07), IEEE, pp. 494–501.
Durillo, J.J., Nebro, A.J., 2011. jMetal: A java framework for multi-objective optimization. Adv. Eng. Softw. 42 (10), 760–771.
Floch, J., Frà, C., Fricke, R., Geihs, K., Wagner, M., Lorenzo, J., Soladana, E., Mehlhase, S., Paspallis, N., Rahnama, H., Ruiz, P., Scholz, U., 2013. Playing MUSIC—Building context-aware and self-adaptive mobile applications. Softw. Pract. Exper. 43 (3), 359–388.
Fraser, G., Arcuri, A., 2012. The seed is strong: Seeding strategies in search-based software testing. In: 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST), IEEE, pp. 121–130.
GoPivotal, I., 2014. AspectJ website. http://www.eclipse.org/aspectj/
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, J. Syst. Softw. 84 (12), 2208–2221.
Hallsteinsen, S., Hinchey, M., Park, S., Schmid, K., 2008. Dynamic software product lines. Computer 41 (4), 93–95.
Hansen, E.A., Zilberstein, S., Danilchenko, V.A., 1997. Anytime heuristic search: First results. Technical Report 97-50. Computer Science Department, University of Massachussetts Amherst.
Harman, M., Jia, Y., Krinke, J., Langdon, B., Petke, J., Zhang, Y., 2014. Search based software engineering for software product line engineering: A survey and directions for future work (keynote paper). In: 18th International Software Product Line Conference (SPLC'14), Florence, Italy, ACM, pp. 5–18.
Hat, R., 2010. JBoss AOP website. http://jbossaop.jboss.org/
IBM. 2005. Autonomic Computing White Paper—An Architectural Blueprint for Autonomic Computing, IBM Corp.
Kastner, C., Thum, T., Saake, G., Feigenspan, J., Leich, T., Wielgorz, F., Apel, S., 2009. FeatureIDE: A tool framework for feature-oriented software development. In: IEEE 31st International Conference on Software Engineering (ICSE'09). Washington, DC, USA: IEEE, pp. 611–614.
Knowles, J.D., Corne, D.W., 2000. Approximating the nondominated front using the pareto archived evolution strategy. Evol. Comput. 8 (2), 149–172.
Kotelysanskii, A., Kapfhammer, G.M., 2014. Parameter tuning for search-based test-data generation revisited: Support for previous results. In: Proceedings of the 14th International Conference on Quality Software (QSIC'14), Washington, DC, USA: IEEE Computer Society, pp 79–84.
Li, J., Liu, X., Wang, Y., Guo, J., 2012. Formalizing feature selection problem in software product lines using 0–1 programming. Pract. Appl. Intell. Syst. 124, 459–465.
Lotufo, R., She, S., Berger, T., Czarnecki, K., Wasowski, A., 2010. Evolution of the linux kernel variability model. In: Proceedings of the 14th International Conference on Software Product Lines: Going Beyond (SPLC'10). Springer-Verlag, Berlin/Heidelberg, pp. 136–150. http://dl.acm.org/citation.cfm?id=1885639.1885653
Mann, H.B., Whitney, D.R., 1947. On a test of whether one of two random variables is stochastically larger than the other. Annal. Math. Stat. 18, 50–60.
Matinlassi, M., 2004. Comparison of software product line architecture design methods: Copa, fast, form, kobra and qada. In: Proceedings of the 26th International Conference on Software Engineering (ICSE'04). IEEE Computer Society, Washington, DC, USA, pp. 127–136.
Mirandola, R., Potena, P., Scandurra, P., 2014. Adaptation space exploration for service-oriented applications. Sci. Comput. Program. 80, 356–384.
Mizouni, R., Matar, M.A., Mahmoud, Z.A. , Alzahmi, S., Salah, A., 2014. A framework for context-aware self-adaptive mobile applications SPL. Expert Syst. Appl, 41, 7549–7564.
Morin, B., Perrouin, G., Lahire, P., Barais, O., Vanwormhoudt, G., Jézéquel, J.-M., 2009. Weaving variability into domain metamodels. In: Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems (MODELS'09). Springer-Verlag, Berlin/Heidelberg, pp. 690–705.
Nebro, A.J., Alba, E., Molina, G., Chicano, F., Luna, F., Durillo, J.J., 2007. Optimal antenna placement using a new multi-objective chc algorithm. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO'07). ACM, New York, NY, USA, pp. 876–883.
Nebro, A.J., Durillo, J.J., Luna, F., Dorronsoro, B., Alba, E., 2009. MOCell: A cellular genetic algorithm for multiobjective optimization. Int. J. Intell. Syst. 24 (7), 726–746.
Pascual, G.G., Pinto, M., Fuentes, L., 2014. Self-adaptation of mobile systems driven by the common variability language. In: Futur. Generat. Comput. Syst., in press.
Rosenmüller, M., Siegmund, N., Pukall, M., Apel, S., 2011. Tailoring dynamic software product lines. In: Proceedings of the 10th ACM international conference on Generative programming and component engineering. New York, NY, USA: ACM, pp. 3–12.
Rouvoy, R., Barone, P., Ding, Y., Eliassen, F., Hallsteinsen, S., Lorenzo, J., Mamelli, A., Scholz, U., 2009. MUSIC: Middleware support for self-adaptation in ubiquitous and service-oriented environments. Softw. Eng. Self-Adaptive Syst. 5525, 164–182.
Sacerdoti, E.D., 1975. A Structure for Plans and Behavior (Ph.D. thesis), Stanford, CA, USA, aAI7605794.
Sarker, R., Coello Coello, C., 2002. Assessment methodologies for multiobjective evolutionary algorithms. In: International Series in Operations Research & Management Science: Vol. 48. Evolutionary Optimization. Springer, USA, pp. 177–195.
Sayyad, A., Ingram, J., Menzies, T., Ammar, H., 2013. Scalable product line configuration: A straw to break the camel's back. In: 2013 IEEE/ACM 28th International Conference on Automated Software Engineering (ASE), IEEE, pp. 465–474.
Seow, S.C., 2008. Designing and Engineering Time: The Psychology of Time Perception in Software. Boston, MA, USA: Addison-Wesley Professional.
Shen, L., Peng, X., Liu, J., Zhao, W., 2011. Towards feature-oriented variability reconfiguration in dynamic software product lines. Top Product. Softw. Reuse, 52–68.
Shi, R., Guo, J., Wang, Y., 2010. A preliminary experimental study on optimal feature selection for product derivation using knapsack approximation. In: 2010 IEEE International Conference on Progress in Informatics and Computing (PIC), vol. 1. Washington, DC, USA: IEEE, pp. 665–669.
Soltani, S., Asadi, M., Gašević, D., Hatala, M., Bagheri, E., 2012. Automated planning for feature model configuration based on functional and non-functional requirements. In: Proceedings of the 16th International Software Product Line Conference, vol. 1, New York, NY, USA: ACM, pp. 56–65.
Trinidad, P., Ruiz-Cortés, A., Pena, J., Benavides, D., 2007. Mapping feature models onto component models to build dynamic software product lines. In: Proceedings of the 11th International Software Product Line Conference (SPLC 2007), pp. 51–56.
Tsang, E., 1993. Foundations of Constraint Satisfaction, vol. 289, Academic Press, London.
Van Veldhuizen, D.A., (1999). Multiobjective evolutionary algorithms: Classifications, analyses, and new innovations (Ph.D. thesis). Air Force Institute of Technology, Wright Patterson AFB, OH, USA, AAI9928483.
Vargha, A., Delaney, H.D., 2000. A critique and improvement of the cl common language effect size statistics of mcgraw and wong. J. Educat. Behav. Stat. 25 (2), 101–132.
von Guericke University Magdeburg, O., 2013a. SPL2go website. http://spl2go.cs.ovgu.de/
von Guericke University Magdeburg, O., 2013b. SPL Conqueror website. http://wwwiti.cs.uni-magdeburg.de/~nsiegmun/SPLConqueror/
Wada, H., Suzuki, J., Yamano, Y., Oba, K., 2012. E^3: A multiobjective optimization framework for sla-aware service composition. IEEE Trans. Serv. Comput. 5 (3), 358–372.
White, J., Schmidt, D., Wuchner, E., Nechypurenko, A., 2007. Automating product-line variant selection for mobile devices. In: 11th International Software Product Line Conference (SPLC'07). Washington, DC, USA: IEEE, pp. 129–140.
White, J., Doughtery, B., Schmidt, D.C., 2008. Filtered cartesian flattening: An approximation technique for optimally selecting features while adhering to resource constraints. In: SPLC, vol. 2, IEEE, pp. 209–216.
White, J., Dougherty, B., Schmidt, D.C., 2009a. Selecting highly optimal architectural feature sets with filtered cartesian flattening. J. Syst. Softw. 82 (8) 1268–1284.
White, J., Dougherty, B., Schmidt, D., Benavides, D., 2009b. Automated reasoning for multi-step feature model configuration problems. In: Proceedings of the 13th International Software Product Line Conference. Pittsburgh, PA, USA: Carnegie Mellon University, pp. 11–20.
Yao, Y., Chen, H., 2009. Qos-aware service composition using NSGA-II 1. In: Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human, ICIS '09. ACM, New York, NY, USA, pp. 358–363.
Zitzler, E., Künzli, S., 2004. Indicator-based selection in multiobjective search. In: X. Yao, E.K. Burke, J.A. Lozano, J. Smith, J.J. Merelo-Guervós, J.A. Bullinaria, J.E. Rowe, P. Tino, A. Kabán, H.-P. Schwefel (Eds.), Parallel Problem Solving from Nature (PPSN) VIII. Lecture Notes in Computer Science, vol. 3242. Springer, Berlin Heidelberg, pp. 832–842.
Zitzler, E., Thiele, L., 1999. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. IEEE Trans. Evol. Comput. 3 (4), 257–271.
Zitzler, E., Laumanns, M., Thiele, L., 2001. SPEA2: Improving the strength pareto evolutionary algorithm.

Gustavo G. Pascual is a Ph.D. Student in the Languages and Computer Science Department at the University of Málaga (Spain). He received the B.Sc. degree in Telecommunications Engineering from the University of Málaga. He is a member of the CAOSD Group and the GISUM Group. His main research areas are component-based software engineering, aspect-oriented software development, dynamic software product lines and self-adaptive middleware platforms.

Roberto E. Lopez-Herrejon is currently a senior postdoctoral researcher at the Johannes Kepler University in Linz, Austria. He was Lise Meitner Fellow (2012–2014) at the same institution. From 2008 to 2014 he was an External Lecturer at the Software Engineering Masters Programme of the University of Oxford, England. From 2010 to 2012 he held an FP7 Intra-European Marie Curie Fellowship on a project for consistency and composition of variable systems with multi-view models. He obtained his Ph.D. degree from the University of Texas at Austin in 2006, funded in part by a Fulbright Fellowship sponsored by the U.S. State Department. From 2005 to 2008, he was a Career Development Fellow at the Software Engineering Centre of the University of Oxford sponsored by Higher Education Founding Council of England (HEFCE). His expertise is software product lines, variability management, feature oriented software development, model driven software engineering, and consistency checking.

Mónica Pinto is an associate professor in the Languages and Computer Science Department at the University of Málaga (Spain). She received her M.Sc. degree in Computer Science in 1998 from the University of Málaga, and her Ph.D degree in 2004 from the same university. Her main research areas are component-based software engineering, aspect-oriented software development, software product lines and dynamic reconfiguration of mobile applications. In the last years she has co-organised the Early Aspects workshop at ICSE, and has been member of the program committee of several workshops and conferences on AOSD and software composition. She has been publicity co-chair at AOSD 2011 and AOSD 2012. She currently participates in several national and international projects on AOSD, software product lines and dynamic reconfiguration. She also participates in the INTER-TRUST STREP project that applies AOSD techniques to dynamically deploy and adapt security policies at runtime.

Lidia Fuentes is head of the CAOSD research group (http://caosd.lcc.uma.es) and professor of computer science at the University of Málaga. Her research interest mainly deals with self-adaptation of future Internet applications, software architectures, software product lines, agent-oriented software engineering and aspect-oriented software development. Her scientific production is very prolific, with more than 150 scientific publications at international forums. Her most significant publications can be found in international journals, such as IEEE Internet Computing, IEEE Transactions on Software Engineering, ACM Computing Surveys or Information and Software Technology journal. Her work has currently around two thousand citations. She has served on the programme committee of multiple conferences such as AOSD, OOPSLA or GPCE. She is actively participating in several European research projects about AOSD, MDD and SPLs (e.g. AOSD-Europe, AMPLE, etc.).

Alexander Egyed is a full professor at the Johannes Kepler University (JKU), Austria. He received his Doctorate degree from the University of Southern California, USA and worked for Teknowledge Corporation, USA (2000–2007) and the University College London, UK (2007–2008). He is most recognized for his work on software and systems modeling—particularly on consistency and traceability of models. Dr. Egyed's work has been published at over a hundred refereed scientific books, journals, conferences, and workshops, with over 3000 citations to date. He was recognized as the 10th best scholar in software engineering in Communications of the ACM, was named an IBM Research Faculty Fellow in recognition to his contributions to consistency checking, received a Recognition of Service Award from the ACM, a Best Paper Award from COMPSAC, and an Outstanding Achievement Award from the USC. He has given many invited talks including four keynotes, served on scientific panels and countless program committees, and has served as program (co-) chair, steering committee member, and editorial board member. He is a member of the IEEE, IEEE Computer Society, ACM, and ACM SigSoft.