Empir Software Eng (2011) 16:29–60
DOI 10.1007/s10664-010-9147-3

A study of the bi-objective next release problem

Juan J. Durillo · Yuanyuan Zhang · Enrique Alba · Mark Harman · Antonio J. Nebro

Published online: 17 December 2010

Editors: Simon Poulding and Massimiliano Di Penta

Abstract One important issue addressed by software companies is to determine which features should be included in the next release of their products, in such a way that the highest possible number of customers get satisfied while entailing the minimum cost for the company. This problem is known as the Next Release Problem (NRP). Since minimizing the total cost of including new features into a software package and maximizing the total satisfaction of customers are contradictory objectives, the problem has a multi-objective nature. In this work, we apply three state-of-the-art multi-objective metaheuristics (two genetic algorithms, NSGA-II and MOCell, and one evolutionary strategy, PAES) for solving NRP. Our goal is twofold: on the one hand, we are interested in analyzing the results obtained by these metaheuristics over a benchmark composed of six academic problems plus a real world data set provided by Motorola; on the other hand, we want to provide insight about the solution to the problem. The obtained results show three different kinds of conclusions: NSGA-II is the technique computing the highest number of optimal

J. J. Durillo (✉) · E. Alba · A. J. Nebro
Computer Science Department, University of Málaga, Málaga, Spain
e-mail: durillo@lcc.uma.es

E. Alba
e-mail: eat@lcc.uma.es

A. J. Nebro
e-mail: antonio@lcc.uma.es

Y. Zhang · M. Harman
CREST Centre, University College London, London, UK

Y. Zhang
e-mail: yuanyuan.zhang@cs.ucl.ac.uk

M. Harman
e-mail: m.harman@cs.ucl.ac.uk

solutions, MOCell provides the product manager with the widest range of different solutions, and PAES is the fastest technique (but with the least accurate results). Furthermore, we have observed that the best solutions found so far are composed of a high percentage of low-cost requirements and of those requirements that produce the largest satisfaction on the customers as well.

Keywords Search based software engineering · Multi-objective optimization · Next release · Requirements engineering

1 Introduction

Traditionally, Search Based Software Engineering (SBSE) has been widely used in Software Testing (Afzal et al. 2009; Harman et al. 2009; Harman 2007; McMinn 2004) and comparatively less so in other fields of Software Engineering. However, there is evidence that SBSE techniques are starting to find their way into all aspects of software engineering activity from the earliest phases of the software development lifecycle concerned with requirements, management and planning right through to the post delivery phases of maintenance and re-engieeering (Harman et al. 2009). This paper focuses on one of these early lifecycle problems, centred on requirements analysis.

In more traditionally considered areas of SBSE, the goal has tended to be one of finding the optimal or near optimal solution to the problem in hand, with respect to a single objective. For example, the very first work on SBSE (Korel 1990; Miller and Spooner 1976; Xanthakis et al. 1992) concerned itself with finding optimal or near optimal test sets.

However, more recently it has been realized that SBSE can also be used as a tool for decision support, using multi-objective approaches. In this mode, the search based approach can be used to provide insight to the Software Engineer, allowing him or her to explore the possible space of candidate solutions with certain properties, revealing structure in the solution space and potential points of attractive trade off. For example, recent work has considered multi-objective formulations of problems in testing (Del Grosso et al. 2005; Everson and Fieldsend 2006; Lakhotia et al. 2007; Walcott et al. 2006; Yoo and Harman 2007), quality assurance (Khoshgoftaar et al. 2004), refactoring (Harman and Tratt 2007) and project management (Alba and Chicano 2007) as well as requirements engineering (Finkelstein et al. 2008; Saliu and Ruhe 2007; Zhang et al. 2007).

This focus on decision support has been technically underpinned by the reformulation of many problems in SBSE as multi-objective problems, to which a Pareto optimal approach can be readily applied. In Pareto optimal approaches, the outcome of the search is not a single (optimal or near optimal solution). It is a *set of candidate solutions*, each of which cannot be improved upon according to one of the multiple objectives to be optimized without a negative impact on another. This set of solutions is called a 'non-dominated' set of solutions, because each is incomparable; no one solution dominates any other in terms of meeting the multiple objectives. In the Pareto optimal approach, all objectives are considered incomparable, so that weighting the different objectives in order to combine them into a single weighted sum objective, is impossible.

Any problem involving some form of cost–benefit analysis can be thought of as a canonical instance of the general class of problems for which a Pareto optimal approach is attractive. In any cost–benefit analysis, it will be hard to determine to what degree a decision maker can yield up a perceived benefit in order to reduce cost. Likewise, such a decision maker will not be able to readily decide, a priori, how much cost increase they would be prepared to tolerate for a commensurate increase in benefit. Cost and benefit simply are not like that; they require subjective human judgements to be made and depend on circumstances.

In such a potentially vague scenario space, in which the exercise of human judgement remains paramount, it may, at first, be difficult to see how one might successfully apply a search based technique. However, the key lies in the manner in which a Pareto approach presents a Pareto front of non-dominated solutions, each of which denotes a cost–benefit pair for which no other pair can be found which improves upon *both* cost and benefit. The shape of this Pareto front gives insight to the decision maker. Though he or she may not be able to determine, a priori, the trade off they are prepared to accept between these two incompatible objectives, the shape of the front can, *a posteriori* direct them to points at which the trade off is most attractive. In this decision support is provided for what has been termed the Next Release Problem (NRP).

The problem here is *not* to tell the product manager, who is the decision maker in this case, what requirement should be in the next release of the software; no self-respecting software engineer would entirely trust and rely upon an automated tool to make such a decision. Rather, the problem is to provide decision support, to help the manager to identify those solutions that best balance the competing concerns of cost and benefit. The approach we adopt is a Pareto optimal one, in which the product manager supplies the tool with estimates of cost and assessments of benefit and the automated part of the analysis uses SBSE to search for a good Pareto front. A 'good' Pareto front is one which provides accuracy and diversity, as will be explained more formally later.

However, there are a number of different algorithms that one may choose to apply to provide pareto fronts, each with their own characteristics of performance and quality. Naturally, the value of any NRP approach is entirely dependent upon these characteristics. That is, the quality of decision support provided to the product manager is only as good as the quality of the results produced by the algorithm, and its usability is affected by the time taken to provide such results. The increasing interest in the problem and in multi objective formulations, indicates that an in-depth experimental and empirical study of different algorithm performances may add value to the available literature on the NRP.

In this paper we seek to explore a set of algorithms available, both experimentally and empirically. We have chosen a set of algorithms that is widely used on related multi objective problems. The experimental aspect is concerned with a laboratory controlled set of problem instances. This experimentation allows us to investigate the degree to which each algorithm performs under variations in problem characteristics (notably size of problem instance). The empirical study is concerned with the performance of each algorithm on a real world case study. Obtaining real world quantitative data on requirements is a challenge. We presently have available to us, only one such data set, kindly provided by Motorola. Therefore, our empirical results are restricted to a case study in the present paper. In future work we hope to

obtain additional real world quantitative requirements data sets on which to perform a more complete empirical study to augment the experimental study which forms the primary contribution of the present paper.

The remainder of this work is structured as follows: The next section contains some background about multi-objective optimization. Section 3 presents the Next Release Problem formally. The algorithms used in this work are described in Section 4. Section 5 is devoted to experimentation. We describe the obtained results in Section 6, and we study the obtained solutions from the point of view of NRP in Section 7. In Section 8, we describe and analyze the results obtained using a real world instance of the problem. Section 9 presents related work. Finally, Section 10 draws the main conclusions and lines of future work.

2 Multi-Objective Background

In this section, we provide the definition of some concepts for a better understanding of this work. In particular, we define the concept of multi-objective optimization problem (MOP), Pareto dominance and Pareto front. In these definitions we are assuming, without loss of generality, that minimization is the goal for all the objectives.

A general MOP can be formally defined as follows: find a vector $\mathbf{x}^* = [x_1^*, x_2^*, \ldots, x_n^*]$ which satisfies the m inequality constraints $g_i(\mathbf{x}) \geq 0, i = 1, 2, \ldots, m$, the p equality constraints $h_i(\mathbf{x}) = 0, i = 1, 2, \ldots, p$, and minimizes the vector function $\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})]^T$, where $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ is the vector of decision variables.

The set of all values satisfying the constraints defines the *feasible region* Ω and any point $\mathbf{x} \in \Omega$ is a *feasible solution*.

Taking into account this definition of a MOP, a solution $x = [x_1, x_2, ..., x_n]$ is said to dominate a solution $y = [y_1, y_2, ..., y_n]$ if and only if $f_i(x) <= f_i(y)$ for $i = 1, 2, ..., m$, and there exist at least one j $(1 \leq j \leq m)$ such that $f_i(x) < f_i(y)$. Conversely, two points are said to be non-dominated whenever none of them dominates the other. Figure 1 depicts some examples of dominated and non-dominated solutions. In this figure, A' dominates to C' because $f_1(A') < f_1(C')$, and $f_2(A') < f_2(C')$. Meanwhile, A' and B' are non-dominated because A' is better than B' in

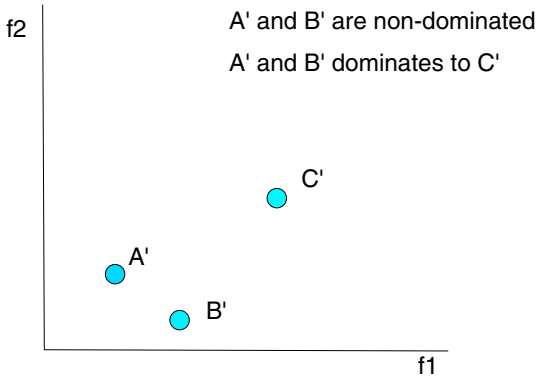


Fig. 1 Examples of dominated and non-dominated solutions

the first objective function ($f_1(A') < f_1(B')$), but B' is better in the other objective function ($f_2(A') > f_2(B')$).

The solution of a given MOP is usually a set of solutions (referred as Pareto optimal set) satisfying:

- Every two solutions into the set are non-dominated.
- Any other solution, y, is dominated by at least one solution in the set.

The representation of this set in the objective space is referred as *Pareto front*. Generating *Pareto front* is the main goal of multi-objective optimization techniques.

In theory, a Pareto front could contain a large number (or even infinitely many) points. In practice, a usable approximate solution will only contain a limited number of them; thus, an important goal is that they should be as close as possible to the exact Pareto front and uniformly spread, otherwise, they would not be very useful to the decision maker. Closeness to the Pareto front ensures that we are dealing with optimal solutions, while a uniform spread of the solutions means that we have made a good exploration of the search space and no regions are left unexplored.

Figure 2 depicts these issues of convergence and diversity. The uppermost front depicts an example of good convergence and bad diversity: the approximation set contains Pareto optimal solutions but there are some unexplored regions of the optimal front. The approximation set depicted in the middle illustrates poor convergence but good diversity: it has a diverse set of solutions but they are not

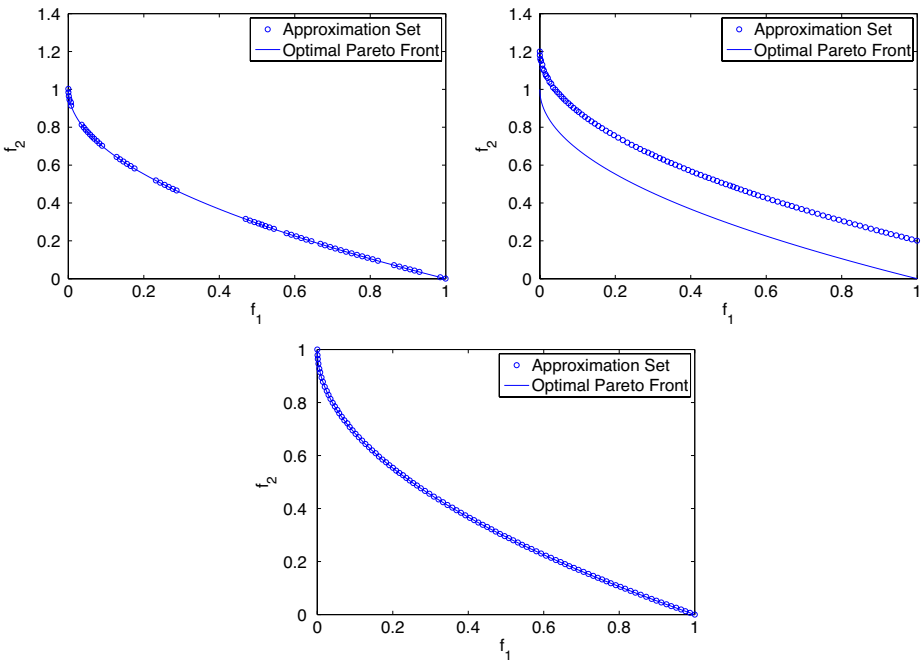


Fig. 2 Examples of Pareto fronts. From *top to bottom*, from *left to right*: **a** good convergence and bad diversity, **b** bad convergence and good diversity, and **c** good convergence and diversity

Pareto optimal. Finally, the lowermost front depicts an approximation front with both good convergence and diversity.

3 Problem Statement

This section formalizes Multi-Objective Next Release Problem.

Given a software package, there is a set, C, of m customers whose requirements have to be considered in the development of the next release of this system. Each customer has associated a value, c_i, which reflects the customers' degree of importance to the software development company.

There is also a set, R, of n requirements to complete. In order to meet each requirement, resources must be spent, which can be transformed into an economical cost: the cost of satisfying the requirement. We denote this as r_j, $(1 \leq j \leq n)$ the economical cost of achieving the requirement j.

We also assume that more than one customer can be concerned with any requirement, and that all the requirements are not equally important for all the customers. In this way, associated with each customer and each requirement, there is a value v_{ij}, which represents the importance of the requirement j for the customer i. All these values can be represented by a matrix.

The MONRP problem is to find a subset, R', of requirements which minimizes the cost and maximizes the total satisfaction of the customers with the finally included requirements. Thus the multi-objective Next Release Problem can be formalized as

$$\text{minimize } f_1 = \sum_{r_i \in R'} r_i \tag{1}$$

$$\text{maximize } f_2 = \sum_{i=1}^{m} c_i \sum_{r_j \in R'} v_{ij}. \tag{2}$$

Since minimizing a given function f is the same as maximizing $(-f)$, in this work we have considered the maximization of $(-f1)$ (i.e., the economical cost for companies), and $f2$ (i.e, the customer satisfaction).

The advantages of considering NRP to be a multi-objective optimization problem, instead of a weighted single objective one, can be drawn from Fig. 3, which shows an example of front obtained for NRP. In this figure "Customer Satisfaction" means

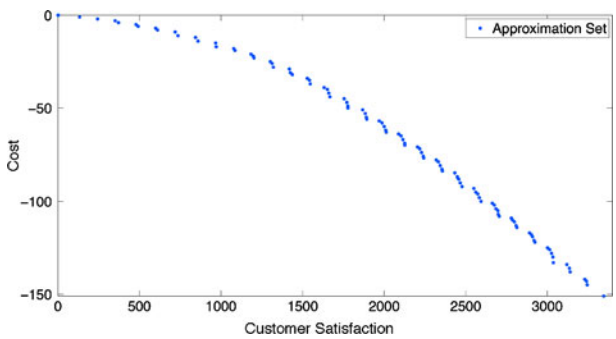


Fig. 3 Examples of Pareto front for NRP

the total satisfaction of the customers, and "Cost" represents the economical cost of a requirement set. (Note that a negative cost means an investment made by the company).

4 Solver Algorithms

In this section we describe the three algorithms which will be evaluated for solving NRP.

NSGA-II, proposed by Deb et al. (2002), is a genetic algorithm which is the 'reference algorithm' in multi-objective optimization (with over 2,500 citations at the time of writing[1]). Its pseudocode is presented as Algorithm 1. NSGA-II makes use of a population (P) of candidate solutions (known as individuals). In each generation, it works by creating new individuals after applying the genetic operators to P, in order to create a new population, Q (lines 5 to 8). Then, both the current (P) and the new population (Q) are joined; the resulting population, R, is ordered according to a ranking procedure and a density estimator known as crowding distance (line 13) (for further details, please see Deb et al. 2002). Finally, the population P is updated with the best individuals in R (line 14). These steps are repeated until a termination condition is fulfilled.

Algorithm 1 Pseudocode of NSGA-II.

```
 1: proc Steps_Up(nsga-II)       //Algorithm parameters in 'nsga-II'
 2: P ← Initialize_Population() // P = population
 3: Q ← ∅                        // Q = auxiliar population
 4: while not Termination_Condition() do
 5:    for i ← 1 to (nsga-II.popSize / 2) do
 6:       parents←Selection(P);
 7:       offspring←Recombination(nsga-II.Pc,parents);
 8:       offspring←Mutation(nsga-II.Pm,offspring);
 9:       Evaluate_Fitness(offspring);
10:      Insert(offspring,Q);
11:    end for
12:    R ← P ∪ Q
13:    Ranking_And_Crowding(nsga-II, R);
14:    P ←Select_Best_Individuals(nsga-II, R)
15: end while
16: end_proc Steps_Up;
```

MOCell (Multi-Objective Cellular Genetic Algorithm), introduced by Nebro et al. (2009), is a cellular genetic algorithm (cGA) which has proven to outperform NSGA-II in some studies (Nebro et al. 2007, 2009). In cGAs, the concept of (small) *neighborhood* is paramount. This means that an individual may only cooperate

[1]Data from Google Scholar: 2,616 citations on 20th September 2009.

with its nearby neighbors in the breeding loop. Overlapped small neighborhoods of cGAs help in exploring the search space because they induce a slow diffusion of solutions through the population, providing a kind of exploration (diversification). Exploitation (intensification) takes place inside each neighborhood by applying the typical genetic operations (crossover, mutation, and replacement).

MOCell includes an external archive to store the non-dominated solutions found as the algorithm progresses. This archive is limited in size and uses the crowding distance of NSGA-II to maintain diversity. The pseudocode of MOCell is presented as Algorithm 2, which corresponds with the version called aMOCell4, described in Nebro et al. (2007).

Algorithm 2 Pseudocode of MOCell.

```
1:  proc Steps_Up(mocell)        //Algorithm parameters in 'mocell'
2:  archive ← ∅ //Creates an empty archive
3:  while not Termination_Condition() do
4:    for individual ← 1 to mocell.popSize do
5:      n_list←Get_Neighborhood(mocell,position(individual));
6:      parent1←Selection(n_list);
7:      parent2←Selection(archive);
8:      offspring←Recombination(mocell.Pc,parent1, parent2);
9:      offspring←Mutation(mocell.Pm,offspring);
10:     Evaluate_Fitness(offspring);
11:     Replacement(position(individual),offspring,mocell);
12:     Insert_Pareto_Front(offspring, archive);
13:   end for
14: end while
15: end_proc Steps_Up;
```

We can observe that, in this version, for each individual we select one parent from its neighborhood and one from the archive, in order to guide the search towards the best solutions found (lines 5 to 8). Then a new solution is created by applying the genetic operators to these parents. The new solution is used to replace the current solution (line 11), and it is considered for inclusion in the archive (line 12). This constitutes a single iteration of the algorithm. The overall algorithm iterates until a termination condition is fulfilled.

PAES is a metaheuristic proposed by Knowles and Corne (1999). The algorithm is based on a simple (1+1) evolution strategy. To find diverse solutions in the Pareto optimal set, PAES uses an external archive of nondominated solutions, which is also used to make decisions about new candidate solutions (Knowles and Corne 1999). An adaptive grid is used as a density estimator in the archive. The most remarkable characteristic of PAES is that it does not make use of any recombination operators (crossover). New solutions are generated only by modifying the current solution. The pseudocode of PAES is presented as Algorithm 3. It commences with a random solution (line 3). In each iteration, a new solution is produced by modifying the current solution (line 5). This new solution is included in the archive and it is considered as a potential replacement for the current solution (lines 7 to 14). These steps are repeated until the maximum number of evaluations is reached.

Algorithm 3 Pseudocode of PAES.

```
1: proc Steps_Up(paes)       //Algorithm parameters in 'paes'
2: archive ← ∅
3: currentSolution ← Create_Solution(paes) // Creates an initial solution
4: while not Termination_Condition() do
5:    mutatedSolution←Mutation(currentSolution);
6:    Evaluate_Fitness(mutatedSolution);
7:    if IsDominated(currentSolution, mutationSolution) then
8:       currentSolution ← mutatedSolution
9:    else
10:      if Solutions_Are_Nondominated(currentSolution, mutationSolution) then
11:         Insert(archive, mutatedSolution)
12:         currentSolution ← Select(paes, archive)
13:      end if
14:   end if
15: end while
16: end_proc Steps_Up;
```

We have included PAES in our study because of its simplicity. PAES does not use any recombination operator, and its only parameter is the number of partitions of the adaptive grid of the archive. Its relative simplicity makes it attractive since there are comparatively few parameters that require tuning in order to know that the algorithm is being applied properly (e.g., population size, crossover probability, mutation probability).

5 Experimental Method

This section is aimed at presenting the indicators used to measure the quality of the obtained results and the benchmark problems we have used. It also describes how the solutions of the problem have been encoded and the genetic operators employed, the configuration of the algorithms, and the methodology we have followed.

5.1 Quality Indicators

Two different issues are normally taken into account for assessing the quality of the results computed by a multi-objective optimization algorithm:

1. To minimize the distance of the computed solution set by the proposed algorithm to the optimal Pareto front (convergence towards the optimal Pareto front).
2. To maximize the spread of solutions found, so that we can have a distribution of vectors as smooth and uniform as possible (diversity).

A number of quality indicators have been proposed in the literature. Among them, we can distinguish between *Pareto compliant* and *non Pareto compliant* indicators (Knowles et al. 2006). Given two Pareto fronts, A and B, if A dominates B, the value of a Pareto compliant quality indicator is higher for A than for B;

meanwhile, this condition is not fulfilled by the non-compliant indicators. Thus, the use of Pareto compliant indicators should be preferable; however, non Pareto compliant indicators can also be used for measuring some particular features of a front. In this work, we apply an indicator of each type: Hypervolume (Zitzler and Thiele 1999) (Pareto compliant), which takes into account the convergence as well as the diversity of the solutions; and Spread (Deb 2001) (non Pareto compliant), which measures the distribution of solutions into a given front. For further details about the formulation of these indicators, please refers to Zitzler and Thiele (1999) and Deb (2001).

To apply these quality indicators, it is usually necessary to know the optimal Pareto front. Of course, typically, we do not know the location of the optimal front. Therefore, we employ as a 'reference Pareto optimal front' the front composed of all the non-dominated solutions out of all the executions carried out (i.e., the best front known for these problems until now). We also consider the number of solutions that are non-dominated with respect to all the solutions computed by all the algorithms.

5.2 Test Problems

In this section, we describe the test problems used to evaluate the performance of NSGA-II, MOCell, and PAES.

The three algorithms were applied to a set composed of six test problems. These problems have been aimed at covering both 'typical' and non 'typical' cases of NRP. Thus, we have generated instances ranging from 2 to 100 customers, and from 20 to 200 requirements. All the values related to each instance have been generated by sampling a random uniform distribution. We have not considered dependencies among requirements.

All the instances have the nomenclature *c_r*, where *c* is the number of customers, and *r* the maximum number of requirements. Specifically, we have considered here the same instances proposed by Zhang et al. (2007): 15_40, 50_80, 2_200, 100_20, 100_25, and 100_40.

5.3 Solution Encoding and Genetic Operators

As described in Section 3, a solution to the problem consists in selecting a subset of requirements to be included in the next release of the software package. In this work, each solution is encoded as a binary string, *s*, of length *n* (the maximum number of requirements), where $s_i = 1$ means that the requirement *i* is included in the next release of the software.

As to the genetic operators, we have used *binary tournament* as the selection scheme. This operator works by randomly choosing two individuals from the population and the one dominating the other is selected; if both solutions are non-dominated one of them is selected randomly. We also use *single point crossover* as crossover operator. It works by creating a new solution in which the binary string from the beginning of a parent solution to a crossover point, randomly chosen, is copied from that parent while the rest is copied from other parent. Finally, the mutation operator used is *random mutation* using which some random bits of the string are flipped.

5.4 Configuration

All approaches were run for a maximum of 25,000 function evaluations, and the results are analyzed when 5,000, 10,000, and 25,000 evaluations have been performed.

The initial population was set to 100 in NSGA-II and MOCell. In MOCell, the archive size was also limited to 100 solutions. In both algorithms the probability of the crossover operator was set to $P_c = 0.9$ and the probability of the mutation operator to $P_m = 1/n$, being n the number or requirements. In PAES, the maximum size of the archive was also set to 100, and the number of divisions of the adaptive grid to 5.

All the algorithms have been implemented using jMetal (Durillo et al. 2006), a Java framework aimed at the development, experimentation, and study of metaheuristics for solving multi-objective optimization problems.

5.5 Methodology

We have executed 100 independent runs for each algorithm and each problem instance. Since we are dealing with stochastic algorithms, we need to perform a statistical analysis of the obtained results to compare them with a certain level of confidence. Next, we describe the statistical test that we have carried out for assuring this. First, a Kolmogorov–Smirnov test is performed in order to check whether the values of the results follow a normal (Gaussian) distribution. If so, the Levene test checks for the homogeneity of the variances. If samples have equal variance (positive Levene test), an ANOVA test is performed; otherwise we perform a Welch test. For non-Gaussian distributions, the non-parametric Kruskal–Wallis test is used to compare the medians of the algorithms. All the tables include the mean and standard deviation of the evaluated indicator.

We always consider in this work a confidence level of 95% (i.e., significance level of 5% or p-value under 0.05) in the statistical tests, which means that the differences are unlikely to have occurred by chance with a probability of 95%. Those tests in which the statistical confidence has been achieved are marked with "+" symbols in the last row in the tables containing the results; conversely, "−" means that we cannot assure anything about the statistical confidence of the results (p-value > 0.05).

For the sake of a better visual comprehension, the best result for each problem is depicted with a grey background.

6 Experimental Analysis

In this section we present the obtained results by the three evaluated algorithms. We start by describing the values of the HV and Δ, the two quality indicators used. Then we consider how many of the computed solutions are among the best solutions found so far. Finally, we have also included the running time of the algorithms.

6.1 Hypervolume Results

Tables 1, 2 and 3 contain the mean and standard deviation for the HV indicator when 5,000, 15,000, and 25,000 function evaluations have been performed, respectively. For this indicator, the higher the value, the better the quality of the obtained results.

Table 1 Mean ($\bar{x}$) and standard deviation (σ) of the results of the HV quality indicator after 5,000 evaluations

	NSGA-II		MOCell		PAES		
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	
15_40	0.655	0.0038	0.654	0.0033	0.633	0.0150	+
50_80	0.562	0.0071	0.559	0.0065	0.524	0.0140	+
2_200	0.461	0.0100	0.458	0.0100	0.417	0.0160	+
100_20	0.612	0.0009	0.612	0.0005	0.604	0.0066	+
100_25	0.631	0.0024	0.630	0.0018	0.613	0.0110	+
100_140	0.501	0.0092	0.496	0.0077	0.446*e*	0.0180	+

Thus, by looking at the tables, we can see that NSGA-II has been the algorithm computing the best results regarding to this indicator when only 5,000 function evaluations have been performed. However, when the number of evaluations increase, the differences between it and MOCell reduce. After 25,000 evaluations, MOCell outperforms NSGA-II for half of the problem instances. PAES is the worst algorithm according to this comparison. In all cases, the difference in the performance of the best algorithm and that of the others is statistically significant.

As described in Section 5.1, the HV indicator measures the non-dominated area covered by a front; thus, the higher the HV value, the larger this area, and, hence, the number of solutions dominated by it. Obviously, the optimal Pareto front has the highest HV value, and the fronts computed by an algorithm should converge towards that value. If we analyze the HV obtained by NSGA-II and MOCell when 10,000 and 25,000 evaluations have been performed, we observe that the differences of the HV value are smaller in the instances with 40 or fewer requirements than in the instances with more requirements. This means that in the first group of instances both algorithms have converged towards an optimal (local or global) Pareto front of the problem. Meanwhile, in the second group, it is still possible to improve the computed fronts.

Figure 4 clarifies this point. It shows the evolution of the HV of the approximated fronts computed by NSGA-II over the number of evaluations carried out in the instances 100_20 and 2_200. In this figure, we can observe that, in the instance with only 20 requirements, the HV indicator has converged towards a fixed value when approximately 4,500 evaluations have been carried out. On the other hand, in the 2_200 instance, the HV indicator increases with the number of evaluations and has not converged towards a fixed value in 25,000 function evaluations; therefore it is possible to compute better fronts by performing a higher number of evaluations. This observation suggests that the instances with more requirements need a higher

Table 2 Mean ($\bar{x}$) and standard deviation (σ) of the results of the HV quality indicator after 10,000 evaluations

	NSGA-II		MOCell		PAES		
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	
15_40	0.663	0.0016	0.663	0.0011	0.645	0.0160	+
50_80	0.588	0.0049	0.587	0.0041	0.540	0.0150	+
2_200	0.522	0.0076	0.513	0.0068	0.450	0.0100	+
100_20	0.613	0.0002	0.613	0.0003	0.605	0.0068	+
100_25	0.635	0.0007	0.635	0.0004	0.618	0.0110	+
100_140	0.541	0.0071	0.534	0.0052	0.469	0.0140	+

Table 3 Mean ($\bar{x}$) and standard deviation (σ) of the results of the HV quality indicator after 25,000 evaluations

	NSGA-II		MOCell		PAES		
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	
15_40	0.665	0.0004	0.666	0.0001	0.651	0.0170	+
50_80	0.604	0.0014	0.604	0.0012	0.560	0.0100	+
2_200	0.578	0.0033	0.568	0.0038	0.483	0.0120	+
100_20	0.613	0.00004	0.613	0.00005	0.608	0.0056	+
100_25	0.636	0.0001	0.636	0.00009	0.623	0.0110	+
100_140	0.573	0.0032	0.568	0.0031	0.495	0.0130	+

number of evaluations than the smaller instances, in order to converge towards the optimal Pareto front.

Thus, in summarizing all these results, some conclusions regarding the HV indicator emerge:

- NSGA-II has been the overall best algorithm.
- NSGA-II has been the fastest algorithm in obtaining an accurate set of solutions (it has obtained the best values taking into account only 5,000 function evaluations).
- When the number of solutions increases, the differences between NSGA-II and MOCell reduce, and MOCell matches the effectiveness of NSGA-II.

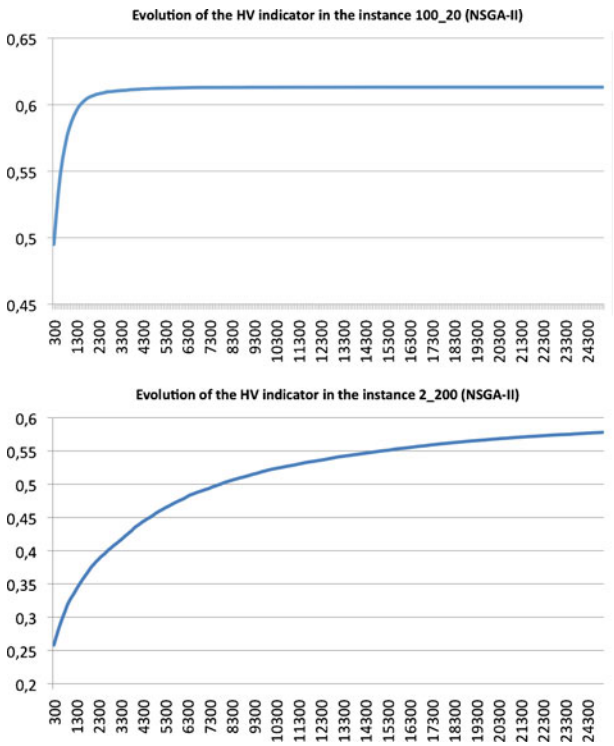


Fig. 4 Evolution of the HV indicator over the number of evaluations in instances 100_20 (*top*) and 2_200 (*bottom*)

Table 4 Mean ($\bar{x}$) and standard deviation (σ) of the results of the Δ quality indicator after 5,000 evaluations

	NSGA-II		MOCell		PAES		
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	
15_40	0.628	0.0340	0.511	0.0400	0.950	0.0600	+
50_80	0.591	0.0400	0.568	0.0510	0.935	0.0370	+
2_200	0.769	0.0490	0.724	0.0370	0.960	0.0330	+
100_20	0.823	0.0220	0.625	0.0150	1.090	0.0500	+
100_25	0.660	0.0310	0.639	0.0240	0.966	0.0590	+
100_140	0.667	0.0520	0.649	0.0460	0.948	0.0390	+

- PAES has computed the least accurate results in this comparison.
- The higher the number of requirements, the harder the problem, with some evidence that the number of requirements has more bearing on problem difficulty than the number of customers.

6.2 *Spread Indicator* Results

We focus now in analyzing the Δ quality indicator, whose values when 5,000, 10,000, and 25,000 evaluations have been performed are included in Tables 4, 5 and 6, respectively. In this indicator, lower values denote better results. The tables show that MOCell is the algorithm obtaining the best results in all the cases. After MOCell, NSGA-II is second best. All result comparisons are statistically significant.

In summary, some observations about the results emerge:

- MOCell was the algorithm computing the fronts with the best distribution of solutions in all the cases.
- PAES has been again the algorithm obtaining the poorest results in accuracy.

An example of the computed fronts for the instances 15_40 and 2_200 by the different algorithms is depicted in Fig. 5. In the instance with 40 requirements (Fig. 5 (top)), we see that the solutions computed by PAES are close to the ones computed by the other techniques; however, NSGA-II and MOCell cover a large number of configurations. In the instance 2_200 (Fig. 5 (bottom)), we observe that the fronts computed by NSGA-II and MOCell dominate the one provided by PAES; furthermore, in this case we can observe the advantages of a front with a good diversity: MOCell has produced a better spread of solutions over the entirety of the Pareto front, while also covering a higher range of different configurations.

Looking again to those figures, we can see that MOCell has been able of computing non-dominatd solutions in areas where NSGA-II and PAES have not found any

Table 5 Mean ($\bar{x}$) and standard deviation (σ) of the results of the Δ quality indicator after 10,000 evaluations

	NSGA-II		MOCell		PAES		
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	
15_40	0.504	0.0360	0.384	0.0350	0.898	0.0600	+
50_80	0.490	0.0380	0.404	0.0310	0.884	0.0460	+
2_200	0.609	0.0330	0.567	0.0380	0.916	0.0310	+
100_20	0.799	0.0110	0.615	0.0054	1.050	0.0480	+
100_25	0.585	0.0260	0.538	0.0190	0.931	0.0670	+
100_140	0.554	0.0350	0.474	0.0350	0.907	0.0360	+

Table 6 Mean ($\bar{x}$) and standard deviation (σ) of the results of the Δ quality indicator after 25,000 evaluations

	NSGA-II		MOCell		PAES		
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	
15_40	0.467	0.0380	0.249	0.0220	0.840	0.0810	+
50_80	0.407	0.0350	0.233	0.0270	0.837	0.0440	+
2_200	0.474	0.0330	0.328	0.0330	0.884	0.0270	+
100_20	0.793	0.0055	0.615	0.0007	1.020	0.0370	+
100_25	0.535	0.0160	0.496	0.0110	0.884	0.0670	+
100_140	0.434	0.0380	0.293	0.0280	0.872	0.0390	+

of them (solutions in the extremes of the Pareto front). This is related to a better exploration of the search space by MOCell. In fact, this is one of the properties of the cellular GA model, in which MOCell is based in, that has been reported in many studies on single-objective optimization (see Alba and Dorronsoro 2008).

6.3 Number of Non Dominated Solutions Found

The instances used in this work have been hand-generated, and the optimal solutions to them are, a priori, unknown. Thus, we cannot be certain that the solutions computed by the algorithms evaluated in this work are optimal; hereinafter we refer to the set of all the non dominated solutions found 'so far' as final solutions.

Tables 7, 8 and 9 contain the number of final solutions computed by each algorithm for different degrees of effort (measured in terms of number of fitness evaluations 'so far'). Starting with Table 7, which shows that information when only 5,000 function evaluations have been carried out, we observe that none of the algorithms

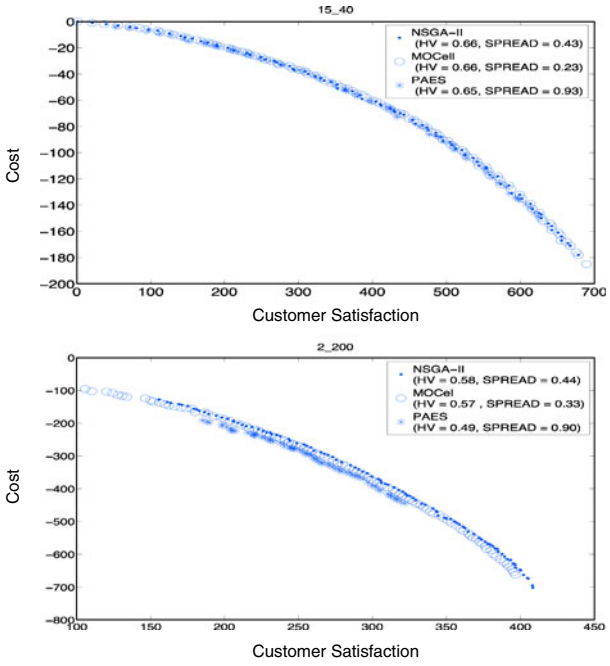


Fig. 5 Examples of the obtained front of solutions in instances 15_40 (15 customers and 40 requirements) and 2_200 (2 customers and 200 requirements)

Table 7 Mean ($\bar{x}$) and standard deviation (σ) of the number of non dominated solutions found after 5,000 function evaluations

	NSGA-II		MOCell		PAES		
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	
15_40	18.5	5.8	12.5	5.5	3.0	2.4	+
50_80	0.0	0.0	0.0	0.0	0.0	0.0	–
2_200	0.0	0.0	0.0	0.0	0.0	0.0	–
100_20	65.4	2.5	63.9	3.0	33.3	6.8	+
100_25	43.9	5.6	40.9	7.4	11.5	5.0	+
100_140	0.0	0.0	0.0	0.0	0.0	0.0	–

has computed final solutions. It is also worth noting that the number of computed final solutions diminish when the number of requirements increases. Among the three algorithms, NSGA-II has been the one computing a higher number, followed by MOCell. The same comments are applicable when 10,000 function evaluations have been carried out (Table 8). However, in this case some final solutions have been computed by NSGA-II in the instance with 80 requirements. Statistical confidence has been found in all the cases where final solutions are obtained.

Finally, consider Table 9, where the number of final solutions found after 25,000 evaluations is included. In this table, we observe that NSGA-II was the only algorithm computing final solutions to the instances with more than 80 requirements. Meanwhile, in the instances with 40 or fewer requirements, MOCell has outperformed NSGA-II for the first time taking into account this indicator.

6.4 Running Time

We have also analyzed the running time required by the algorithms. All the time values are included in Tables 10, 11 and 12, which show the time in milliseconds to perform 5,000, 10,000, and 25,000 function evaluations, respectively. These values refer to execution on an Intel iQ7 processor at 2.8 GHz, with 6 GB RAM memory, running linux (kernel version 2.6.28–15) and the Java Virtual Machine provided by Sun (jdk version 1.6.0_14).

Considering the values shown in these tables, we observe that PAES was the fastest algorithm in our comparison in practically all the instances; only in the instance with 140 requirements and 100 customers the running times of PAES and NSGA-II are comparable. Notwithstanding this, it is important to notice that, in each case, all values are under one second. Looking at the time in each instance, we observe that the higher the number of requirements and customers, the higher the required time. The last column shows that the differences between the results are statistically significant.

Table 8 Mean ($\bar{x}$) and standard deviation (σ) of the number of non dominated solutions found after 10,000 function evaluations

	NSGA-II		MOCell		PAES		
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	
15_40	43.8	5.5	36.5	6.9	7.98	3.8	+
50_80	0.2	0.6	0.0	0.0	0.0	0.0	+
2_200	0.0	0.0	0.0	0.0	0.0	0.0	–
100_20	68.0	1.7	72.6	1.7	39.9	6.1	+
100_25	62.2	4.8	65.9	4.7	18.4	5.3	+
100_140	0.0	0.0	0.0	0.0	0.0	0.0	–

Table 9 Mean ($\bar{x}$) and standard deviation (σ) of the number of non dominated solutions found after 25,000 function evaluations

	NSGA-II		MOCell		PAES		
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	
15_40	48.3	4.6	59.8	4.7	15.3	4.4	+
50_80	1.9	3.4	0.8	1.1	0.0	0.0	+
2_200	1.02	8.9	0.0	0.0	0.0	0.0	+
100_20	68.3	1.5	74.9	0.2	44.5	4.3	+
100_25	72.8	3.3	82.7	3.9	27.2	6.1	+
100_140	1.0	3.2	0.0	0.0	0.0	0.0	+

Table 10 Mean ($\bar{x}$) and standard deviation (σ) of the running time after 5,000 function evaluations (ms)

	NSGA-II		MOCell		PAES		
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	
15_40	457	33	501	26	236	19	+
50_80	617	27	784	33	470	19	+
2_200	1,160	28	1,400	43	1,010	21	+
100_20	472	37	507	16	244	18	+
100_25	510	37	592	34	299	27	+
100_140	1,250	59	1,410	45	1,160	46	+

Table 11 Mean ($\bar{x}$) and standard deviation (σ) of the running time after 10,000 function evaluations (ms)

	NSGA-II		MOCell		PAES		
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	
15_40	621	26	754	30	343	20	+
50_80	1,010	45	1,230	44	822	41	+
2_200	2,200	48	2,580	53	2,010	32	+
100_20	649	32	690	35	380	35	+
100_25	736	42	863	38	482	40	+
100_140	2,220	72	2,570	70	2,230	130	+

Table 12 Mean ($\bar{x}$) and standard deviation (σ) of the running time after 25,000 function evaluations (ms)

	NSGA-II		MOCell		PAES		
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	
15_40	1,000	33	1,160	26	619	34	+
50_80	2,060	80	2,380	77	1,820	81	+
2_200	5,180	57	5,870	72	4,920	48	+
100_20	1,100	54	1,110	44	759	57	+
100_25	1,310	56	1,460	62	974	77	+
100_140	5,160	170	6,010	210	5,160	260	+

7 Studying the Computed Solutions

In the previous section, we have analyzed the quality of the solutions obtained when three different multi-objective optimizers are applied for solving NRP; in this section, we are interested in analyzing which requirements are included in the best solutions found by these algorithms.

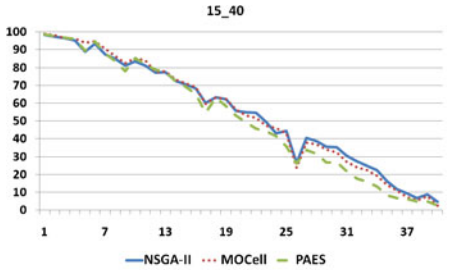
Let us suppose that a requirements engineer has to select a subset of requirements to be included in the next release of a software package and only wants to optimize the cost of fulfilling these requirements. In this situation, it is clear that the software engineer should include, in this set, those requirements that are cheaper. Something similar happens if the interest is only to maximize the satisfaction of the users of that system. Those requirements which satisfy the customer the most should be selected. But which requirements should be considered if the goal is to optimize both objectives? Intuitively, in this case, the optimal Pareto front should be composed of a set of solutions with different numbers of requirements, where those requirements offering a higher ratio of satisfaction per unit cost are more likely to be the candidate solutions found on the final Pareto front.

To analyze which requirements are included in the best solutions found by each algorithm, we have proceeded as follows. First, we sorted all the requirements by the ratio of satisfaction per unit cost (i.e., the satisfaction that each requirement provides to the customers divided by the cost of implementing it). Then, for each requirement we computed the mean of the times that it is included by each algorithm in each solution. Finally, we plotted this information in Fig. 6. The horizontal axis represents, in descending order, the requirements sorted by means of the ratio satisfaction by unit cost. The vertical axis represents the percentage of solutions which implement the requirement represented by the horizontal axis.

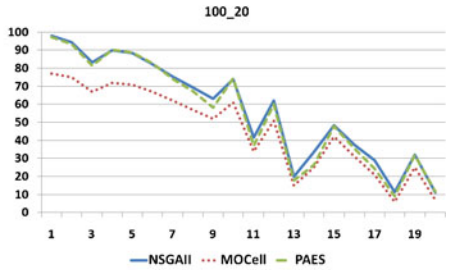
Let us start by analyzing the instance with the lowest number of requirements, i.e., that one known as 100_20. Figure 6d shows the information related to this instance. Although there are some exceptions, there is also a clear tendency to include those requirements with a higher ratio of satisfaction per unit cost with more probability. Considering the differences between the algorithms, we make two further observations. On the one hand, NSGA-II and MOCell appear to have adopted consistently different design strategies. That is, each requirement appears in a higher number of configurations in NSGA-II than in MOCell. However, considering the results of the previous section, both have obtained very accurate results. On the other hand, NSGA-II and PAES have included each requirement the same number of times but, as Section 5 showed, NSGA-II has obtained fronts of better quality than PAES.

Figure 6a, b and e present the information belonging to the instances 100_25, 5_40, and 50_80, respectively. The requirements offering a higher ratio of satisfaction by unit cost are included in a higher number of solutions. Furthermore, NSGA-II and MOCell include each of these requirements as part of a solution more times than PAES. The requirements with the highest ratio have been used more by MOCell than by the other two algorithms, and the requirements with the smallest ratios were more used by NSGA-II.

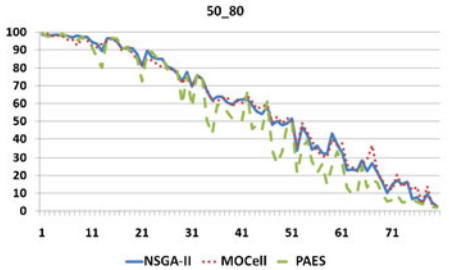
Finally, the results for instances 100_140, and 2_200 are depicted in Fig. 6c and f, respectively. We observe that there is also a tendency to use those requirements with



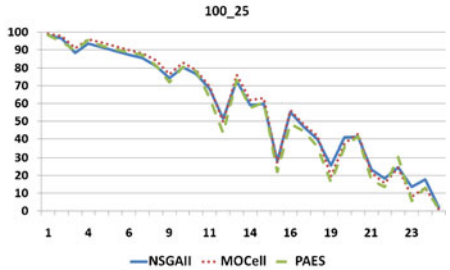
(**a**) 15 customers; 40 requirements



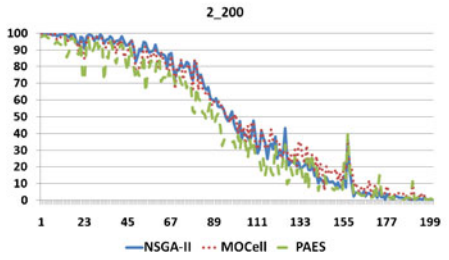
(**d**) 100 customers; 20 requirements



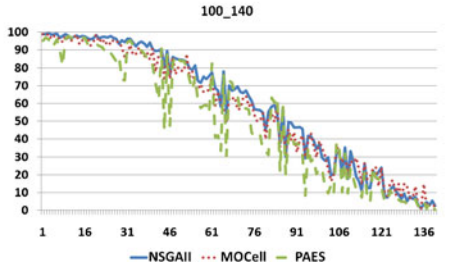
(**b**) 50 customers; 80 requirements



(**e**) 100 customers; 25 requirements



(**c**) 2 customers; 200 requirements



(**f**) 100 customers; 140 requirements

Fig. 6 Percentage of solutions which include each requirement (*horizontal axis*). Requirements are sorted by means of the ratio satisfaction provided by unit cost. In these graphics, the *x*-axis represents the requirements order by mean of ratio of provided customer satisfaction per unit cost; meanwhile, the *y*-axis represent the percentage of use of each requirements in the computed solutions

a better ratio of satisfaction/cost in a higher number of solutions. However, we also see that these figures present many oscillations. These oscillations could potentially indicate that the algorithms have yet to fully converge, as we stated in Section 5.

Thus, in all the instances, the best solutions found so far by each algorithm are composed by those requirements which more satisfy the customers by unit cost. This fact corroborates the observations we made in the conference version of this paper (Durillo et al. 2009), where we stated that the better solutions found were composed of a high percentage of the cheapest requirements, and those which more satisfy the customers are generally those providing the higher ratio satisfaction per unit cost.

8 A Case Study: The Motorola Data Set

This section is aimed at solving a real world instance of NRP problem provided by a large international company, Motorola. We start by presenting the problem. After that, we analyze the obtained results in terms of the quality of the computed fronts, and also in terms of the composition of the solutions. Finally, we deep in the analysis of the obtained results.

8.1 Obtained Results

The Motorola data set concerns a set of 35 requirements for hand held communication devices. The stakeholders are four mobile telephony service providers, each of which has a different set of priorities with respect to the features that they believe ought to be included in each handset. Motorola also maintain cost data in the form of the estimated cost of implementation of each requirement. Each of these stakeholders is equally important for the company (i.e., the value c_i is the same for $i = 1..4$). There exists a main difference between this problem and the test instances studied in this work; each requirement is only desired by one customer.

Table 13 includes the results of the HV, when 5,000, 10,000, and 25,000 function evaluations have been performed. As happened with the test problems, NSGA-II and MOCell computed the best fronts regarding this indicator. Actually, although the results are quite similar, when 25000 evaluations have been performed MOCell has been able to compute better fronts than NSGA-II. In fact, if we show the boxplot distribution of the HV values obtained for both algorithms (see Fig. 7), we can see that the values obtained by MOCell have been higher (then better) than the ones obtained by NSGA-II. Furthermore, the non-overlapped notches in each box means that there is statistical significance between the obtained HV by both algorithms. Additionally, as the last column in the table indicates, the differences in the distributions of results have been statistically significant in all the cases.

We now consider the Δ values, presented in Tables 14. We observe that MOCell is the algorithm computing the fronts with the best value for this indicator; NSGA-II has obtained the second best values.

Thus, the values obtained by the algorithms in both indicators lead us to conclude that MOCell has been the most remarkable technique for solving this problem.

Figure 8 shows examples of fronts computed by NSGA-II and MOCell after 25,000 evaluations. As we can observe, both algorithms have converged towards the same front. However, it is possible to see at a glance that MOCell has obtained a better distribution of solutions. These observations verify the results obtained by the quality indicators: both algorithms have obtained similar values of HV, but MOCell has outperformed to NSGA-II in the Δ indicator.

Table 13 Mean ($\bar{x}$) and standard deviation (σ) of the results of the HV quality indicator in the problem provided by Motorola

Number	NSGA-II		MOCell		PAES		
of evaluations	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	
5,000	0.778	0.0021	0.777	0.0026	0.745	0.0350	+
10,000	0.783	0.0005	0.782	0.0006	0.752	0.0350	+
25,000	0.784	0.0001	0.784	0.0001	0.762	0.0220	+

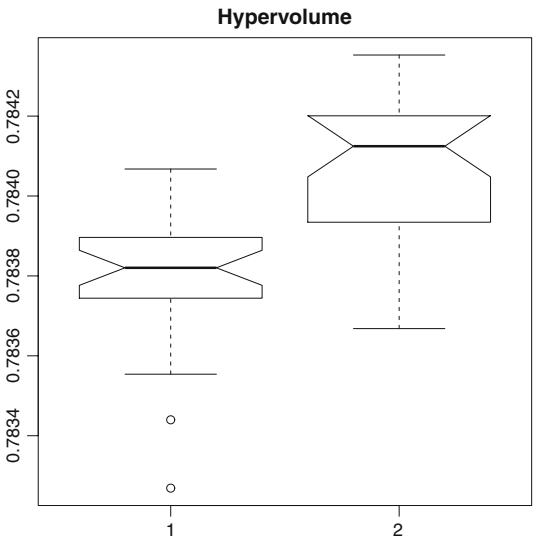


Fig. 7 Distribution of the HV value of the fronts computed by NSGA-II and MOCell

Considering the shape of the fronts depicted in that figure, some conclusions can be drawn. On the one hand, we observe that by increasing the investment from 5 to 1,000 cost units (vertical axis) it is possible to obtain a customer satisfaction between 0 and 50 satisfaction units. On the other hand, there is a section of the front along which an increase in customer satisfaction would require a much larger investment by the company. For example, increasing the customer satisfaction from 50 to 80 units requires an increase in investment of 700%.

With regard to the number of final solutions found by the algorithms, Table 15 summarizes the number of them found after performing 5,000, 10,000, and 25,000 evaluations. Also in this indicator, the algorithms behave in a similar manner to that in which they did for the test instances: when less than 10.000 function evaluations have been carried out, NSGA-II is the algorithm which finds the highest number of final solutions; however, when the number of evaluations increases MOCell outperforms to NSGA-II.

The running times of the algorithms are presented in Table 16. In all the cases, the time required by PAES is less than half the time required by the other two algorithms. However, as we can see, all the times are under one second.

Considering the composition of the computed solutions, Fig. 9 shows the distribution of the requirements used by each algorithm. We can see that those requirements with the better ratio satisfaction by unit cost are included in a higher number of

Table 14 Mean ($\bar{x}$) and standard deviation (σ) of the results of the Δ quality indicator in the problem provided by Motorola

Number	NSGA-II		MOCell		PAES		
of evaluations	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	
5,000	0.910	0.0430	0.546	0.0310	1.100	0.0620	+
10,000	0.842	0.0330	0.502	0.0230	1.070	0.0630	+
25,000	0.812	0.0204	0.473	0.0067	1.050	0.0640	+

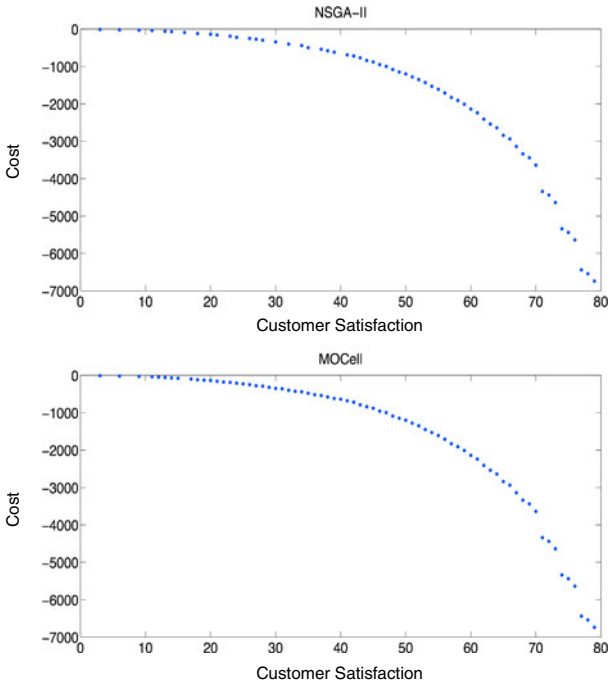


Fig. 8 Examples of fronts computed by NSGA-II (*top*) and MOCell (*bottom*) in the problem provided by Motorola

solutions. This replicates the results observed for the test instances of the MONRP presented earlier.

Thus, the obtained results in this instance confirm the algorithm behavior observed in the previous test instances in terms of the quality of the solutions found, diversity of solutions and the composition of those solutions.

8.2 Post-analysis of the Motorola Problem Results

In the last section, we focused on computing the Pareto front of solutions for the Motorola Problem and on analyzing the composition (in terms of the implemented requirements) of that front. In this section, we go an step forward on the analysis of the solutions computed by the most outstanding algorithm for that instance: MOCell. In particular, we want to analyze two different issues: on the one hand, we want to figure out if is there any relationship between a requirement and the cost and provided satisfaction of a solution which implements it; and, on the other hand, we are also interested in the fairness (i.e., to what extent can a solution be shown to be

Table 15 Mean ($\bar{x}$) and standard deviation (σ) of the number of non dominated solutions found in the problem provided by Motorola

Number	NSGA-II		MOCell		PAES		
of evaluations	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	
5,000	25.6	4.8	20.2	5.5	9.2	4.1	+
10,000	42.6	3.2	37.5	4.4	15.9	5.1	+
25,000	48.8	2.8	53.9	3.0	24.4	7.2	+

Table 16 Mean ($\bar{x}$) and standard deviation (σ) of the running time in the problem provided by Motorola (time is given in ms)

Number of evaluations	NSGA-II		MOCell		PAES		
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	
5,000	353	15	371	5.9	168	8.1	+
10,000	481	6.3	496	6.3	242	9.4	+
25,000	800	5.7	801	16	437	14	+

a fair allocation of resources) of each solution into the computed Pareto front. We make use of the best front found so far by MOCell as the reference Pareto front.

The fairness of an allocation of resources has been previously studied in the SBSE field. Concretely, in Finkelstein et al. (2008), a multi-objective algorithm has been used for computing solutions which maximizes the final satisfaction of customers, minimizes the cost, while maximizing the mean of fulfilled requirement for each customer, i.e., a new objective function is considered for measuring the fairness. The approach followed here is different: we compute the Pareto front of solutions only in terms of cost and satisfaction of the customers, and once it has been computed, we analyze how fair are those solutions.

Thus, summarizing, this section is aimed at answering to the next two questions, respectively:

- Does a requirement determine the place into the Pareto front where a solution implementing it is located?
- Can a given solution into the Pareto front benefiting a customer while damaging other ones?

For answering the first of the above suggested questions, we need to study if the implementation of a requirement and the objective functions (cost and provided customer satisfaction) are correlated. To come with this issue, we have made use of the Spearman's correlation coefficient.

The Spearman Rank Correlation (Kendall and Gibbons 1990) statistical analysis test assess whether two measurement variables are correlated. The test consist in calculating the, so called, correlation coefficient, rs, which can take value between -1 and $+1$. A coefficient $rs = -1$ means two variables have a perfect negative correlation (as one increases, the other decreases). Accordingly, $rs = +1$ means two variables have a perfect positive correlation (as one increases, so does the other);

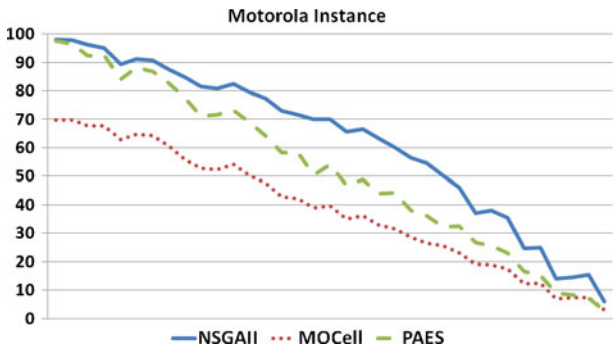


Fig. 9 Distribution of the use of requirements in the Motorola problem. Again, in this graphic, the x-axis represents the requirements order by mean of ratio of provided customer satisfaction per unit cost, and the y-axis represent the percentage of use of each requirements in the computed solutions

$rs = 0$ means two variables are entirely independent; there is no correlation between them.

In our problem, a requirement can only have two states: it is implemented or not. As a consequence, to determine the correlation between each requirement and the objectives functions may have not sense. Thus, in order to study this issue we have group the requirements attending to the ratio of provided satisfaction by unit cost, and after that what we have done is to study the correlation between each group and the objective functions. This way, we can obtain insight knowledge about the relationship between each requirement and the objectives functions by means of its ratio.

Thus, taking into account the above, we have proceed as follows. First, we have sorted all the requirement by means of the ratio satisfaction per unit cost, and we have normalized the ratio to the interval [0, 1]; then, we have classify all the requirements in four different groups attending to the value of its ratio; this classification is based on the information provided by the instance and can be done before computing the optimal solutions. In particular, for this instance, we have computed the following groups: the first group, that we have called *very good*, consist of requirements having a ratio between 0.5 and 1; the second group, *good*, consist of requirements having a ratio between 0.1 and 0.5. The third group (*poor*) consist of those requirements whose ratio is lower than 0.1, and finally, the last group, that we have called *very poor*, consist of those requirements with a very small ratio.

Once the groups have been determined, we have computed the correlation coefficient between each group and the solutions obtained. The obtained results are summarized in Table 17.

Attending to that table, we can observe that for the requirements classified as *very good*, the correlation coefficient with the cost and provided satisfaction are 0.19 and 0.4, respectively. This means that the use of requirements belonging to this group are very weakly correlated with cost and provided satisfaction, and hence, they can be part of whatever solution regardless of them. In that table, it is also possible to observe that the correlation coefficient increases when the ratio value of a group is decreased. This indicates that the lowest the ratio, the stronger the correlation. As a consequence, a requirement with a small ratio of satisfaction by unit cost will be rarely present on solutions of low cost; meanwhile, they will be found more often in those solutions of higher cost and provided satisfaction.

Accordingly, the answer to the first question is that, for requirements with a high ratio it is not possible to determine a priori the cost and customer satisfaction of a solution implementing it; meanwhile, for requirements with a small ratio it is possible to estimate them. This result was somehow expected because, from the last section, we have that requirements with higher ratio than others are implemented in a higher number of solutions. Furthermore, it is reasonable that a good requirement (in terms

Table 17 Spearman's rank correlation between requirements and cost of implementing

Group	Correlation coefficient with cost	Correlation coefficient with customer satisfaction
Very good	0.19	0.4
Good	0.53	0.83
Poor	0.86	0.94
Very poor	0.92	0.66

of its ratio) should be taken in many different configurations, while a bad one (also in terms of its ratio) should be implemented in those solutions which maximize the provided satisfaction (regardless of the cost).

We turn now to analyze the fairness of the computed solutions. The idea here is to determine the correlation between the satisfaction achieved for the different customers. Thus, if the satisfaction of two customers, A and B, are correlated, we can argue that whenever A gets satisfied, B also achieves a high grade of satisfaction.

For computing the correlation coefficient between the different customers we have proceed as follows. First, for each solution into the Pareto front, we have computed the number of implemented requirements targeted by each customer. Then, we have normalized those values to the $[0, 1]$ interval. These values give us a measure of how many requirements have been implemented satisfying each customer in each solution. After that, we have computed the correlation coefficient between each pair of customers. Table 18 summarizes the computed values for the correlation coefficient.

Looking at this table, we see that the coefficient correlation is higher than 0.84 in practically all the cases. Actually, the values under 0.91 involves a comparison with customer 4, which is the most difficult to satisfy due to it is only interested in one out of the 35 requirements. This means that there is a strong correlation between the satisfaction achieved for each pair of customers. As a consequence, the interpretation of those results is that each solution into the Pareto front try to satisfy the customers the same.

The answer to the second proposed question is that the solutions computed by MOCell are not only good solutions in terms of the objective functions, but in addition they seem to be fair in terms of the satisfaction provided to each customer. This is desirable but unexpected result due to neither of the three algorithms employed in this work makes use of any information about the fairness of a solution.

In addition, the product manager is provided with a set of non-dominated solutions (all the solutions are equally important a priori) instead of a single solution. This allows the product manager to choose the best solution, depending on different situations and taking into account factors about which only he or she may be aware. For example, in difficult economic periods the product manager would select solutions involving a lower cost for the company (solutions in the left part of the front); on the one hand, in times of high competition with other companies, the choice could be to select a solution which provides a high degree of customer satisfaction (solutions in the right part of the front). Meanwhile, in a single objective formulation of the problem, dealing with changes on these external situations may require a redefiniton of the considered weights, and to recompute the solution. Of course, these are relatively over-simplified scenarios, intended merely for illustrative purposes.

Moreover, the shape of the Pareto front could help the product manager in working out relationships between the objectives. These relationships may yield

Table 18 Spearman's rank correlation between the satisfaction of different customers

	Customer 2	Customer 3	Customer 4
Customer 1	0.95	0.95	0.85
Customer 2		0.91	0.84
Customer 3			0.85

insights into the nature of the problem, and also it can give a number of choices to select the most adequate solutions.

In practice, the product manager will bring to the scenario a complex interwoven set of managerial, technical, sociological, economic and political concerns, for which it would be impractical to seek any machine-readable formulation. Nevertheless, the search based approach can complement this rich human domain knowledge, by setting out the best choices available, based solely on the cost–benefit analysis information provided. In this way, the machine and human work hand-in-hand. The machine focuses on what it does best (unbiased consideration of an enormous number of potential solutions, guided by cost–benefit data supplied by the human). The human considers the range of options selected in this purely mechanistic manner and uses the shape of the Pareto front to locate interesting locations in the solution space to which to direct further attention and consideration.

9 Related Work

From the industry point of view, many companies feel that they cannot control the release planning challenge, because many of them may only rely on the product or project manager to investigate the implicit characteristics of requirements and study the competing interests of the stakeholders.

In the literature, Yeh and Ng (1990) argued that a target system benefited directly from ranking and prioritising requirements in 1990. Karlsson (1996) adopted two types of techniques for selecting and prioritising software requirements: Quality Function Deployment (QFD) (Sullivan 1986) and Analytical Hierarchy Process (AHP) (Saaty 1980) in 1996. In QFD the stakeholders prioritize the requirements on an ordinal scale (using a numerical assignment). The drawback of this is that there is no clear and obvious definition of the distinction among the absolute number assigned to each requirement. Moreover, relationships between requirements are not supported by QFD. The most serious drawback is that QFD cannot manage functional requirements, because there is no degree of fulfilment for functional requirements.

In 1997, Karlsson and Ryan (1997) proposed a cost-value approach, using AHP, applying it to compare all the candidate requirements in order to determine which of the two is of higher priority and to what extent its priority is higher. Moreover, in 1998, they evaluated six different methods for selecting and prioritising requirements (Karlsson et al. 1998) and found that AHP is the most promising method. However, the disadvantage of using a pairwise comparison technique is the huge number of required pairwise comparisons. The method becomes laborious and inefficient as the scale of the project increases. In addition, this prioritizing process has a lack of support for requirement interdependencies.

AHP caters for human judgements that may be both partial and inconsistent in order to arrive at a robust requirement prioritization. The prioritization problem is clearly related to the Next Release Problem (NRP), because one could select a subset simply as a prefix of the prioritized sequence. However, such a subset selection cannot, by definition, be better than that which can be located by selection of requirements within the same budget and is less amenable to multi-objective generalization.

This is a motivation for the consideration of the separate, but related problem of requirements selection. Compared with priority-based methods, we can provide more than one (usually many) optimal alternative solutions within a certain criterion (such as under specific project budget). As such, the requirements engineer has the opportunity to observe the impact of including or excluding certain requirements, and can use this to choose the best from the different alternatives, without affecting the quality of solutions. An analogous choice between prioritization and selection formulations can be found in SBSE approaches to regression test case selection and prioritization (Harman et al. 2009).

The work of Karlsson et al. has had an enormous impact in the field of requirements engineering and is now the underpinning of the popular tool FocalPoint, marketed by TeleLogic, which is now subsidiary of IBM.

Within the SBSE community, a recent trend has emerged in which search-based optimization techniques have been used to solve requirements selection and optimization problems. This would seem to be a natural and realistic extension of the initial work on SBSE.

The NRP was first formulated as a single-objective SBSE problem by Bagnall et al. (2001). The paper described various metaheuristic optimization algorithms, including greedy algorithms, branch and bound, simulated annealing and hill climbing. The authors did not give any *value* property to each requirement. They only used an associated *cost*. The task of the work was to find a subset of stakeholders, whose requirements are to be satisfied. The objective was to maximize the cumulative measure of the stakeholder's importance to the company under resource constraints.

Feather and Menzies (2002) built an iterative model to seek the near-optimal attainment of requirements. The authors proposed a Defect Detection and Prevention (DDP) process based on a real-world instance: a NASA pilot study. The DDP combined the requirements interaction model with the summarization tool to provide and navigate the near-optimal solutions in the risk mitigation/cost trade-off space. The paper was one of the first to use Pareto optimality in SBSE for requirements, though, unlike the work in our paper, the Pareto fronts were not produced using multi-objective optimization techniques (as with more recent work), but were produced using the iterative application of a weighting based single objective formulation by applying simulated annealing.

Greer and Ruhe (2004), Ruhe and Greer (2003) and Ruhe and Ngo-The (2004) proposed the genetic algorithm based approaches known as the EVOLVE family which aimed to maximize the benefits of delivering requirements in an incremental software release planning process. Their approaches balance the required and available resources; assessing and optimizing the extent to which the ordering conflicts with stakeholder priorities. They also took requirement changes and two types of requirements interaction relationship into account and provided candidate solutions for the next release in an iterative manner. As with previous work, this piece of work still adopted a single objective formulation, taking the resource budget as a constraint.

Using search-based techniques in order to choose components to include in different releases of a system was studied by Baker et al. (2006), Harman et al. (2006). The work considered requirements problems as feature (component) subset selection problems, like Feather et al., presenting results for a single objective

formulation applied to a real world data set: the Motorola Data Set. The work of AlBourae et al. (2006) was focused more on the requirements change handling. That is, re-planning of the product release. A greedy replan algorithm was adopted to reduce risks and increase the number of requirements achieved in the search space under change.

In addition, Cortellessa et al. (2006, 2008a, b) described an optimization framework to provide decision support for COTS and in-house components selection. The Integer Linear Programming (LINGO model solver) based optimization models (CODER, DEER) were proposed to automatically satisfy the requirements while minimizing the cost.

The aforementioned work on this problem has tended to treat the requirements selection and optimization as a single objective problem formulation, in which the various constraints and objectives that characterize the requirements analysis problem are combined into a single objective fitness function. Single objective formulations have the drawback that the maximization of one concern may be achieved at the expense of the potential maximization of another resulting in a bias guiding the search to a certain part of the solution space.

More recently, there has been work on multi-objective formulations of the problem. Zhang et al. (2007) proposed a multi-objective formulation of the next release problem (NRP) to optimise value and cost, upon which we base the formulation in the present paper. This was the first paper to use a Pareto optimal, multi-objective approach to the NRP, migrating it from NRP to MONRP. Independently, at the same time, Saliu and Ruhe (2007) also adopted a Pareto optimal multi-objective approach to the related problem of balancing implementation objectives and requirements objectives. This was the first work to establish and study the link between requirements optimization and the corresponding tension with the implementation. All previous work had considered requirements in isolation, independent from the architectural constraints that choices of requirements impose upon the implementation of the chosen requirement set.

Finkelstein et al. (2008, 2009) considered the problem of fairness analysis in requirements optimization. This was the first paper to introduce techniques for analysis of the trade-offs between different stakeholders' notions of fairness in requirements allocation, where there are multiple stakeholders with potentially conflicting requirement priorities and also possibly different views of what would constitute fair and equitable solution.

As can be seen, the field of multi-objective requirements analysis is growing and developing into a well-defined subfield of activity within the overall areas of SBSE. A position paper on recent trends in requirements analysis optimization can be found in the work of Zhang et al. (2008).

Much of the previous work has been concerned with development of new models, formulations and frameworks for search based requirements. This previous work suggests that requirement analysis can be transformed from a purely qualitative process of human value judgement to a decision support environment in which human judgement is informed by quantitative assessments of choices, optimized using metaheuristic techniques. This growing interest in SBSE for requirement necessitates a more detailed empirical and experimental analysis of the algorithmic choices available to engineers seeking to use SBSE techniques in requirements analysis.

The present paper seeks to take a step toward the provision of this detailed experimental analysis. In the conference version of this paper (Durillo et al. 2009), we provided an initial set of experimental results to investigate the effectiveness of NSGA-II and MOCell for the MONRP. In the present journal version of the paper, we extend these previous results in the following ways:

- We broaden the scope to include another multi-objective optimizer, PAES (Knowles and Corne 1999).
- We analyze the development of Pareto fronts as the algorithms progress. Since all the algorithms are essentially 'anytime' algorithms, this analysis explores how quickly the algorithms converge to a final Pareto front and the increase in Pareto front quality as the algorithms progress.
- We extend the study to consider the efficiency (run time performance) of each of the algorithms studied.
- We consider, additionally, a real world case study of the MONRP, to see whether the experimental results suggested by the detailed study of different problem instances are borne out in practice.
- We further analyze the obtained fronts, not only in terms of the composition of each solution but also considering the fairness of each point into the Pareto front.

10 Conclusions and Future Work

In this paper we have studied the Next Release Problem, with the intention of analyzing the performance of three different multi-objective algorithms, and the solutions they have provided over both test cases and a real instance of the problem.

To come with those issues, we have evaluated three state-of-the-art multi-objective optimization algorithms: NSGA-II, MOCell, and PAES. This comparison has been done on the basis of two quality indicators, HV and Δ, the number of non-dominated solutions obtained by those algorithms, and running time.

In terms of convergence towards the optimal Pareto front, NSGA-II and MOCell have been the best solvers in our comparison. The former algorithm has obtained the best results, performing a lower number of evaluations. Furthermore, it has obtained the best fronts in the two instances with the highest number of requirements. Regarding the distribution of solutions contained in the fronts computed by the algorithms, MOCell has been the most outstanding algorithm in our comparison.

As to the number of obtained solutions, NSGA-II is also the algorithm which has shown the best performance; it is the technique computing the best non-dominated solutions found so far in the two biggest instances. If we attend to the composition of those solutions, we have observe that they are composed in a high percentage of those requirements offering the highest ratio of customer satisfaction by unit cost.

The simplest algorithm in our comparison, PAES, has been the fastest algorithm in our comparison. However, it has obtained the least accurate results according to all the indicators. This highlights the importance of both a population and the use of a recombination operator in order to better explore the search space of MONRP.

The best solutions found so far by the algorithms are composed by those requirements which more satisfy the customers by unit cost. Furthermore, we have observed

that algorithms using the same number of times the same requirements can provide the software engineer with solutions of different quality.

Additionally, we have also made use of the computed front as a tool for analyzing the fairness of the obtained solutions. Specifically, we have analyzed the satisfaction of customers provided by each solution into the Pareto fronts computed by MOCell, the best algorithm for that instance. The results have shown that solutions into those fronts try to satisfy the customers the same.

Thus, by considering a multi-objective approach, it is possible to allow the software engineers to use Pareto front evaluation as a comparative tool for a number of objectives. First of all, the analyst can pick up the best solutions on the Pareto front in different circumstances based on their priorities. For example, at a specific budget level for a project, one or more optimal solutions might be found when moving along the Pareto front. Meanwhile, the stakeholders' satisfaction can also be concerned. Each satisfaction level has its own corresponding cost (resource allocation, spending) according to the Pareto front. This can help the analyst make rough estimates and adjustments for a project budget.

Moreover, the Pareto front not only gives solutions themselves, but also may yield interesting insights into the nature of the problem. The shape of the Pareto front (concave, convex, discontinuous, knee point, nadir point, etc.) reflects the structure of data in an intuitive way, and provides the analyst with very valuable information about the trade-off among the different objectives and helps fully understand the problem and reach practical solutions. Particularly, in the real instance we have identified areas of the fronts where a small increment in customers satisfaction demanded a huge one in the investment.

Future work will verify these findings by applying search techniques to a larger number of real word problems. This will provide valuable feedback to researchers and practitioners in search techniques and in software engineering communities. Other formulations of the problem considering different sets of objectives and constraints and the design of techniques which assist software engineers in the decision making are also issues to study. This, in turn, may give rise to the need for the development of more efficient solution techniques. It is also interesting to investigate how these techniques scale when the number of requirements and/or customer increases. In order to reach this goal, a procedure will be needed which allows the systematic creation of instances with the desired features; in this sense, we plan to design a problem generator for MONRP instances.

Acknowledgements J. J. Durillo, A. Nebro, and E. Alba acknowledge founds from the "Consejería de Innovación, Ciencia y Empresa", Junta de Andalucía under contract P07-TIC-03044 DIRICOM project (http://diricom.lcc.uma.es), and the Spanish Ministry of Science and Innovation and FEDER under contract TIN2008-06491-C04-01 (the M* project). J. J. Durillo is also supported by grant AP-2006-03349 from the Spanish Ministry of Education and Science. Mark Harman is partly supported by EPSRC grants EP/G060525 (CREST: Centre for Research on Evolution, Search and Testing, Platform Grant), and EP/D050863 (SEBASE: Software Engineering By Automated SEarch), which also fully supports Yuanyuan Zhang.

References

Afzal W, Torkar R, Feldt R (2009) A systematic review of search-based testing for non-functional system properties. Inf Softw Technol 51(6):957–976

Alba E, Chicano JF (2007) Software project management with gas. Inf Sci 177(11):2380–2401
Alba E, Dorronsoro B (2008) Cellular genetic algorithms. Operations research/computer science interfaces, vol 42. Springer-Verlag, Heidelberg
AlBourae T, Ruhe G, Moussavi M (2006) Lightweight replanning of software product releases. In: Proceedings of the 1st international workshop on software product management (IWSPM '06). IEEE Computer Society, Minneapolis, pp 27–34
Bagnall AJ, Rayward-Smith VJ, Whittley IM (2001) The next release problem. Inf Softw Technol 43(14):883–890
Baker P, Harman M, Steinhofel K, Skaliotis A (2006) Search based approaches to component selection and prioritization for the next release problem. In: ICSM '06: proceedings of the 22nd IEEE international conference on software maintenance. IEEE Computer Society, Washington, pp 176–185
Cortellessa V, Marinelli F, Potena P (2006) Automated selection of software components based on cost/reliability tradeoff. In: Proceedings of the 3rd European workshop on software architecture (EWSA '06). LNCS, vol 4344. Springer, Nantes, pp 66–81
Cortellessa V, Crnkovic I, Marinelli F, Potena P (2008a) Experimenting the automated selection of COTS components based on cost and system requirements. J Univers Comput Sci 14(8):1228–1255
Cortellessa V, Marinelli F, Potena P (2008b) An optimization framework for "build-or-buy" decisions in software architecture. Comput Oper Res 35(10):3090–3106
Deb K (2001) Multi-objective optimization using evolutionary algorithms, 1st edn. Wiley
Deb KD, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Trans Evol Comput 6(2):182–197
Del Grosso C, Antoniol G, Di Penta M, Galinier P, Merlo E (2005) Improving network applications security: a new heuristic to generate stress testing data. In: GECCO '05: proceedings of the 2005 conference on Genetic and evolutionary computation. ACM, New York, pp 1037–1043
Durillo JJ, Nebro AJ, Luna F, Dorronsoro B, Alba E (2006) jMetal: a java framework for developing multi-objective optimization metaheuristics. Technical Report ITI-2006-10, Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, E.T.S.I. Informática, Campus de Teatinos
Durillo JJ, Zhang Y, Alba E, Nebro AJ (2009) A study of the multi-objective next release problem. In: SSBSE '09: proceedings of the 2009 1st international symposium on search based software engineering. IEEE Computer Society, Washington, pp 49–58
Everson RM, Fieldsend JE (2006) Multiobjective optimization of safety related systems: an application to short-term conflict alert. IEEE Trans Evol Comput 10(2):187–198
Feather MS, Menzies T (2002) Converging on the optimal attainment of requirements. In: RE '02: proceedings of the 10th anniversary IEEE joint international conference on requirements engineering. IEEE Computer Society, Washington, pp 263–272
Finkelstein A, Harman M, Mansouri SA, Ren J, Zhang Y (2008) Fairness analysis” in requirements assignments. In: RE '08: proceedings of the 2008 16th IEEE international requirements engineering conference. IEEE Computer Society, Washington, pp 115–124
Finkelstein A, Harman M, Mansouri SA, Ren J, Zhang Y (2009) A search based approach to fairness analysis in requirement assignments to aid negotiation, mediation and decision making. Requir Eng 14(4):231–245
Greer D, Ruhe G (2004) Software release planning: an evolutionary and iterative approach. Inf Softw Technol 46(4):243–253
Harman M (2007) The current state and future of search based software engineering. In: FOSE '07: 2007 future of software engineering. IEEE Computer Society, Washington, pp 342–357
Harman M, Tratt L (2007) Pareto optimal search based refactoring at the design level. In: GECCO '07: proceedings of the 9th annual conference on genetic and evolutionary computation. ACM, New York, pp 1106–1113
Harman M, Skaliotis A, Steinhöfel K (2006) Search-based approaches to the component selection and prioritization problem. In: Proceedings of the 8th annual conference on genetic and evolutionary computation (GECCO '06). ACM, Seattle, pp 1951–1952
Harman M, Mansouri SA, Zhang Y (2009) Search based software engineering: a comprehensive analysis and review of trends techniques and applications. Technical Report TR-09-03
Karlsson J (1996) Software requirements prioritizing. In: Proceedings of the second international conference on requirements engineering (RE '96). IEEE Computer Society, Colorado Springs, CO, pp 110–116

Karlsson J, Ryan K (1997) A cost-value approach for prioritizing requirements. IEEE Softw 14(5):67–74
Karlsson J, Wohlin C, Regnell B (1998) An evaluation of methods for prioritizing software requirements. Inf Softw Technol 39(14–15):939–947
Kendall M, Gibbons JD (1990) Rank correlation methods, 5th edn. A Charles Griffin Title
Khoshgoftaar TM, Liu Y, Seliya N (2004) Module-order modeling using an evolutionary multiobjective optimization approach. In: Proceedings of the 10th IEEE international symposium on software metrics (METRICS '04). IEEE Computer Society, pp 159–169
Knowles JD, Corne DW (1999) The Pareto Archived Evolution Strategy : A new baseline algorithm for pareto multiobjective optimisation. In: Proceedings of the 1999 congress on evolutionary computation (CEC'99), pp 98–105
Knowles J, Thiele L, Zitzler E (2006) A tutorial on the performance assessment of stochastic multiobjective optimizers. TIK Report 214, Computer Engineering and Networks Laboratory (TIK), ETH Zurich
Korel B (1990) Automated software test data generation. IEEE Trans Softw Eng 16(8):870–879
Lakhotia K, Harman M, McMinn P (2007) A multi-objective approach to search-based test data generation. In: GECCO '07: proceedings of the 9th annual conference on genetic and evolutionary computation. ACM, New York, pp 1098–1105
McMinn P (2004) Search-based software test data generation: a survey: research articles. Softw Test Verif Reliab 14(2):105–156
Miller W, Spooner D (1976) Automatic generation of floating-point test data. IEEE Trans Softw Eng 2:223–226
Nebro AJ, Durillo JJ, Luna F, Dorronsoro B, Alba E (2007) Design issues in a multiobjective cellular genetic algorithm. In: Obayashi S, Deb K, Poloni C, Hiroyasu T, Murata T (eds) Evolutionary multi-criterion optimization. 4th international conference, EMO 2007. Lecture notes in computer science, vol 4403. Springer, pp 126–140
Nebro AJ, Durillo JJ, Luna F, Dorronsoro B, Alba E (2009) Mocell: a cellular genetic algorithm for multiobjective optimization. Int J Intell Syst 24(7):726–746
Ruhe G, Greer D (2003) Quantitative studies in software release planning under risk and resource constraints. In: Proceedings of the international symposium on empirical software engineering (ISESE '03). IEEE, Rome, pp 262–270
Ruhe G, Ngo-The A (2004) Hybrid intelligence in software release planning. International Journal of Hybrid Intelligent Systems 1(1–2):99–110
Saaty TL (1980) The analytic hierarchy process, planning, priority setting, resource allocation. McGraw-Hill
Saliu MO, Ruhe G (2007) Bi-objective release planning for evolving software systems. In: ESEC-FSE '07: proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering. ACM, New York, pp 105–114
Sullivan L (1986) Quality function deployment. Quality progress, pp 39–50
Walcott KR, Soffa ML, Kapfhammer GM, Roos RS (2006) Timeaware test suite prioritization. In: ISSTA '06: proceedings of the 2006 international symposium on software testing and analysis. ACM, New York, pp 1–12
Xanthakis S, Ellis C, Skourlas C, Gall, AL, Katsikas S, Karapoulios K (1992) Application of genetic algorithms to software testing. In: Proceedings of the 5th international conference on software engineering and applications. Toulouse, France, pp 625–636
Yeh R, Ng P (1990) Software requirements—a management perspective. In: Thayer RH, Dorfman M (eds) System and software requirements engineering. IEEE Computer Society Press Tutorial, pp 450–461
Yoo S, Harman M (2007) Pareto efficient multi-objective test case selection. In: ISSTA '07: proceedings of the 2007 international symposium on software testing and analysis. ACM, New York, pp 140–150
Zhang Y, Finkelstein A, Harman M (2008) Search based requirements optimisation: existing work and challenges. In: REFSQ '08: proceedings of the 14th international conference on requirements engineering. Springer-Verlag, Berlin, pp 88–94
Zhang Y, Harman M, Mansouri SA (2007) The multi-objective next release problem. In: GECCO '07: proceedings of the 9th annual conference on genetic and evolutionary computation. ACM, New York, pp 1129–1137
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