Knowledge-Based Systems 83 (2015) 105–115

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

Software requirement optimization using a multiobjective swarm intelligence evolutionary algorithm

José M. Chaves-González *, Miguel A. Pérez-Toledano, Amparo Navasa

Computer Science Department, University of Extremadura, Cáceres, Spain

ARTICLE INFO

Article history:
Received 24 July 2014
Received in revised form 12 February 2015
Accepted 14 March 2015
Available online 24 March 2015

Keywords:
Next release problem
Multiobjective evolutionary algorithm
Software requirement selection
Search-based software engineering
Swarm intelligence
Artificial bee colony

ABSTRACT

The selection of the new requirements which should be included in the development of the release of a software product is an important issue for software companies. This problem is known in the literature as the Next Release Problem (NRP). It is an NP-hard problem which simultaneously addresses two apparently contradictory objectives: the total cost of including the selected requirements in the next release of the software package, and the overall satisfaction of a set of customers who have different opinions about the priorities which should be given to the requirements, and also have different levels of importance within the company. Moreover, in the case of managing real instances of the problem, the proposed solutions have to satisfy certain interaction constraints which arise among some requirements. In this paper, the NRP is formulated as a multiobjective optimization problem with two objectives (cost and satisfaction) and three constraints (types of interactions). A multiobjective swarm intelligence metaheuristic is proposed to solve two real instances generated from data provided by experts. Analysis of the results showed that the proposed algorithm can efficiently generate high quality solutions. These were evaluated by comparing them with different proposals (in terms of multiobjective metrics). The results generated by the present approach surpass those generated in other relevant work in the literature (e.g. our technique can obtain a HV of over 60% for the most complex dataset managed, while the other approaches published cannot obtain an HV of more than 40% for the same dataset).

© 2015 Elsevier B.V. All rights reserved.

1. Introduction

The number and difficulty of the tasks to be performed by current software systems are increasing ever more rapidly. One consequence is a growth in the complexity and the extension of modern software systems, and a concomitant increase in development effort (both time and cost). Software development companies have to efficiently satisfy large sets of requirements by minimizing the production costs of software projects. In most cases, it is not possible to develop all the new features originally suggested.

Software requirement optimization is an important task in Software Engineering, and is especially relevant when managing incremental software development approaches, such as the agile group of methods.

In these methods, the software product is developed by generating releases which are produced in short iterative cycles. In each iteration, a new set of requirements is proposed, tailored to fit the clients' needs and the development costs. In this context, the challenge consists of defining which requirements should be developed taking into consideration several complex factors (priorities given to different clients which have different levels of importance to the company, development efforts, cost restrictions, interactions between different requirements, etc.). This complex problem, called the Next Release Problem (NRP, [1] in the related literature, has no simple solution.

NRP is an NP-hard problem [14] which simultaneously manages two conflicting and independent objectives: development cost (effort) and clients' satisfaction. Thus, it cannot be managed by traditional exact optimization methods. In these kinds of cases, multiobjective evolutionary algorithms (MOEAs) are very appropriate strategies [6,7] because they take into account simultaneously several conflicting objectives without the artificial adjustments which form part of classical single-objective optimization methods. However, most of the related work in the literature takes a simplified approach by using a kind of aggregation function, and tackles the problem as if there were a single objective. Other work does not address the interactions that arise between the requirements in real NRP instances of the problem. In this present communication, we propose a novel technique

* Corresponding author.
E-mail addresses: jm@unex.es (J.M. Chaves-González), toledano@unex.es (M.A. Pérez-Toledano), amparonm@unex.es (A. Navasa).

http://dx.doi.org/10.1016/j.knosys.2015.03.012
0950-7051/© 2015 Elsevier B.V. All rights reserved.

corresponding to the Search-Based Software Engineering (SBSE) research field [16] to deal with a real multiobjective version of the NRP (MONRP).

Specifically, we propose an adapted version of the artificial bee colony algorithm (ABC, [18], in which several multiobjective features have been included in order to obtain high-quality results for a realistic MONRP. Such swarm intelligence approaches have recently been applied to a wide range of complex optimization problems with very satisfactory results [23,24,5,21]. As will be described below, our technique provides better results for MONRP than other approaches published in the literature.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 describes the basic background of the problem, and the multiobjective formulation which we adopted. Section 4 presents our proposed approach: a multiobjective artificial bee colony (MOABC) algorithm for the software requirement selection problem. The experiments performed and their results are presented and analyzed in Section 5. Finally, Section 6 summarizes the conclusions and future lines of work.

2. Related work

The requirement selection problem consists of selecting a certain number of requirements that will be developed for the next release of a specific software package such that those requirements minimize the development costs of the project and maximize the clients' satisfaction. The problem involves two conflicting objectives which have to be considered on equal terms. In the literature, Karlsson [20] proposed two kinds of method for selecting and prioritizing software requirements: an Analytical Hierarchy Process (AHP) and Quality Function Deployment (QFD). In AHP the requirements are classified by a pairwise cost-value, and in QFD they are prioritized on an ordinal scale. However, neither kind of method supports requirement interdependencies, and they have to perform a very large number of comparisons for projects with large sets of requirements.

The requirement selection problem was originally formulated in a single-objective form in the Search-Based Software Engineering (SBSE) field by Bagnall et al. [1]. SBSE is the research field in which search-based optimization algorithms are proposed and tested to tackle problems in Software Engineering [16]. The problem formulated by Bagnall et al. [1] was solved by applying different metaheuristic algorithms. All the proposals were single-objective evolutionary algorithms which combined the objectives by using an aggregation function. The same is the case with the works of Greer and Ruhe [15], and Baker et al. [2] which also adopted a single objective formulation for different evolutionary algorithms. None of these works considered the interactions which arise among requirements. Moreover, the single-objective formulation has the drawback of performing a biased search in the search space of solutions because the objectives have to be artificially aggregated in some way (for example, with a weighted sum or a weighted product).

More recently, the NRP was formulated as a multiobjective optimization problem (MOOP). Zhang et al. [36] proposed the first multiobjective formulation of the original NRP (MONRP). This version was based on Pareto dominance [6]. The method tackles each objective separately (without any combination function), thus allowing the algorithm to explore non-dominated solutions (the solutions of greater quality). The work of Finkelstein et al. [13,12] also used multiobjective optimization for the analysis of trade-offs among multiple clients with potentially conflicting requirement priorities, but the authors did not consider the interactions that arise among requirements. The same is the case with the work of Durillo et al. [11], Jiang et al. [17], and Charan Kumari et al. [4], in which different multiobjective evolutionary algorithms are proposed for solving NRP, but without addressing dependencies among requirements. In Durillo et al. [11], the authors used the well-known algorithms PAES (Pareto Archived Evolution Strategy, [22]), NSGA-II (fast Non-dominated Sorting Genetic Algorithm, [8]), and MOCell (MultiObjective Cellular genetic algorithm, [25]. Jiang et al. [17] solved NRP by using an Ant Colony Optimization (ACO) algorithm [10]. Finally, Charan Kumari et al. [4] proposed the use of a hybrid differential evolution strategy [26].

The only two studies in which, to the best of our knowledge, the MONRP was tackled considering the requirement interactions are those of Sagrado et al. [28] and Souza et al. [34]. They both propose the use of Ant Colony Optimization [10] to solve the problem, but only the work of Sagrado el al. [28] published the datasets used, so that, in Section 5, we shall compare our results with those of that study. In this respect, it is worth mentioning that many of the related studies do not make all the information about the datasets used public (probably for commercial reasons), so it is not possible to make any numerical comparison with their results.

Here, we present a multiobjective search-based approach based on the artificial bee colony (ABC) algorithm [18]. We have adapted the algorithm to work with an MONRP formulation in which different types of requirement interactions and effort constraints are considered. Our technique searches for high quality sets of solutions that balance the clients' priorities and the cost limitations while keeping the requirement interactions. Recent publications have shown the applicability of this approach to different domains. Thus, in Chaves-Gonzalez et al. [5], MOABC is applied to the generation of DNA sequences for reliable DNA computing. In Li et al. [23], the algorithm is used in machine learning to optimize boiler efficiency. [27] propose the use of MOABC to solve the routing problem in optical networks. Silva-Maximiano et al. [30] apply the algorithm to solving the frequency assignment problem. In [29], Phylogenetic Inference was solved with MOABC. [32] successfully solves the minimum spanning tree problem with an adapted version of the artificial bee colony, and [19] use ABC to solve constrained optimization problems. In those studies, the results obtained with the artificial bee colony algorithm are compared with those results obtained from other approaches (NSGA-II, SPEA2, PSO, DE, etc.). We shall here apply MOABC to another research field, and we shall show in Section 5 that the results given by our technique are better than those obtained by other published multiobjective approaches.

3. The multiobjective next release problem

This section explains the MONRP for the selection of software requirements. As mentioned above, the NRP problem was originally formulated by Zhang et al. [36], but here we shall update the formulation in order to handle real instances of the problem in which different types of interactions occur among the requirements to be managed. But first we shall introduce some multiobjective concepts needed for a clearer understanding of what follows.

3.1. Multiobjective background

Two of the most important concepts in multiobjective optimization (MOO) are Pareto dominance and Pareto front. In MOO, a problem does not have a unique optimal solution, but a Pareto front of solutions [6]. The Pareto front is a vector of decision variables which satisfy the problem constraints and optimize the objective functions being considered. Thus, the Pareto front contains a set of Pareto solutions which are not dominated by any

other solution. In this context, a solution $x = [x_1, x_2, \ldots, x_n]$, where n is the number of problem objectives, is said to dominate a solution $y = [y_1, y_2, \ldots, y_n]$, if and only if y is not better than x for any objective $i = 1, 2, \ldots, n$, and there exists at least one objective, x_i, in x which is better than the corresponding y_i objective in y. Inversely, two solutions are said to be non-dominated whenever neither of them dominates the other. Fig. 1 shows some examples of dominated and non-dominated solutions. As can be observed, in the case that the objective functions $f1$ and $f2$ are to be minimized, the solution A dominates the solution D because $f1(A) < f1(D)$ and $f2(A) < f2(D)$. The solutions A, B and C are non-dominated because they are better than the other solutions in some of the objective functions in every case. Thus, A is better than B and C in $f1$, C is better than A and B in $f2$, and B is better than A in $f2$ and better than C in $f1$.

Therefore, as mentioned above, the solution of a given MOO problem is not a specific solution, but a set of solutions, referred to as a Pareto optimal set, which satisfies two properties. First, every two solutions in the set are non-dominated solutions, and second, any other solution found is dominated by at least one solution in the Pareto optimal set. The representation of this set in the objective space is known as the Pareto front (Fig. 1). Theoretically, a Pareto front could contain a large number of non-dominated solutions, but in practice this set only contains a limited number of them. For this reason, the number of non-dominated solutions found for a specific problem is important, as also is that they be uniformly spread along the Pareto front (as illustrated in Fig. 1). Otherwise, they would not be as useful as they should be for the decision maker.

3.2. The requirement selection problem

The software requirement selection problem consists of selecting a set of requirements which involves a set of conflicting criteria that have to be simultaneously optimized. These criteria are usually related to the minimal cost of producing the new release of a software package, and to the greatest client satisfaction. This problem is particularly relevant in agile software development approaches which promote the iterative development of releases of a software product in short development cycles in order to improve productivity (and to introduce checkpoints with the clients). Moreover, the clients involved in the NRP are classified according to their level of importance to the company. This level of importance may depend on the position they occupy in the company, but also on market factors which could lead to modification of the clients' weight (e.g., the company may be interested in satisfying the newest clients' needs). In addition, each requirement will have a different consideration for each client, and this will make the final level of importance for a particular software requirement depend on the *priority* of that requirement for each client and on the level of *importance* of each client being considered. Also, each requirement means an effort in terms of development *costs*, and the company's resources are limited so that

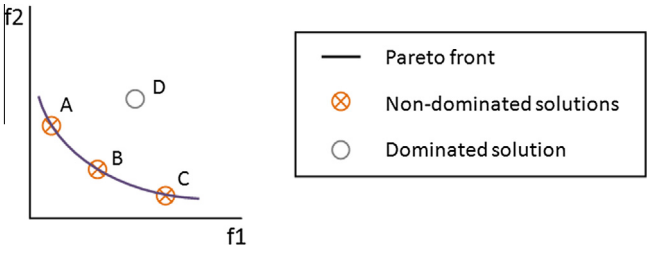


Fig. 1. Example of Pareto front – dominated and non-dominated solutions.

requirements which imply high costs are not usually preferred. Finally, the requirements also present different interaction constraints with other requirements. For example, some requirements must be developed only after some others are developed, or some requirements cannot be developed if some others have already been included in the next release of the software package. The requirement interactions will be handled as problem constraints in the multiobjective model proposed in this work. Carlshamre et al. [3] proposed a classification for the interactions that depended on different software development projects. This classification can be grouped into four main categories [28]: (a) *implication* or *precedence*: $r_i \Rightarrow r_j$, which means that a requirement r_i cannot be chosen if the requirement r_j has not been previously chosen; (b) *combination* or *coupling*: $r_i \oplus r_j$, which means that a requirement r_i has compulsorily to be included with the requirement r_j; (c) *exclusion*: $r_i \otimes r_j$, which means that a requirement r_i cannot be included together with the requirement r_j; (d) *modification*: the development of the requirement r_i implies that some other requirements modify their implementation cost or the satisfaction they provide to the clients.

Therefore, the requirement selection problem can be formulated as an MOOP in which the two objectives are the overall *clients' satisfaction* which has to be *maximized*, and the *software development cost* which has to be *minimized*. Furthermore, these objectives are subject to the problem of the *interactions* that arise among the requirements, which have to be handled as constraints for the MOOP. Considering the requirement selection problem as an MOOP instead of a weighted single-objective problem has the advantage of being able to provide the decision maker with a set of non-dominated solutions, instead of a single best solution. Thus, the decision maker can choose the best solution depending on different situations, and considering complex technical, sociological, economic, and political factors which are beyond any mathematical formulation. The search-based proposal described in this paper can complement this rich, human-domain knowledge by providing the best choices available according to a cost-benefit analysis. The multiobjective proposed approach reduces the enormous number of potential solutions so that the human expert can study the range of solutions provided in the Pareto front generated by the multiobjective algorithm.

3.3. Multiobjective formulation

This subsection is devoted to describing formally the Multiobjective Next Release Problem (MONRP), which is an extension of the traditional NRP proposed by Bagnall et al. [1]. In order to perform a fair comparison of the results generated, we follow the formulation proposed by Sagrado et al. [28], but including in the mathematical model a new constraint: the cost thresholds applied to the solutions that are generated.

Let $R = \{r_1, r_2, \ldots, r_n\}$ be the set of n requirements which have to be developed for the next release of a software package. These requirements represent improvements to the current software system which are suggested by a set of m clients, $C = \{c_1, c_2, \ldots, c_m\}$. Each client has a degree of importance for the company that is reflected by a weight factor. The set of clients' weights is denoted by $W = \{w_1, w_2, \ldots, w_m\}$. Moreover, each requirement r_j in R has an associated cost, e_j, which represents the effort needed for the development of that requirement. Let $E = \{e_1, e_2, \ldots, e_n\}$ be the set of costs which are associated with the n requirements.

It is assumed that each client gives a different importance to each requirement. The importance that a requirement r_j has for a particular client c_i is given by a value $v_{ij} > 0$. A zero value for v_{ij} means that the client c_i has not suggested the requirement r_j. A priority matrix of $m \times n$ holds all the importance values v_{ij}. The overall satisfaction s_j of a given requirement r_j is calculated as

the weighted sum of its importance values for all the clients considered, and can be expressed as indicated in Eq. (1). The set of the global satisfactions calculated in that way is denoted by $S = \{s_1, s_2, \ldots, s_n\}$.

$$s_j = \sum_{i=1}^{m} w_i \bullet v_{ij} \tag{1}$$

The MONRP consists of finding a decision vector, X, which determines the requirements that will be developed for the next software release. X is a subset of R, and it contains the requirements that maximize the clients' satisfaction, and minimize the development costs. These are the two problem objectives, but in addition, all requirements in X have to satisfy the constraints of the problem. These constraints are mainly related to the restrictions generated for the requirement interactions. In this study, we consider two types of interactions: implication interactions – $r_i \Rightarrow r_j$ (if r_i belongs to X, r_j must be also in X) and combination interactions – $r_i \oplus r_j$ (if r_i belongs to X, r_j must be also in X, and vice versa). Furthermore, every solution proposed has to satisfy a cost threshold constraint L_C, as expressed in Eq. (2).

$$\sum_{j \in X} e_j \leqslant L_C \tag{2}$$

Thus, MONRP can be formulated as expressed in Eq. (3):

$$\begin{aligned} &\textit{Maximize } S(X) = \sum_{j \in X} s_j \\ &\textit{Minimize } E(X) = \sum_{j \in X} e_j \\ &\text{Subject to } \begin{cases} \textit{cost threshold constraint} \\ \textit{interaction constraints} \end{cases} \end{aligned} \tag{3}$$

The solution to the problem will be a set of nd non-dominated solutions $\{X_1, X_2, \ldots, X_{nd}\}$ which satisfy the conditions of Eq. (3).

4. Multiobjective artificial bee colony for software requirement selection

Having introduced the background to the MONRP, in this section we shall detail the main features of the proposed multiobjective evolutionary algorithm, but first we shall present the encoding of the solutions dealt with by the proposed approach.

4.1. Solution encoding

The solution is encoded with the purpose of giving support to all the information which is needed to represent and evaluate valid solutions for the MONRP. Fig. 2 shows the representation of the individual used by the multiobjective metaheuristic proposed in this paper. The individuals are the solutions which are the objects of the evolutionary algorithms, so it is important to have a good design that provides fast processing when the genetic operators work with them. Therefore we encoded the solution as a requirement vector of Booleans (X) with n positions. Each position of X indicates whether or not the requirement j is selected for the next release of the software package. If so, that specific position is equal to 1. Otherwise, it is 0. The total number of requirements in X, σ, is also saved in the data structure that maintains the individual ($X_{ReqsNumber}$). Finally, the overall quality for the solution is evaluated through the values of the two objectives being dealt with (cost and overall satisfaction). These values are calculated as explained in Section 3.3, and are also saved in the data structure (see Fig. 2).

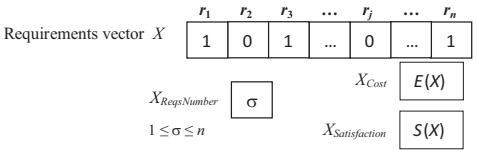


Fig. 2. Data structure for the individual.

4.2. Multiobjective artificial bee colony

In this paper, we propose a multiobjective swarm intelligence evolutionary algorithm based on the artificial bee colony (ABC) algorithm to solve the requirement selection problem. As explained in Section 3, the problem was formulated as an MOOP, so that the original ABC scheme was modified to work as a multiobjective evolutionary algorithm. ABC [18] is motivated by the collective intelligent behavior of honey bee swarms. The algorithm is divided into three main stages which correspond to the behavior of three types of bee: employed, onlooker, and scout bees. The employed bees search for new food sources, and when they find a new one, they come back to the hive and then dance to indicate the new area which they have found. Onlooker bees watch the dances of employed bees, and choose food sources depending on those dances in order to search more deeply in that area. Finally, scout bees explore new areas of the search space with the aim of finding new sources of food (which represent new solutions to the problem).

Algorithm 1. Pseudocode for MOABC to solve the requirement selection problem.

Input: *nEB, nOB, Prob*$_{Mut}$, *limitForScout*
Output: *NDS_archive*
1: /* *Generate and evaluate the first half of the colony C* */
2: *C* ⇐ randomGenerationOfEmployedBees (*nEB*)
3: *C* ⇐ fastNonDominantedSort (*C, nEB*)
4: *C* ⇐ crowdingDistanceCalculation (*C, nEB*)
5: *NDS_archive* ⇐ ∅
6: **while** (not stop condition satisfied) **do**
7: /* *Employed bees search stage* */
8: **for** *i* = 1 to *nEB* **do**
9: *mutatedBee* ⇐ exploreSolution (*C*$_i$, *Prob*)
10: *C*$_i$ ⇐ updateEmployedBee (*C*$_i$, *mutatedBee*)
11: **end for**
12: /* *Generate a probability vector using the employed bees* */
13: *probVector* ⇐ generateProbabilityVector (*C, nEB*)
14: /* *Onlooker bees search stage* */
15: **for** *i* = *nEB* to *nOB* **do**
16: *C*$_{chosen}$ ⇐ chooseEmployedBee (*probVector, C, nEB*)
17: *mutatedBee* ⇐ exploreSolution (*C*$_{chosen}$, *Prob*)
18: *C*$_{chosen}$ ⇐ updateOnlookerBee (*C*$_{chosen}$, *mutatedBee*)
19: **end for**
20: /* *Scout bees search stage* */
21: **for** *i* = 1 to *nEB* **do**
22: **if** *C*$_i$.*iterations* > *limitForScout* **then**
23: *C*$_i$ ⇐ replaceWithScoutBee ()
24: **end if**
25: **end for**
26: /* *Sort the colony by quality and update the Pareto solutions archive* */
27: *C* ⇐ fastNonDominantedSort (*C*)
28: *C* ⇐ crowdingDistanceCalculation (*C*)
29: *NDS_archive* ⇐ updateNDSArchive (*C*)
30: **end while**

Furthermore, in order to address the multiobjective nature of the problem, our technique includes some techniques taken from well-known multiobjective evolutionary algorithms (MOEAs). Thus, from the fast non-dominated sorting genetic algorithm (NSGA-II, [8], we took the idea of non-dominated sorting which organizes the solutions in the population into different groups according to their relation of dominance (as explained in Section 3.1). From the Pareto archived evolutionary strategy (PAES, [22], we took the concept of non-dominated solution archive, which keeps updated the best solutions found during execution of the algorithm. Algorithm 1 presents the pseudocode for the multiobjective artificial bee colony algorithm proposed in this work.

4.2.1. Solution generation and evaluation

The first operation performed in our technique is the random generation of the first half of the colony *C*, which corresponds to the employed bees (Line 2 in Algorithm 1). The employed bees represent here *nEB* solutions which are randomly generated, each with σ requirements (see Fig. 2). These solutions are valid individuals which verify the problem constraints (see Section 3.2). This means that the interactions specified for the solutions generated in the problem instance are respected. After that, each solution is evaluated to determine its quality according to the formulation explained in Section 3.3. Thus, the clients' satisfaction for that specific solution is calculated from Eq. (1), and the cost associated with that solution from Eq. (2).

After the evaluation of each employed bee in *C*, they are ranked according to different Pareto fronts (see Section 3.1), and then each solution is sorted according to non-dominated ranks (Line 3 in Algorithm 1). In this respect, it is said that *a particular solution is of greater quality than another if it is dominated by fewer solutions*, and, in multiobjective optimization, it is known that *a particular solution dominates another if the solution is better in at least one of the objectives and is not worse in any of the others*. In the case that two solutions present equal ranks, they are classified according to the crowding distance [8]. This multiobjective measure is also calculated for each employed bee at the beginning of the algorithm (Line 4). Thus, *a particular solution is better than another if it has the same or lower dominance ranking and presents a greater crowding distance*, because solutions which are located in less crowded regions are preferred [8]. Finally, prior to the main loop of the algorithm, the set of non-dominated solutions (*NDS_archive*), which will contain the best solutions found during the algorithm's execution, is initialized (Line 5).

The main loop of the algorithm (Lines 6–30) is divided into three search stages, one for each type of bee, and a sorting and update stage at the end. These search stages will be explained in the following subsections. Once the three search stages have been completed, the whole colony is sorted again by quality (Lines 26–28) and the best half of the colony will become the new employed bees for the next iteration of the algorithm. Moreover, in each iteration of the algorithm, the best ranked solutions in the population are used to update the *NDS_archive* (non-dominated solutions archive), which maintains the best solutions found (Line 30). This record will be processed after the algorithm's iterations have finished in order to obtain the best ranked solutions in the overall process.

4.2.2. Employed bees search stage

In the employed bees search stage (Lines 7–11 in Algorithm 1), the first half of the colony *C* explores the search space by applying a random mutation to each individual with a mutation probability of $Prob_{Mut}$ (Line 9). It is important to highlight here that the mutation operation includes a verification process in which every individual is fixed in case some constraint specified by the problem instance is not satisfied. This feature makes the algorithm's performance very fast in comparison with other published approaches, as will be discussed in Section 5.2. Fig. 3 shows a simple example for the individual mutation operation, and the subsequent procedure to verify and fix the constraints which are affected.

As can be observed in the example, the random mutation changes requirements r_2, r_3, and r_j relative to the original requirements vector C_i (they are marked with a discontinuous line in C_i'). Then, the interaction constraints are checked. In the example, there is a combination constraint between r_2 and r_3 ($r_2 \oplus r_3$), which means that if the requirement r_2 is chosen, the requirement r_3 has also to be chosen, and an exclusion constraint between r_2 and r_n ($r_2 \otimes r_n$), which means that if r_2 is chosen, r_n cannot be chosen. In the example, the interactions are checked and the requirements with problems are fixed so that no interaction is violated. The result is given in C_i'' (the changes are again marked with a discontinuous line). Finally, every solution must satisfy the cost threshold constraint L_C, which in the case of the example is set to 30% of the total development effort. Thus, in Fig. 3 one can see that requirements r_2 and r_3 are unselected to fulfill this constraint, and the final solution is represented as indicated in the *mutatedBee* final solution. In the case that the individual is changed to fulfill some requirement constraint, it is checked again to verify that no new restrictions have been violated due to the changes made. This process stops when the final generated individual, *mutatedBee*, is a valid solution which satisfies all the constraints specified by the problem instance (see Section 5.1).

The procedure for the constraint verification allows the mutated solutions to be checked efficiently and repaired if necessary, so that every solution generated by the algorithm can be considered a valid solution. After the verification process, the algorithm checks whether the valid mutated bee (*mutatedBee*) is better (in terms of dominance and crowding distance, as explained above) than the original solution C_i. In that case, C_i is replaced by *mutatedBee*; otherwise, C_i stays in the colony and the mutated individual is discarded. Once the employed bees have been processed, a probability vector is generated (Line 13). This vector contains the probability of an employed bee's being selected for the next stage of the algorithm, and it depends on the quality of the solution.

4.2.3. Onlooker bees search stage

The second search stage consists of processing the second half of the colony (Lines 14–19 in Algorithm 1), which contains the onlooker bees. Every employed bee generated, $C_{selected}$, has to be selected according to the previously generated probability vector. This probability vector gives the best solutions a better chance of being selected, and the probability is reduced in accordance with the reduction in the quality of the solutions. The mutation

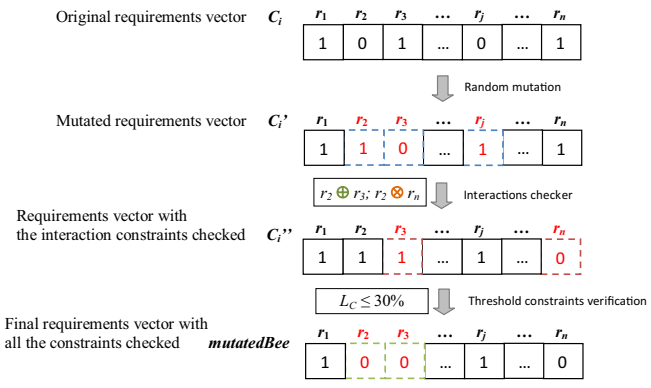


Fig. 3. Mutation operation and procedure for the constraints verification.

Table 1
Requirement development cost (effort), requirement priority level for each client, and interactions for Dataset 1.

		r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}	r_{18}	r_{19}	r_{20}
Effort		1	4	2	3	4	7	10	2	1	3	2	5	8	2	1	4	10	4	8	4
Priority level	cl_1	4	2	1	2	5	5	2	4	4	4	2	3	4	2	4	4	4	1	3	2
	cl_2	4	4	2	2	4	5	1	4	4	5	2	3	2	4	4	2	3	2	3	1
	cl_3	5	3	3	3	4	5	2	4	4	4	2	4	1	5	4	1	2	3	3	2
	cl_4	4	5	2	3	3	4	2	4	2	3	5	2	3	2	4	3	5	4	3	2
	cl_5	5	4	2	4	5	4	2	4	5	2	4	5	3	4	4	1	1	2	4	1
Interactions		$r_4 \Rightarrow r_8$			$r_4 \Rightarrow r_{17}$		$r_8 \Rightarrow r_{17}$		$r_9 \Rightarrow r_3$		$r_9 \Rightarrow r_6$		$r_9 \Rightarrow r_{12}$		$r_9 \Rightarrow r_{19}$		$r_{11} \Rightarrow r_{19}$		$r_3 \oplus r_{12}$		$r_{11} \oplus r_{13}$	

operation is performed following a process similar to that followed for the employed bees (Fig. 3), but in this case the mutation operation and the constraints verification process is applied only to the selected bee, C_{chosen} (Line 17), and not to all the onlooker bees. The solutions that represent these bees have to be selected to operate with them, and it is possible that not all of them are processed.

After the mutation process, it is checked whether or not the new generated bee, *mutatedBee*, presents a value better than or equal to that of the original solution, C_{chosen}, in terms of dominance (Line 18). If so, the *mutatedBee* solution is stored in the colony; otherwise, the selected bee, C_{chosen}, is saved and *mutatedBee* is discarded. Notice that in this search stage a new generated solution with an equal dominance value is allowed to replace the original solution. This feature of the search procedure helps to promote diversity in the population being dealt with.

4.2.4. Scout bees search stage

The third search stage consists of generating the scout bees (Lines 20–25). In this stage, the *limitForScout* parameter (Line 22) plays a key role because, if the solution associated with a bee is not improved in *limitForScout* iterations, it will be discarded and replaced by a newly generated scout bee. In such a case, the candidate solution is considered a local optimum for the dataset being dealt with, so that this individual is transformed into a scout bee which will be randomly generated and improved by using a local search that tries to improve the cost and the satisfaction – see Eq. (3) – of the newly generated solutions. Scout bees are generated with the aim of exploring different undiscovered regions of the search space by looking for new high-quality valid solutions.

5. Experiments and results

In this section, we first describe the method followed in the experiments which we performed, and the datasets used to evaluate the performance of the proposed algorithm. Then, we present the results using different quality indicators and compare them with the results of other approaches published in the literature.

5.1. Experimental method and the datasets used

All the experiments performed in this study were run in the same environment: an Intel Xeon 2.33 GHz processor with 4 GB RAM. On the software side, we used the *gcc* 4.4.5 compiler on a Scientific Linux 5.3 64-bit OS. Since we are dealing with a stochastic algorithm, we carried out 100 independent runs for each experiment. The results provided in the following subsections are the mean values of these independent runs. It is important to note here that the arithmetic mean is a valid statistical measure because the results were found to fit a normal distribution, as was confirmed by a p-value greater than 0.05 for the Shapiro–Wilk test [9]. Moreover, all the results presented a very low dispersion (as will be seen in the tables in the following subsection), so that they can be considered statistically reliable.

The effectiveness of our proposed approach was tested using two different real datasets. Each dataset was constrained with four different cost boundaries applied to the total effort for the development of all the requirements of the software project (30%, 50%, 70%, and 100% of the total development effort), so that our technique was tested with a total of eight instances of the next release problem.

The first dataset was taken from Greer and Ruhe [15]. It includes 20 requirements and 5 clients. Table 1 shows the development effort associated with each requirement, the level of priority assigned to each requirement for each client, and the interaction constraints. The priority level for each requirement takes values from 1 to 5 depending on its level of importance. These values can be interpreted as representing a requirement that is: (1) not important, (2) minor, (3) important, (4) very important, and (5) extremely important. Moreover, each requirement has an associated development cost (effort) which is estimated in terms of a score from 1 to 10. Finally, a set of implication and combination interactions between requirements is also considered. Exclusion interactions are not present in this dataset.

Each client has a relative level of importance for the company. The overall priority that a particular client gives to a specific requirement will depend on the priority level assigned (expressed in Tables 1 and 2) and the weight assigned to that client – see Eq. (1). The clients' weights for the two datasets that we used are listed in Table 3. The values run from 1 (the least important clients) to 5 (the most important client).

The dataset presented in Table 1 is not very large, but, with respect to this, we want to clarify that, due to the privacy policies followed by software development companies, as far as we know, the only two available real datasets are the ones included in this present study. In addition, these datasets have previously been used in other work [28], so that we were able to use these datasets to compare the present results with those of other studies in the literature (Section 5.2).

The second dataset, proposed by Sagrado et al. [28], includes 100 requirements, 5 clients, and 44 (combination and implication) requirement interactions. Table 2 gives the development effort associated with each requirement, the level of priority assigned to each requirement for each client, and the interaction constraints. The relative importance of each client is given in the second row of Table 3. One observes that this second dataset is more complex than the previous one. Indeed, both the number of requirements (100) and the development costs (which, in this case, range from 1 to 20) are taken from real agile software project developments. The greatest development effort for a requirement is set at 20 effort units, which can be translated into 4 real weeks. This temporal limit is usually defined as a time box in agile software engineering methods. For instance, the scrum method proposes iterations of between 2 and 4 weeks [33]. The priority levels in this case run from 1 to 3 because, when clients have to

Table 2
Requirement development cost (effort), requirement priority level for each client, and interactions for Dataset 2.

		r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}	r_{18}	r_{19}	r_{20}
Effort		16	19	16	7	19	15	8	10	6	18	15	12	16	20	9	4	16	2	9	3
Priority level	cl_1	1	2	1	1	2	3	3	1	1	3	1	1	3	2	3	2	2	3	1	3
	cl_2	3	2	1	2	1	2	1	2	2	1	2	3	3	2	1	3	2	3	3	1
	cl_3	1	1	1	2	1	1	1	3	2	2	3	3	3	1	3	1	2	2	3	3
	cl_4	3	2	2	1	3	1	3	2	3	2	3	2	1	3	2	3	2	1	3	3
	cl_5	1	2	3	1	3	1	2	3	1	1	2	2	3	1	2	1	1	1	1	3
		r_{21}	r_{22}	r_{23}	r_{24}	r_{25}	r_{26}	r_{27}	r_{28}	r_{29}	r_{30}	r_{31}	r_{32}	r_{33}	r_{34}	r_{35}	r_{36}	r_{37}	r_{38}	r_{39}	r_{40}
Effort		2	10	4	2	7	15	8	20	9	11	5	1	17	6	2	16	8	12	18	5
Priority level	cl_1	2	1	1	1	3	3	3	3	1	2	2	3	2	1	2	2	1	3	3	2
	cl_2	3	3	3	2	3	1	2	2	3	3	1	3	2	2	1	2	3	2	3	3
	cl_3	2	1	2	3	2	3	3	1	3	3	3	2	1	2	2	1	1	3	1	2
	cl_4	1	1	1	2	3	3	2	1	1	1	1	2	2	2	3	2	2	3	1	1
	cl_5	1	1	3	3	3	2	2	3	2	3	1	1	3	3	2	2	1	1	2	1
		r_{41}	r_{42}	r_{43}	r_{44}	r_{45}	r_{46}	r_{47}	r_{48}	r_{49}	r_{50}	r_{51}	r_{52}	r_{53}	r_{54}	r_{55}	r_{56}	r_{57}	r_{58}	r_{59}	r_{60}
Effort		6	14	15	20	14	9	16	6	6	6	6	2	17	8	1	3	14	16	18	7
Priority level	cl_1	2	2	3	1	1	1	2	2	3	3	3	3	1	3	2	1	3	1	3	1
	cl_2	3	3	1	1	3	2	2	2	1	3	3	3	1	2	2	3	3	2	1	1
	cl_3	1	3	1	3	3	3	3	1	3	2	3	1	2	3	2	3	2	1	2	3
	cl_4	3	1	1	3	1	2	1	1	3	2	2	1	3	2	1	3	3	1	2	3
	cl_5	3	1	1	2	1	2	3	3	2	2	1	3	3	2	3	1	2	1	3	2
		r_{61}	r_{62}	r_{63}	r_{64}	r_{65}	r_{66}	r_{67}	r_{68}	r_{69}	r_{70}	r_{71}	r_{72}	r_{73}	r_{74}	r_{75}	r_{76}	r_{77}	r_{78}	r_{79}	r_{80}
Effort		10	7	16	19	17	15	11	8	20	1	5	8	3	15	4	20	10	20	3	20
Priority level	cl_1	2	2	3	3	1	3	1	3	2	3	1	3	2	3	1	1	2	3	3	1
	cl_2	1	3	2	3	1	2	1	2	3	1	1	3	1	3	2	1	3	3	1	2
	cl_3	1	1	2	3	3	1	3	3	3	1	3	1	3	1	1	2	3	3	1	2
	cl_4	2	2	3	3	3	1	2	1	2	1	2	3	3	2	2	2	1	3	3	1
	cl_5	2	2	1	2	1	3	2	1	2	1	2	2	3	2	1	3	2	3	1	3
		r_{81}	r_{82}	r_{83}	r_{84}	r_{85}	r_{86}	r_{87}	r_{88}	r_{89}	r_{90}	r_{91}	r_{92}	r_{93}	r_{94}	r_{95}	r_{96}	r_{97}	r_{98}	r_{99}	r_{100}
Effort		10	16	19	3	12	16	15	1	6	7	15	18	4	7	2	7	8	7	7	3
Priority level	cl_1	2	1	3	1	2	2	2	1	3	2	2	3	1	1	1	2	1	3	1	1
	cl_2	1	2	1	2	2	1	3	2	2	2	3	2	2	3	2	2	1	3	1	1
	cl_3	1	2	3	2	3	1	2	2	3	3	3	3	2	1	1	2	3	3	2	3
	cl_4	3	1	2	2	2	1	1	1	3	1	1	3	3	1	2	1	2	3	1	3
	cl_5	3	2	1	2	2	2	2	1	3	3	3	1	1	3	1	3	3	3	3	3
Interactions		$r_2 \Rightarrow r_{24}$	$r_3 \Rightarrow r_{26}$	$r_3 \Rightarrow r_{27}$	$r_3 \Rightarrow r_{28}$	$r_3 \Rightarrow r_{29}$	$r_4 \Rightarrow r_5$	$r_6 \Rightarrow r_7$	$r_7 \Rightarrow r_{30}$	$r_{10} \Rightarrow r_{32}$	$r_{10} \Rightarrow r_{33}$	$r_{14} \Rightarrow r_{32}$									
		$r_{14} \Rightarrow r_{34}$	$r_{14} \Rightarrow r_{37}$	$r_{14} \Rightarrow r_{38}$	$r_{16} \Rightarrow r_{39}$	$r_{16} \Rightarrow r_{40}$	$r_{17} \Rightarrow r_{43}$	$r_{29} \Rightarrow r_{49}$	$r_{29} \Rightarrow r_{50}$	$r_{29} \Rightarrow r_{51}$	$r_{30} \Rightarrow r_{52}$										
		$r_{30} \Rightarrow r_{53}$	$r_{31} \Rightarrow r_{55}$	$r_{32} \Rightarrow r_{56}$	$r_{32} \Rightarrow r_{57}$	$r_{33} \Rightarrow r_{58}$	$r_{36} \Rightarrow r_{61}$	$r_{39} \Rightarrow r_{63}$	$r_{40} \Rightarrow r_{64}$	$r_{43} \Rightarrow r_{65}$	$r_{46} \Rightarrow r_{68}$										
		$r_{47} \Rightarrow r_{70}$	$r_{55} \Rightarrow r_{79}$	$r_{56} \Rightarrow r_{80}$	$r_{57} \Rightarrow r_{80}$	$r_{62} \Rightarrow r_{83}$	$r_{62} \Rightarrow r_{84}$	$r_{64} \Rightarrow r_{87}$													
		$r_{21} \oplus r_{22}$	$r_{32} \oplus r_{33}$	$r_{46} \oplus r_{47}$	$r_{65} \oplus r_{66}$																

Table 3
The clients' relative importances.

Clients' weights	cl_1	cl_2	cl_3	cl_4	cl_5
Dataset 1	1	4	2	3	4
Dataset 2	1	5	3	3	1

Table 4
The datasets' main properties and HV reference points.

Dataset 1	20 requirements, 5 clients, 10 interaction constraints r_{min} (*cost*, *satisfaction*) = (0, 0) r_{max} (*cost*, *satisfaction*) = (85, 893)
Dataset 2	100 requirements, 5 clients, 44 interaction constraints r_{min} (*cost*, *satisfaction*) = (0, 0) r_{max} (*cost*, *satisfaction*) = (1037, 2656)

make an assignment related to the benefit of including a new requirement, they prefer to use a coarse-grained scale. In particular, they simply place the requirements into one of three categories: (1) inessential, (2) desirable, and (3) mandatory [31,35].

As we were working in a multiobjective environment, the parameter configuration of our proposed approach was established according to the quality of the Pareto front produced in each test. We used four quality indicators aimed at being able to conduct a comparative study with other relevant published work.

The first quality indicator was the *hypervolume* (*HV*, [37]. This measure calculates the volume (in the objective space) covered by members of a non-dominated set of solutions Q – see Eq. (4). *HV* measures the convergence and diversity of the Pareto fronts obtained. When a Pareto front has a greater *HV* than another one, this could be due to either of two factors: some solutions in the better front dominate solutions in the other, or solutions in the better front are more widely distributed than in the other. Since both properties are considered to be good, algorithms with higher values of *HV* are considered to be desirable. In order to calculate this metric, two reference points were required. Since the problem we were addressing has two objectives, these points were r_{min} ($obj1_{min}$, $obj2_{min}$) and r_{max} ($obj1_{max}$, $obj2_{max}$), i.e., points containing the minimum and maximum values for the two objectives (development cost and overall client satisfaction). Note that *HV* is not free from arbitrary scaling of the objectives, so that the value of this metric may be distorted if the ranges of the two objective functions are different. Thus, before calculating the hypervolume, all the objective function values had to be normalized. The normalization points used for each dataset are given in Table 4.

$$HV = volume\left(\bigcup_{i=1}^{|Q|} v_i\right) \tag{4}$$

The second was the spread achieved by the set of solutions (*Δ-Spread*). This indicator measures the diversity of the solutions by using the Euclidean distances between consecutive solutions in the Pareto front. Pareto fronts with a smaller spread are preferred. *Δ-Spread* is defined by Eq. (5):

$$\Delta = \frac{d_f + d_l + \sum_{i=1}^{N-1} |d_i - \overline{d}|}{d_f + d_l + (n-1)d} \tag{5}$$

where d_i is the Euclidean distance between two consecutive solutions, $\overline{d}$ is the mean distance between each pair of solutions, N is the total number of solutions in the Pareto front, and d_f and d_l are, respectively, the Euclidean distance from the first and the last solution in the Pareto front to the extreme solutions of the optimal Pareto front in the objective space.

The third was the *number of non-dominated solutions* (*NDS*) found. Pareto fronts with a greater number of non-dominated solutions are preferred.

And the fourth was the *execution time* of the algorithm. Execution times are included in order to provide a measure of efficiency for the algorithm.

Finally, with respect to the algorithm's configuration, in order to perform fair comparisons with other work, we set the same stop condition for our technique: 10,000 fitness function evaluations [28]. The other algorithm parameters were tuned separately to obtain the best results for the problem being tackled. Thus, the initial population was set to 40 individuals. Half of the population are the employed bees (parameter *nEB* in Algorithm 1), and the other half the onlooker bees (*nOB* in Algorithm 1). The probability of the mutation operator was set to $Prob_{Mut} = 0.5$, and the *limitForScout* parameter was set to 3.

5.2. Results, discussion, and comparison with other work

In this subsection, we present the results obtained with our proposed approach, and compare them with those published in other work. We start by analyzing the values of the *HV* and *Δ-Spread* quality indicators. Then we perform a comparative study of the number of non-dominated solutions (*NDS*) obtained by different approaches, and finally compare the execution times of different algorithms.

Table 5
Average *HV* and standard deviation of the results for the 4 instances of Dataset 1.

Dataset 1 Effort boundary	MOABC Mean ± std. dev.	ACO Mean ± std. dev.	NSGA-II Mean ± std. dev.	GRASP Mean ± std. dev.
30%	41.880% ± 1.15e−5	10.283% ± 6.57e−2	9.015% ± 1.12	7.708% ± 3.66e−1
50%	54.715% ± 2.64e−4	23.912% ± 6.75e−2	20.652% ± 1.60	19.114% ± 3.50e−1
70%	60.855% ± 9.49e−4	38.464% ± 7.08e−2	32.157% ± 2.30	32.242% ± 4.96e−1
Without effort limit	63.780% ± 1.28e−3	–	–	–

Table 6
Average *HV* and standard deviation of the results for the 4 instances of Dataset 2.

Dataset 2 Effort boundary	MOABC Mean ± std. dev.	ACO Mean ± std. dev.	NSGA-II Mean ± std. dev.	GRASP Mean ± std. dev.
30%	41.232% ± 1.14e−2	8.517% ± 6.21e−2	7.920% ± 2.49e−1	4.088% ± 8.55e−3
50%	51.212% ± 1.17e−2	19.159% ± 9.94e−2	18.006% ± 5.20e−1	15.454% ± 6.88e−2
70%	58.212% ± 7.00e−3	32.777% ± 1.14e−1	31.710% ± 8.92e−1	27.943% ± 7.50e−2
Without effort limit	61.702% ± 4.94e−3	–	–	–

Table 7
Average Δ-*Spread* and standard deviation of the results for the 4 instances of Dataset 1.

Dataset 1 Effort boundary	MOABC Mean ± std. dev.	ACO Mean ± std. dev.	NSGA-II Mean ± std. dev.	GRASP Mean ± std. dev.
30%	0.52 ± 0.01	0.52 ± 0.03	0.76 ± 0.09	0.64 ± 0.09
50%	0.48 ± 0.01	0.52 ± 0.01	0.79 ± 0.07	0.73 ± 0.07
70%	0.43 ± 0.01	0.48 ± 0.02	0.80 ± 0.07	0.69 ± 0.06
Without effort limit	0.39 ± 0.05	–	–	–

Table 8
Average Δ-*Spread* and standard deviation of the results for the 4 instances of Dataset 2.

Dataset 2 Effort boundary	MOABC Mean ± std. dev.	ACO Mean ± std. dev.	NSGA-II Mean ± std. dev.	GRASP Mean ± std. dev.
30%	0.45 ± 0.02	0.68 ± 0.06	0.80 ± 0.07	0.60 ± 0.04
50%	0.42 ± 0.02	0.66 ± 0.06	0.81 ± 0.06	0.74 ± 0.04
70%	0.38 ± 0.02	0.61 ± 0.06	0.77 ± 0.05	0.70 ± 0.03
Without effort limit	0.35 ± 0.03	–	–	–

Table 9
Mean number of *NDS* and standard deviation of the results for the 4 instances of Dataset 1.

Dataset 1 Effort boundary	MOABC Mean ± std. dev.	ACO Mean ± std. dev.	NSGA-II Mean ± std. dev.	GRASP Mean ± std. dev.
30%	15 ± 0.00	13.66 ± 13.66	9.69 ± 2.09	11.37 ± 1.47
50%	23.66 ± 0.48	17.75 ± 0.61	11.30 ± 1.82	17.65 ± 2.22
70%	32.35 ± 0.99	20.57 ± 20.57	11.70 ± 1.90	20.26 ± 2.18
Without effort limit	40.55 ± 1.25	–	–	–

Table 10
Mean number of *NDS* and standard deviation of the results for the 4 instances of Dataset 2.

Dataset 2 Effort boundary	MOABC Mean ± std. dev.	ACO Mean ± std. dev.	NSGA-II Mean ± std. dev.	GRASP Mean ± std. dev.
30%	125.37 ± 7.57	47.12 ± 5.44	54.34 ± 8.51	57.99 ± 3.66
50%	135.93 ± 9.60	57.68 ± 5.69	65.54 ± 11.86	75.81 ± 5.81
70%	139.31 ± 9.93	70. 98 ± 5.27	83.32 ± 10.52	120.14 ± 7.27
Without effort limit	147.51 ± 9.90	–	–	–

5.2.1. Hypervolume (HV) results

Tables 5 and 6 summarize the comparative results in terms of the hypervolume indicator. We compare the results obtained with our proposed approach, MOABC, with those of other algorithms proposed by Sagrado et al. [28]: ACO, NSGA-II, and GRASP. The results are given in terms of the mean (and standard deviation) hypervolume of 100 independent runs for the two datasets under study. Moreover, we show the results for each dataset with the four cost boundaries considered (30%, 50%, 70%, and 100% of the total development cost). Therefore, a total of 8 instances of the problem were studied. The results with a higher value of *HV* are of better quality. One observes that MOABC is the algorithm which gives the best results regarding this multiobjective indicator for every problem instance. In addition, the results obtained with MOABC present very low dispersions for every cost boundary tested. It can therefore be said that the overall improvement achieved by our swarm technique, MOABC, is quite notable in terms of this multiobjective measure. This means that MOABC is able to explore the search space of solutions better than the other approaches, and consequently the solutions provided for the MONRP will be of better quality. One sees that GRASP is the metaheuristic which gives the poorest results. This could be because GRASP is a trajectory-based metaheuristic and does not work with a population of individuals as the other approaches do, so that the exploration of the search space is more limited.

5.2.2. Δ-Spread results

In this subsection, we shall focus on analyzing the *Δ-Spread* quality indicator for the 8 instances of the problem under study. In the case of this multiobjective measure, lower values denote better results. Tables 7 and 8 summarize the results for this indicator.

One observes from Tables 7 and 8 that MOABC is the evolutionary algorithm yielding the best results in all cases, so that it can be said that our technique is the algorithm that computes the Pareto fronts with the best distribution of solutions in all cases.

5.2.3. Number of non-dominated solutions (NDS)

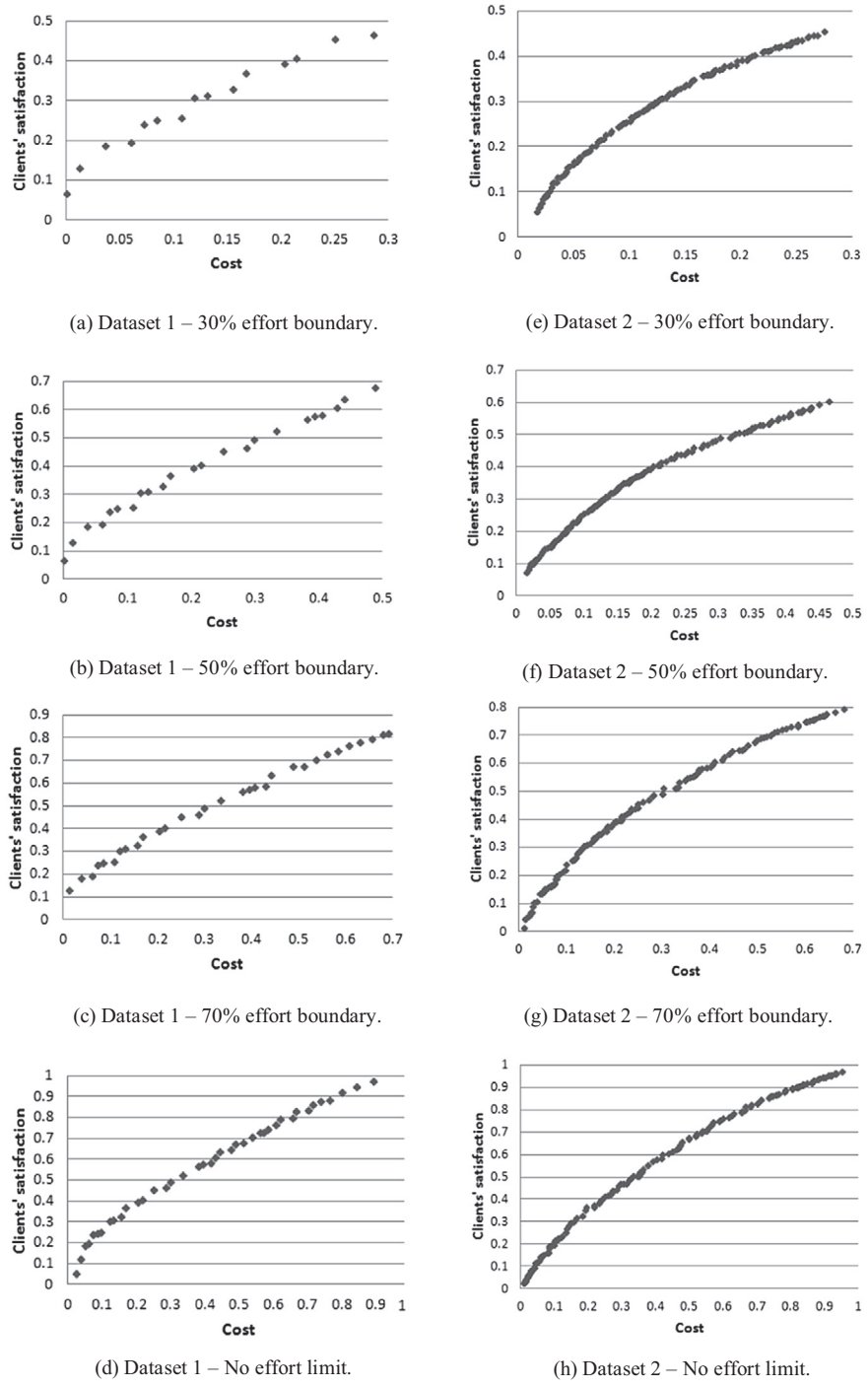
The optimal solutions for the problem tackled in this paper are *a priori* unknown, so that it cannot be guaranteed that the solutions computed by the algorithms are optimal. For this reason, the set of all the final high-quality solutions found by the algorithms are termed *non-dominated solutions*. Tables 9 and 10 give the mean number of non-dominated solutions obtained by our approach (MOABC) and by the other algorithms published in the literature [28] for the different development effort boundaries applied to the two datasets under study.

Considering the results in Tables 9 and 10, one notices that MOABC yields a markedly greater number of non-dominated solutions (NDS) in every case. The number of *NDS* generated is greater for Dataset 2, which is more complex, and the differences between our technique and the other approaches that have been published in the literature are also greater for this more complex dataset. In fact, the number of *NDS* obtained by MOABC is twice the number of *NDS* generated by ACO for Dataset 2 (Table 10). Fig. 4 presents the non-dominated solutions obtained by our proposed approach (MOABC) for each dataset studied.

5.2.4. Execution time

We also studied the execution time that the proposed algorithm takes to provide the solutions. However, we cannot directly compare the execution time of our proposed approach with those of the other approaches published [28] since the hardware and software conditions for those algorithms were not provided. In the present case, as was mentioned above, MOABC was run on an Intel Xeon 2.33 GHz processor with 4 GB RAM, and the execution times for Dataset 1 were shorter than 300 ms in every case. The processing of Dataset 2 took less than 2 s in all cases.

For the algorithms proposed in the literature [28], the fastest for Dataset 1 is ACO, with execution times between 600 and 800 ms, approximately. Therefore, the present technique would seem to be at least twice as fast as the previous fastest algorithm for Dataset 1. The fastest algorithm proposed for Dataset 2 is NSGA-II (although it is not the one that gives the best results, as one observes from Tables 5–10), with execution times between 28 and 38 s, approximately. Moreover, the ACO algorithm, which is the most competitive published proposal, takes longer than 10 min to obtain any such competitive results. Thus, our technique is at least 10 times faster than the fastest algorithm, and greatly faster (by another order of magnitude) than the best algorithm proposed in the literature for the problem being tackled.



(a) Dataset 1 – 30% effort boundary.

(e) Dataset 2 – 30% effort boundary.

(b) Dataset 1 – 50% effort boundary.

(f) Dataset 2 – 50% effort boundary.

(c) Dataset 1 – 70% effort boundary.

(g) Dataset 2 – 70% effort boundary.

(d) Dataset 1 – No effort limit.

(h) Dataset 2 – No effort limit.

Fig. 4. Pareto fronts obtained by MOABC.

Nevertheless, as was observed above, in no way can this be considered a rigorous comparative study because we do not know the hardware and software conditions under which the aforementioned literature algorithms were executed.

6. Conclusions and future work

In this paper, we have studied the Next Release Problem, and have proposed a multiobjective swarm intelligence approach (MOABC) to tackle real instances of the problem. Specifically, we have formally defined a constrained multiobjective version of the requirement selection problem in which different types of interaction between requirements and several development effort boundaries were considered.

We have evaluated MOABC in terms of several quality indicators by comparing the results with other approaches published in the literature (ACO, NSGA-II, GRASP). We found that our proposed approach can obtain the best sets of requirements in an efficient way. In particular, it yields sets of non-dominated solutions more of which are in the Pareto fronts, with a greater hypervolume and a lower spread between the solutions. In sum therefore, the results indicate that our technique can efficiently generate high quality sets of requirements that can allow software engineers to

make decisions about the set of features that have to be included in the next release of a software package.

Eight different problem instances taken from two real-world datasets were used to check the effectiveness of our evolutionary algorithm. These datasets included different numbers of requirements, client priorities, and requirement interactions, and both of them had previously been employed by other published work so that we were able to make comparisons with the results of the present study.

Due to the good results obtained with MOABC, in future work, it might be interesting to work with other multiobjective approaches based on swarm intelligence that can be applied to the problem. An example might be a hybrid version of MOABC with some other multiobjective approach. It could also be interesting to study the use of this search technique applied to larger real-world datasets. To this end, it would be necessary to develop a MONRP dataset generator for the systematic generation of instances. This improvement should be addressed with the help of software design experts, because the question of the interactions among the requirements in a complex software package is in no way trivial. Other formulations of the problem, considering other objectives or more complex constraints, would also be interesting lines for future work.

Acknowledgments

This research was partially funded by the Spanish Ministry of Science and Innovation under the Contract TIN2012-34945, by the Department of Employment, Enterprise and Innovation of the Government of Extremadura (GR10129), and by the European Regional Development Fund.

References

[1] A.J. Bagnall, V.J. Rayward-Smith, I. Whittley, The next release problem, Inform. Softw. Technol. 43 (14) (2001) 883–890.
[2] P. Baker, M. Harman, K. Steinhofel, A. Skaliotis, Search based approaches to component selection and prioritization for the next release problem, in: Proceedings of the 22nd IEEE International Conference on Software Maintenance, Washington, DC, 2006, pp. 176–185.
[3] P. Carlshamre, K. Sandahl, M. Lindvall, B. Regnell, An industrial survey of requirements interdependencies in software product release planning, in: Proceedings of 5th IEEE International Symposium on Requirements Engineering (RE 2001), IEEE Computer Society, Toronto, 2001, pp. 84–93.
[4] A. Charan Kumari, K. Srinivas, M.P. Gupta, Software requirements optimization using multi-objective quantum-inspired hybrid differential evolution, in: O. Schüze et al. (Eds.), EVOLVE – A Bridge between Probability, AISC 175, Springer, New York, 2013, pp. 107–120.
[5] J.M. Chaves-González, M.A. Vega-Rodríguez, J.M. Granado-Criado, A multiobjective swarm intelligence approach based on artificial bee colony for reliable DNA sequence design, Eng. Appl. Artif. Intell. 26 (9) (2013) 2045–2057.
[6] C.A.C. Coello, G.B. Lamont, D.A.V. Veldhuizen, Evolutionary Algorithms for Solving Multiobjective Problems, Springer, New York, 2007.
[7] K. Deb, Multi-Objective Optimization Using Evolutionary Algorithms, John Wiley & Sons, New Jersey, 2001.
[8] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast elitist multi-objective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2000) 182–197.
[9] J. Demsar, Statistical comparison of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.
[10] M. Dorigo, T. Stützle, Ant Colony Optimization, MIT Press, Cambridge, 2004.
[11] J. Durillo, Y. Zhang, E. Alba, A.J. Nebro, A study of the bi-objective next release problem, Empirical Softw. Eng. 16 (2009) 29–60.
[12] A. Finkelstein, M. Harman, S.A. Mansouri, J. Ren, Y. Zhang, A search based approach to fairness analysis in requirement assignments to aid negotiation, mediation and decision making, Requir. Eng. J. 14 (2009) 232–245 (RE '08 Special Issue).
[13] A. Finkelstein, M. Harman, S.A. Mansouri, J. Ren, Y. Zhang, "Fairness analysis" in requirements assignments, in: Proc. 16th IEEE Int. Requirements Engineering Conf., Washington, DC, 2008. pp. 115–124.
[14] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP Completeness, Freeman, New York, 1990.
[15] D. Greer, G. Ruhe, Software release planning: an evolutionary and iterative approach, Inform. Softw. Technol. 46 (4) (2004) 243–253.
[16] M. Harman, A. Mansouri, Y. Zhang, Search based software engineering: trends, techniques and applications, ACM Comput. Surv. 45 (1) (2012) 11.
[17] H. Jiang, J. Zhang, J. Xuan, Z. Re, Y. Hu, A hybrid ACO algorithm for the next release problem, in: Proc. of the 2nd International Conference on Software Engineering and Data Mining, Chengdu, 2010, pp. 166–171.
[18] D. Karaboga, B. Akay, A survey: algorithms simulating bee swarm intelligence, Artif. Intell. Rev. 31 (2009) 61–85.
[19] D. Karaboga, B. Basturk, Artificial Bee Colony (ABC) optimization algorithm for solving constrained optimization problems, in: P. Melin, et al. (Eds.), IFSA 2007, LNAI 4529, 2007, pp. 789–798.
[20] J. Karlsson, Software requirements prioritizing, in: Proceedings of the Second International Conference on Requirements Engineering (RE '96), Colorado Springs, 1996, pp. 110–116.
[21] M.S. Kıran, E. Özceylan, M. Gündüz, T. Paksoy, Swarm intelligence approaches to estimate electricity energy demand in Turkey, Knowl.-Based Syst. 36 (2012) 93–103.
[22] J. Knowles, D. Corne, The Pareto archived evolution strategy: a new baseline algorithm for Pareto multiobjective optimization, Proc. Congr. Evol. Comput. (CEC) (1999) 98–105.
[23] G. Li, P. Niu, Y. Ma, H. Wang, W. Zhang, Tuning extreme learning machine by an improved artificial bee colony to model and optimize the boiler efficiency, Knowl.-Based Syst. 67 (2014) 278–289.
[24] M. Lozano, A. Duarte, F. Gortázar, R. Martí, A hybrid metaheuristic for the cyclic antibandwidth problem, Knowl.-Based Syst. 54 (2013) 103–113.
[25] A.J. Nebro, J.J. Durillo, F. Luna, B. Dorronsoro, E. Alba, Mocell: a cellular genetic algorithm for multiobjective optimization, Int. J. Intell. Syst. 24 (7) (2009) 726–746.
[26] K. Price, R. Storn, Differential evolution – a simple evolution strategy for fast optimization, Dr. Dobb's J. 22 (4) (1997) 18–24.
[27] A. Rubio-Largo, M.A. Vega-Rodriguez, J.A. Gómez-Pulido, J.M. Sánchez-Pérez, Multiobjective approach based on artificial bee colony for the static routing and wavelength assignment problem, Soft Comput. 17 (2) (2013) 199–211.
[28] J. Sagrado, I.M. del Águila, F.J. Orellana, Multi-objective ant colony optimization for requirements selection, J. Empirical Softw. Eng. (2014), http://dx.doi.org/10.1007/s10664-013-9287-3.
[29] S. Santander-Jiménez, M.A. Vega-Rodríguez, Applying a multiobjective metaheuristic inspired by honey bees to phylogenetic inference, BioSystems 114 (1) (2013) 39–55.
[30] M. Silva-Maximiano, M.A. Vega-Rodríguez, J.A. Gómez-Pulido, J.M. Sánchez-Pérez, A new multiobjective artificial bee colony algorithm to solve a real-world frequency assignment problem, Neural Comput. Appl. 22 (7) (2013) 1447–1459.
[31] E. Simmons, Requirements triage: what can we learn from a "medical" approach?, IEEE Softw 21 (4) (2004) 86–88.
[32] A. Singh, An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem, Appl. Soft Comput. 9 (2) (2009) 625–631.
[33] K. Schwaber, M. Beedle, Agile Software Development with Scrum, Prentice Hall, 2001.
[34] J.T. Souza, C.L. Brito Maia, T.N. Ferreira, R.A. Ferreira, M.M. Albuquerque, An ant colony optimization approach to the software release planning with dependent requirements, in: Proc. of the 3rd Int. Symposium on Search Based Software Engineering (SBSE'11), 2011, pp. 142–157.
[35] K.E. Wiegers, Software Requirements, Microsoft Press, Redmond, WA, USA, 2003.
[36] Y. Zhang, M. Harman, S.A. Mansouri, The multi-objective next release problem, in: Proc. of the 9th Conference on Genetic and Evolutionary Computation (GECCO '07), New York, 2007, pp. 1129–1137.
[37] E. Zitzler, L. Thiele, Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach, IEEE Trans. Evol. Comput. 3 (4) (1999) 257–271.