Parallel multi-objective artificial bee colony algorithm for software requirement optimization

Hamidreza Alrezaamiri, Ali Ebrahimnejad & Homayun Motameni

Requirements Engineering

ISSN 0947-3602

Requirements Eng
DOI 10.1007/s00766-020-00328-y

ONLINE FIRST

Springer

Requirements Engineering
https://doi.org/10.1007/s00766-020-00328-y

ORIGINAL ARTICLE

Parallel multi-objective artificial bee colony algorithm for software requirement optimization

Hamidreza Alrezaamiri[1] · Ali Ebrahimnejad[2] · Homayun Motameni[3]

Received: 19 November 2017 / Accepted: 17 January 2020

Abstract

In incremental software development approaches, the product is developed in various releases. In each release, a set of requirements is proposed for the development. Usually, due to lack of funds, lack of time and dependency between requirements, there is no possibility to develop all the required requirements. There are two conflicting objectives for choosing an optimal subset of the requirements: increasing customer satisfaction and reducing development costs. This problem is known as the next release problem (NRP) and is categorized as an NP-hard problem. Unlike the standard version of the NRP, we formulate this problem as a restricted multi-objective optimization problem. There exist metaheuristic algorithms for solving this problem performed as serials. In this paper, we introduce a parallel algorithm based on the master–slave model in order to improve the quality of the solutions. Based on the criteria of multi-objective problems, the quality of the obtained solution is compared with several metaheuristic algorithms. Two scenarios and two different datasets are used for experiments. Results indicate that the proposed method in the first scenario would highly improve the quality of solutions. Moreover, the method reduces execution time significantly through improvement in the quality of the solution in the second scenario.

Keywords Software requirements · Multi-objective algorithm · Next release problem · Master–slave model

1 Introduction

In the past years, we have seen the growth of complexity and software capabilities. Large software systems have sensitive and complex development processes. One of the most important stages of the software development is the "recognition requirements" stage. Carelessness at this stage of the software development may lead to an increase in estimated costs of product development, increase in product development time, reducing product quality, declining customer satisfaction and even project failure [22]. Therefore, it has always been tried to recognize customer needs and demands as fast and accurately as possible using new methods and to consider them in the software product [16, 35]. In some software development methods such as incremental development methods, the product is developed in several releases. In these methods, like in agile methods, the developer faces a set of requirements that need to be developed in each release [13]. Due to technical problems in the development of some requirements, lack of time, lack of funds and conflicting requirements of projects, it is almost impossible to develop all proposed requirements in every release [29]. Therefore, it is important to choose an optimal subset of the developmental requirements that can provide maximum satisfaction to customers, with minimal time and cost for the development team. In large projects with a large number of requirements, it is very difficult to choose the optimal subset. That is why it is essential to propose a method to determine the optimal subset of requirements and to present the result of the method to software engineers to help them in decision making. Choosing the optimal subset of requirements which is called the next release problem (NRP) is a NP-hard problem because of having at least two conflicting objectives. The

✉ Ali Ebrahimnejad
a.ebrahimnejad@qaemiau.ac.ir

Hamidreza Alrezaamiri
hamidreza.alreza@baboliau.ac.ir

Homayun Motameni
motameni@iausari.ac.ir

[1] Department of Computer Engineering, Babol Branch, Islamic Azad University, Babol, Iran

[2] Department of Mathematics, Qaemshahr Branch, Islamic Azad University, Qaemshahr, Iran

[3] Department of Computer Engineering, Sari Branch, Islamic Azad University, Sari, Iran

cost of software development is defined as the sum of individual developed requirements [36]. If customer satisfaction increases and cost reduction is taken into account, it is clear that by reducing the cost of software development, requirements are less developed and customer satisfaction reduces accordingly. On the other hand, by developing most of the proposed requirements, customer satisfaction increases, but it also leads to an increase in the cost of development. Because of the contradictory nature of these two objectives, traditional and single-objective optimization methods cannot find the optimal subset of requirements [38]. For this reason, the use of multi-objective evolutionary algorithms for solving this issue is necessary [1, 7]. In recent years, researchers have used different multi-objective evolutionary algorithms for solving NRPs. All these algorithms were implemented serially and obtained solutions with different qualities due to their ability to find the solution space. Examples of these algorithms are greedy randomized adaptive search procedure (GRASP) [37], non-dominated sorting genetic algorithm (NSGA-II) [8], ant colony system (ACS) [10], multi-objective artificial bee colony (MOABC) [4] and differential evolution with Pareto tournament (DEPT) [3] algorithms. Among them, the multi-objective ABC algorithm produced the highest quality solutions. In this paper, we intend to use the parallel processing technique and execute several MOABC algorithms [4] in the master–slave model simultaneously in order to obtain higher-quality solutions than those of the serial mode.

Due to the complexities and limitations of the NRP, serial metaheuristic algorithms may not be able to probe the problem space well and provide acceptable solutions in a short time. We run some MOABC algorithms in parallel with a new technique to alleviate this problem. Because each MOABC algorithm has a different random initial population and also operators have random choices, it may be possible to find solutions in each run of the MOABC algorithm that other algorithms could not detect. In this technique, a set of non-dominated solutions (NDSs) is derived from each algorithm. Eventually, these sets are merged to form the overall solution set. The overall solution set of the problem derived from the implementation of this technique based on multi-objective criteria such as the number of NDSs, HV and the spread of solutions has better quality than the set of solutions obtained from the serial implementation of a MOABC algorithm. Also, the presented technique can reduce the runtime by adopting some policies which will be discussed later. This is the first time a parallel algorithm is used for search-based software engineering (SBSE) [20] to solve the multi-objective NRP. Results of the parallel execution of the proposed algorithm are compared with those of serial parallel algorithms such as NSGA-II, ACO, MOABC and GRASP. The results confirm that the proposed approach yields higher-quality solutions than the serial executions.

The rest of the paper is organized as follows. In Sect. 2, we present the literature review. In Sect. 3, the multi-objective NRP is discussed and formulated. In Sect. 4, the master–slave parallel processing model and the proposed method are introduced. In Sect. 5, the experiments and results obtained are analyzed. Finally, in Sect. 6, implications of the study, future works in this domain and conclusion are presented.

2 Literature review

The selection of an optimal subset of requirements for the development in the next software release is a very complex task. In fact, it is considered as one of the NP-hard-related problems. This means that traditional methods cannot find the optimal solution to this problem at the correct time. Search-based software engineering (SBSE) is one of the research areas in which search-based optimization algorithms are used to solve problems related to the software engineering. The NRP is also covered in this area [20]. Pitangueira et al. [33] carried out classification, analysis and proper assessment of previous works on issues related to the selection and prioritization of the requirements of the software. Recently, suggested methods for solving the NRP were mostly in the form of metaheuristic optimization algorithms.

One of the first efforts to solve the NRP is done by Karlsson [24] in which the author used two methods, namely analytical hierarchy process (AHP) and quality function deployment (QFD), to select and prioritize software requirements. In the QFD method, the requirements were prioritized, and in the AHP method, the requirements were classified. In large projects with a large number of requirements, these methods are not effective due to the many comparisons drawn and long run times. Bagnall et al. [2] introduced three methods for selecting the optimal subset of requirements. The first method used linear programming to obtain the exact solution of the problem. In the second method, three greedy algorithms were used for solving the same problem. The third method used two local search algorithms, namely hill climbing and simulated annealing. In NRPs with a low number of requirements, the linear programming method finds the exact solution of the problem at the correct time. However, the linear programming method cannot find the solution in a reasonable time when the size of problem is large. The other two methods presented in this article could not find qualified solutions due to their low search capabilities in the solution space.

Metaheuristic optimization algorithms used to solve the NRP fall into two categories: single objective and multi-objective. In the single-objective optimization algorithms, the algorithm turns the multi-objective problem into a single-objective problem by assigning a specific weight to each goal and merging them. Greer and Ruhe [19] used the genetic algorithm to select the optimal subset of requirements. De Souza et al. [12] solved the NRP using the ACO algorithm. Jiang et al. [23] combined the ACO algorithm with a local search algorithm to select the optimal subset of requirements. The aim of this approach was to increase the ability of the algorithm to find quality solutions.

In recent years, the NRP has been solved mostly in the form of a multi-objective optimization problem. In multi-objective problems, none of the objectives is superior to others. In fact, the importance of all objectives is the same. The NRP was first formulated as multi-objective by Zhang et al. [42]. Durillo et al. [14] analyzed the multi-objective optimization algorithms, Pareto archived evolution strategy (PAES) [25], NSGA-II [8] and multi-objective cellular genetic algorithm (MOCell) [30] to solve the NRP. Del Sagrado et al. [10] proposed the multi-objective ACS algorithm to solve this problem. In that paper, the interaction between requirements was incorporated in the datasets for the first time. Chaves-González and Pérez-Toledano [3] used the differential evolution algorithm to solve this problem and considered the interaction between requirements. Parejo et al. [31] used the integer linear programming method in both single-objective and multi-objective models. These models were able to find the exact solution in the single-objective model and small-sized multi-objective problems. But their proposed method in large multi-objective problems will not have proper run time. Chaves-González et al. [4] applied the MOABC algorithm to select the optimal set of requirements. High-quality results were obtained by this algorithm. Here, the interaction between requirements was considered and two datasets assessed in the article were made available. In the present study, we intend to run the MOABC algorithm introduced in Chaves-González et al. [4] through the parallel way and in the master–slave model. The union of the results obtained by running all algorithms in parallel gives a significant and quality solution set. So far, no parallel algorithm has been introduced for solving the NRP. For this reason, the quality of the results obtained by the proposed method is compared with those of the serial algorithm introduced in previous studies.

3 The problem of selecting the optimal subset of multi-objective requirements

Single-objective problems are problems with inherently one objective. If there are several different objectives in the problem, a main objective will be formed when certain weights are assigned to each objective, and they are then combined into a single objective [6]. Suppose there are three goals, namely f_1, f_2 and f_3 in a problem. In addition, we intend to aggregate them into a single-objective problem through assigning certain weights to these objectives. In formula (1), it can be seen that the combination of these three objectives forms the main objective of the problem:

$$W_1 * f_1 + W_2 * f_2 + W_3 * f_3 = F. \tag{1}$$

In single-objective problems, there is usually a unique solution to the problem. However, in multi-objective problems, the objectives are not combined together, and there is no unique solution. Rather, there is a set of solutions, none having superiority over another. This solution set is referred to as non-dominated solutions. This solution set in the multi-objective problems figured out the Pareto frontier [31]. Suppose we have an n-objective problem, and let $x = [x_1, x_2, \dots, x_n]$ and $y = [y_1, y_2, \dots, y_n]$ denote the two feasible solutions of this problem. Solution x is dominant over solution y if x is better than or equal to y for all n objectives. Also, there should be at least one objective x_i where x is strictly better than in y. Otherwise, the two objectives are non-dominant, i.e., neither objective is dominant over another. Three solutions are shown in Fig. 1. The complete solutions to the objectives f_1 and f_2 are at point 0. Solution A is dominant over solution C because, for every two objectives, solution A is of higher quality. $f_1(A) < f_1(C)$ and $f_2(A) < f_2(C)$. However, the two solutions A and B are non-dominated by one another for each one is dominant over the other in on objective.

3.1 The problem of selecting the requirements

In all software engineering projects, proper selection and elicitation of requirements are an important and critical task. To identify and elicit the requirements properly in the early stages of product development, various methods

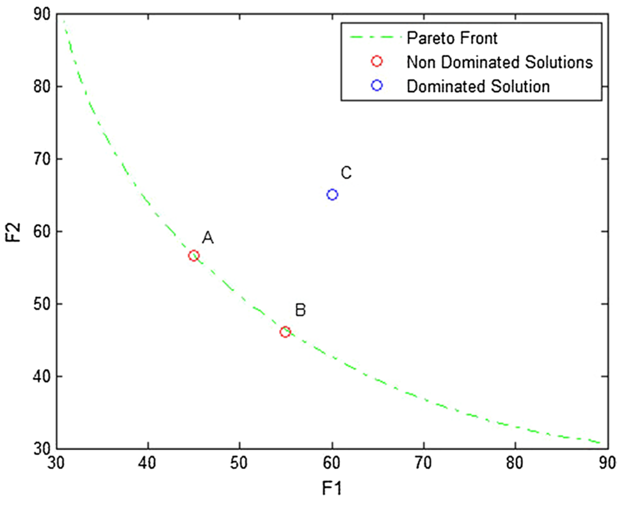

Fig. 1 The Pareto front diagram

have been introduced [9, 17, 27, 32]. However, due to the presence of numerous and sometimes contradictory factors, proper selection of the requirements that need to be developed poses a great challenge to software engineers. In agile development methods in which a product is developed in various releases, the very challenge becomes even more serious [26]. Problems such as the presence of customers with different views and interests, difference between the levels of importance of customers for developer companies, factors affecting the market and companies' tendencies to consider the needs of new customers make selection very difficult. Despite all these problems, software engineers have to select an optimal subset of proposed requirements allowing for maximum satisfaction for customers, as well as the lowest cost of development for a software company [10]. In addition, the presence of interactions between requirements creates a serious limitation in the selection of requirements. There are different types of interactions and dependencies between requirements which appear to impose restrictions on the problem. Del Sagrado et al. [10] classified the interactions between requirements into four major groups:

- *Implication* $r_i \Rightarrow r_j$. A requirement r_j cannot be implemented if a requirement r_i has not been chosen yet.
- *Combination* $r_i \oplus r_j$. A requirement r_i cannot be chosen separately from a requirement r_j.
- *Exclusion* $r_i \otimes r_j$. A requirement r_i cannot be chosen together with a requirement r_j.
- *Modification* The development of the requirement r_i implies that some other requirements change their satisfaction or implementation effort.

There is a major advantage in considering the NRP as a multi-objective one rather than a weighted single-objective one. In the multi-objective problem, software engineers find a set of non-dominated solutions rather than one good solution. The solution set makes up the Pareto front which enables the developer team to analyze the cost and satisfaction. For example, the developer team can conclude from the Pareto front graph how much it would additionally cost it if it requests a 15 percent increase in customer satisfaction. It can also compute the amount of saved money by decreasing customer satisfaction by 5 percent [42].

3.2 Multi-objective NRP formulation

In the multi-objective NRP (MONRP), n requirements are proposed for development in the next release. We consider this set as $R = \{r_1, r_2, \dots, r_n\}$. Each requirement has a cost of development represented as $E = \{e_1, e_2, \dots, e_n\}$. Assume that n requirements of the set R were proposed by m clients as $C = \{c_1, c_2, \dots, c_m\}$. Each client has an importance level for the company. The importance level is assumed to be $W = \{w_1, w_2, \dots, w_m\}$.

It is assumed that each client in C gives a value to each requirement in R. The value that a requirement r_j has for a particular client c_i is given by an amount $v_{ij} \geq 0$. A matrix of $m \times n$ holds all the importance amounts v_{ij}. Total satisfaction s_j of a given requirement r_j is calculated as the weighted sum of its values for all the clients considered and can be expressed as indicated in Eq. (2). The set of the total satisfactions calculated in that way is denoted by $S = \{s_1, s_2, \dots s_n\}$.

$$s_j = \sum_{i=1}^{m} w_i * v_{ij} \quad j = 1, 2, \dots, 6, \tag{2}$$

where j is the index of requirements set. Since there are six requirements, the index j is varied from 1 to 6.

In the MONRP, each solution is displayed as an n-cell vector where the ith cell on the vector corresponds to the ith requirement in the set R, for $i = 1, 2, \dots, n$. A solution is shown in Fig. 2.

In each solution, each requirement selected to be developed in the next release takes value 1 in its corresponding cell and value 0 otherwise. We should be looking to find solutions that are non-dominated regarding the satisfaction and cost objectives. Of course, the suggested solutions should meet the limitations considered for the problem. The first limitation is the dependence between requirements which were explained in the previous section. The second limitation is the maximum amount of development costs which is considered by the developer company. The maximum cost of a product denoted by l_c is calculated by formula (3) where x is a solution of this problem.

$$\sum_{j \in x} e_j \leq l_c. \tag{3}$$

Each solution is valid if it meets the two limitations, namely the dependency between requirements and the maximum cost of development [4].

3.3 The multi-objective ABC algorithm and its applications

The results obtained by the multi-objective ABC algorithm [4] to solve the NRP on the basis of multi-objective problem measurement criteria were of higher quality than those of other algorithms for solving this problem. Their analysis of the results showed that the proposed algorithm can efficiently generate high-quality solutions. These were evaluated by

r_1	r_2	r_3	…	r_n
0	1	0		1

Fig. 2 Structure of a solution

comparing them with different proposals (in terms of multi-objective metrics). The results generated by the ABC algorithm surpass those generated in other relevant work in the literature like NSGA II, GRASP and ACS. For example, their 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. Because of the complexity of the NRP, we intend to continue to improve the quality of the solutions to this problem. In this article, we intend to perform the simultaneous running of several multi-objective ABC algorithms in parallel in a master–slave model, and obtain a higher-quality solution set compared to that of the serial MOABC algorithm by means of the union of their non-dominated archive of solutions. In this section, the MOABC algorithm [4] which is the basis of our article is described. The algorithm is composed of three main steps, and each step is related to the activity of one type of bees. In this algorithm, there are three types of bees: employed, onlooker and scout bees. Employed bees search the environment to find a new food source and return the hive upon finding one. Through dance performances, they inform the onlooker bees of the location of the food source. Every onlooker bee in the hive selects a food source at random after obtaining the employed bees' information in order to launch more searches in that area. The better the food source, the greater the chances of selection by onlooker bees. The third type of bees is scout bees. Employed bees that abandon their food source become a scout bee. They move in the area randomly to find another food source. In [4], the authors combined two interesting ideas from other famous multi-objective algorithms with their own algorithm. Their first idea was to utilize the non-dominated sorting technique from the NSGA-II algorithm [8]. According to this technique, the population solutions were grouped according to dominance rank. The second idea of the PAES algorithm [25] was to use the concept of a non-dominated solution archive in order to store the best solutions found during implementation in their archives.

In Algorithm 1, MOABC pseudocode is shown to solve the NRP [4]. In this algorithm, NEB is the number of employed bees; NOB is the number of onlooker bees; Prob is the mutation probability; LimitForScout is the parameter related to the scout bees. The related algorithm generates a random solution for every employed bee in the first step so that the colony is shaped (second line in algorithm 1). In the colony created, all limitations defined in the problem are met so that the solutions generated are valid. After creating the colony, solutions are evaluated based on formulas (2) and (3) which determine their level of satisfaction and cost, respectively (third and fourth lines in Algorithm 1). Colony-based solutions are sorted according to non-dominated ranks and crowding distance. If a solution has less dominance ranking than another solution, or if equal, it has a greater crowding distance, considered a better solution. Before the main loop of algorithm starts, the non-dominated solution set (NDS_archive) is set to null. The main loop of the algorithm (lines 6 to 30) includes three steps of searching, one step of sorting, and storing the solutions. At the stage of the employed bees' search (lines 7 to 11), each employed bee creates mutations on its corresponding solution to create a new solution. To ensure that the solutions resulting from the mutation do not violate the limitations of the problem, a function validates them. It corrects the solution so that the solutions will always remain valid [4]. If the new solution is of higher quality (regarding the dominance and crowding distance criteria) than the parent solution, it will be replaced and would otherwise be discarded. After the employed bees' work (in line 13) is finished, a vector including the probability of selection of any food source is created for the next step. The higher the quality of the food source, the more likely its being selected. Lines 15 to 19 are related to the stage of the onlooker bees search. Each onlooker bee that is randomly generated based on the probability vector at the end of the previous stage selects a food source to search and apply the practice of mutation. The solution resulting from mutation is checked to ensure its validity, and if necessary, it is repaired. Like the previous stage, if the new solution is of higher quality than the parent one, it will be replaced and would otherwise be discarded. Lines 20 to 25 are related to scout bees.

```
1: % Generate and evaluate the colony C
2: C = random Generation Of Employed Bees (|NEB)
3: C = fast Non Dominated Sort (C, NEB)
4: C = crowding Distance Calculation (C, NEB)
5: NDS _ archive = empty
6: while (not stop condition)
7:     % Employed bees search stage
8:     for i = 1 to NEB
9:        mutated Bee = explore Solution ($C_i$, Prob)
10:       $C_i$ = update Employed Bee ($C_i$, mutated Bee)
11:    end
12:    % Generate a probability vector using the employed bees
13:    prob Vector = generate Probability Vector (C, NEB)
14:    % Onlooker bees search stage
15:    for i = NEB to NOB
16:       $C_{chosen}$ = choose Employed Bee (prob Vector, C, NEB)
17:       mutated Bee =  explore Solution ($C_{chosen}$, Prob)
18:       $C_{chosen}$ = update Onlooker Bee ($C_{chosen}$, mutated Bee)
19:    end
20:    %Scout bees search stage
21:    for i = 1 to NEB
22:       if $C_i$. Iterations > limit For Scout then
23:          $C_i$ = replace With Scout Bee ()
24:       end
25:    end
26:    %Sort the colony by quality and update the Pareto
       solutions archive
27:    C = fast Non Dominated Sort (C)
28:    C = crowding Distance Calculation (C)
29:    NDS _ archive = update NDS Archive (C)
30: end
```

Each employed bee that is not successful as many as the LimitForScout parameter value in its local queries, leaves the food source and turns into a scout bee. Searching the problem space, the bee produces a new random solution, and from then on, works on this food source. In lines 27 and 28, the colony solutions are sorted based on non-dominated ranks and crowding distance. In line 29, the non-dominated solutions are stored in NDS_archive. The main loop is repeated until the termination condition is reached. In each iteration, the non-dominated solutions are stored in NDS_archive. Due to its high ability, the ABC algorithm is used in many problems. The algorithm has many applications in solving various problems either as multi-objective or as single objective. In [15, 39, 40], the single-objective (ABC) algorithm and in [5, 11, 34], the multi-objective (ABC) algorithm were used to solve the problem.

4 Parallel multi-objective artificial bee colony

In the last section, the multi-objective ABC algorithm was introduced as a serial algorithm. The metaheuristic algorithms in spite of all their ability run into trouble in some of the major and complex problems, and there is always a need for their improvement. In this section, we intend to improve the solutions of the NRP using the master–slave parallel programming model. Therefore, the parallel multi-objective artificial bee colony (PMOABC) is introduced. In what follows, detailed description of this algorithm is given. We first describe the master–slave model.

4.1 The master–slave model

In recent years, the use of parallel programming for metaheuristic algorithms has gained greater attention, especially when serial metaheuristic algorithms did not have satisfactory runtime and solution quality. Parallel evolutionary algorithms can be classified from different aspects.

From an architectural perspective, these algorithms can fall into either the shared memory or distributed memory architecture. In the shared memory architecture, algorithms run in parallel on the cores of a computer or threads of a processor, and communication is established through writing on the shared memory. However, in the distributed memory architecture, algorithms run on different computers, and communication between the computers is achieved through message passing [18].

From another aspect, classification of these algorithms fall into two categories, namely distributed population models and distributed dimension models. Distributed population models include the master–slave model, the Iceland model, the cell model and the hierarchical model which parallelize an evolution task at population, individual or operation levels. The second group includes distributed dimensions models such as coevolution and multi-agent models which focus on reducing the problem size [18].

In the master–slave model, a processor or core is selected as the master, and the rest of the core serves as the slaves for the master core. The master usually delegates the hard work or heavy computing tasks to the slaves and waits for results of their works. The master–slave model is shown in Fig. 3. In this figure, the master sends individuals to slaves for determining their fitness and then receives the fitness amounts from slaves.

In recent years, metaheuristic algorithms which were usually run in parallel in the master–slave model appeared as two common forms. In the first form, the master core executed the main part of the algorithm. It also assigned the tasks of calculating the fitness of solutions and updating solutions to the core of slaves and would then wait for a response from them [21]. The master–slave model might appear inefficient for problems where the processing time related to the computations assigned to the slaves is not high. This is because communication between masters and slaves imposes more time. In the second common form of using the master–slave model, the master would carry out all computations related to the algorithm in each iteration, and then, it would assign the obtained solutions to the slaves for further searches (local search) and would receive the results from them. A solution replaces the parent solution if a higher-quality solution is obtained in the local search [41]. The performance of distributed computations is measured according to two criteria, namely speedup and efficiency. The speedup rate equals the serial execution time of the algorithm divided by the parallel execution time of the algorithm. The amount of efficiency is defined as the speedup rate divided by the number of cores (or processors) participating in the process. In practice, these two criteria may be affected due to the computational overhead, low speed communications or slowest CPU performance [18].

In this paper, a new technique is used to parallelize the master–slave model. The master–slave model is one of the most popular parallelization models. The reason of choosing this model in our proposed approach is to utilize all the CPU hardware power. Hence, it is possible to use all the cores in the CPU to achieve the most desirable solutions and to reduce the runtime.

4.2 The proposed method

In the previous section, the multi-objective ABC algorithm was described. We have seen that the result of the run of this algorithm is a set of solutions called non-dominated solutions (NDSs). The solution set forms the Pareto front in the multi-objective problems. The number of non-dominated

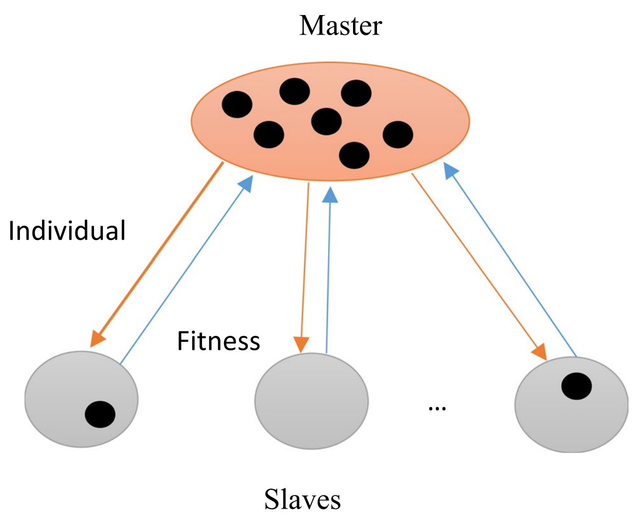

Fig. 3 An example of the master–slave model

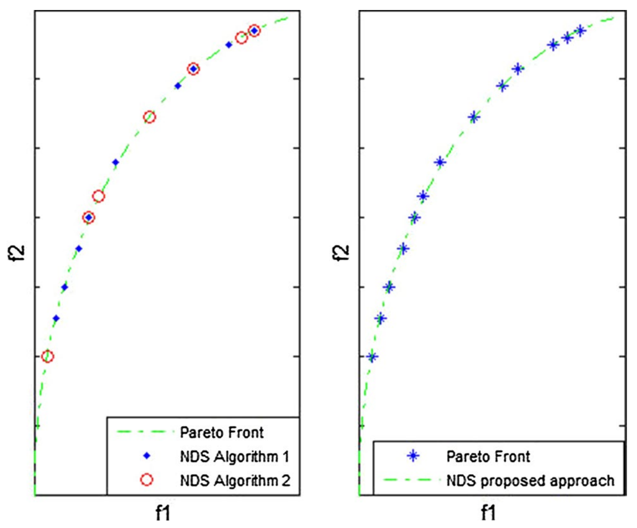

Fig. 4 An example of taking the union of the set of NDSs

solutions is considered one of the criteria for evaluation of multi-objective algorithms. Thus, the more the number of non-dominated solutions obtained by the algorithm, the better. It also has a positive effect on other criteria for evaluation of the algorithm. Therefore, in the proposed method, we intend to create a parallel algorithm similar to the master–slave model in order to obtain higher-quality solutions from the serial algorithms. In this method, we intend to use the master–slave model in the shared memory architecture with a new innovation. In the proposed method, one of the cores (e.g., core 1) becomes the master and the other cores are set as slaves. All cores (masters and slaves) run a multi-objective (ABC) algorithm on themselves completely under a same condition. That is, during the execution of all cores, the number of iterations, the number of colony solutions and other parameters of the algorithm are equal. Each core will create a colony randomly. At the end of the execution of the MOABC algorithm, all slaves send their NDSs to the master core. The master core arranges the NDS sets of slaves and NDS resulting from the execution on its own core. It further removes repeated solutions so that we could have a general solution set of non-dominated solutions without any repetitions. In this case, the number of unique solutions will be far greater than the NDS set obtained from a serial algorithm and will result in a set of higher-quality solutions to the problem. To better understand the proposed method, the results of concurrent execution of the two multi-objective algorithms are shown in Fig. 4. In this example, the objective f_1 is minimized and the objective f_2 is maximized. In Fig. 4, the Pareto front is shown by a green dashed line on the left; the NDSs related to Algorithm 1 (9 solutions) are shown by blue dots; the NDSs related to Algorithm 2 (7 solutions) are shown by red circles. In Fig. 3 (right-hand side), the union of the set of NDSs derived from two algorithms produced a solution set with a higher number of non-dominated solutions (13 solutions). In the proposed method, the iterative solutions will be considered only once. The union of the set of NDSs derived from two algorithms is shown with blue stars in Fig. 4.

Unlike the conventional methods of using a master–slave model in which slaves were idle until they were assigned the execution of part of the algorithm, in the proposed method, the execution of a whole algorithm is assigned to slaves. In this case, the hardware capacity is fully used, and slaves (cores) are not idle. This innovation in the master–slave model results in the optimal use of the hardware. It should be noted that most computers are equipped with multi-processor and multi-core CPUs. Therefore, the proposed method is applicable on a wide range of computers, even personal ones [28].

In the proposed method, unlike other modes of the master–slave model, the number of communications between masters and slaves is very small. In fact, slaves send their set of NDS to the master only once at the end of the execution of algorithms. The master core will only wait for the set of NDSs from the slaves, at the end of the execution of its algorithm, and will further sort them.

The proposed algorithm is a synchronism method because the master must wait until the end of the execution of algorithm for receiving the solution set of all slaves. However, the waiting time would be very negligible. This is because the processing power of the cores of a computer is the same (homogeneity), and the algorithm being executed on all cores is exactly the same. In case the processing power of cores is different, the master must wait until the solution set of the slowest core is obtained. In this situation, taking into account the number of less iterations for slower cores can reduce the run time of this algorithm such that we do not have to be forced to bear their execution delay. In the proposed method, the work load is approximately equal on all cores. It is only the master core that has an extra responsible for sorting the incoming solution set and removing the iterative solutions. If we ignore the time the parallel processing overhead imposes, the time to send the NDSs to the master once and the time needed to remove the iterative solutions from the overall set of solutions, it is safe to say that the execution time of the algorithm in parallel mode does not differ from that of the serial algorithm, yet we will obtain a higher-quality set of solutions. We recall that our main purpose of proposing this method is to improve the quality of the solution set of the NRP. Figure 5 shows the flowchart of the proposed method.

The right-hand side of Fig. 5 shows n MOABC algorithms run parallel on separate cores. After the end of each algorithm, the NDS archive of that algorithm will be sent to the master's core. The master sorting out the NDS sets derived by the slaves determines the unique set of NDS as the final solution of the problem and displays the statistical indices.

The development team chooses one solution of the final NDS according to the cost and satisfaction level for development in the next release. In fact, each solution of the NDS corresponds to a subset of the proposed requirements. For example, the solution from NDS, which provides the highest level of satisfaction with a 20% cost limit, is $\{r_4, r_7, r_{11}, r_{12}, r_{19}, r_{33} r_{41}, r_{48}\}$.

In this problem, each solution is related to a subset of the requirements. In fact, when a development team chooses a solution, they should develop those related requirements. In the initial phase of each release, developer team faces with a set of requirements and should choose the best subset of the requirements for developing according to this method.

In the first step, the proposed method selects an optimal subset of requirements for development in the next release. The selected subset can help the development team to make a better decision. That is, the development team uses a smart decision-making approach instead of traditional and tasteful choices. After that, in the second step, the development team implements the selected subset. The amount the satisfaction of each requirement is obtained from clients surveys. If the client prioritization requirements have changed significantly during the development, the development team can make changes to the selected subset subject to the constraints (such as cost limit) are not violated.

Remark 4.1 In this paper, a new approach of using the master–slave model in a shared memory architecture has presented. In this approach, the load of the processing of the master and the slaves is almost equal. In a multi-core computer, with starting the PMOABC algorithm, a core is selected as the master core and the rest of the cores are chosen as slaves. By commanding the master, all the cores run the MOABC algorithm simultaneously and independently. Each core generates a random colony to run the MOABC algorithm. Here, the conditions for the execution of all algorithms are the same. After the run of the MOABC algorithm, the cores send the NDS sets to the master core. The master core obtains a total set of solutions based on the union of these NDS sets and removing the repeated solutions. Due to the complexity of the problem, in the run of each core, several unique solutions may be discovered that have not been discovered in the other cores run. The union of these sets of solutions creates a set with a greater number of NDS than the serial mode.

5 Experiments and results

In this section, experiments, datasets and scenarios evaluated for the problem are reviewed and explained. Then, the results of the proposed method are presented based on multiple criteria and further compared with those of similar works [4].

5.1 Datasets and test criteria

All tests are conducted on a system with an Intel core i7, 1.60 GHz processor, a 4 GB RAM and Windows 7, 64 bits. The software used to execute the proposed method is MATLAB version R2014b. Since we have used random algorithms in these tests, each algorithm is executed 100 times independently, and their mean, standard deviation and related results are presented. The proposed method has been tested on two datasets. In addition, four development cost limits (30%, 50%, 70% and 100% of the total cost of development) are considered on each dataset. Therefore, it can be said that the proposed method is tested with four samples from each dataset, or in fact with eight samples from the NRP. The first dataset is adopted from [19]. This dataset includes 20 requirements and 5 clients. In Table 1, the development cost of each requirement, the priority level of each requirement for each client and the interaction between the requirements are listed. The clients assigned a priority level in the range [1,5] to each requirement. These scores show the customer satisfaction rate toward that requirement for development in the next release. The lowest score given to each requirement is 1. In fact, this means that the client has the least interest in the development of this requirement in the next release. However, score 5 means that clients are most interested in the development of requirement in the next release. The development cost of each requirement is estimated to

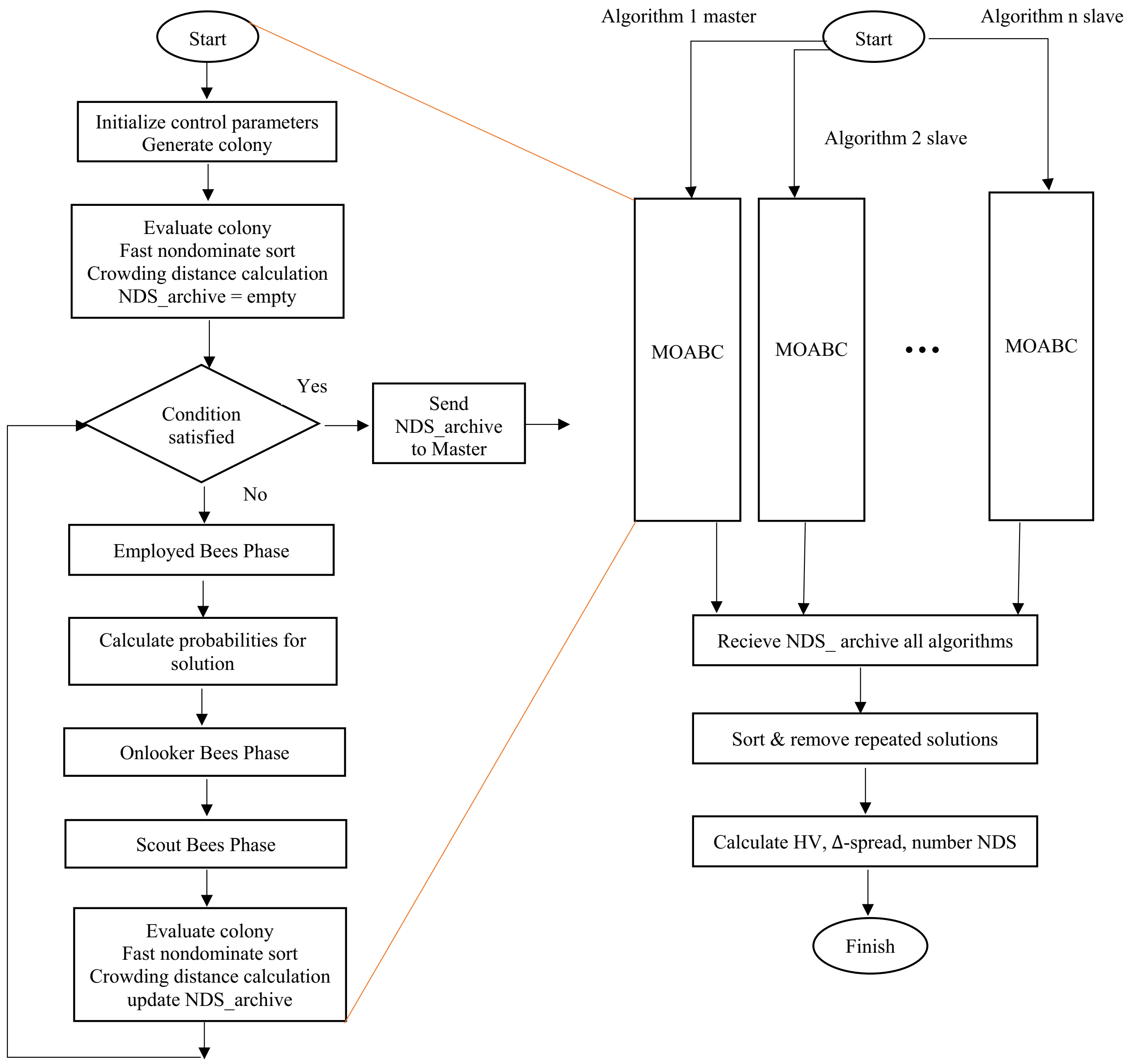

Fig. 5 Flowchart

range between 1 and 10 units. In this dataset, two interactions, namely implication and combination, are considered between the requirements.

The second dataset was introduced by Del Sagrado et al. [10] which included 100 requirements, 5 clients and 44 interactions between requirements. In Table 2, the scores of the priority assigned by each customer to each requirement, interactions between requirements and the development cost of each requirement that is a value between 1 and 20 are listed. This dataset is much more complicated than the previous one which is adopted from a real software development project. This dataset has 40 implication interactions and 4 combination interactions between requirements.

Each customer carries a different level of importance (weight) for the software developer company. The level of importance is used in formula (2). Table 3 shows the weight of each client in the two datasets. In this table, 1 is the least importance and 5 is the most.

Since the problem is defined in a multi-objective environment, the quality of the solutions for each test is evaluated in terms of the criteria of multi-objective problems. We use three indicators to assess the quality in accordance with previous works carried out for this problem.

The first quality index is hypervolume (HV). This index calculates the volume covered by the non-dominated set of solutions Q by means of formula (4).

Table 1 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}
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
Effort	1	4	2	3	4	7	10	2	1	3	2	5	8	2	1	4	10	4	8	4
Interactions	$r_4 \Rightarrow r_8$		$r_4 \Rightarrow r_{17}$		$r_8 \Rightarrow r_{17}$		$r_9 \Rightarrow r_3$		$r_9 \Rightarrow r_6$		$r_9 \Rightarrow r_{12}$		$r_9 \Rightarrow r_{19}$		$r_{11} \Rightarrow r_{19}$		$r_3 \otimes r_{12}$		$r_{11} \otimes r_{13}$	

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

HV measures the diversity and convergence of the obtained Pareto fronts. A Pareto front has a greater HV than the other if solutions in the better front are more widely distributed than in the other or some solutions in the better front dominate solutions in the other.

Algorithms with higher amounts of HV are better. In order to calculate this indicator, two reference points are required. Since the problem under consideration has two objectives, these points were $r_{\min}$ ($\mathrm{obj1}_{\min}$,$\mathrm{obj2}_{\min}$) and $r_{\max}$ ($\mathrm{obj1}_{\max}$,$\mathrm{obj2}_{\max}$). This points containing the maximum and minimum amounts for the two objectives. For maximum the hypervolume, both objective function amounts had to be normalized. The normalization points used for each dataset are presented in Table 4.

The second quality indicator was the spread achieved by the set of solutions (Δ—spread). This indicator calculates 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 formula (5).

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

where d_i is the Euclidean distance between two consecutive solutions, $\bar{d}$ is the mean distance between each pair of solutions, n is the 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 quality indicator was the number of non-dominated solutions (NDS) found. Pareto fronts with a greater number of non-dominated solutions are better.

We evaluate our proposed method in two scenarios. In the first scenario, the proposed method will terminate in all modes (with any number of execution cores, 2 or 4 cores) after 10,000 iterations. In this scenario, our purpose is to show the quality of the solutions obtained by the proposed method compared to the results given in Chaves-González et al. [4]. Therefore, all conditions and parameters of the algorithm are set according to those given in Chaves-González et al. [4]. The number of population in all algorithms is 40 individuals; the probability of mutation operator (Prob) is 0.5; and the value of the parameter limitForScout is 3. This scenario is applied on both datasets.

The second dataset has a long run time due to its complexity and largeness (having 100 requirements). The second scenario will be introduced for this complex dataset. In the second scenario, we intend to reduce the run time as the

Table 2 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}
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
Effort	16	19	16	7	19	15	8	10	6	18	15	12	16	20	9	4	16	2	9	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}
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	2	3	2	2	3	2	3	1	1	3	3	2	2	1	1	2	1
Effort	2	10	4	2	7	15	8	20	9	11	5	1	17	6	2	16	8	12	18	5
	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}
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
Effort	6	14	15	20	14	9	16	6	6	6	6	2	17	8	1	3	14	16	18	7
	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}
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
Effort	10	7	16	19	17	15	11	8	20	1	5	8	3	15	4	20	10	20	3	20
	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}
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

Table 2 (continued)

	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
Interactions	$r_2 \Rightarrow r_{24}$			$r_3 \Rightarrow r_{26}$			$r_3 \Rightarrow r_{27}$		$r_3 \Rightarrow r_{28}$		$r_3 \Rightarrow r_{29}$		$r_4 \Rightarrow r_5$		$r_6 \Rightarrow r_7$		$r_7 \Rightarrow r_{30}$		$r_{10} \Rightarrow r_{32}$	
	$r_{10} \Rightarrow r_{33}$			$r_{14} \Rightarrow r_{32}$			$r_{14} \Rightarrow r_{34}$		$r_{14} \Rightarrow r_{37}$		$r_{14} \Rightarrow r_{38}$		$r_{16} \Rightarrow r_{39}$		$r_{16} \Rightarrow r_{40}$		$r_{17} \Rightarrow r_{43}$		$r_{29} \Rightarrow r_{49}$	
	$r_{29} \Rightarrow r_{50}$			$r_{29} \Rightarrow r_{51}$			$r_{30} \Rightarrow r_{52}$		$r_{30} \Rightarrow r_{53}$		$r_{31} \Rightarrow r_{55}$		$r_{32} \Rightarrow r_{56}$		$r_{32} \Rightarrow r_{57}$		$r_{33} \Rightarrow r_{58}$		$r_{36} \Rightarrow r_{61}$	
	$r_{39} \Rightarrow r_{63}$			$r_{40} \Rightarrow r_{64}$			$r_{43} \Rightarrow r_{65}$		$r_{46} \Rightarrow r_{68}$		$r_{47} \Rightarrow r_{70}$		$r_{55} \Rightarrow r_{79}$		$r_{56} \Rightarrow r_{80}$		$r_{57} \Rightarrow r_{80}$		$r_{62} \Rightarrow r_{83}$	
	$r_{62} \Rightarrow r_{84}$			$r_{21} \otimes r_{22}$			$r_{32} \otimes r_{33}$		$r_{46} \otimes r_{47}$		$r_{65} \otimes r_{66}$									

Table 3 The importance of clients to the developer company

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 Points of reference for two datasets

Dataset 1
r_{min} ($cost_{min}$, $satisfaction_{min}$) = (0, 0)
r_{max} ($cost_{max}$, $satisfaction_{max}$) = (85, 893)
Dataset 2
r_{min} ($cost_{min}$, $satisfaction_{min}$) = (0, 0)
r_{max} ($cost_{max}$, $satisfaction_{max}$) = (1037, 2656)

first objective and, to some extent, improve the quality of solutions as the second objective. Therefore, to reduce run time in the proposed method, increasing the number of cores reduces the total number of iterations. In this scenario, serial algorithms are run with 10,000 iterations. In addition, 16,000 iterations in total are considered for the proposed method; that is, provided that the proposed method is run with two cores, the algorithm of each core has 8000 iterations (8000 × 2 = 16,000). In the event that it runs with four cores, each algorithm will have 4000 iterations (4000 × 4 = 16,000), and finally, in case it runs with eight cores, each algorithm will have 2000 iterations (2000 × 8 = 16,000). The number of the colony members and other parameters are similar to the first scenario, which is the same in all algorithms. In this scenario, the proposed method performs 6000 iterations more than the serial mode. This yields higher-quality solutions compared to those of the serial mode. However, considerable improvement is achieved in terms of runtime. For example, in the eight-core mode, each core performs 2000 iterations instead of one core doing 10,000 iterations. In the next subsection, the results and analysis of the two scenarios are presented.

5.2 Experiments and results

In this section, the results of the proposed method are presented. They are further compared with those of previous studies. Chaves-González et al. [4] used the results of algorithms MOABC, ACO, NSGA-II and GRASP to solve the NRP. The first scenario is applied on both datasets. In Table 5, the mean and standard deviation of the results, which were obtained by 100 independent runs of the proposed method and other algorithms on dataset 1, are presented.

In the first scenario, the proposed method was run by two algorithms, namely PMOABC 2 and PMOABC 4. The number of iterations in all algorithms is 10,000, and the number of colony members is 40. Four cost constraints, i.e., 30%, 50%, 70% and 100%, were considered. As given in Table 5, the GRASP algorithm has achieved the weakest results due to its greedy and local searches. The NSGA II algorithm by the use of crossover and mutation operators searched the problem space slightly better than the GRASP algorithm in all tests and obtained a bit better HV value than the GRASP algorithm. However, both algorithms have poor results. The ACS algorithm, using ants swarm intelligence, obtained higher HV and NDS values than the two GRASP and NSGA II algorithms. Among the serial algorithms, the MOABC algorithm yielded the best solutions. In fact, this algorithm could search the problem space better than the other serial algorithms. In addition to uniformly distributed NDS on the Pareto front, the more amounts of HV and number of NDS confirm superior of this algorithm rather than other serial algorithms. We now discuss the results of the proposed algorithms. The PMOABC 2 algorithm runs by two MOABC algorithms simultaneously. Then, it selects the union of the NDS set as its own solution set. It can be seen that the number of NDS in this algorithm has exceeded that of the serial algorithms. Subsequently, the spread criterion is decreased and the *HV* criterion is increased. For example, in the 70% test of PMOABC 2 algorithm, it is detected about 16 NDS more than GRASP and ACS algorithms. Moreover, HV and spread values have improved significantly. In all tests, the superiority of the PMOABC 2 algorithm is evident compared to MOABC serial algorithm. In the Pareto front diagram, the more uniformly the NDSs are distributed, the better it is. This, in turn, reduces the amount of the spread criterion. In addition, the more the available solution set in NDS yields larger area bounded by the graph, the higher quality it is. As mentioned in the previous subsection, to calculate the *HV* criterion, the fitness solutions are normalized for both objectives. Subsequently, the *HV* value is calculated. The PMOABC 4 algorithm that runs four MOABC algorithms simultaneously could produce the best results in all criteria. For example, for

Table 5 Results of the execution of algorithms in the first scenario on dataset 1

Effort limit	Hypervolume	Spread	Number of NDSs	Time (s)
Dataset 1 30%				
GRASP	7.708% ± 3.66e−1	0.64 ± 0.09	11.37 ± 1.47	–
NSGA-II	9.015% ± 1.12	0.76 ± 0.09	9.69 ± 2.09	–
ACS	10.283% ± 6.57e−2	0.52 ± 0.03	13.66 ± 13.66	–
MOABC	41.88% ± 1.15e−5	0.52 ± 0.01	15.00 ± 0.00	7.81
PMOABC 2	45.04% ± 2.29e−3	0.48 ± 0.10	17.16 ± 1.45	8.07
PMOABC 4	46.65% ± 2.70e−4	0.46 ± 0.16	20.26 ± 0.87	9.51
Dataset 1 50%				
GRASP	19.114% ± 3.50e−1	0.73 ± 0.07	17.65 ± 2.22	–
NSGA-II	20.652% ± 1.60	0.79 ± 0.07	11.30 ± 1.82	–
ACS	23.912% ± 6.75e−2	0.52 ± 0.01	17.75 ± 0.61	–
MOABC	54.715% ± 2.64	0.48 ± 0.01	23.66 ± 0.48	7.89
PMOABC 2	56.43% ± 2.19	0.46 ± 0.09	26.31 ± 1.12	8.14
PMOABC 4	57.29% ± 1.12	0.45 ± 0.08	30.63 ± 0.71	9.56
Dataset 1 70%				
GRASP	32.242% ± 4.96e−1	0.69 ± 0.06	20.26 ± 2.18	–
NSGA-II	32.157% ± 2.30	0.80 ± 0.07	11.70 ± 1.90	–
ACS	38.464% ± 7.08e−2	0.48 ± 0.02	20.57 ± 20.57	–
MOABC	60.855% ± 9.49e−4	0.43 ± 0.01	32.35 ± 0.99	7.92
PMOABC 2	61.95% ± 0.13	0.41 ± 0.07	36.31 ± 2.07	8.18
PMOABC 4	62.67% ± 0.22	0.40 ± 0.06	37.54 ± 1.34	9.59
Dataset 1 100%				
GRASP	–	–	–	–
NSGA-II	–	–	–	–
ACS	–	–	–	–
MOABC	63.780% ± 1.28e−3	0.39 ± 0.05	40.55 ± 1.25	7.83
PMOABC 2	65.18% ± 0.13	0.37 ± 0.03	43.11 ± 1.31	8.06
PMOABC 4	66.05% ± 0.22	0.36 ± 0.03	44.89 ± 1.02	9.32

the test 50%, the most amounts of the NDS (the mean 36.63) resulted among the all algorithms belong to the PMOABC 4.

It can be seen that the serial (MOABC) algorithm had shorter run time than that of the parallel mode. The reason is the presence of parallel processing overhead and communication between master and slave cores. As the number of cores participating in the proposed method increases, the total run time slightly increases. Since the hardware and software executing the algorithms are different [40], the run time of algorithms GRASP, NSGA-II and ACS was not compared. It can be seen from the table that quality of the solutions obtained by the proposed method (PMOABC $\times$) for all modes of the cost constraints (30%, 50%, 70% and 100%) was better than that of the serial algorithms executed. Dataset 1 has had little complexity and few requirements. Let us now analyze dataset 2, which is much more complicated from the first scenario. In Table 6, the mean and standard deviation of the results obtained based on 100 independent implementation of the proposed method and other algorithms on dataset 2 are given.

As given in Table 6, the proposed method yielded a higher-quality solution set due to the concurrent execution of several algorithms and taking the union of their NDSs set where the number of NDSs was more than that of the serial algorithms, HV increased and spread of the solution set reduced. For example, for the test 30%, PMOABC 4 algorithm could find about 10 NDS more than MOABC, about 70 NDS more than NSGA II and GRASP and also 88 NDS more than the ACS algorithm. For the test 50%, the value of HV in the PMOABC 4 algorithm is more than three times of the HV value in the ACS algorithm. In the same test, the value of HV in the PMOABC 4 algorithm is about 17% more than the best serial algorithm (MOABC). In all tests, the spread value of PMOABC 2 algorithm is lower than that of the serial algorithms. The same criterion is also more appropriate in the PMOABC 4 algorithm than the one in the PMOABC 2 algorithm.

The discovery of most of the NDSs in Pareto front resulted in a reduction in the average distance between successive solutions. On the other hand, the increase in the number of NDSs in Pareto front and their proportional distribution led to an increase in the area bounded by Pareto front (HV) graph. Within all cost constraints considered, the proposed method has performed better than the serial algorithms. In the proposed method, the number of executing algorithms (slave core) increased which caused the quality of solutions and also the runtime increases such that the best solution was produced by the PMOABC 4 algorithm. It should be noted that Tables 5 and 6 contain the results of the run of the algorithms in the first scenario with 10,000 iterations per algorithm. Due to its high complexity (having 100 requirements and 44 interactions between requirements), dataset 2 has caused algorithms applied on it to have relatively long run time. Therefore, we propose the second scenario for this complex dataset. In the second scenario, we intend to reduce the run time. Therefore, we adjust the total number of iterations of the proposed method proportionate to the number of executor algorithms.

Since the algorithms in serial mode are executed with 10,000 iterations, we consider 16,000 reps in total for the proposed method. Our purpose is to do more iteration to obtain higher-quality solutions than those of the serial mode. In addition, to reduce the run time, we distribute these 16,000 reps between the executor algorithms (executor cores). In the PMOABC 2 algorithm, both algorithms perform 8000 reps; in the PMOAB 4 algorithm, all of four algorithms perform 4000 reps; and in the PMPABC 8 algorithm, all of eight algorithms perform 2000 reps. The rest of the parameters of the simulation for this scenario are similar to that of the previous one. In Table 7, the results of the second scenario applied on dataset 2 are given. As the number of executor cores increases, the quality of solutions improves and the run time decreases dramatically. The PMOABC 8 algorithm which forms its solution set through taking the union of eight sets of NDSs has produced the highest quality solutions. A comparison of the solutions obtained by the application of the three algorithms in the proposed PMOABC $\times$ method indicates that the use of more algorithms with fewer reps is more desirable than the use of fewer algorithms with more reps. This issue can be seen through comparison of PMOABC 8 with PMOABC 2. In these tests, spread and HV values have improved slightly and slowly, but the NDS amount reflects the differences better. For example, for the test 30% test, PMOABC 8 algorithm has detected about 83 NDS more than the ACS algorithm, about 76 NDS more than the NSGA II algorithm and 72 NDS more than the GRASP algorithm. For the test 70%t, the PMOABC 8 algorithm has detected about 4 NDS more than the MOABC algorithm, about 61 NDS more than the NSGA II algorithm and 74 NDS more than the ACS algorithm.

In various PMOABC $\times$ algorithms, the run time is decreased as the number of their iterations is reduced (e.g., 4000 reps to 2000 reps). However, due to the presence of the parallel processing overhead and sending NDS sets from slaves to masters, this reduction in time is not in accordance with the ideal time that is half of this amount. Because ideally, it is expected that the algorithm run time is reduced approximately by half when the rep count is halved, yet, in practice, this will not happen due to the presence of overheads. In Table 7, the speedup and efficiency rates related to the algorithms of the proposed method are given. The PMOABC 8 algorithm displayed the best speedup rate among the algorithms of the proposed method for each of its algorithms performs only 2000 iterations, while the serial algorithm performs 10,000 iterations. In addition, the PMOABC 2

Table 6 Results of the execution of algorithms in the first scenario on dataset 2

Effort limit	Hypervolume	Spread	Number NDS	Time (s)
Dataset 2 30%				
GRASP	4.088 ± 8.55e−3	0.60 ± 0.04	57.99 ± 3.66	–
NSGA-II	7.920 ± 2.49e−1	0.80 ± 0.07	54.34 ± 8.51	–
ACS	8.517 ± 6.21e−2	0.68 ± 0.06	47.12 ± 5.44	–
MOABC	41.232 ± 1.14e−2	0.45 ± 0.02	125.37 ± 7.57	34.65
PMOABC 2	42.634 ± 0.081	0.44 ± 0.06	131.21 ± 11.20	36.55
PMOABC 4	42.877 ± 0.072	0.42 ± 0.09	135.33 ± 12.54	40.93
Dataset 2 50%				
GRASP	15.454 ± 6.88e−2	0.74 ± 0.04	75.81 ± 5.81	–
NSGA-II	18.006 ± 5.20e−1	0.81 ± 0.06	65.54 ± 11.86	–
ACS	19.159 ± 9.94e−2	0.66 ± 0.06	57.68 ± 5.69	–
MOABC	51.212 ± 1.17e−2	0.42 ± 0.02	135.93 ± 9.60	33.46
PMOABC 2	58.672 ± 0.033	0.38 ± 0.06	141.08 ± 13.13	35.71
PMOABC 4	60.312 ± 0.047	0.38 ± 0.05	143.73 ± 13.81	39.99
Dataset 2 70%				
GRASP	27.943 ± 7.5e−2	0.70 ± 0.03	120.14 ± 7.27	–
NSGA-II	31.710 ± 8.92e−1	0.77 ± 0.05	83.32 ± 10.52	–
ACS	32.777 ± 1.14e−1	0.61 ± 0.06	70.98 ± 5.27	–
MOABC	58.212 ± 7.00e−3	0.38 ± 0.02	139.31 ± 9.93	33.38
PMOABC 2	60.965 ± 0.022	0.37 ± 0.05	145.17 ± 13.56	34.27
PMOABC 4	61.872 ± 0.026	0.35 ± 0.06	152.22 ± 14.27	39.14
Dataset 2 100%				
GRASP	–	–	–	–
NSGA-II	–	–	–	–
ACS	–	–	–	–
MOABC	61.702 ± 4.94e−3	0.35 ± 0.03	147.51 ± 9.90	32.78
PMOABC 2	62.310 ± 0.028	0.34 ± 0.04	158.12 ± 14.28	33.42
PMOABC 4	62.798 ± 0.071	0.33 ± 0.03	161.39 ± 15.01	40.76

algorithm has the highest rate of efficiency since it only uses two cores to run the algorithm. In all algorithms simulated in both scenarios, it can be seen that the algorithm run time within the cost constraint of 100% is less than the run time within other cost constraints (30%, 50% and 70%). This is due to the presence of an auxiliary function when repairing the solutions obtained by the mutation so that the desired solution does not violate the limitations of the NRP, i.e., cost constraints and interaction and/or dependencies between requirements. In case the cost constraint is considered to be 100%, there will be no cost constraints. Therefore, repairing the related solutions is done faster which further makes the overall run time slightly shorter than the other modes with cost constraints.

The results given in Tables 5, 6 and 7 demonstrate that PMOABC could find more number of the NDS than MOABC. This proves that the implementation of several independent algorithms in parallel situation can result better solution rather than serial algorithms.

Providing more number of NDS automatically improves criteria of HV and spread. MOABC was the strongest algorithm among other serial algorithms based on previous researches. Then, PMOABC result is very better rather to other serial algorithms.

6 Conclusion

In this paper, we examined the NRP and formulated it as a restricted multi-objective optimization problem. We also proposed a parallel algorithm, based on a master–slave model for solving the NRP for the first time. In the proposed model, implemented in the shared memory architecture, multiple MOABC algorithms are simultaneously run on the master core and slave cores. At the end of their performance, the slave cores send their NDSs to the master core. The master core considers a general solution set resulting from the execution of all cores as the general solution to the problem through sorting and deleting duplicate solutions. This method, since the NDSs are obtained by the execution of several algorithms, obtained higher-quality solutions compared to those of the serial algorithm. In the

Table 7 Results of the second scenario on dataset 2

Effort limit	Hypervolume	Spread	Number of NDSs	Time	Speedup	Efficiency
Dataset 2 30%						
GRASP	4.088 ± 8.55e−3	0.60 ± 0.04	57.99 ± 3.66	–	–	–
NSGA-II	7.920 ± 2.49e−1	0.80 ± 0.07	54.34 ± 8.51	–	–	–
ACS	8.517 ± 6.21e−2	0.68 ± 0.06	47.12 ± 5.44	–	–	–
MOABC	41.232 ± 1.14e−2	0.45 ± 0.02	125.37 ± 7.57	34.65	–	–
PMOABC 2	41.423 ± 0.072	0.45 ± 0.07	127.32 ± 12.14	28.75	1.20	0.60
PMOABC 4	41.911 ± 0.068	0.45 ± 0.09	128.84 ± 11.94	14.18	2.44	0.61
PMOABC 8	42.603 ± 0.091	0.44 ± 0.08	130.21 ± 13.16	7.88	4.39	0.55
Dataset 2 50%						
GRASP	15.454 ± 6.88e−2	0.74 ± 0.04	75.81 ± 5.81	–	–	–
NSGA-II	18.006 ± 5.20e−1	0.81 ± 0.06	65.54 ± 11.86	–	–	–
ACS	19.159 ± 9.94e−2	0.66 ± 0.06	57.68 ± 5.69	–	–	–
MOABC	51.212 ± 1.17e−2	0.42 ± 0.02	135.93 ± 9.60	33.46	–	–
PMOABC 2	51.643 ± 0.041	0.44 ± 0.05	138.88 ± 10.96	27.04	1.23	0.62
PMOABC 4	51.928 ± 0.055	0.42 ± 0.06	139.92 ± 13.22	13.76	2.43	0.61
PMOABC 8	52.171 ± 0.050	0.41 ± 0.03	140.62 ± 13.78	7.63	4.38	0.55
Dataset 2 70%						
GRASP	27.943 ± 7.5e−2	0.70 ± 0.03	120.14 ± 7.27	–	–	–
NSGA-II	31.710 ± 8.92e−1	0.77 ± 0.05	83.32 ± 10.52	–	–	–
ACS	32.777 ± 1.14e−1	0.61 ± 0.06	70.98 ± 5.27	–	–	–
MOABC	58.212 ± 7.00e−3	0.38 ± 0.02	139.31 ± 9.93	33.38	–	–
PMOABC 2	58.831 ± 0.052	0.37 ± 0.05	142.42 ± 12.86	27.30	1.22	0.61
PMOABC 4	59.343 ± 0.036	0.37 ± 0.02	143.93 ± 13.77	13.63	2.45	0.61
PMOABC 8	60.344 ± 0.042	0.37 ± 0.03	144.64 ± 14.24	8.48	3.94	0.49
Dataset 2 100%						
GRASP	–	–	–	–	–	–
NSGA-II	–	–	–	–	–	–
ACS	–	–	–	–	–	–
MOABC	61.702 ± 4.94e−3	0.35 ± 0.03	147.51 ± 9.90	32.78	–	–
PMOABC 2	61.890 ± 0.033	0.35 ± 0.04	151.73 ± 12.66	26.44	1.24	0.62
PMOABC 4	61.983 ± 0.062	0.35 ± 0.03	152.31 ± 15.08	13.13	2.50	0.62
PMOABC 8	62.021 ± 0.069	0.34 ± 0.05	154.03 ± 16.22	6.83	4.79	0.60

proposed method, the concurrent use of several algorithms has led to the inapplicability of the general solution in local optimization. In addition, the proposed method makes an efficient use of the hardware capacity and divides the workload almost evenly between the cores. Since the proposed algorithm was the first parallel algorithm for solving the NRP, the quality of the solutions obtained by the proposed method was compared with those of serial algorithms such as MOABC, ACO, NSGA-II and GRASP. To conduct the experiments, two datasets were evaluated in previous works. These datasets contained various requirements and clients with different levels of importance. Two types of limitations were defined for this problem. The first limitation was the dependencies between the requirements, and the second was the cost of product development. Applying these limitations makes us face a restricted optimization problem. In addition, we considered two different scenarios for doing the experiments. In the first scenario, the aim of the experiment was to increase the quality of the solutions obtained by the proposed method compared to that of serial algorithm; and we could make a significant improvement in the quality of the solutions. In the second scenario, the first objective was to reduce the run time, and the second objective was to improve the quality of solutions, relatively. In this scenario, increasing the number of algorithms participating in the proposed approach and reducing the number of iterations in these algorithms were realized both as objectives of the scenario. The run time reduced significantly and the quality of the solutions slightly improved. We now mention a few examples of researches to be conducted in this area in the future. The implementation of other parallel processing models to improve the quality of solutions and to reduce run time might be interesting topics to work on. The use of other metaheuristic algorithms in the proposed method, or a combination of all metaheuristic algorithms together, and their implementation in the proposed method can improve the quality of solutions.

Acknowledgements The authors would like to thank the anonymous reviewers and the editor in chief for their insightful comments and suggestions.

References

1. Alrezaamiri H, Ebrahimnejad A, Motameni H (2018) Software requirement optimization using a fuzzy artificial chemical reaction optimization algorithm. Soft Comput 23:1–16
2. Bagnall AJ, Rayward-Smith VJ, Whittley IM (2001) The next release problem. Inf Softw Technol 43(14):883–890
3. Chaves-González JM, Pérez-Toledano MA (2015) Differential evolution with Pareto tournament for the multi-objective next release problem. Appl Math Comput 252:1–13
4. Chaves-González JM, Pérez-Toledano MA, Navasa A (2015) Software requirement optimization using a multiobjective swarm intelligence evolutionary algorithm. Knowl-Based Syst 83:105–115
5. Chaves-González JM, Vega-Rodríguez MA, Granado-Criado JM (2013) A multiobjective swarm intelligence approach based on artificial bee colony for reliable DNA sequence design. Eng Appl Artif Intell 26(9):2045–2057
6. Colanzi TE, Vergilio SR (2016) A feature-driven crossover operator for multi-objective and evolutionary optimization of product line architectures. J Syst Softw 121:126–143
7. Deb K (2001) Multi-objective optimization using evolutionary algorithms, vol 16. Wiley, Hoboken
8. Deb K, Pratap A, Agarwal S, Meyarivan TAMT (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197
9. De la Hidalga AN, Hardisty A, Jones A (2016) SCRAM–CK: applying a collaborative requirements engineering process for designing a web based e-science toolkit. Requir Eng 21(1):107–129
10. Del Sagrado J, Del Aguila IM, Orellana FJ (2015) Multi-objective ant colony optimization for requirements selection. Empir Softw Eng 20(3):577–610
11. Delgarm N, Sajadi B, Delgarm S (2016) Multi-objective optimization of building energy performance and indoor thermal comfort: a new method using artificial bee colony (ABC). Energy Build 131:42–53
12. De Souza JT, Maia CLB, Ferreira TN, Carmo RAF, Brasil MMA (2011) An ant colony optimization approach to the software release planning with dependent requirements. In: Cohen MB, Ó Cinnéide M (eds) Search based software engineering. SSBSE 2011. Lecture notes in computer science, vol 6956. Springer, Berlin, Heidelberg
13. Dragicevic S, Celar S, Turic M (2017) Bayesian network model for task effort estimation in agile software development. J Syst Softw 127:109–119
14. Durillo JJ, Zhang Y, Alba E, Harman M, Nebro AJ (2011) A study of the bi-objective next release problem. Empir Softw Eng 16(1):29–60
15. Ebrahimnejad A, Tavana M, Alrezaamiri H (2016) A novel artificial bee colony algorithm for shortest path problems with fuzzy arc weights. Measurement 93:48–56
16. Femmer H, Fernández DM, Wagner S, Eder S (2017) Rapid quality assurance with requirements smells. J Syst Softw 123:190–213
17. Ferrari A, Spoletini P, Gnesi S (2016) Ambiguity and tacit knowledge in requirements elicitation interviews. Requir Eng 21(3):333–355
18. Gong YJ, Chen WN, Zhan ZH, Zhang J, Li Y, Zhang Q, Li JJ (2015) Distributed evolutionary algorithms and their models: a survey of the state-of-the-art. Appl Soft Comput 34:286–300
19. Greer D, Ruhe G (2004) Software release planning: an evolutionary and iterative approach. Inf Softw Technol 46(4):243–253
20. Harman M, Mansouri SA, Zhang Y (2012) Search based software engineering: Trends. Tech Appl, ACM Comput Surv 5:11
21. Iimura I, Hamaguchi K, Ito T, Nakayama S (2005) A study of distributed parallel processing for queen ant strategy in ant colony optimization. In: Sixth international conference on parallel and distributed computing, applications and technologies, 2005. PDCAT 2005, IEEE, pp 553–557
22. Jayatilleke S, Lai R, Reed K (2018) A method of requirements change analysis. Requir Eng 23(4):493–508
23. Jiang H, Zhang J, Xuan J, Ren Z, Hu Y (2010) A hybrid ACO algorithm for the next release problem. In: 2010 2nd international conference on software engineering and data mining (SEDM), IEEE, pp 166–171
24. Karlsson J (1996) Software requirements prioritizing. In: Proceedings of the second international conference on requirements engineering, IEEE, pp 110–116
25. Knowles J, Corne D (1999) The pareto archived evolution strategy: a new baseline algorithm for pareto multiobjective optimisation. In: Proceedings of the 1999 congress on evolutionary computation, 1999. CEC 99, IEEE, vol 1, pp 98–105
26. Lindsjørn Y, Sjøberg DI, Dingsøyr T, Bergersen GR, Dybå T (2016) Teamwork quality and project success in software development: a survey of agile development teams. J Syst Softw 122:274–286
27. Liu L, Zhou Q, Liu J, Cao Z (2017) Requirements cybernetics: elicitation based on user behavioral data. J Syst Softw 124:187–194
28. Meade A, Deeptimahanti DK, Buckley J, Collins JJ (2017) An empirical study of data decomposition for software parallelization. J Syst Softw 125:401–416
29. Misaghian N, Motameni H (2016) An approach for requirements prioritization based on tensor decomposition. Requir Eng. https://doi.org/10.1007/s00766-016-0262-6
30. 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
31. Parejo JA, Sánchez AB, Segura S, Ruiz-Cortés A, Lopez-Herrejon RE, Egyed A (2016) Multi-objective test case prioritization in highly configurable systems: a case study. J Syst Softw 122:287–310
32. Prakash D, Prakash N (2017) A multifactor approach for elicitation of Information requirements of data warehouses. Requir Eng. https://doi.org/10.1007/s00766-017-0283-9
33. Pitangueira AM, Maciel RSP, Barros M (2015) Software requirements selection and prioritization using SBSE approaches: a systematic review and mapping of the literature. J Syst Softw 103:267–280
34. Rubio-Largo Á, Vega-Rodríguez MA, González-Álvarez DL (2015) Multiobjective swarm intelligence for the traffic grooming problem. Comput Optim Appl 60(2):479–511
35. Thew S, Sutcliffe A (2018) Value-based requirements engineering: method and experience. Requir Eng 23(4):443–464
36. Veerapen N, Ochoa G, Harman M, Burke EK (2015) An integer linear programming approach to the single and bi-objective next release problem. Inf Softw Technol 65:1–13
37. Vianna DS, Arroyo JEC (2004) A GRASP algorithm for the multi-objective knapsack problem. In: 24th international conference of

the Chilean computer science society, 2004. SCCC 2004, IEEE, pp 69–75
38. Wessing S, Preuss M (2016) On multiobjective selection for multimodal optimization. Comput Optim Appl 63(3):875–902
39. Xiang Y, Peng Y, Zhong Y, Chen Z, Lu X, Zhong X (2014) A particle swarm inspired multi-elitist artificial bee colony algorithm for real-parameter optimization. Comput Optim Appl 57(2):493–516
40. Yurtkuran A, Emel E (2015) An adaptive artificial bee colony algorithm for global optimization. Appl Math Comput 271:1004–1023
41. Yu W, Zhang W (2006) Study on function optimization based on master-slave structure genetic algorithm. In: 2006 8th international conference on signal processing, IEEE, vol 3
42. Zhang Y, Harman M, Mansouri SA (2007) The multi-objective next release problem. In: Proceedings of the 9th annual conference on Genetic and evolutionary computation, ACM, pp 1129–1137