Soft Comput (2018) 22:6341–6361
https://doi.org/10.1007/s00500-017-2687-3

METHODOLOGIES AND APPLICATION

Many-objective artificial bee colony algorithm for large-scale software module clustering problem

Amarjeet[1] · **Jitender Kumar Chhabra**[1]

Published online: 6 July 2017

Abstract The meta-heuristic search algorithms have been widely applied to solve the various science and engineering optimization problems. However, the performance of these algorithms is highly sensitive toward the number of objective functions and number of decision variables. Recently, it has been explored by the researchers that the performance of such algorithm degrades when the number of objective functions and decision variables increases by some limit. Hence, these algorithms can be hardly acceptable to the real-world optimization problems such as software module clustering problem (SMCP), which contains a large number of objective functions and decision variables. Previous researchers have proposed several approaches to address the many-objective optimization problems by revising existing meta-heuristic algorithms. Recently, an artificial bee colony algorithm (ABC), a meta-heuristic algorithm, effectively used to address the several multi-objective optimization problems. Even though in most of the cases ABC algorithm performs better compared to other meta-heuristic algorithms, it faces the same problems as other meta-heuristic algorithms for a large number of objective functions and decision variables. This paper proposes a many-objective artificial bee colony (MaABC) algorithm to solve many-objective SMCPs. In this contribution, we revised the original ABC by using, quality indicator, L_p-norm-based ($p < 1$) distances, and two external archives concepts. To validate the proposed approach, an extensive comparative study is performed with the existing many-objective optimization algorithms (i.e., Two-Arch2, NSGA-III, MOEA/D, and IBEA) over seven SMCPs. The statistical analysis of the results show that the proposed MaABC outperforms existing many-objective approaches in terms of modularization quality (MQ), cohesion, coupling, and inverted generational distance (IGD).

Keywords Artificial bee colony · Meta-heuristic algorithm · Many-objective optimization · Software module clustering

Communicated by V. Loia.

✉ Amarjeet
amarjeetnitkkr@gmail.com

Jitender Kumar Chhabra
jitenderchhabra@gmail.com

[1] Department of Computer Engineering, NIT Kurukshetra, Haryana, India

1 Introduction

The optimization problem containing more than three objective functions is usually regarded as many-objective optimization problem (MaOP) (Khare et al. 2003; Praditwong and Yao 2007). The MaOP is a special category of multi-objective optimization problem (MuOP). To solve the low-dimensional science and engineering MuOPs, various multi-objective evolutionary algorithms (MOEAs) (e.g., PESA-II (Corne et al. 2001), SPEA2 (Zitzler et al. 2002), and NSGA-II (Deb et al. 2002)) have been used successfully.

Traditional MOEAs based on the Pareto-dominance selection principles face great trouble when dealing with MuOPs with high-dimensional decision variables and many-objective functions (Wang et al. 2015). The empirical studies (Khare et al. 2003; Praditwong and Yao 2007) have also demonstrated that the effectiveness of MOEAs based on the Pareto-dominance selection principles degrades over large-scale optimization problems. The first reason of such performance degradation is that as the number of objective functions increases, the number of non-dominated solutions increases dramatically in the population that deteriorate the

required selection pressure for the candidate solution to move in the next generation (Wang et al. 2015; Hughes 2008). The second reason is that the large number of objective functions felicitates the sparse distribution of candidate solution in the high-dimension objective space which hurdle diversity management.

To improve the performance of MOEAs in case of MaOPs, researchers have proposed several alternatives of traditional Pareto-dominance selection, such as L-optimality (Zou et al. 2008), ε-dominance (Laumanns et al. 2002; Hadka and Reed 2013), fuzzy dominance (Wang and Jiang 2007), θ-dominance (Yuan et al. 2015), and preference order ranking (Pierro et al. 2007). We are not providing details about many other approaches regarding the many-objective optimization. The works (Bingdong et al. 2015) provided the detailed investigation about the various classes of many-objective optimization approaches that have been used to solve the many-objective optimization problems. In brief, the works (Bingdong et al. 2015) have categorized the many-objective optimization approaches in following major group: objective reduction (e.g., Cinneide et al. 2012), an indicator based (e.g., Zitzler and Künzli 2004), preference or reference set (e.g., Deb and Jain 2014), decompositions (e.g., Zhang and Li 2007). The main limitations of these are as follows: (1) The dimensionality reduction may fail in lessening the objective functions or yielding a solution set that does not contain the whole Pareto front (Bingdong et al. 2015), (2) in indicator-based approach, the hyper-volume calculation increases exponentially by the increase in the number of objectives (Cai et al. 2014), (3) in the preference-based approach, the overhead of decision making associated with the preference elicitation is one of the main drawbacks (Asafuddoula et al. 2015), (4) the main difficulty in the decomposition-based approach is to define proper weight vector (Cai et al. 2014).

In summary, there has been much advancement carried out to enhance the performance of MOEAs for MaOPs. However, these enhancements have not been widely explored to address the real-world, many-objective optimization problems (e.g., many-objective SMCPs (MaSMCPs)). The search-based software engineering (SBSE) exploited various meta-heuristic search algorithms to address the real-world software engineering problems (Harman and Jones 2001). Recently, the software module clustering problem (SMCP) has been regarded as an important search-based software engineering problem where meta-heuristic search algorithms were found more effective against deterministic approaches (Praditwong et al. 2011; Parashar and Chhabra 2016). The SMCP is natural many-objective optimization problem where many conflicting objective functions require to be optimized. Most of the previous work (Praditwong et al. 2011; Kumari et al. 2013; Kumari and Srinivas 2016) addresses the SMCP as multi-objective optimization problem using multi-objective optimization approaches (e.g., NSGA-II). The work (Mkaouer et al. 2015) first formulated the SMCP as many-objective software remodularization problems and solved using NSGA-III (Deb and Jain 2014) a genetic-based many-objective evolutionary algorithm. Even though NSGA-III has been used successfully to solve the MaSMCP, the applicability of other widely used meta-heuristic algorithms (e.g., artificial bee colony algorithm (Karaboga 2005)) has not been explored.

To solve the MaSMCP, we exploited various positive concepts, e.g., quality indicator (Zitzler and Künzli 2004), L_p-norm-based ($p < 1$) distances (Wang et al. 2015), and two external archives (Wang et al. 2015) from the existing meta-heuristic approaches and proposed a new approach, namely many-objective artificial bee colony (MaABC) algorithm. The researchers (Plevris and Papadrakakis 2011; Li and Yin 2012) have also reported that a good combination of two or more concepts into search algorithm may be advantageous, and they can perform considerably better than the single pure search algorithm in handling large-scale real-world optimization problems. The major contributions of the proposed work are as follows:

- The SMCP is transformed as search-based many-objective optimization problem, namely many-objective SMCP (MaSMCP). Two MaSMCP formulations, extended maximizing cluster approach (E-MCA) and extended equal-size cluster approach (E-ECA) based on multi-objective software module clustering formulations (Praditwong et al. 2011), have been designed.
- To address the MaSMCP, a new many-objective meta-heuristic algorithm, namely many-objective artificial bee colony (MaABC) algorithm, has been proposed. The MaABC is designed by incorporating the various positive concepts (e.g., quality indicator, L_p-norm-based ($p < 1$) distances, and two external archives from the existing meta-heuristic approaches into the original ABC algorithm).
- To confirm the supremacy of the proposed MaABC approach, an extensive comparative study has been conducted and the obtained results are compared with the relevant existing many-objective optimization algorithms (i.e., Two-Arch2 (Wang et al. 2015), NSGA-III (Deb and Jain 2014), MOEA/D (Zhang and Li 2007), and IBEA (Zitzler and Künzli 2004) over seven practical software module clustering problems. The statistical analysis of the results indicate that the proposed approach outperforms existing approaches in terms of modularization quality (MQ), cohesion, coupling and IGD.

The rest part of this article is arranged as follows: Section 2 briefly provides some basic concepts and back-

ground. Section 3 discusses related work relevant to the proposed approach. Section 4 presents description of proposed MaABC for object-oriented software (OOS) systems. Section 5 gives the experimental setup. Section 6 presents the results and compares with the best performing many-objective algorithms from the existing literature to show the supremacy of the MaABC algorithm. Section 7 discusses the threats to validity. Finally, Sect. 8 gives the concluding remarks and future research directions.

2 Software module clustering

This section provides the basic background and many-objective formulation related to search-based software module clustering problem.

2.1 Software module clustering problem

To ensure the program design quality such as comprehensibility, maintainability, and extendibility, the software developer organizes the highly cohesive source code elements (i.e., modules) into the same group (i.e., cluster) and non-cohesive elements into different groups. The task of organizing software modules into clusters based on some criteria is generally referred as software module clustering problem (SMCP). Formally, the SMCP can be defined as follows: Given a set $N = 1, 2, ..., n$ of software modules which are connected with each other with connection weights $w(i, j)$ where $i, j \epsilon N$, the SMCP consists of generating a partition $\mathrm{P} = \{\mathrm{C}_1, \mathrm{C}_2, ..., \mathrm{C}_m\}$ of n^*m clusters, such that $1 \leq m \leq n$ with $\mathrm{C}_i \neq \Phi, \mathrm{C}_i \cap \mathrm{C}_j = \Phi, i \neq j$, and $\cup_{i=1}^{m} C_i = N$ so as to optimize some design criteria. In SMCP, the choice of what constitutes a software elements and a cluster depends on the software abstraction level where clustering is performed. This paper considers object-oriented classes of the source code as modules and packages (used as group of classes) as clusters.

2.2 Software module clustering as an optimization problem

The main goal of the SMCP is to find one or more feasible partitions which correspond to extreme values of certain quality properties. Based on the number of quality criteria, SMCPs can be addressed by formulating as mono-objective, multi-objective, and many-objective optimization problem. The brief descriptions of these formulations are given below.

- Mono-objective optimization The formulation of SMCP as a mono-objective optimization problem (MoSMCP) deals with the task of determining a clustering solution that optimizes a single quality criterion as fitness function. From a given cluster design vector c, an optimal clustering solution c^* can be determined as follows:

$$f(\mathrm{c}^*) = \min / \max f(\mathrm{c}) | \mathrm{c} \in \quad \Psi \tag{1}$$

The symbol Ψ denotes the set of all feasible clustering solution. The function f can be a minimization or maximization function.

- Multi-objective Optimization: The formulation of SMCP as a multi-objective optimization problem (MuSMCP) deals with the task of determining a set of trade-off clustering solutions that optimizes more than one and less than or equal to three design criteria as objective functions simultaneously. From a given cluster design vector c, a set of trade-off clustering solutions c^* can be determined as follows:

$$f(c^*) = \begin{cases} \min \mathrm{f}_1(\mathrm{c}), \mathrm{f}_2(\mathrm{c}), \ldots, \mathrm{f}_\mathrm{M}(\mathrm{c})^T & 1 < M \\ \mathrm{g}_j(\mathrm{c}) \geq 0 & j = 1, ..., P \\ \mathrm{h}_\mathrm{k}(\mathrm{c}) = 0 & k = 1, ..., Q \\ \mathrm{c}_i^L \leq \mathrm{c}_i \leq \mathrm{c}_i^U & i = 1, ..., n \end{cases} \tag{2}$$

The symbols M and f_i represent the number of design criteria (i.e., objective functions) and the ith design criterion, respectively. The P, Q, c_i^L, and c_i^U denote the number of inequality design constraints, number of equality design constraints, lower bound of the decision variable x_i, and upper bound of the decision variable x_i, respectively.

2.3 Software module clustering as a many-objective optimization problem

The formulation of SMCP as a many-objective optimization problem (MaSMCP) deals with the task of determining a set of trade-off clustering solutions that optimizes more than three design criteria as objective functions simultaneously. From a given cluster design vector c, a set of trade-off clustering solutions c^* can be determined as follows:

$$f(c^*) = \begin{cases} \min \mathrm{f}_1(\mathrm{c}), \mathrm{f}_2(\mathrm{c}), \ldots, \mathrm{f}_\mathrm{M}(\mathrm{c})^T & M > 3 \\ \mathrm{g}_j(\mathrm{c}) \geq 0 & j = 1, \ldots, P \\ \mathrm{h}_\mathrm{k}(\mathrm{c}) = 0 & k = 1, \ldots, Q \\ \mathrm{c}_i^L \leq c_i \leq \mathrm{c}_i^U & i = 1, \ldots, n \end{cases} \tag{3}$$

The symbols M and f_i represent the number of design criteria (i.e., objective functions) and the ith design cri-

terion, respectively. The P, Q, c_i^L, and c_i^U denote the number of inequality design constraints, number of equality design constraints, lower bound of the decision variable x_i, and upper bound of the decision variable x_i, respectively. In the following, we present the two many-objective formulations, namely extended maximizing cluster approach (E-MCA) and extended equal-size cluster approach (E-ECA) for software module clustering problem. These formulations are, respectively, the extended versions of maximizing cluster approach (MCA) and equal-size cluster approach (ECA) proposed in the works (Praditwong et al. 2011).

Extended maximizing cluster approach (E-MCA) The main aim of E-MCA many-objective software module clustering formulation is to achieve good clustering having low coupling and high cohesion, while minimizing the number of isolated clusters, maximizing the number of clusters, minimizing the cluster cyclic dependencies, minimizing average shortest path length between a source and all other reachable clusters. The objective functions defined under the E-MCA approach are:

- Maximizing the sum of intracluster dependencies.
- Minimizing the sum of intercluster dependencies.
- Maximizing the number of clusters.
- Maximizing the modularization quality (MQ).
- Minimizing the number of isolated clusters.
- Minimizing cluster cyclic dependencies.
- Minimizing average shortest path length between a source and all other reachable clusters.

Extended equal-size cluster approach (E-ECA) The main goal of the E-ECA approach is to encourage the generation of clusters of nearly equal size. The objective functions defined in this formulation are:

- Maximizing the sum of intracluster dependencies.
- Minimizing the sum of intercluster dependencies.
- Maximizing the number of clusters.
- Maximizing the modularization quality (MQ).
- Minimizing the difference between maximum and minimum number of modules in a cluster.
- Minimizing cluster cyclic dependencies.
- Minimizing average shortest path length between a source and all other reachable clusters.

Modularization quality (MQ) was first introduced by the works (Mancoridis et al. 1999), later the works (Praditwong et al. 2011) redefined it. In this paper, we use the MQ definition the same as defined in the works (Praditwong et al. 2011).

3 Basic concepts

3.1 Original ABC algorithm

The artificial bee colony (ABC) is a meta-heuristic search technique inspired by the intelligent foraging behavior of honey bees. It was originally designed by Karaboga (Karaboga 2005). Recently ABC gained wide attention and found to be effective and well situated for solving the various types of optimization problems in science and engineering fields (Dahiya et al. 2010; Jadhav and Bamane 2016; Xianneng and Guangfei 2016; Hashim et al. 2016; Amarjeet and Chhabra 2017a). The working procedure of ABC algorithm is mainly divided into three essential parts: food source positions, nectar amount, and different types of honey bees. Each food source position corresponds to the feasible candidate solution of the problem, and the nectar amount corresponds to the food source quality (i.e., fitness of the candidate solution). The honey bees are classified into three categories: employed bees, onlooker bees, and scout bees. Each type of honey bees performs particular operation to generate new candidate solutions (food source positions). The brief description of the original ABC algorithm is as follows:

- *Population initialization phase* The population of ABC algorithm contains a certain number of food sources (i.e., Np), and initially these food sources are produced randomly according to the problem's search space. For a continuous optimization problem following equation is used to initialize the food source.

$$v_{id} = v_{id}^{\min} + r \times (v_{id}^{\max} - v_{id}^{\min}) \tag{4}$$

  where i represents food source number in the population with the range of $i = 1, 2, \ldots Np$, d represents the dimension (i.e., decision variable) of each food source having range of $d = 1, 2, \ldots D$, r represents a uniform random number distributed over the interval [0, 1], and $v_{id}^{\max}$ and $v_{id}^{\min}$ represent the upper and lower bounds for the decision variable d, respectively. After the population initialization, each food source is associated a "limit" variable with 0 value. If a food source in the population could not be improved after a certain number of trials (i.e., limit) then that food source is abandoned.
- *Employed bee phase* In the employed bee phase, the ith food source v_i of the population is assigned to the ith employed bee, which perform search operation randomly around the neighbor of current food source, and determines a new food source as follows.

$$v_{\text{new},d} = v_{id} + r \times (v_{id} - v_{kd}) \tag{5}$$

  where $i \in \{1, ..., Np\}$, and selection strategy of index k is random with the condition $k \in \{1, ..., Np\} \wedge k \neq i$.

The v_{new} represents a new food source which is randomly chosen from the neighbor. After generating new solution v_{new} its fitness is calculated and compared to the original food source v_i; thereafter, the solution with the highest fitness value is selected.

- *Onlooker bee phase* The onlooker bees make a decision on food sources whether to select or not of the food source selected by the employed bees. To perform this, the onlooker bees use the probability values, calculated using Eq. (6), to select the food source for determining promising regions in the search space.

$$p_i = \frac{fit_i}{\sum_{i=n}^{SN} fit_i} \tag{6}$$

where fit_i is the fitness value of the ith food source.

- *Scout bee phase* If a food source in employed and onlooker bee phase cannot be further improved through a fixed iteration, then that food source is supposed to be discarded. The scout bee performs a random search operation over the whole feasible search space and generates a new food source, and the abandoned food source is replaced with it.

3.2 Indicator-based ranking

The Pareto-dominance-based MOEAs face challenges in converging toward the true Pareto front in many-objective optimization problems. In order to solve this problem, the ranking strategy of MOEAs was redefined in different ways (e.g., θ dominance (Yuan et al. 2015), grid dominance (Yang et al. 2013), preference order ranking (Pierro et al. 2007)). These improved approaches have successfully used to optimize large-scale many-objective optimization problems. Min–max strategy (Coello 1996; Coello and Christiansen 1998) is also an alternate which can be used for the ranking of non-dominated solutions. In this paper, to rank the non-dominated solution we use the concept of quality indicator $I_{\varepsilon+}$ given in IBEA (Zitzler and Künzli 2004). The $I_{\varepsilon+}$ is used to calculate the minimum distance that one solution requires, in order to dominate another solution in the objective space. The value of $I_{\varepsilon+}$ between two solutions c_1 and c_2 can be computed as follows:

$$f(c_i) = \sum_{c_j \in P \setminus \{c_i\}} -e^{-I_{\varepsilon+}(c_j, c_i)/0.05} \tag{7}$$

$$I_{\varepsilon+}(c_1, c_2) = \min_{\varepsilon}(f_i(c_1) - \varepsilon \leq f_i(c_2))1 \leq i \leq m \tag{8}$$

where $f(c_i)$ is the fitness of a candidate solution c_i of population P, and m is the number of objective functions. This method of fitness assignment improves the efficiency and effectiveness of the candidate solution selection process from the current population for the next generation.

3.3 L_p-norm-based ($p < 1$) distance

The choice of distance metric in high-dimensional applications is not clear; the notion for the similarity calculation is very heuristic (Aggarwal and Hinneburg 2001). Many applications with high-dimensional algorithms and indexing structures use the Euclidean distance metric. The study (Aggarwal and Hinneburg 2001) demonstrated that the efficiency and effectiveness of Euclidean distance (L_2-norm) degrades as the number of dimensions increases. However, the fractional distances (L_k-norm, $k < 1$) and Manhattan distance (L_1-norm) perform better in a high-dimensional space. The concluding remark of the study (Aggarwal and Hinneburg 2001) also showed that the fractional distances (L_k-norm, $k < 1$) are more effective compared to Manhattan distance (L_1-norm) in a high-dimensional space. The L_k-norm is defined as follows:

$$L_k(x, y) = \sum_{i=1}^{d} (||x^i - y^i||^k)^{1/k} | x, y \in R^d, k \in Z \tag{9}$$

For MaSMCPs with many numbers of objective functions, an Lk-norm-based ($k < 1$) distance measure with a constant k may not suit all the cases. Hence, the value of k in this paper is set as $k = 1/m$(m is the number of objectives) similar to the work (Wang et al. 2015).

4 Many-objective artificial bee colony (MaABC)

ABC algorithm has demonstrated several advantages: very few control parameters, relatively easy implementation, and good exploration and exploitation capability. Previous works (Karaboga and Basturk 2007, 2008; Karaboga and Akay 2009; Akay and Karaboga 2012) demonstrated that ABC has better performance than that of an ant colony algorithm (ACO), particle swarm optimization (PSO), differential evolutionary (DE), and genetic algorithm (GA) in solving large and complex optimization problems. The applicability and usefulness of the ABC algorithm has not been studied by any researcher till date to solve the software multi-objective SMCPs (more specifically many-objective SMCPs (MaSMCPs)).

Based on the optimization concept of ABC and full consideration software module clustering feature, this paper proposes a many-objective artificial bee colony (MaABC) algorithm to solve the MaSMCP. Unlike Pareto-dominance-based multi-objective ABC, the proposed MaABC approach is designed by evaluating the solution on the basis of quality indicator (Zitzler and Künzli 2004) and Lp-norm ($p<1$) distances (Wang et al. 2015). Additionally, the approach used

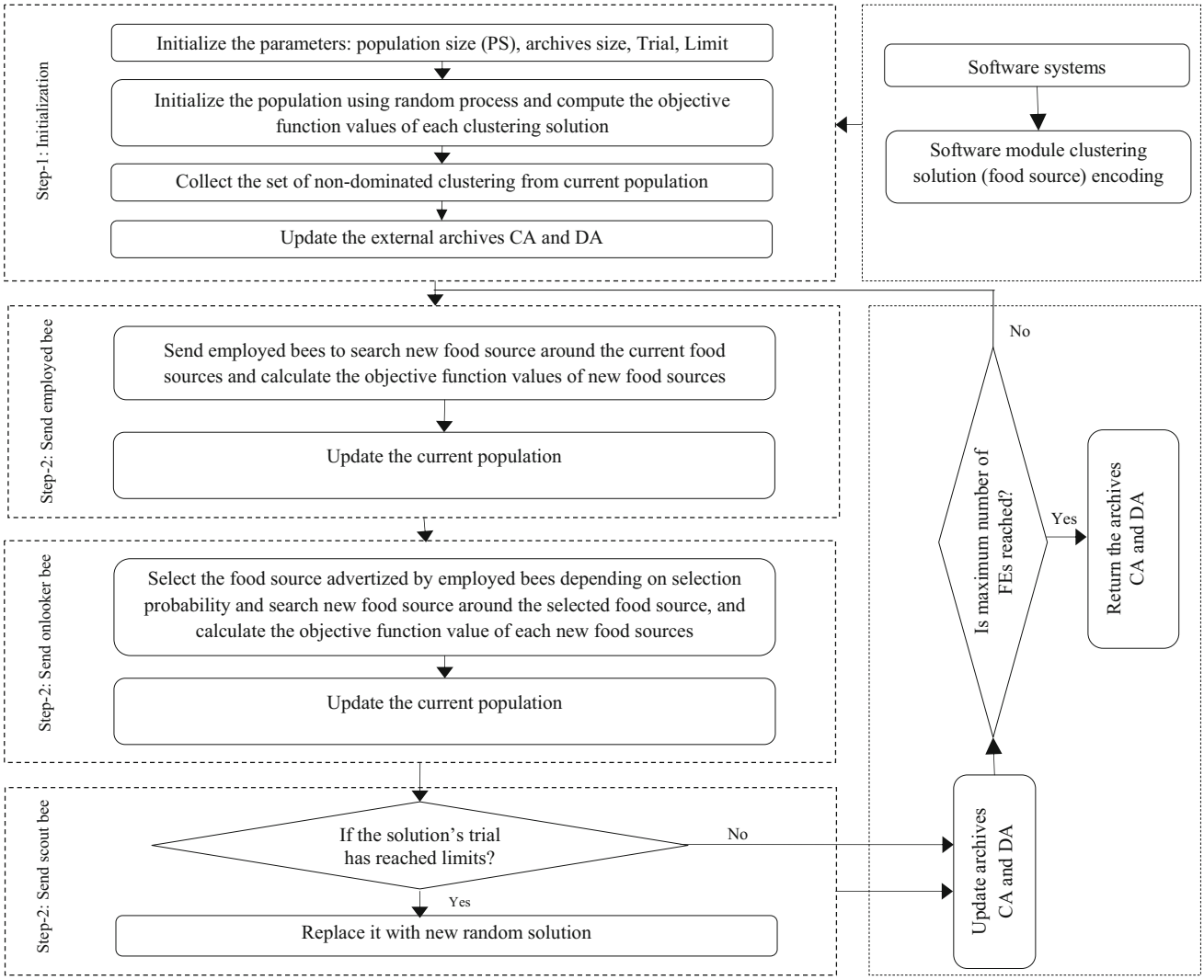

Fig. 1 Framework of proposed MaABC algorithm

two external archives, namely convergence archive (CA) and divergence archive (DA). The overall process of MaABC is presented in Fig. 1. Similar to the other meta-heuristic search optimization algorithms our proposed MaABC is an iterative process. It begins with an initial population of randomly generated food source position (i.e., module clustering solutions), and then the following steps are repeated until a stopping criterion is satisfied: (1) update the external archives CA and DA, (2) send the employed bees to search the new food source around the neighbor of food source which is already present in her memory, (3) send onlooker bees on food sources based on food source nectar amounts (i.e., fitness values) advertised by employed bees, (4) place scout bees randomly in the search area for finding the new food sources, and (5) store the best food sources found so far into the external two archives, (6) repeat the step 2 to 5 until the termination criterion is satisfied.

4.1 Problem representation

To solve any optimization problems using a search-based meta-heuristic algorithm, the problem must be defined and encoded corresponding to the different operators of the algorithms so that problem can be tackled easily and effectively. The MaSMCP is a discrete many-objective combinatorial optimization problem. The original ABC algorithm was designed to address the continuous optimization problem. To increase its applicability to the discrete combinatorial optimization problems, researchers always converted the continuous domain into the discrete domain. And this overhead causes difficulties to the ABC algorithm.

In this paper, the particular solution of MaSMCP problem is encoded as a vector of n decision variables where the index of the vector and value at that index correspond to the modules and clusters, respectively. Let us consider a particular module clustering solution c_i which is denoted with a vector

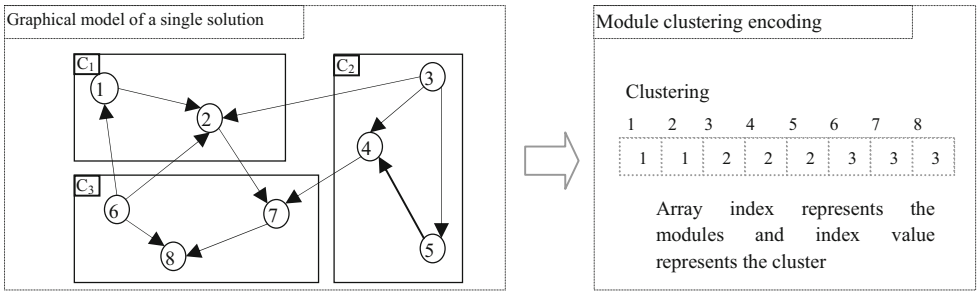

Fig. 2 Representation of a software module clustering solution

Table 1 Basic symbols and their meaning corresponding to ABC algorithm

	Symbols	ABC definition	MaSMCP definition
1.	$\vec{C_i} = (c_i^1, c_i^2, \ldots, c_i^{\|D\|})$	Food source	Candidate solution/clustering solution
2.	$\vec{C_i}$	ith food source	ith module clustering solution
3.	|D|	Number of features of a food source	Number of decision variables (i.e., modules)
4.	d	dth of features of a food source	dth of modules of a clustering solution
4.	c_i^d	Value of dth feature of the ith food source	Value of dth module of the ith solution
5.	UB^d	Upper bound for the dth feature	Upper bound for the dth dimension
6.	LB^d	Lower bound for the dth feature	Lower bound for the dth dimension
7.	|Np|	Number of food sources in population	Number of clustering solutions in population

c_i and decision variables d: $\vec{C_i} = (c_i^1, c_i^2, ..., c_i^{|D|})$. In this solution representation, each element c_i^d of vector c_i refers to a particular dimension of decision variable of the considered problem. To demonstrate it, let us consider a hypothetical software system depicted in Fig. 2.

In Fig 2, the module clustering solution for a hypothetical SMCP contains eight modules distributed in three clusters C_1, C_2 and C_3. Hence, according to our encoding method, it is represented as a vector c = [1, 1, 2, 2, 2, 3, 3, 3], where vector index from 1 to 8 represents the modules and values 1 to 3 correspond to the clusters in which that modules exist. For example modules 1 and 2 are grouped in the same cluster C_1; hence the value of the vector indexed with 1 and 2 is 1. Similarly, modules 3, 4, and 5 are in cluster C_2 and modules 6, 7, and 8 are in cluster C_3. The basic symbols and their meaning corresponding to ABC algorithm and MaSMCP are given in Table 1.

4.2 Population initialization

In MaABC each population member (i.e., food source/candidate solution) is initialized randomly using the formula $c_i^d = \text{RandInt}(UB_i^d - LB_i^d)$. The notations UB_i^d and LB_i^d correspond to the upper and lower bound on the dth decision variable for the ith member of the population; RandInt () denotes a random function that selects a random integer value between LB_i^d and UB_i^d. The main steps of the population initialization are given in Algorithm 1.

Algorithm 1: Population initialization

```
1. Generate candidate solutions/food source for the initial population
2.     For i=1 to |Np|                          /* |Np| represents the population size */
3.         For d=1 to |D|                       /* |D| represents the number of dimension */
4.             c_i^d = RandInt(UB_i^d − LB_i^d)   /* LB_i^d =1 and UB_i^d =n (i.e., Total number of classes) */
5.         End for
6.     End for
7.  Calculate each objective function for c_i candidate solution
8.  Calculate fitness of each candidate solution and Initiate Trial_1, Trial_2, ..., Trial_|Np| by 0
```

4.3 Send employed bees

In this phase, each employed bees fly toward the current module clustering solution (food source) that she has in her memory and exploit a new food source around the vicinity of the current solution. To force the employed bees toward a better neighboring solution the employee

bees are guided by current population and two archives (CA and DA). To this contribution, the value of each decision variables for a new solution is determined as follows:

$$v_i^d = \begin{cases} \text{RandInt}(c_1^d, c_2^d, \ldots, c_{|N_p|}^d) & if\, r \geq 0.5 \\ \text{RandInt}(c_1^d, c_2^d, \ldots, c_{|CA+DA|}^d) & if\, r < 0.5 \end{cases} \quad (10)$$

The RandInt function generates a value by the randomly selected dimension from the population and archives (CA + DA) with 0.5 probabilities each. The step-by-step process of the employed bees is given in Algorithm 2.

4.4 Send onlooker bees

In this step of the algorithm, each onlooker bees make the selection decision for module clustering solution advertized by the employed bees on the basis of fitness value (nectar amount). The onlooker bee selects a solution based on the probability which is calculated as follows:

$$p_i = \frac{fit(c_i)}{\sum_{m=1}^{\text{Food Number}} fit(c_m)} \quad (11)$$

where p_i is the selection probability of the module clustering solution c_i. The $fit(c_i)$ is the fitness of the module clustering solution c_i advertized by the each employed bee i. The detailed steps are given in Algorithm 3.

Algorithm 2: SendEmployedBees (population, CA, DA)

```
Send employed bees for each food sources of the population
For i=1 to |Np|        /* Determine a new single dimension d, d ∈ {1, 2, . . . , |D|} for the current food source */
    If r ≥ 0.5     /* r is random uniform random number between [0,1],*/
        v_i^d = RandInt(c_1^d, c_2^d, ..., c_|Np|^d),   /* Select a random decision variable values from population*/
    Else
        v_i^d = RandInt(c_1^d, c_2^d, ..., c_|CA+DA|^d)    /* Select a random decision variable values from
    |CA+DA|*/
    End If
    If v_i ≠ c_i Then              /* The new solution and current solution is not equal */
        Calculate the objective functions of new module clustering solution: v_i
        Calculate fitness of the new module clustering solution v_i using Eq. (7)
        If fitness of the new module clustering solution v_i improved Then
            Replace old module clustering solution c_i with new module clustering solution v_i
        Else
            Increment Trial_i by 1
        End If
    End if
End For
```

Algorithm 3: SendOnlookerBees (population, CA, DA)

```
Calculate probability value p_i based on Eq. (11) of each module clustering c_i
Set i=1, t=1;
While i< |Np| +1      /* Send the onlooker bees */
    If r_1 < p_i Then   /* r_1 is random uniform random number between [0,1],*/
        i=i+1
        If r ≥ 0.5     /* r is random uniform random number between [0,1] */
            v_i^d = RandInt(c_1^d, c_2^d, ..., c_|Np|^d),   /* Select a random decision variable values from population*/
        Else
            v_i^d = RandInt(c_1^d, c_2^d, ..., c_|CA+DA|^d)    /* Select a random decision variable values from |CA+DA|*/
        End If
        If v_i ≠ c_i Then              /* The new solution and current solution is not equal */
            Calculate the objective functions of new module clustering solution: v_i
            Calculate fitness of the new module clustering solution v_i using Eq. (7)
            If fitness of the new module clustering solution v_i improved Then
                Replace old module clustering solution c_i with new module clustering solution v_i
            Else
                Trial_i = Trial_i +1    /* increment Trial_i value by one */
            End If
        End If
    t=t+1
    If t > |Np| Then
        i=1     /* Reset the value of i */
    End If
End While
```

4.5 Send scout bees

In this phase, if the food source corresponding to the module clustering solution does not improve in certain iteration, then it is abandoned from the population. The employed bees related to the abandoned food source turn into a scout bee, and the corresponding food source is changed with a module clustering solution that is generated in the same manner as that in the population initialization phase. The detailed steps are given in Algorithm 4.

Algorithm 4: SendScoutBees (population, Limit)

```
For i=1 to |Np|   If there exists some module clustering solution c_i | {trial_i > t},
    If max (Trial_i)>limit
        For d=1 to |D|                    /* |D| represents the number of dimension */
            v_i^d = RandInt (UB_i^d − LB_i^d)    /* LB_i^d =1 and UB_i^d =n (i.e., Total number of classes) */
        End For
        Calculate the objective functions of new module clustering solution: v_i
        Calculate fitness of the new module clustering solution v_i
        If fitness of the new module clustering solution v_i improved Then
            Replace old module clustering solution c_i with new module clustering solution v_i
        Else
            Retain old module clustering solution c_i
        End If
    End If
    Set Trial_i=0,
End For
```

4.6 Update CA and DA archives

To guide the employed and onlooker bees in a good direction, the MaABC algorithm uses the two external archives concepts inspired by the work presented in Wang et al. (2015). The two archives are used to store best module clustering solutions found in each generation. These archives are named as CA (convergence archive) and DA (diversity archive) with equal fixed size. Both CA and DA archives are updated according to Algorithm 5.

Algorithm 5: UpdateArchives(CA, DA)

```
Parameters: FS-food source, CA, DA, Total size |CA+DA|=Np;
Collect FS_nd the set of non-dominated food sources from the current population   /* Addition Strategy */
For i=1 to | FS_nd|  do
    If FS_nd[i] cannot be dominated by any food source in either DA or CA archive then
        If FS_nd[i] dominates any food source stored in either DA or CA archive then
            The dominated food sources stored in DA and CA archive are removed
            FS_nd[i].FLAG=1. /* The non-dominated food source with domination */
        Else
            FS_nd[i].FLAG=0. /* The non-dominated food source without domination */
            Add the FS_nd[i] to archive CA
        End If
    Else
        FS_nd[i].FLAG=-1. /* The dominated food source */
    End If
End For
Insert the food sources with FLAG==1 to CA archive and food sources with FLAG==0 to DA archive
If |DA|> |Np/2|  and |CA|>|Np/2| the total size of DA and CA archives overflows
        Apply the selection mechanism to delete the extra food sources from DA and CA archives
End if
End For
```

4.7 Selection in overflowed CA and DA

If the number of solutions in the CA and DA increases in the size of CA and DA, then the algorithm activates the selection method to remove the module clustering solution corresponding to the food sources from both CA and DA archives. To select the module clustering solution from the CA archive, we use the concept of quality indicator $I_{\varepsilon+}$ given in IBEA (Zitzler and Künzli 2004) discussed in Sect. 3.2 The food source with low rank is selected from the CA and then removed. This selection strategy specially is used to ensure the diversity in the CA archive. Finally, an updated CA with a fixed number of module clustering solutions can be obtained. On the other hand, in case of DA overflow, we select and delete the module clustering solution of DA that has a minimum Lp-norm ($p < 1$) (i.e., discussed in Sect. 3.3) distance from the module clustering solutions of the CA archives.

4.8 Termination

The steps send employed bee, send onlooker bee, send scout bee, and update archive iterate cycle by cycle until the termination condition is met.

Algorithm 6: MaABC Algorithm

Input: Class dependency matrix, |PS|, CA, DA, Limit
Number of dimensions |D|
Output: CA and DA archives with Pareto optimal solutions

1. **Initialization:** Initialize the population, trials
2. UpdateArchives (CA,DA)
3. **While** (convergence criterion is satisfied)
4. SendEmployedBees (population, CA , DA)
5. SendOnLookerBees (population, CA , DA)
6. SendScoutBees (population,limit)
7. UpdateArchives (CA,DA)
8. **End While**
9. **Return** CA and DA archives

By the termination of the MaABC algorithm, the external archive CA and DA are returned as the output. In our implementation, the MaABC terminates after a predefined number of fitness evaluations.

5 Experimental setup

This section provides detailed description about the experimental setup to assess the proposed MaABC algorithm.

5.1 Software systems

In order to evaluate the performance of our proposed MaABC algorithm, we have tested it on seven open-source software systems (i.e., JFreeChart, JHotDraw, JavaCC, JUnit, Java Servlet API, XML API DOM, and DOM 4 J) with different characteristics. These software systems have also been used to evaluate the similar clustering techniques by the previous researchers (Amarjeet and Chhabra 2014; Mkaouer et al. 2015; Amarjeet and Chhabra 2015, 2017b). Table 2 provides a brief summary of the software systems.

Table 2 Characteristics of selected software projects

Systems	Version	#Modules	#Dependencies
JFreeChart	0.9.21	401	1420
JHotDraw	6.0*b*1	398	2175
JavaCC	1.5	154	722
JUnit	3.81	100	276
Java Servlet API	2.3	63	131
XML API DOM	1.0.*b*2	119	209
DOM 4 J	1.5.2	195	930

5.2 Research questions

To evaluate the effectiveness of MaABC, we answer the following research questions.

RQ1. Does proposed approach produce clustering solution having a better MQ compared to existing approaches?
RQ2. Does proposed approach produce clustering solution having a better coupling compared to existing approaches?
RQ3. Does proposed approach produce clustering solution having a better cohesion compared to existing approaches?
RQ4. Does proposed approach produce clustering solution having a better IGD compared to existing approaches?

To answer these research questions, we cluster the seven software systems using the proposed MaABC and existing many-objective meta-heuristic search algorithms.

5.3 Existing algorithms and parameter settings

In order to verify the performance of the proposed many-objective algorithm, the four state-of-the-art many-objective meta-heuristic search algorithms (i.e., Two-Arch2 (Wang et al. 2015), NSGA-III (Deb and Jain 2014), MOEA/D (Zhang and Li 2007), and IBEA (Zitzler and Künzli 2004)) are considered. These algorithms are implemented in jMetal, a multi-objective meta-heuristic framework, on a 16-core 2.60 GHz Intel Xeon CPU with 8 GB RAM. The parameter settings of each algorithm (i.e., proposed and state-of-the-art algorithms) are given in Table 3. To have a fair comparison of considered algorithms, each of the algorithms is given equal

Table 3 Algorithms and their parameter values

#	Algorithms	Parameter	Values
1	NSGA-III	Population size	10*N
		Number of fitness evaluations	200* N
		Crossover weight	0.8–1.0
		Mutation weight	0.04 *$\log_2(N)$
		Reference points	100
		Number of division	5
2	IBEA	Archive size	10*N
		Number of fitness evaluations	200* N
3	MOEA/D	Neighborhood size	20
		Max replacement	5
		H	200
		Number of fitness evaluations	200* N
4	Two_Arch 2	Population size	10*N
		Crossover weight	0.8 to 1.0
		Mutation weight	0.04 *$\log_2(N)$
		Number of fitness evaluations	200* N
		Size of CA	5*N
		Size of DA	5*N
5	MaABC	Number of food sources (population size)	10*N
		Number of fitness evaluations	200* N
		Size of CA (convergence archive)	5*N
		Size of DA (divergence archive)	5*N
		Number of employed bees	10*N
		Number of onlooker bees	10*N
		Number of scout bees	0.05*N

* N represents the number of modules of the studies software systems

number of number of fitness evaluations (N_{FE}) (Črepinšek et al. 2014).

5.4 Collecting results and statistical tests

The many-objective meta-heuristic search algorithms are stochastic optimizer, i.e., they can generate different results on each run over the same problem. In this experiment, the results are collected from the experiment by executing each of the meta-heuristics on each problem instance by 31 independent simulation runs.

6 Results and analysis

In order to evaluate the behavior of our proposed algorithm, we compare MaABC with Two_Arch2, NSGA-III, MOEA/D, and IBEA on 7 many-objective software module clustering problems with 7 numbers of objective functions. The compared meta-heuristic algorithms are all representatives of multi-objective evolutionary algorithms for many-objective optimization problems. The results of MaABC, Two_Arch2, NSGA-III, MOEA/D, and IBEA on the each problem are shown in Tables 4, 5, 6, 7, 8, 9, 10, 11, where the algorithms are statistically analyzed by using Wilcoxon's rank sum test with 95% confidence level ($\alpha = 0.05$) (Arcuri and Fraser 2013). Sections 6.1, 6.2, 6.3, and 6.4 present comparison among MaABC, Two_Arch2, NSGA-III, MOEA/D, and IBEA algorithms regarding MQ, coupling, cohesion, and IGD, respectively.

6.1 The MQ value as assessment criterion

In this section, the proposed software module clustering approach, MaABC, was compared with Two_Arch2, NSGA-III, MOEA/D, and IBEA in terms of MQ values. Table 4 presents metric MQ in terms of median and standard deviation obtained from different meta-heuristic algorithms with E-MCA formulation on the seven problem suite. Similarly, Table 5 presents metric MQ in terms of median and standard deviation obtained from different meta-heuristic algorithms with E-ECA formulation on the seven problem suite.

If we see the results presented in Table 4, it clearly indicates that the MQ values achieved by the MaABC

Table 4 Median MQ values obtained from algorithms with E-MCA formulation

Systems	MaABC	Two_Arch2	NSGA-III	MOEA/D	IBEA
JFreeChart	18.973 (0.351)	18.033 (0.230)[−]	17.584 (0.505)[−]	18.028 (0.057) [−]	17.852 (0.223)[−]
JHotDraw	12.671 (0.531)	11.853 (0.497)[−]	12.628 (0.611)[≈]	11.886 (0.548)[−]	11.871 (0.477)[−]
JavaCC	6.529 (0.126)	6.928 (0.108) [+]	6.462 (0.103)[−]	5.851 (0.060)[−]	5.659 (0.150)[−]
JUnit	6.479 (0.274)	6.162 (0.262)[−]	6.290 (0.123)[−]	6.370 (0.182)[≈]	6.139 (0.139)[−]
Java Servlet API	3.514 (0.342)	3.056 (0.207)[−]	2.914 (0.144)[−]	2.884 (0.120)[−]	2.874 (0.149)[−]
XML API DOM	5.550 (0.472)	5.299 (0.283)[≈]	5.153 (0.441)[−]	5.123 (0.371)[−]	4.883 (0.292)[−]
DOM 4 J	12.476 (0.589)	11.568 (0.407)[−]	11.461 (0.541)[−]	11.399 (0.435)[−]	11.546 (0.554)[−]

Wilcoxon's rank sum test at a 0.05 significance level is performed between MaABC and each of Two_Arch2, NSGA-III, MOEA/D, and IBEA. The symbols "−", "+," and "≈" denote the performance of the corresponding algorithm is significantly worse than, better than, and not significantly different than that of the proposed MaABC, respectively

Table 5 Median MQ values obtained from algorithms with E-ECA formulation

Systems	MaABC	Two_Arch2	NSGA-III	MOEA/D	IBEA
JFreeChart	19.167 (0.434)	18.195 (0.216)[−]	18.096 (0.233)[−]	18.077 (0.277)[−]	18.116 (0.250)[−]
JHotDraw	12.923 (0.264)	12.347 (0.394)[≈]	12.347 (0.662)[≈]	12.182 (0.379)[−]	11.996 (0.440)[−]
JavaCC	7.464 (0.153)	6.962 (0.110)[≈]	6.943 (0.200)[≈]	6.754 (0.192)[−]	6.619 (0.228)[−]
JUnit	7.197 (0.234)	6.227 (0.220)[−]	6.397 (0.139)[−]	6.438 (0.132)[−]	6.130 (0.256)[−]
Java Servlet API	3.856 (0.217)	3.588 (0.123)[≈]	3.474 (0.149)[−]	3.423 (0.173)[−]	3.460 (0.182)[−]
XML API DOM	6.158 (0.528)	5.415 (0.342)[−]	5.246 (0.339)[−]	5.140 (0.804)[−]	5.418 (0.483)[−]
DOM 4 J	12.369 (0.482)	11.668 (0.725)[≈]	11.283 (0.555)[−]	11.472 (0.606)[−]	11.556 (0.415)[−]

Wilcoxon's rank sum test at a 0.05 significance level is performed between MaABC and each of Two_Arch2, NSGA-III, MOEA/D, and IBEA. The symbols "−", "+," and "≈" denote the performance of the corresponding algorithm is significantly worse than, better than, and not significantly different than that of the proposed MaABC, respectively

Table 6 Median coupling values obtained from algorithms with E-MCA formulation

Systems	MaABC	Two_Arch2	NSGA-III	MOEA/D	IBEA
JFreeChart	656.51 (47.45)	687.81 (42.51)[≈]	704.52 (46.10)[−]	741.29 (43.36)[−]	765.13 (89.28)[−]
JHotDraw	789.76 (33.12)	813.10 (41.07)[−]	815.45 (45.40)[−]	827.06 (44.95)[−]	832.97 (62.22)[−]
JavaCC	214.03 (23.78)	227.97 (33.44)[≈]	230.77 (26.50)[−]	231.42 (13.98)[−]	245.32 (32.29)[−]
JUnit	76.25 (21.21)	91.74 (14.67)[−]	92.65 (10.52)[−]	82.81 (8.24)[−]	91.97 (20.85) [−]
Java Servlet API	21.13 (11.78)	26.03 (8.85)[≈]	25.65 (4.71)[≈]	31.74 (8.44)[−]	31.97 (9.45)[−]
XML API DOM	65.31 (14.48)	76.65 (11.37)[−]	79.74 (20.66)[−]	81.48 (12.50)[−]	82.97 (9.52)[−]
DOM 4 J	241.54 (83.18)	234.94 (74.66)[≈]	255.45 (47.64)[≈]	243.19 (71.03)[≈]	253.26 (69.39)[≈]

Wilcoxon's rank sum test at a 0.05 significance level is performed between MaABC and each of Two_Arch2, NSGA-III, MOEA/D, and IBEA. The symbols "−", "+," and "≈" denote the performance of the corresponding algorithm is significantly worse than, better than, and not significantly different than that of the proposed MaABC, respectively

approach are significantly better over the most of the problem instances. Most specifically, if we compare the MQ values of MaABC with MQ values of each individual algorithm, we observed that (1) the comparative results between MaABC and Two_Arch2 algorithms show that the MaABC is able to achieve better Two_Arch2 in six problems out of seven, in which five cases are significantly better. (2) The comparison results between MaABC and NSGA-III show that the MaABC approach performs better in six problems out of seven problems, in which 4 cases are significantly better. (3) The comparative results between MaABC and other two algorithms (i.e., MOEA/D and IBEA) show that the MaABC approach outperforms MOEA/D and IBEA in most of the cases.

The MQ values achieved by each of the algorithm with E-ECA formulation presented in Table 5 show that the proposed MaABC approach outperforms in most of the cases. The MaABC performs better than Two_Arch2 in all cases, in which three cases are significantly better. The MaABC also performs better than NSGA-III in all cases, in which five

Table 7 Median coupling values obtained from algorithms with E-ECA formulation

Systems	MaABC	Two_Arch2	NSGA-III	MOEA/D	IBEA
JFreeChart	641.71 (42.15)	665.03 (38.62)[≈]	691.00 (45.70)[−]	684.90 (42.58)[−]	684.42 (44.91)[−]
JHotDraw	731.10 (63.24)	803.10 (74.15)[−]	817.13 (73.08)[−]	780.77 (63.81)[≈]	757.68 (68.03)[≈]
JavaCC	195.33 (45.17)	197.55 (49.16)[−]	234.55 (26.16)[≈]	226.97 (59.54)[≈]	227.32 (32.15)[≈]
JUnit	63.97 (15.39)	74.35 (10.39)[−]	80.35 (80.35)[−]	82.61 (5.44)[−]	74.65 (8.58)[−]
Java Servlet API	18.13 (7.65)	30.00 (8.82)[−]	29.97 (8.35)[−]	28.58 (3.07)[−]	30.13 (5.91)[−]
XML API DOM	71.53 (4.31)	76.55 (10.08)[≈]	79.52 (13.19)[≈]	76.23 (16.03)[≈]	79.55 (5.81)[−]
DOM 4 J	235.53 (63.18)	241.74 (75.22)[≈]	284.48 (42.05)[−]	262.61 (70.79)[−]	283.39 (33.24)[−]

Wilcoxon's rank sum test at a 0.05 significance level is performed between MaABC and each of Two_Arch2, NSGA-III, MOEA/D, and IBEA. The symbols "−", "+," and "≈" denote the performance of the corresponding algorithm is significantly worse than, better than, and not significantly different than that of the proposed MaABC, respectively

Table 8 Cohesion values obtained from algorithms with E-MCA formulation

Systems	MaABC	Two_Arch2	NSGA-III	MOEA/D	IBEA
JFreeChart	1454.43 (56.36)	1414.19 (42.51)[≈]	1397.48 (46.10)[−]	1360.71 (43.36)[−]	1336.87 (89.28)[−]
JHotDraw	1391.22 (25.15)	1361.90 (41.07)[−]	1359.55 (45.40)[−]	1347.94 (44.95)[−]	1342.03 (62.22)[−]
JavaCC	514.23 (32.81)	494.03 (33.44)[≈]	491.23 (26.50)[≈]	490.58 (13.98)[−]	476.68 (32.29)[−]
JUnit	201.65 (23.13)	184.26 (14.67)[−]	183.35 (10.52)[−]	193.19 (8.24)[−]	184.03 (20.85)[−]
Java Servlet API	114.74 (11.25)	104.97 (8.85)[≈]	105.35 (4.71)[≈]	99.26 (8.44)[−]	99.03 (9.45)[−]
XML API DOM	148.31 (23.57)	132.34 (11.19)[−]	129.63 (20.43)[−]	127.75 (12.37)[−]	126.34 (9.53)[−]
DOM 4 J	693.14 (65.21)	685.06 (74.66)[≈]	674.55 (47.64)[≈]	686.81(71.03)[≈]	676.74 (69.39)[≈]

Wilcoxon's rank sum test at a 0.05 significance level is performed between MaABC and each of Two_Arch2, NSGA-III, MOEA/D, and IBEA. The symbols "−", "+," and "≈" denote the performance of the corresponding algorithm is significantly worse than, better than, and not significantly different than that of the proposed MaABC, respectively

Table 9 Cohesion values obtained from algorithms with E-ECA formulation

Systems	MaABC	Two_Arch2	NSGA-III	MOEA/D	IBEA
JFreeChart	1471.14 (43.13)	1436.97 (38.62)[≈]	1411.00 (45.70)[−]	1417.10 (42.58)[≈]	1417.58 (44.91)[−]
JHotDraw	1421.26 (23.24)	1371.90 (74.15)[−]	1357.87 (73.08)[−]	1394.23 (63.81)[≈]	1417.32 (63.03)[≈]
JavaCC	531.14 (49.25)	524.45 (49.16)[−]	487.45 (26.16)[−]	495.03 (59.54)[−]	494.68 (32.15)[−]
JUnit	213.13 (23.31)	201.65 (10.39)[≈]	195.65 (18.41)[−]	193.39 (5.44)[−]	201.35 (8.58)[≈]
Java Servlet API	115.83 (5.13)	101.00 (8.82)[−]	101.03 (8.35)[−]	102.42 (3.07)[−]	100.87 (5.91)[−]
XML API DOM	153.45 (10.21)	132.84 (10.16)[≈]	130.19 (13.58)[≈]	133.28 (16.03)[≈]	129.63 (5.80)[−]
DOM 4 J	694.42 (63.21)	688.26 (75.22)[≈]	645.52 (42.05)[−]	667.39 (70.79)[≈]	646.61 (33.24)[−]

Wilcoxon's rank sum test at a 0.05 significance level is performed between MaABC and each of Two_Arch2, NSGA-III, MOEA/D, and IBEA. The symbols "−", "+," and "≈" denote the performance of the corresponding algorithm is significantly worse than, better than, and not significantly different than that of the proposed MaABC, respectively

cases are significantly better. The MOEA/D and IBEA perform significantly worst MaABC in all problem instances. The results also show that the MQ values achieved by MaABC with E-ECA formulation are more effective compared to E-MCA formulation.

Figure 3 presents the results of MQ achieved by each of the considered algorithms with E-MCA and E-ECA formulation over all seven problem instances. From Fig. 3, it is clear that the proposed MaABC is able to generate clustering solution having better MQ values compared to Two_Arch2, NSGA-III, MOEA/D, and IBEA.

6.2 Coupling as an assessment criterion

To answer the research question RQ2, the software module clustering generated by proposed approach and rival algorithms has been evaluated in terms of coupling as an assessment criterion. The values of coupling obtained by MaABC and existing many-objective optimization algorithms with E-MCA many-objective software clustering formulation over all seven problems are shown in Table 6. The values of coupling obtained by MaABC and existing many-objective optimization algorithms with E-ECA many-objective soft-

Table 10 IGD values obtained from algorithms with E-MCA formulation

Systems	MaABC	Two_Arch2	NSGA-III	MOEA/D	IBEA
JFreeChart	5.721×10^{-4}	$5.740 \times 10^{-4}[\approx]$	$6.283 \times 10^{-4}[-]$	$6.615 \times 10^{-4}[-]$	$6.721 \times 10^{-4}[-]$
JHotDraw	7.765×10^{-3}	$7.954 \times 10^{-3}[-]$	$7.785 \times 10^{-3}[\approx]$	$8.008 \times 10^{-3}[-]$	$8.092 \times 10^{-3}[-]$
JavaCC	5.464×10^{-3}	$5.492 \times 10^{-3}[\approx]$	$5.271 \times 10^{-3}[+]$	$5.721 \times 10^{-3}[-]$	$5.734 \times 10^{-3}[-]$
JUnit	4.001×10^{-4}	$4.194 \times 10^{-4}[-]$	$4.229 \times 10^{-4}[-]$	$4.321 \times 10^{-4}[-]$	$4.343 \times 10^{-4}[-]$
Java Servlet API	6.813×10^{-4}	$7.062 \times 10^{-4}[-]$	$7.212 \times 10^{-4}[-]$	$7.314 \times 10^{-4}[-]$	$7.257 \times 10^{-4}[-]$
XML API DOM	6.182×10^{-3}	$6.365 \times 10^{-3}[-]$	$6.413 \times 10^{-3}[-]$	$6.478 \times 10^{-3}[-]$	$6.495 \times 10^{-3}[-]$
DOM 4 J	7.856×10^{-3}	$8.172 \times 10^{-3}[-]$	$8.215 \times 10^{-3}[-]$	$8.392 \times 10^{-3}[-]$	$8.412 \times 10^{-3}[-]$

Wilcoxon's rank sum test at a 0.05 significance level is performed between MaABC and each of Two_Arch2, NSGA-III, MOEA/D, and IBEA. The symbols "−", "+," and "≈" denote the performance of the corresponding algorithm is significantly worse than, better than, and not significantly different than that of the proposed MaABC, respectively

Table 11 IGD values obtained from algorithms with E-ECA formulation

Systems	MaABC	Two_Arch2	NSGA-III	MOEA/D	IBEA
JFreeChart	5.521×10^{-4}	$5.533 \times 10^{-4}[\approx]$	$6.261 \times 10^{-4}[-]$	$6.723 \times 10^{-4}[-]$	$6.645 \times 10^{-4}[-]$
JHotDraw	7.665×10^{-3}	$7.734 \times 10^{-3}[-]$	$8.241 \times 10^{-3}[-]$	$8.123 \times 10^{-3}[-]$	$8.012 \times 10^{-3}[-]$
JavaCC	5.261×10^{-3}	$5.386 \times 10^{-3}[-]$	$5.671 \times 10^{-3}[-]$	$5.721 \times 10^{-3}[-]$	$5.734 \times 10^{-3}[-]$
JUnit	4.004×10^{-4}	$4.104 \times 10^{-4}[\approx]$	$4.106 \times 10^{-4}[\approx]$	$4.124 \times 10^{-4}[-]$	$4.123 \times 10^{-4}[-]$
Java Servlet API	6.662×10^{-4}	$7.125 \times 10^{-4}[-]$	$7.212 \times 10^{-4}[-]$	$7.314 \times 10^{-4}[-]$	$7.341 \times 10^{-4}[-]$
XML API DOM	6.183×10^{-3}	$6.365 \times 10^{-3}[-]$	$6.324 \times 10^{-3}[-]$	$6.502 \times 10^{-3}[-]$	$6.328 \times 10^{-3}[-]$
DOM 4 J	8.021×10^{-3}	$8.218 \times 10^{-3}[-]$	$8.187 \times 10^{-3}[-]$	$8.356 \times 10^{-3}[-]$	$8.512 \times 10^{-3}[-]$

Wilcoxon's rank sum test at a 0.05 significance level is performed between MaABC and each of Two_Arch2, NSGA-III, MOEA/D, and IBEA. The symbols "−", "+," and "≈" denote the performance of the corresponding algorithm is significantly worse than, better than, and not significantly different than that of the proposed MaABC, respectively

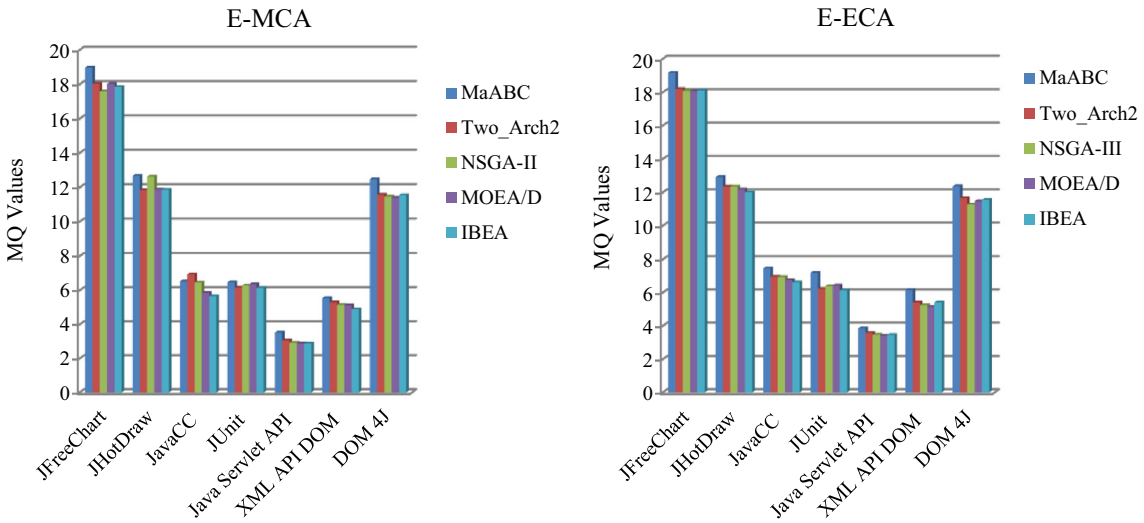

Fig. 3 Comparison of MQ values obtained from algorithms with E-EMA and E-ECA

ware clustering formulation are shown in Table 7. To make the distinction among each algorithm, the coupling results are also demonstrated using bar charts which are demonstrated in Fig. 4.

In cases of E-MCA formulation, the coupling results reported in Table 6 show that the MaABC performs better than Two_Arch2, NSGA-III, MOEA/D, and IBEA approaches in all seven software module clustering problems, in which most of the cases are significantly better in favor of MaABC. Similarly, in case of E-ECA formulation the coupling results demonstrated in Table 7 show that the MaABC also performs better Two_Arch2, NSGA-III, MOEA/D, and IBEA

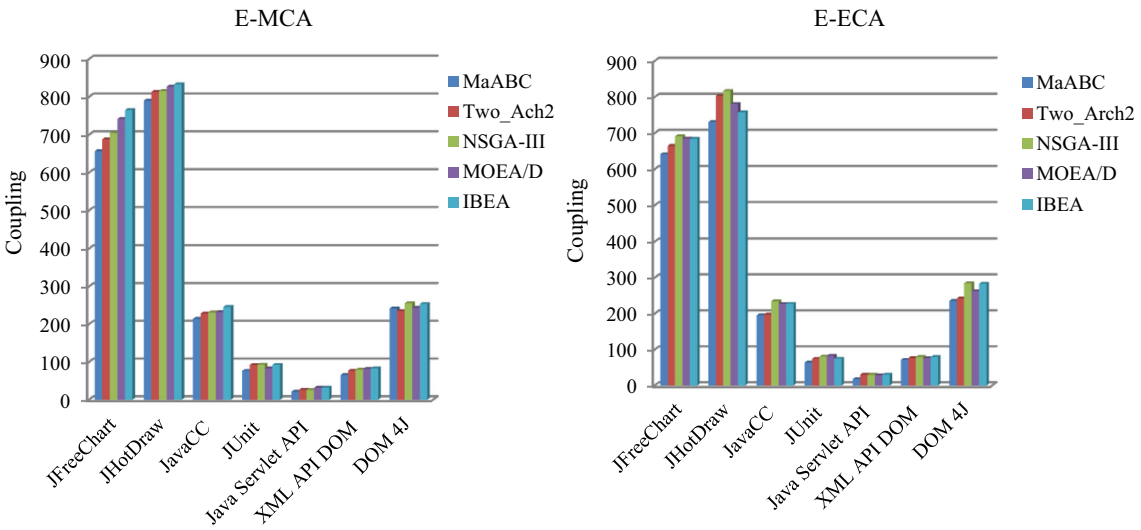

Fig. 4 Comparison of coupling values obtained from algorithms with E-EMA and E-ECA

approaches in all seven problem instances in which most of the cases are significantly better in favor of MaABC.

Now if we compare the coupling results between E-MCA and E-ECA formulation, the coupling results demonstrated in Tables 6 and 7 also show that the module clustering solution obtained by MaABC with E-ECA is more effective compared to E-MCA formulation.

The achieved coupling results of MaABC, Two_Arch2, NSGA-III, MOEA/D, and IBEA are also demonstrated in Fig. 4. It is clearly observed that the coupling values of MaABC presented in Fig. 4 are better in most of the cases compared to Two_Arch2, NSGA-III, MOEA/D, and IBEA with both E-MCA and E-ECA formulations.

6.3 Cohesion as an assessment criterion

To answer the research question RQ2, this section compares the cohesion of module clustering solutions produced by the proposed MaABC approach and existing many-objective meta-heuristic algorithms (i.e., Two_Arch2, NSGA-III, MOEA/D, and IBEA). To validate the answer of approach, each algorithm was analyzed with two software module clustering many-objective formulations (i.e., E-MCA and E-ECA) over all seven module clustering problems.

Table 8 compares the median and standard deviation results in terms of cohesion between MaABC and each of the existing rival many-objective meta-heuristic algorithms (i.e., Two_Arch2, NSGA-III, MOEA/D, and IBEA) with E-MCA formulation over seven SMCPs. Similarly, Table 9 compares the median and standard deviation results in terms of cohesion between MaABC and each of the existing rival many-objective meta-heuristic algorithms with E-MCA formulation over seven SMCPs.

For all the problem instances, the cohesion values presented in Table 8 clearly indicate that the proposed MaABC is able to achieve better cohesion when compared with each algorithm, in which most of the cases are statistically significant in favor of MaABC. Similarly, the cohesion results of the E-ECA formulation presented in Table 9 also indicate that the MaABC is able to achieve better cohesion values compared to existing algorithms in all cases in which most of the cases are significantly better.

Now if we compare the cohesion results between E-MCA formulation (for all problem instances and for all algorithms) and E-ECA formulation (for all problem instances and for all algorithms), it can be easily observed that the each algorithm on each problem instance the E-ECA formulation is able to generate better cohesion results compared to E-MCA formulation.

Figure 5 presents the results of cohesion achieved by each of the considered algorithms with E-MCA and E-ECA formulation over all seven problem instances. From Fig. 5, it is clear that the proposed MaABC is able to generate clustering solution having better cohesion values compared to Two_Arch2, NSGA-III, MOEA/D, and IBEA.

6.4 IGD as assessment criterion

In the previous sections, we compared MaABC with existing many-objective algorithms (i.e., Two_Arch2, NSGA-III, MOEA/D, and IBEA) in terms of structural quality metrics (i.e., MQ, coupling, and cohesion). In this section, we compare the MaABC with existing algorithms in terms of IGD values for both E-MCA and E-ECA formulations. Tables 10 and 11 show the median IGD values for MaABC and all rival algorithms under comparison. For the E-MCA formulation, Table 10 shows that MaABC outperforms other algorithms,

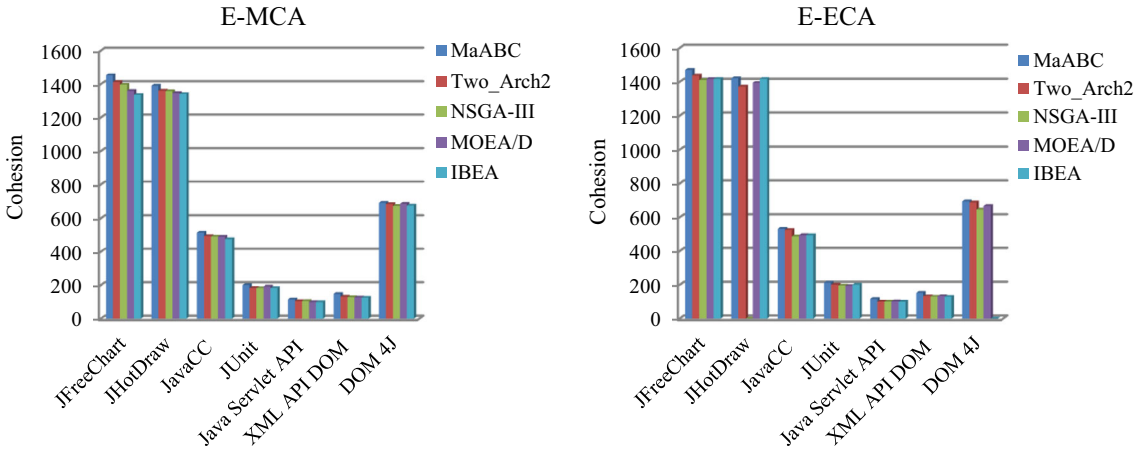

Fig. 5 Comparison of cohesion values obtained from algorithms with E-EMA and E-ECA

except in the JavaCC problem instance where MaABC performs significantly worse NSGA-III. In existing algorithms, Two_Arch2 performs better than NSGA-II, MOEA/D, and IBEA in most of the cases. Additionally, NSGA-III seems to be better MOEA/D and IBEA. For E-ECA formulation, the IGD values presented in Table 11 indicate that the MaABC performs significantly better existing approaches in most of the cases.

6.5 Results summary

This section discusses the contributions and implications of our proposed MaABC approach in solving the MaSM-CPs using proposed MaABC. The main contribution of the MaABC with respect to existing MOEAs is that the MaABC integrates the external archive and indicator-based concepts into the ABC algorithm to achieve a good balance between exploration and exploitation in case of MaSM-CPs. The results presented in Sects. 3.1 to 3.4 showed that the MaABC performed better compared to the existing many-objective optimization approaches (i.e., Two_Arch2, NSGA-II, MOEA/D, and IBEA) in terms of MQ, coupling, cohesion, and IGD in most of the cases. Hence, it can be concluded that the proposed approach can be a good alternate for solving the software clustering problems containing many-objective functions compared to other existing approaches.

7 Threats to validity

To clarify the limitations and strengths of our proposed multi-objective software module clustering approach, we explore the factors that could influence the validity of the achieved results. There are two major categories of threats (i.e., external validity and internal validity) that could affect the results of proposed approach.

External validity (or selection validity) corresponds to the degree to which obtained results (samples) of the approach can be generalized to the broad perspective of problems. In search-based software engineering, this is a very important threat to validity of results because of the large number of diverse software projects available to any study of software module clustering. In our experimentation, this threat to validity has been mitigated by the fact that the proposed approach is concerned with module dependency graph (MDG), an abstract representation of software systems. Since there is a many to one relation between the software systems and MDG (i.e., many individual software systems can map into a single MDG), the findings for a set of MDGs of a particular size are relevant to wider MDGs. In order to mitigate the possible external threats to validity, the experimentation uses various size of MDGs, both un-weighted and weighted.

Internal validity corresponds to the degree to which conclusions can be drawn about the causal effect of independent variables on the dependent variables. In this empirical study, possible internal threats come from violations of statistical assumptions or inappropriate statistical tests, inaccurate underlying analysis, and the extent to which the variables used in the experiment precisely measure the concepts they claim to measure. In this empirical study, we use Wilcoxon's rank sum test. Wilcoxon's rank sum test is more appropriate when the type of distribution of data is unknown.

8 Related works

To address the SMCPs, large numbers of analytical and search-based approaches have been proposed. The formu-

lation of SMCPs as search-based optimization problem and solving it using search-based meta-heuristic algorithms gained wide attention. Based on the problem formulation, the search-based software module clustering approaches can be categorized into single-objective, multi-objective, and many-objective software module clustering. The subsequent subsections discuss literature background of these approaches.

8.1 Single-objective optimization approaches

In the works by Mancoridis et al. (1998), the authors proposed a single-objective search technique to cluster the software systems. In this contribution, they adapted the genetic algorithm (GA) as search algorithm and designed a modularization quality (MQ) metric to guide the search process. In order to improve the MQ's effectiveness, the metric was redefined over the years (Mitchell and Mancoridis 2002; Praditwong et al. 2011). Later the authors (Mancoridis et al. 1999) proposed a single-objective optimization tool named as Bunch for software module clustering. The effectiveness of the Bunch was evaluated over medium- and large-size software systems.

Doval et al. (1999) proposed a single-objective optimization technique for finding the good partition of the module dependency graph (MDG). The MDG is a graphical representation of software systems, where nodes represent the software elements and edges represent the dependency between elements. They used GA as search technique and MQ as fitness function. The work (Mitchell and Mancoridis 2002) adapted genetic algorithm, simulated annealing, and hill-climbing algorithms to cluster the software systems.

Harman et al. (2002) introduced an encoding scheme for representation of the software clustering solution which reduces the size of the search space and proposed new crossover method to support the retention and formation of building blocks. They claimed that the new designed crossover method is more effective for genetic algorithms compared to the standard crossover method. Mahdavi et al. (2003) presented a multiple hill-climbing algorithm to cluster the software systems. They evaluated the proposed approach over 19 software projects and found that presented approach has generated good results. The works (Mitchell et al. 2004) illustrated that designing the search landscape for search-based meta-heuristic optimization is a good approach for gaining insight into the search algorithms.

Harman et al. (2005) performed an empirical study for evaluation of robustness of two fitness function (i.e., MQ and EVM). The robustness results of MQ and EVM were compared. The comparison results showed that the both fitness function degrades slowly as the percentage of mutation is applied, but the EVM fitness function emerges to be more robust compared to MQ fitness function. Praditwong (2011) proposed a new grouping genetic algorithm (GGA) to address the single-objective software module clustering problems.

The works (Jinhuang and Jing 2016) introduced a new quality metric, namely modularization similarity (MS), for software clustering problem. The MS is based on the structural similarity and experience and knowledge of software design. The main goal of the MS is to guide the module clustering process toward good-quality clustering solutions. They also compared the clustering results obtained through MS with the results of MQ. The comparison results demonstrated that the MS outperforms the basic MQ. Table 12 provides a brief description of about the single-objective software module clustering approaches.

8.2 Multi-objective optimization approaches

Praditwong et al. (2011) presented two multi-objective formulations, namely equal-size cluster approach (ECA) and maximizing cluster approach (MCA), for software module clustering problems. Each of the MCA and ECA multi-objective formulations consists of five different partial conflicting objective functions. To optimize each MCA and ECA formulation, they used a Two-Archive-based GA (Praditwong and Yao 2006). They evaluated the effectiveness of MCA and ECA formulations over the real-world module clustering problems. Their experimental results claimed that the proposed MCA and ECA-based multi-objective approaches are able to generate significantly better clustering results than the existing single-objective clustering approaches.

Barros (2012) conducted an empirical study to assess the effectiveness and efficiency of two multi-objective formulations while searching clustering solution with multi-objective meta-heuristic algorithms. The works (Kumari et al. 2013) were claimed to be the first for applying the hyper-heuristics multi-objective approach to address the software module clustering problems. They evaluated the supremacy of the proposed work on six software systems with multiple conflicting objective functions. They reported that the hyper-heuristics-based multi-objective approach requires less computational cost in generating better-quality clustering solutions compared to existing approaches.

The works (Amarjeet and Chhabra 2014) presented a multi-objective software module clustering approach aiming to preserve the core components of the existing software modularization while clustering process of software systems. The approach was evaluated on seven software clustering problems with MCA and ECA multi-objective module clustering formulations. Their empirical results claimed that the approach was able to preserve the core components and at the same time it also achieved the desired clustering qual-

Table 12 Summary of relevant single- and multi-objective optimization approaches

Reference	Objective function	Search technique	No. of systems	No. of systems
Mancoridis et al. (1999)	MQ	GA	5	Single-objective
Mancoridis et al. (1999)	MQ	GA, HC, SA	1	Single-objective
Doval et al. (1999)	MQ	GA	1	Single-objective
Mitchell and Mancoridis (2002)	MQ	GA, HC, SA	5	Single-objective
Harman et al. (2002)	Coupling, cohesion	GA, HC	7	Single-objective
Mahdavi et al. (2003)	MQ	HC	19	Single-objective
Mitchell et al. (2004)	MQ	GA	1	Single-objective
Harman et al. (2005)	MQ, EVM	HC	6	Single-objective
Praditwong et al. (2011)	MQ	GGA	17	Single-objective
Praditwong et al. (2011)	MCA, ECA	Two-Archive GA	17	Multi-objective
Barros (2012)	MCA, ECA	NSGA-II	13	Multi-objective
Kumari et al. (2013)	MCA, ECA	Hyper-heuristics	6	Multi-objective
Amarjeet and Chhabra (2014)	MCA, ECA	NSGA-II	7	Multi-objective
Amarjeet and Chhabra (2017c)	MCA, ECA	NSGA-II	6	Multi-objective
Kumari and Srinivas (2016)	MCA, ECA	Hyper-heuristics	12	Multi-objective
Jinhuang and Jing (2016)	MQ, MS	GA, HC, MAEA	17	Single-objective
Amarjeet and Chhabra (2016)	Fitness	HS, GA, DE, ABC, HC	8	Single-objective
Amarjeet and Chhabra (2017a)	IMCI, BMCI, MCI, MSI	NSGA-II	8	Multi-objective

ity. Amarjeet and Chhabra (2017c) formulated the software module clustering problem as a problem of improving the package structure within the existing clustering. To this contribution, they proposed different objective functions that help in improving the existing package structure of the software systems.

The works (Kumari and Srinivas 2016) presented the hyper-heuristic approach named as multi-objective hyper-heuristic evolutionary algorithm (MHypEA) to solve the multi-objective software module clustering problems. They evaluated the hyper-heuristic approach using MCA and ECA multi-objective formulation given by Praditwong et al. (2011). The clustering results obtained through MHypEA compared with clustering achieved through NSGA-II and Two-Archive algorithm (Praditwong and Yao 2006). The comparison results showed that the MHypEA performs better than the NSGA-II and Two-Archive algorithms. The works (Amarjeet and Chhabra 2017c) presented a multi-objective approach to address the software module clustering problem. They proposed a new objective functions based on the lexical and structural information. The results demonstrated that the combined use of lexical and structural information is able to generate better clustering solution compared to individual structural or lexical information.

8.3 Many-objective optimization approaches

The study (Praditwong et al. 2011) reported that the multi-objective formulation of the software module clustering problem and solving it using multi-objective evolutionary algorithms has been found more effective compared to the single-objective formulation of software module clustering problem. This is mainly due to the balanced exploration and exploitation ability of multi-objective evolutionary algorithms. Despite the success of multi-objective evolutionary algorithms, it does not perform well in problems that belong to the many-objective optimization field (Mkaouer et al. 2015).

In the literature of multi-objective software module clustering, many studies use the multi-objective evolutionary algorithms to solve the software clustering problems (e.g., Praditwong et al. 2011; Barros 2012; Kumari et al. 2013). The main reasons of performance degradation of multi-objective evolutionary algorithms with many-objective optimization are as follows: (1) As the number of objective functions in optimization problem increases by more than three, it faces difficulties in differentiating the solutions of population, (2) the execution time taken by non-dominated sorting process of multi-objective evolutionary algorithms increases, resulting in increase in overall time complexity of algorithm (Hughes 2008).

To address the challenges of the multi-objective evolutionary algorithms, many approaches have been proposed in the literature of multi-objective evolutionary algorithms. Interested researchers/academicians can find more details in the literature survey presented by works (Wang et al. 2015). They classified the many-objective optimization approaches into the following groups: (1) objective reduction approaches

Table 13 Summary of relevant many-objective approaches applied in SBSE

Reference	Algorithm	Category	Area	Number of objectives
Praditwong et al. (2011)	Two-Archive GA	Reference	Design phase	5
Sayyad et al. (2013a, b)	IBEA	Indicator	Requirement analysis phase	5
Sayyad et al. (2013a, b)	IBEA	Indicator	Requirement analysis phase	5
Yao (2013)	Two-Archive GA	Reference	Design phase	5
Kalboussi et al. (2013)	P-MOET	Preference	Testing phase	7
Mkaouer et al. (2014)	NSGA-III	Preference	Maintenance phase	15
Ramirez et al. (2014)	SPEA2, NSGA-II, ε-MOEA, MOEA/D, GrEA	Decomposition	Design phase	6
Olaechea et al. (2014)	GIA, IBEA	Indicator	Requirement phase	7
Mkaouer et al. (2015)	NSGA-III	Reference	Maintenance phase	8
Mkaouer et al. (2016)	NSGA-III	Reference	Maintenance phase	8

(e.g., Cinneide et al. 2012; Gong et al. 2013), (2) preference-based approaches (e.g., Said and Bechikh 2013) , (3) reference set-based approaches (e.g., Deb and Jain 2014), (4) indicator-based approaches (e.g., Zitzler and Künzli 2004; Bader 2011), (5) aggregation-based approaches (e.g., Garza-Fabre et al. 2009), (6) relaxed dominance-based approaches (e.g., GarzaspsFabre et al. 2010), (7) decomposition-based approaches (e.g., Zhang and Li 2007), and (8) external archive-based approaches (e.g., Praditwong and Yao 2006; Wang et al. 2015). Table 13 provides the summary of the many-objective methodologies of the software engineering field.

9 Conclusions and future works

This work investigated the incorporation of indicator-based ranking and two external archive techniques into artificial bee colony algorithm in order to solve the software module clustering problems with many objectives (i.e., more than three objective functions). The proposed ABC-based module clustering approach is named as many-objective artificial bee colony algorithm (MaABC). Two versions of many-objective software module clustering formulations (i.e., extended equal-size cluster approach (E-ECA) and extended maximizing cluster approach (E-MCA)) have been introduced. To verify the performance of proposed MaABC, it has been evaluated on seven software clustering problems obtained from different module dependency graphs (MDGs) of object-oriented software systems. The clustering results obtained from MaABC have also been compared with existing approaches (i.e., Two_Arch2, NSGA-II, MOEA/D, and IBEA) in terms of MQ, coupling, cohesion, and IGD. The achieved clustering results demonstrated that MaABC is able to generate better clustering compared to existing approaches. Future works include a detailed empirical study on different ranking strategies (e.g., θ-dominance, grid dominance, preference order ranking) for non-dominated solutions including min–max method to solve the many-objective software remodularization problem.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest

Human participants This study does not contain any studies with human participants or animals performed by any of the authors.

References

Aggarwal CC, Hinneburg A, Keim DA (2001) On the surprising behavior of distance metrics in high dimensional space. Springer, New York

Akay B, Karaboga D (2012) A modified artificial bee colony algorithm for real parameter optimization. Inf Sci 192:120–142

Amarjeet P, Chhabra JK (2014) An empirical study of the sensitivity of quality indicator for software module clustering. In: 2014 seventh international conference on contemporary computing (IC3), Noida, pp 206–211

Amarjeet P, Chhabra JK (2015) Improving package structure of object-oriented software using multi-objective optimization and weighted class connections. J King Saud Univ Comput Inf Sci. Available online 2 November 2015

Amarjeet P, Chhabra JK (2016) Harmony search based remodularization for object-oriented software systems. Comput Lang, Syst Struct 47:153–169

Amarjeet P, Chhabra JK (2017a) TA-ABC: two-archive artificial bee colony for multi-objective software module clustering problem. J Intell Syst. doi:10.1515/jisys-2016-0253

Amarjeet P, Chhabra JK (2017b) Improving modular structure of software system using structural and lexical dependency. Inf Softw Technol 82:96–120

Amarjeet P, Chhabra JK (2017c) Improving package structure of object-oriented software using multi-objective optimization and weighted class connections. J King Saud Univ-Comput Inf Sci 29(3):349–364

Arcuri A, Fraser G (2013) Parameter tuning or default values? An empirical investigation in search-based software engineering. Empir Softw Eng 18(3):594–623

Asafuddoula M, Ray T, Sarker R (2015) A decomposition-based evolutionary algorithm for many objective optimization. IEEE Trans Evolut Comput 19(3):445–460

Bader J, Zitzler E (2011) HypE: an algorithm for fast hypervolume-based many-objective optimization. Evolut Comput 1(19):45–76

Barros M (2012) An analysis of the effects of composite objectives in multi-objective software module clustering. In: Proceedings of the fourteenth international conference on genetic and evolutionary computation, pp 1205–1212

Bingdong L, Jinlong L, Tang K, Xin Y (2015) Many-objective evolutionary algorithms: a survey. ACM Comput Surv 48(1):1–37

Cai D, Yuping W, Miao Y (2014) A new evolutionary algorithm based on contraction method for many-objective optimization problems. Appl Math Comput 247:191–205

Cinnéide M, Tratt L, Harman M, Counsell S, Moghadam IH (2012) Experimental Assessment of Software Metrics Using Automated Refactoring. In: Proceedings of the ACM-IEEE international symposium on empirical software engineering and measurement. pp 49–58

Coello CA (1996) An empirical study of evolutionary techniques for multiobjective optimization in engineering design. PhD thesis, Department of Computer Science, Tulane University, New Orleans, LA

Coello CA, Christiansen AD (1998) Two new GA-based methods for multiobjective optimization. Civil Eng Syst 15(3):207–243

Corne D, Jerram N, Knowles J, Oates M (2001) PESA-II: region-based selection in evolutionary multiobjective optimization. In: Proceedings of the 3rd annual conference on genetic evolutionary computation, pp 283–290

Črepinšek M, Liu SH, Mernik M (2014) Replication and comparison of computational experiments in applied evolutionary computing: common pitfalls and guidelines to avoid them. Appl Soft Comput 19:161–170

Dahiya SS, Chhabra JK, Kumar S (2010) application of artificial bee colony algorithm to software testing. In: 2010 21st Australian software engineering conference, Auckland, pp 149–154

Deb K, Jain H (2014) An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints. IEEE Trans Evolut Comput 18(4):577–601

Deb K, Agrawal S, Pratap A, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197

Doval D, Mitchell BS, Mancoridis S (1999), Automatic clustering of software systems using a genetic algorithm. In: Proceedings of IEEE conference on software technology and engineering practice (1999), pp 73–81

Garza-Fabre N, Pulido GT, Coello CAC (2009) Ranking methods for many-objective optimization. In: Advances in artificial intelligence MICAI 2009, pp 633–645

Garza-Fabre N, Pulido GT, Coello CAC (2010) Alternative fitness assignment methods for manyobjective optimization problems. In: Artificial evolution, pp 146–157

Gong D, Sun J, Ji X (2013) Evolutionary algorithms with preference polyhedron for interval multiobjective optimization problems. Inf Sci 233:141–161

Hadka D, Reed P (2013) Borg: an auto-adaptive many-objective evolutionary computing framework. Evolut Comput 21(2):231–259

Harman M, Hierons R, Proctor M (2002) A new representation and crossover operator for search-based optimization of software modularization. In: Proceedings of the genetic and evolutionary computation conference, pp 1351–1358

Harman M, Jones BF (2001) Search-based software engineering. Inf Softw Technol 43(14):833–839

Harman M, Swift S, Mahdavi K (2005) An empirical study of the robustness of two module clustering fitness functions. In: Proceedings of the 7th annual conference on Genetic and evolutionary computation

Hashim A, Hashim BO, Ayinde MA, Abido M (2016) Optimal placement of relay nodes in wireless sensor network using artificial bee colony algorithm. J Netw Comput Appl 64:239–248

Hughes EJ (2008) Fitness assignment methods for many-objective problems. In: Multi-objective problem solving from nature. From concepts to application, pp 307–329

Jadhav HT, Bamane PD (2016) Temperature dependent optimal power flow using g-best guided artificial bee colony algorithm. Int J Electr Power Energ Syst 77:77–90

Jinhuang H, Jing L (2016) A similarity-based modularization quality measure for software module clustering problems. Inf Sci 342(10):96–110

Kalboussi S, Bechikh S, Kessentini M, Said LB (2013), Preference-based many-objective evolutionary testing generates harder test cases for autonomous agents. In: Proceedings of the 5th international symposium on search-based software engineering, pp 245–250

Karaboga D (2005) An idea based on honey bee swarm for numerical optimization. Technical Report-TR06, Erciyes University, Engineering Faculty, Computer Engineering Department

Karaboga D, Akay B (2009) A comparative study of artificial bee colony algorithm. Appl Math Comput 214(1):108–132

Karaboga D, Basturk B (2007) A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J Glob Optim 39(3):459–471

Karaboga D, Basturk B (2008) On the performance of artificial bee colony (ABC) algorithm. Appl Soft Comput 8(1):687–697

Khare V, Yao X, Deb K (2003) Performance scaling of multi-objective evolutionary algorithms. In: Evolutionary multi-criterion optimization. Lecture notes in computer science. vol 2632, pp 376–390

Kumari AC, Srinivas K (2016) Hyper-heuristic approach for multi-objective software module clustering. J Syst Softw 117:384–401

Kumari AC, Srinivas K, Gupta MP (2013) Software module clustering using a hyper-heuristic based multi-objective genetic algorithm. In: 2013 IEEE 3rd international advance computing conference (IACC), Ghaziabad, pp 813–818

Laumanns M, Thiele L, Deb K, Zitzler E (2002) Combining convergence and diversity in evolutionary multi-objective optimization. Evolut Comput 10(3):263–282

Li X, Yin M (2012) Hybrid differential evolution with artificial bee colony and its application for design of a reconfigurable antenna array with discrete phase shifters. IET Microw Antenna Propag 6:1573–1582

Mahdavi K, Harman M, Hierons RM (2003) A multiple hill climbing approach to software module clustering. In: Proceedings of the international conference on software maintenance, pp 315–324

Mancoridis S, Mitchell BS, Chen YF, Gansner ER (1999) Bunch: a clustering tool for the recovery and maintenance of software system structures. In: Proceedings of the IEEE international conference software maintenance, 1999, pp 50–59

Mancoridis S, Mitchell BS, Chen YF, Rorres C, Gansner ER (1998) Using automatic clustering to produce high-level system organizations of source code. In: Proceedings of the international workshop program comprehension, pp 45–53

Mitchell BS, Mancoridis S (2002) Using heuristic search techniques to extract design abstractions from source code. In: Proceedings of the genetic and evolutionary computation conference, pp 1375–1382

Mitchell B.S, Mancoridis S, Traverso M (2004) Using interconnection style rules to infer software architecture relations. In: Proceedings

of the conference on genetic and evolutionary computation, pp 1375-1387

Mkaouer MW, Kessentini M, Bechikh S (2016) On the use of many quality attributes for software refactoring: a many-objective search-based software engineering approach. Empir Softw Eng 21(6):2503–2545

Mkaouer MW, Kessentini M, Bechikh S, Deb K, Cinnéide MO (2014) High dimensional search-based software engineering: finding tradeoffs among 15 objectives for automated software refactoring using NSGA-III. In: Proceedings of the genetic and evolutionary computation conference, pp 1263–1270

Mkaouer MW, Kessentini M, Shaout A, Koligheu P, Bechikh S, Deb K, Ouni A (2015) Many objective software remodularization using NSGA-III. ACM Trans Softw Eng Methodol 24(3):1–17

Olaechea R, Rayside D, Guo J, Czarnecki K (2014) Comparison of exact and approximate multi-objective optimization for software product lines. In: Proceedings of the 18th international software product line conference, pp 92–101

Parashar A, Chhabra JK (2016) An approach for clustering class coupling metrics to mine object oriented software components. Int Arab J Inf Technol (IAJIT) 13(3):239–248

Pierro FD, Khu S-T, Savi'c DA (2007) An investigation on preference order ranking scheme for multi-objective evolutionary optimization. IEEE Trans Evolut Comput 11(1):17–45

Pierro FD, Khu ST, Savic DA (2007) An investigation on preference order ranking scheme for multiobjective evolutionary optimization. IEEE Trans Evolut Comput 11(1):17–45

Plevris P, Papadrakakis M (2011) A hybrid particle swarm-gradient algorithm for global structural optimization. Comput Aided Civil Infrastruct Eng 26:48–68

Praditwong K (2011) Solving software module clustering problem by evolutionary algorithms. In: Eighth international joint conference on computer science and software engineering, pp 154–159

Praditwong K, Harman M, Yao X (2011) Software module clustering as a multi-objective search problem. IEEE Trans Softw Eng 37(2):264–282

Praditwong K, Yao X (2006) A new multi-objective evolutionary optimization algorithm: the two-archive algorithm. In: Cheung Y-M, Wang Y, Liu H (eds) Proceedings of the international conference computational intelligence and security, vol 1, pp 286–291

Praditwong K, Yao X (2007) How well do multi-objective evolutionary algorithms scale to large problems. In: Proceedings IEEE congress evolutionary computing (CEC), Singapore, pp 3959–3966

Ramirez A, Romero JR, Ventura S (2014) On the performance of multiple objective evolutionary algorithms for software architecture discovery. In: Proceedings of the 2014 conference on genetic and evolutionary computation, pp 1287–1294

Said LB, Bechikh S, Ghédira K (2013) The r-dominance: a new dominance relation for interactive evolutionary multicriteria decision making. Proc IEEE Trans Evolut Comput 14(5):801–818

Sayyad AS, Ingram J, Menzies T, Ammar H (2013) Scalable product line configuration: a straw to break the camel's back. In: IEEE/ACM 28th international conference on automated software engineering, pp 465–474

Sayyad AS, Menzies T, Ammar H (2013) On the value of user preferences in search-based software engineering: a case study in software product lines, pp 492–501

Wang G, Jiang H (2007) Fuzzy-dominance and its application in evolutionary many objective optimization. In: Proceedings of the international conference on computational intelligence and security workshops, pp 195–198

Wang H, Jiao L, Yao X (2015) Two_Arch2: an improved two-archive algorithm for many-objective optimization. IEEE Trans Evolut Comput 19(4):524–541

Xianneng L, Guangfei Y (2016) Artificial bee colony algorithm with memory. Appl Soft Comput 41:362–372

Yao X (2013) Some recent work on multi-objective approaches to search-based software engineering. In: Proceeding of the 5th symposium on search based software engineering, pp 4–15

Yang S, Li M, Liu X, Zheng J (2013) A grid-based evolutionary algorithm for many-objective optimization. IEEE Trans Evolut Comput 17(5):721–736

Yuan Y, Xu H, Wang B, Yao X (2015) A new dominance relation based evolutionary algorithm for many-objective optimization. IEEE Trans Evolut Comput 20(1):16–37

Zhang Q, Li H (2007) MOEA/D: a multiobjective evolutionary algorithm based on decomposition. Proc IEEE Trans Evolut Comput 11(6):712–731

Zitzler E, Künzli S (2004) Indicator-based selection in multi-objective search. Parallel problem solving from nature–PPSN VIII. Springer, Berlin, pp 832–842

Zou X, Chen Y, Liu M, Kang L (2008) A new evolutionary algorithm for solving many-objective optimization problems. IEEE Trans Syst Man Cybern-Part B 38(5):1402–1412

Zitzler E, Laumanns M, Thiele L (2002) SPEA2: improving the strength Pareto evolutionary algorithm. In: Proceedings of the evolutionary methods design optimization control application, pp 95–100