Front. Comput. Sci., 2023, 17(3): 173209
https://doi.org/10.1007/s11704-022-1449-2

RESEARCH ARTICLE

Multi-dimensional information-driven many-objective software remodularization approach

Amarjeet PRAJAPATI (✉)[1], Anshu PARASHAR[2], Amit RATHEE[3]

1 Department of Computer Science Engineering & Information Technology, Jaypee Istitute of Information Technology, Noida 201307, India
2 Department of Computer Science & Engineering, Thapar Institute of Engineering & Technology, Punjab 147004, India
3 Department of Computer Science, Government College Barota, Haryana 131301, India

Abstract Most of the search-based software remodularization (SBSR) approaches designed to address the software remodularization problem (SRP) areutilizing only structural information-based coupling and cohesion quality criteria. However, in practice apart from these quality criteria, there require other aspects of coupling and cohesion quality criteria such as lexical and changed-history in designing the modules of the software systems. Therefore, consideration of limited aspects of software information in the SBSR may generate a sub-optimal modularization solution. Additionally, such modularization can be good from the quality metrics perspective but may not be acceptable to the developers. To produce a remodularization solution acceptable from both quality metrics and developers' perspectives, this paper exploited more dimensions of software information to define the quality criteria as modularization objectives. Further, these objectives are simultaneously optimized using a tailored many-objective artificial bee colony (MaABC) to produce a remodularization solution. To assess the effectiveness of the proposed approach, we applied it over five software projects. The obtained remodularization solutions are evaluated with the software quality metrics and developers view of remodularization. Results demonstrate that the proposed software remodularization is an effective approach for generating good quality modularization solutions.

Keywords software restructuring, remodularization, multi-objective optimization, software coupling and cohesion

1 Introduction

Frequentchanges made in the source code during maintenance often degrade the design of the software system [1]. To improve the design of a degraded software system, remodularization of source code is generally carried out [2]. In the previous two decades, many remodularization approaches based on deterministic automated software clustering and search-based automated software clustering have been proposed (e.g., [2−7]). It has been found that for a large and complex software system, the search-based automated software clustering approaches are more effective for modularizing the source code compared to the deterministic automated software clustering approaches [1−7].The concept of transforming the software remodularization problem (SRP) as a search-based single/multi/many-objective optimization problem makes it more promising and opens ample opportunity for the application of metaheuristic optimization algorithms [1]. Such a technique of addressing the SRPs is generally regarded as search-based software remodularization (SBSR). The SBSR performance depends on the suitability of the metaheuristic optimization algorithm and the effective design of the objective functions.

Most of the existing metaheuristic optimization algorithms designed to address the SRP have exploited the existing framework of the canonical metaheuristic optimization algorithms (e.g., [7−11]). Overall, many SBSR approaches have successfully exploited the framework of the existing canonical metaheuristic optimization algorithm and provided a customized version of the SBSR algorithm. Similar to the considerable progress in the development of the customized or tailored version of the SBSR algorithm, tremendous growth in designing various objective functions reflecting the different aspects of software quality has also been observed [2−7]. The modularization quality (MQ) [2] is a widely used software quality metric for objective/fitness function in different single-objective SBSR approaches [5,8]. The other multi-objective SBSR approaches [6−7] use the refined version of modularization quality (MQ) [6]. The definition of MQ metrics uses only structural information;therefore, it only represents the structural aspect of software quality. To improve the meaningfulness of the generated remodularization solutions, the approaches [5,9,12−13] use the lexical aspects of software quality along with the structural aspects of software quality. Recently, some studies [1,9,14] exploited the conceptual aspects of software quality along with the structural and lexical aspects of software quality.

Received August 16, 2021; accepted May 24, 2022

E-mail: amarjeetnitkkr@gmail.com

Even the existing SBSR approaches perform well in addressing a particular aspect of software remodularization; still, many challenges remained untouched. For example, most of the software remodularization approaches either consider limited dimensions of artefacts or give equal importance to each dimension of information in defining the objective functions. However, it is commonly observed that the different dimensions of structural, lexical, and changed history information generally contribute differently in the software remodularization process instead they contribute equally. Apart from defining the objective functions, some issues remain untouched in designing the search-based metaheuristic optimization algorithm. Most of the SBSR approaches use the traditional multi-objective optimization algorithms in solving the remodularization problems having a large number (more than three) of objective functions. The traditional multi-objective optimization algorithms work well with optimization problems having a small number (less than three) of objective functions and do not work well with a large number of objective functions.

To address the aforementioned issues concerning the definition of the objective functions and designing the metaheuristic optimization algorithm for the SRP, we introduce an improved definition of objective functions and many-objective metaheuristic optimization algorithm. In particular, in the existing definition of software package coupling and cohesion, we incorporate the different dimensions of structural, lexical, and changed-history information and introduced different categories of coupling and cohesion metrics. In each type (i.e., structural, lexical, and changed-history) of source code information used for defining the coupling and cohesion metrics, we also consider different dimensions of structural information, different dimensions of lexical information, and different dimensions of changed-history information with their relative importance. Apart from different types of object functions defined in terms of structural, lexical, and changed-history information, we also consider the other supportive objectives, i.e., the number of clusters, and the difference between the minimum and the maximum number of modules in a cluster. The major contributions of this paper are summarized as follows:

- A variety of class coupling methods based on the different dimensions and combinations of structural, lexical, and changed-history information is introduced and further, these class couplings are used to define the different types of software coupling and cohesion for the remodularization of software systems.
- In the computation of class coupling, the different dimensions of structural, lexical, and changed-history information are given different weightage according to their importance in the coupling.
- To optimize the different objectives to produce the software remodularization solutions, an existing many-objective artificial bee colony (MaABC) has also been tailored by incorporating various strategies.
- The proposed approach's supremacy is validated by applying it over the different instances of many-objective SRPs. Especially, five object-oriented software projects having varying chrematistics are considered as the test problems

The remaining part of the paper is organized as follows. Section 2 presents related works based on the structural, lexical, and changed history remodularization. Section 3 provides a detailed description of proposed software remodularization methodology. Section 4 explains the experimentation configuration designed for the experimentation. Section 5 presents a discussion of the results. Section 6 covers the various types of threats that can affect the validation of the results and their mitigation strategies. Section 7 concludes the work with the suggestion of future directions.

2 Related works

In the literature on search-based software engineering, a variety of SBSR approaches addressing the different aspects of software design improvements have been proposed (e.g., [1−11]). These SBSR approaches exploit the framework of the existing metaheuristic optimization algorithms and tailor them according to the suitability of the SRPs. Based on the number of objectives involved in the SRP, the SBSR approaches can be divided into single-objective SBSR approaches, multi-objective SBSR approaches, or many-objective SBSR approaches. Further, based on the type of information used in designing the objective functions these SBSR approaches can be further divided into 1) structural information-based SBSR approaches (S-SBSR approaches), 2) structural + lexical information-based SBSR approaches (SL-SBSR approaches), 3) structural + lexical + changed history information-based SBSR approaches (SLC-SBSR approaches).

2.1 S-SBSR approaches

The designing of the objective functions in many of the SBSR approaches is based on the source code's structural information. Over the last two decades, many SBSR approaches based on structural information have been proposed to address the different aspects of SRPs. The availability of various tools for extracting the different dimensions of structural information from the source code made the structural information based SBSR approaches more popular in the research community.

In the structural information-based SBSR, the source code implementation's structural information is used to define the different modularization criteria as objective functions. The different programming paradigms use different construction to realize the different implementation requirements. Therefore, the structural information can vary according to the underlying programming language used for system implementation. For example, in Java-based object-oriented software, the classes can be considered as modules and method calls and inheritance relationships can be used to compute the class coupling. On the other hand, in a procedural programming language such as C programming, the source file can be considered as modules and function calls can be used to compute the module coupling. The structural information-based SBSR approaches based on the number of quality criteria used as the objective function can be divided into 1) single-objective structural information-based SBSR,

2) multi/many-objective structural information-based SBSR.

In the literature on SBSR, many approaches have used structural information to define the modularity criteria as objective/fitness functions for the formulation of SRP as single or multi-objective SRP. In the context of single-objective SRP, the structural information is used to define a single modularization quality criterion as a single objective function. Mancoridis et al. (1998) [2] in the direction of single-objective structural information-based SBSR is the first approach, where the structural source code information is used to define the single-objective function and optimized using the genetic-based metaheuristic optimization algorithm. In particular, the approach uses the structural information and defined modularization quality (MQ) metrics to assess the software modularity quality and further it is used as an objective function in the genetic algorithm-based software remodularization framework. Moreover, this approach's input is in the form of a module dependency graph (MDG) and the MQ is defined in terms of inter-connectivity and intra-connectivity of the graph partition. A similar definition of MQ has also been used in [8,15] to quantify the quality of software partitioning for software remodularization. Doval et al. (1999) [15] treated the software partitioning problem as an optimization problem and used the Genetic Algorithm (GA) to find a good modularization solution from the search space based on the MQ as fitness. Mahdavi et al. (2003) [8] transformed the software system into the weighted and un-weighted MDG and used the multiple hill-climbing optimizers to find a good partitioning solution based on the MQ as the fitness function.

Abdeen et al. (2011) [7] also treated the SRP as a restructuring of the existing package structure of an object-oriented software system. Their approach optimizes software modularization by minimizing the direct package cyclic dependencies. For this, a fitness function based on package coupling and cohesion is used to guide the Simulated Annealing (SA) driven optimization process of the software remodularization. To compute the package coupling and package cyclic dependencies different types of class relationships such as method calls and class inheritance have been used. Prajapati and Chhabra (2017) [9] uses the different dimensions of structural information to quantify the modularization quality of object-oriented software systems. Further, they used the modularization quality as a fitness function to optimize the system's modular. To explore and exploit the search space to find a good modularization solution, their optimization approach uses a Harmony Search (HS) based metaheuristic optimization algorithm.

Apart from the single-objective structural information-based SBSR approaches, there also has been tremendous growth in the direction of multi-objective structural information-based SBSR approaches In the multi-objective formulation of SRP, the different types of structural information are used to define the different aspects of software quality metrics and they are optimized simultaneously using the multi-objective metaheuristic optimization algorithm for the software remodularization. In the direction of multi-objective structural information-based SBSR, the approach reported by Praditwong et al. (2011) [6] is regarded as the base work. This work defines two sets of remodularization criteria as objective functions. It uses the evolutionary-based multi-objective metaheuristic optimizer to optimize each set of objectives simultaneously to improve the software modular structure.

To evaluate the effectiveness of composite objectives [6] with inclusion and exclusion of some supportive objectives in the multi-objective formulation of software module clustering problem, Barros (2012) [16] presented an empirical analysis. Their study concluded that the exclusion of some objective (i.e., supportive objective) from the composite objective can also help in generating effective and efficient results. To improve the metaheuristic optimization algorithm's effectiveness and efficiency for the multi-objective software module clustering problem, Kumari and Srinivas (2016) [17] presented a hyper-heuristic based evolutionary algorithm. The approach generated a better module clustering solution compared to the existing GA-based approaches under the same set of composite objectives [6]. In the direction of improving the metaheuristic optimization algorithm for the multi-objective SRP, Prajapati and Geem (2020) [18] proposed a Harmony Search (HS) based software remodularization approach for software architecture reconstruction activity.

2.2 SL-SBSR approaches

The different dimensions of the structural relationship existing among the software entities are the most widely used source code information in software remodularization. Many software remodularization approaches have used structural relationships in defining the software modularity criteria as fitness/objective functions evaluating the modularization solution. However, other source code information such as linguistic information embedded in the form of comments and identifiers has not gained more attention in the search-based software engineering community. Some researchers have demonstrated the usefulness of lexical information of the source in restructuring the software systems for different purposes.

In some work, the researchers have combined the lexical information with the structural information to guide the remodularization process. In some work, researchers have used the lexical information separately. In the combined structural and lexical information based SBSR approaches, the single or various dimensions of structural and lexical information are together used to define the objective functions. Further, they are optimized using single or multi-objective metaheuristic optimization algorithms. The structural and lexical information based SBSR approaches has been found more effective in generating remodularization solutions as intended by the software developers compared to the separate structural information-based SBSR.

To evaluate the effectiveness of combined structural and lexical similarity in software remodularization, Anquetil and Lethbridge (1999) [5], uses the various features corresponding to the different types of structural and lexical information. Especially, they used the formal and non-formal features of the source code to quantify the coupling of entities and uses the Bunch framework to restructure the software systems. To

generate the meaningful decomposition of the software packages, the approach presented in [13] uses the semantic and structural relationships existing between the packages' classes. They combined the structural coupling measures, i.e., information-flow-based coupling (ICP), and semantic coupling measure, i.e., the conceptual coupling between classes (CCBC) with their relative weights into a single coupling measure. Even if the approach does not use the search-based software engineering concepts, the coupling measures can be easily used as fitness functions.

To improve the accuracy of meaningfulness of remodularization solution of the multi-objective software remodularization approach, Prajapati and Chhabra (2017) [9] suggested the use of the different aspects of structural and lexical class dependency information to compute the package coupling and cohesion for the object-oriented software systems. They used the NSGA-II based multi-objective evolutionary algorithm as the optimization technique for software remodularization. In case of unavailability of tool to extract the structural information, Kargar et al. (2017) [19] suggested the use of lexical information embedded in the source code of the software system. The authors proposed the semantic dependency graph as an alternative for the call dependency graph (CDG) based on the lexical information. To partition the SDG to generate a clustering solution, they used the hill-climbing algorithm as a search optimizer. Their results demonstrate that SDG can be a good alternative if the software systems are developed in different programming languages and the extraction of CDG is not feasible.

2.3 SLC-SBSR approaches

It is commonly observed that the software developers generally modularize the different software elements based on the various types of information not only the single aspects of design information. Keeping these facts in mind, various software remodularization approaches have been proposed by utilizing different types of artefacts. Even the structural and lexical-based software remodularization approaches performing well to produce a good quality software remodularization solution, incorporating some other artefacts such as changed-history, can make the modularization solution more effective.

In the literature on SBSR, only a few papers have considered the combined structural, lexical, and changed history information in the formulation of SBSR problem. Mkaouer et al. (2015) [1] presented the first study, where the structural, lexical, and changed history information is exploited to define the different types of modularity metrics. They used the concept of many-objective optimization to optimization all conflicting modularization quality criteria. Later some other researchers and academicians have also provided some contributions in this direction.

Rathee and Chhabra (2018) [20] utilized the structural, lexical, and changed history information with their relative importance in defining the modularization criteria. Additionally, they explored the multiple dimensions of structural and lexical information that can improve modularization criteria' effectiveness. Recently, Prajapati et al. (2020) [14] also exploited the structural, lexical, and changed history information to compute the class coupling strength for the computation of object-oriented software package coupling and cohesion.

3 Proposed approach

In the proposed SBSR, the major contributions are divided into two parts. The first part focuses on designing the remodularization objective model and the second part concentrates on the developmentof the customized many-objective optimization algorithm.

3.1 SRP formulation

Software remodularization is the problem of optimizing the software entities' distributions into existing modules to improve the quality of software systems. The definition of software entities and modules can vary according to the programming paradigms used for the implementation and abstractions. In this work, we are mainly focusing on the object-oriented software systems where classes are assumed as software entities and modules as packages. A well-distribution of source code classes into packages have to satisfy many quality criteria (often conflicting). The number and definition of quality criteria used in remodularization process vary according to the purpose and quality requirements. The quality requirements for the large and complex systems are getting very large. Moreover, well-modularized software systems have to satisfy various dimensions of software quality. To make the remodularization solution more useful, we consider the following software quality for the remodularization based on structural, lexical, and changed history information.

Structural software package coupling and cohesion The structural package coupling and structural software package cohesion of an object-oriented software project measure the degree of inter-relatedness of packages and the degree of intra-relatedness of packages corresponding to the structural relationships, respectively. To compute the structural package coupling and cohesion, using the structural coupling of the classes is a common and effective approach. To capture the structural coupling between the classes, many coupling metrics have been proposed in object-oriented software engineering literature. In this direction, the study conducted by Li and Henry [21] has formulated various types of structural information-based class coupling metrics (e.g., Data abstraction coupling (DAC) and message passing coupling (MPC)) to compute the class coupling.

The response for a class (RFC) and coupling between the object (CBO) introduced by Chidamber and Kemerer [22] are the other two most widely used class coupling metrics to compute the structural class coupling. Martin [23] also introduced Afferent Coupling (Ca) and Efferent Coupling (Ce) metrics based on the structural class dependency information to compute the class coupling. Briand et al. [24] designed several structural information-based coupling metrics to compute the coupling between the classes. Further, Briand et al. [25] build a unified framework based on the coupling metrics developed in the literature [26−27] to computation class coupling of object-oriented software systems. To exploit

the importance of polymorphism in coupling computation, Lee et al. [28] introduced an information flow-based coupling (ICP) metric.

The aforementioned coupling metrics can be useful to quantify and evaluate the quality of object-oriented software systems. But they cannot be effective if used as a fitness/ objective function to guide the remodularization process in search-based software modularization. This software coupling metrics uses only limited types of structural class information to compute the coupling metrics. However, the remodularization process requires class coupling metrics as fitness/objective function that consists of various dimensions of class coupling information. Because the software developers generally use various dimensions of the class coupling information according to their relative importance in modularizing the software modules in the software components. Therefore, in this study, we also exploit various structural information dimensions to compute the class coupling for the use of the software package coupling and cohesion computation. The definitions of different types of structural class dependencies are derived from these studies [9,29]. These are the 1) extends relationship (EX), 2) implementsrelationship (IM), 3) is-of-type relationship (IT), 4) reference (RE), 5) method callsrelationship (CA), 6) has parameterrelationship (HP), 7) returnsrelationship (RT), and 8) throws relationship (TH).

The structural software package coupling and cohesion are the two most important remodularization quality criteria which are highly required and widely used in SBSR as objective functions. The definition of these two quality criteria can vary according to the considered structural information. The package coupling and cohesion are defined as follows.

$$SPcou(M) = \sum_{i=1}^{n} \sum_{j \notin T_i} \varnothing\left(c_i, c_j\right), \tag{1}$$

$$SPcoh(M) = \sum_{i=1}^{n} \sum_{j \in T_i} \varnothing\left(c_i, c_j\right), \tag{2}$$

where M is the particular modularization containing n number of classes. The T_i is the set of classes that are in the same package as of class c_i. The $\varnothing\left(c_i, c_j\right)$ computes the connection strength between class c_i and c_j. The value of $\varnothing\left(c_i, c_j\right)$ is computed as follows.

$$\varnothing\left(c_i, c_j\right) = \sum\nolimits_{r \in R} w_r \times n_r\left(c_i, c_j\right). \tag{3}$$

R is the set of different types of structural relationships i.e., R={EX, IM, IT, RE, CA, HP, RT, TH}, w_r is the relative weight of a particular relationship $r \in R$, and n_r is the frequency of r-type relationship.

To compute the relative weights of the relationships is an essential and challenging task because it has a significant impact on class coupling value and quantification of remodularization quality. In this study, we use the relative weight computation approach, as presented in [9].

Lexical software package coupling and cohesion Once the underlying domain vocabulary at package level is identified and vector representation is used for efficiently processing recognized domain vocabulary. The next step is to compute concept relatedness using vector representation for identified domain vocabulary. This paper considers determining relatedness using a freely available software package called WordNet. WordNet is a widely acknowledged approach to determining the semantic similarity between two words. It is an extensive lexical database (containing semantic/ conceptual relations among different words) of English that organizes different verbs, adverbs, nouns, and adjectives into different hierarchical structures called synsets based on underlying semantic relations such as synonymy, autonomy, and hyponymy. The degree of semantic/ lexical relatedness among two words/ concepts says c_1 and c_2 are measured using the concept of path length between concepts (synsets) represented by c_1 and c_2, respectively. The path length based lexical relatedness in this paper is measured using the metric as proposed by Wu et al. [30]. The following expression gives the mathematical formulation for measuring this conceptual/ lexical similarity between two concepts c_1 and c_2:

$$Sim(c_1, c_2) = \frac{2 \times \mathrm{dep}(c)}{len(c_1, c) + len(c_2, c) + 2 \times \mathrm{dep}(c)}, \tag{4}$$

Here, c in the considered mathematical formulation represents the lowest common subsumer (LCS) between any two different considered concepts c_1 and c_2. LCS is commonly defined as the most common ancestor/ parent of both c_1 and c_2. Moreover, $len(c_1, c)$ and $len(c_2, c)$ represents the total number of edges that exist in the path described by two nodes.

The source code of a software system is a rich source of acquiring domain-specific vocabulary [20]. The authors of this chapter think that this domain-specific vocabulary can be easily built by tokenizing six main parts of the underlying source code of a class. These essential parts include comments sections introduced by developers for elaborating the particular section, identifiers used for naming classes, member variables declaration section, signatures used to describe different methods, identifiers used as parameter names, and body section of methods. Further, the authors in this paper believe that the semantic information represented by these considered six parts can is of different relevance. Therefore, tokens extracted from each part are considered different and a unique weight is assigned to each part while combining overall lexical strength. The weights assigned to different parts are estimated by formulating a probabilistic lexical model and utilizing the Expectation-Maximization (EM) algorithm as used in [31].

Based on the concept of building an efficient domain vocabulary model (by extracting tokens from six different parts and assigning unique weights to them) and determining lexical similarity between any two concepts using the proposed Eq. (4). The next step is to measure the overall values for considered two quality parameters viz cohesion, and coupling at the package level, say P. The mathematical formulation used for measuring these parameters is as following:

$$LexStrength_k(N,T) = \frac{\sum_{i=1}^{N}\sum_{j=1;j\neq i}^{T} Sem(c_i,c_j)}{N*(N-1)}, \quad (5)$$

$$SemCoh(P) = \text{Average}\left(\sum_{k=1}^{6} w_k * LexStrength_k(N,N)\right), \quad (6)$$

$$SemCup(P) = \text{Average}\left(\sum_{k=1}^{6} w_k * LexStrength_k(N,T)\right). \quad (7)$$

Here, N is the total number of lexemes present in the DV of package P extracted from the kth part and T is the total number of lexemes present in the DV of the rest of the subsystem (the obtained modular structure except the package P being considered) extracted from the same kth part.

Changed-history software package coupling and cohesion During the life cycle of the software, system changes are made to the software components. The developer made changes and record them as change commits in the version history or repository [3,4]. So, the frequent co-change pattern among the components like classes can be considered as change-coupling among them. In literature, most of the coupling and cohesion measurement is computed statically by analysing the source code [12,22,32]. The majority of the source code metrics include LOC, coupling, cohesion, function points, inheritance [12,32−34]. Static analysis of the source code sometimes shows stagnation as it does not entirely reflect software evolution. In the present development scenario, software evolution is recorded using version or configuration management systems as software repositories like Github and Sourceforge [3,4]. Such software repositories consist of valuable information about the evolution of the software applications in terms of their change history. Change history lists the changes that are made in the past and collects the software changelogs.

Apart from the structural and semantic analysis, change commits should be analyzed to compute the dependency metrics that can contribute to the re-modularization, restructuring, and clustering of the legacy or existing software system modules. Various studies have investigated evolutionary data for computing cohesion and coupling [32−34], co-change pattern [35], change impact analysis etc. [3,4,36−38] exploited the change history. It computed different metrics to further restructure the software systems by applying various classification and clustering techniques. Additionally, Amarjeet et al. [14] also used structural and change coupling information to re-modularise the software system. While traditional dependency measures are widely utilized to measure the quality of the software system. But in the present development scenario, it is necessary to analyze the change repositories to predict the various types of dependencies to compute coupling and cohesion and structural and semantic measures. Through this, a developer can have a view of the historical co-change pattern of software entities. The literature has observed that none of the studies has fully utilized the structural, semantic, and change history for dependency measurements to apply search-based techniques further. Through this, the maintainer can implement future changes and also restructure the system effectively.

Considering the importance of past co-change behaviour of software entities, change-coupling metrics have been computed based on software change history. We have explored the classes that are changed together in the past and may have some proximity. If several change-commits indicate co-change pattern of a few classes, then such co-change pattern should be considered as possible change-coupling among these classes. Here, we intend to measure change-coupling pattern at attribute, method and class levels. For this purpose, metrics namely Attribute-level change coupling, Method-level change coupling, Class-level change coupling are computed. These coupling measures are further used to calculate the overall coupling and cohesion among the packages as mentioned below:

$$ChC(c_i) = \alpha_1 * AChC(c_i) + \alpha_2 * MChC(c_i) + \alpha_3 * CChC(c_i), \quad (8)$$

$$ChCoH(c_i) = \beta_1 * AChC(c_i) + \beta_2 * MChC(c_i) + \beta_3 * CChC(c_i), \quad (9)$$

where $ChC(c_i)$- Change-coupling of class c_i, $AChC(c_i)$- Attribute level change-coupling of c_i, $MChC(c_i)$- Method level change coupling of class c_i, $CChC(c_i)$- Class level change-coupling of class c_i, and α_1, α_2, and α_3 are the coupling weights assigned to each type of the coupling. β_1, β_2, and β_3 are the cohesion weights assigned respectively. The demonstration of change coupling is given Fig. 1.

The two classes c_i and c_j may have attribute level change-coupling if some attributes of c_j and c_i are frequently changed together. Such information types need to be extracted from the changelogs or commits recorded in the version or change history of the software system. The definition of $AChC(c_i)$ is mentioned below.

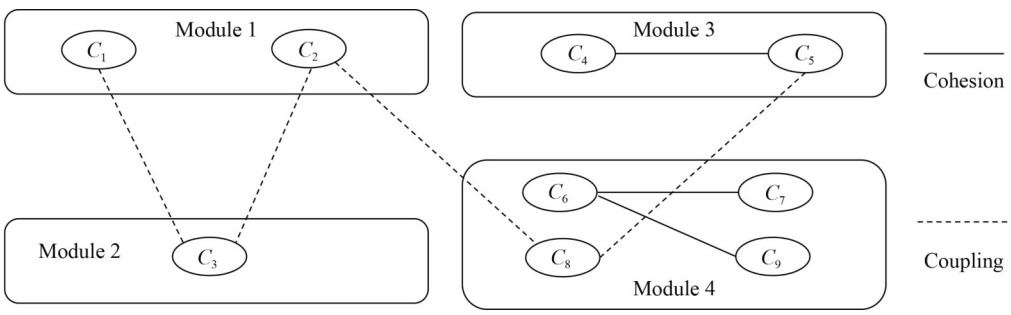

Fig. 1 Demonstration of change coupling

$$AChC(c_i) = \sum_{j=1, j\neq i}^{n} ACh_{prox}(c_i, c_j), \tag{10}$$

where, $AChC(c_i)$- Attribute level change coupling of class c_i, $ACh_{prox}c_i, c_j)$- Attribute level change coupling (proximity) between the classes c_i and c_j. And n is the total number of classes. The metrics to compute the Attribute level change coupling (proximity) i.e.,$ACh_{prox}c_i, c_j)$ between the class's c_i and c_j is described as under:

$$ACh_{prox}\left(c_i, c_j\right) = \sum_{k=1}^{NAc_i} \sum_{l=1}^{NAc_j} \frac{|c_i c_k \cap c_j c_l|}{|c_i a_k \cup c_j a_l|}, \tag{11}$$

where NAc_i and NAc_j are the number of attributes of the class's c_i and c_j, respectively. $c_i a_k$ is the list of attributes of class c_i (k=1 to NAc_i), $c_j a_l$ is the list of attributes of class c_j (k=1 to NAc_i). $|c_i a_k \cup c_j a_l|$ gives the number of changes commits in which the attributes of classes c_i and c_j that are co-changed together. $|c_i a_k \cup c_j a_l|$ gives the number of changes commits in which the attributes of classes c_i or c_j or both are changed.

The two classes c_i and c_j may have method level change coupling is few methods of c_i and c_j are frequently changed together in the past. Such types of information need to be extracted from the change logs or commits recorded in the version or change history of the software system. The definition of $MChC(c_i)$ is mentioned below.

$$MChC(c_i) = \sum_{j=1, j\neq i}^{n} MCh_{prox}(c_i, c_j). \tag{12}$$

Here, $MChC(c_i)$- Method level change coupling of class c_i, $MCh_{prox}c_i, c_j)$- Method level change coupling (proximity) between the classes c_i and c_j. n-Total number of classes. The metrics to compute the method level change coupling (proximity) i.e.,$MCh_{prox}c_i, c_j)$ between the class's c_i and c_j is described as under:

$$MCh_{prox}\left(c_i, c_j\right) = \sum_{k=1}^{NMc_i} \sum_{l=1}^{NMc_j} \frac{|c_i m_k \cap c_j m_l|}{|c_i a_k \cup c_j a_l|}, \tag{13}$$

where NMc_i and NMc_j are the number of methods of the classs'sc_i and c_j, respectively. $c_i m_k$ is the list of methods of class c_i (k=1 to NMc_i), $c_j a_l$ is the list of methods of class c_j (k=1 to NMc_i). $|c_i m_k \cap c_j m_l|$ gives the number of changes commits in which the methods of classes c_i and c_j that are co-changed together. $|c_i a_k \cup c_j a_l|$ gives the number of changes commits in which the methods of classes c_i or c_j or both are changed.

The two classes c_i and c_j may have class level change coupling, a) if class c_i inherits from another class c_j, b) if class c_i realizes interface of class c_j. Such classes may frequently co-change together in the past, and this type of coupling information could be extracted from the change logs or commits recorded in the version or change history of the software system. The definition of $CChC(c_i)$ is mentioned below.

$$CChC(c_i) = \sum_{j=1, j\neq i}^{n} CCh_{prox}(c_i, c_j), \tag{14}$$

where $CChC(c_i)$- Class level change coupling of class c_i, $CCh_{prox}\left(c_i, c_j\right)$- Class level change coupling (proximity) between the classes c_i and c_j, n-total number of classes. The metrics to compute the class level change coupling (proximity) i.e.,$CCh_{prox}\left(c_i, c_j\right)$ between the class's c_i and c_j is described as under:

$$CCh_{prox}\left(c_i, c_j\right) = \sum_{j=1, j\neq i}^{n} \frac{c_i \cap c_j}{c_i \cup c_j}. \tag{15}$$

Here, $c_i \cap c_j$ gives the number of changes commits in which the classes c_i and c_j that are co-changed together. $c_i \cap c_j$ gives the number of changes commits in which the classes c_i or c_j or both are changed.

The overall process of computing the different levels of change-coupling involvethe following three significant steps: Step-1: Mining software repository to extract all the changelogs/commits of the subjected software application. Step-2: Filtration and pre-processing of the available change-commits. Step-3: The relevant change-commits are explored to extract the co-change pattern to compute $AChC(c_i)$, $MChC(c_i)$, $CChC(c_i)$ and $ChC(c_i)$ change coupling metrics as described above. It is demonstrated in Fig. 2.

Software remodularization as many-objective optimization problem Inmany-objective optimization model of any problem, a large number (i.e., more than three) of objective functions (often conflicting) along withsome equality and inequality constraints has to be optimized to generate the solutions.The many-objective optimization definition of the software remodularization problem can be given as follows:

$$\begin{aligned} &\min \mathrm{F}(d) = [f_1(d), f_2(d), \ldots, f_M(d)]^{\mathrm{T}}, M > 3, \\ &d_i^{Lower} \leqslant d_i \leqslant d_i^{Upper}, \quad i = 1, ..., n. \end{aligned} \tag{16}$$

In the context of software remodularization problem, $f_1(d), f_2(d), \ldots, f_M(d)$ are the quality criteria defined in terms of decision variable d. The d_i^{Lower} and d_i^{Upper} are the ith decision variable's lower and upper bound and n is the number of decision variables.In software remodularization, we have defined the different set of objective functions based on the

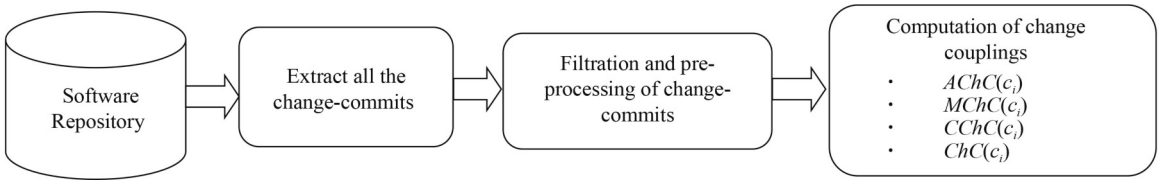

Fig. 2 Methodology for computing different level of change coupling

structural, lexical, and changed-history dependency information. The definition of each group of many-objective SRP (i.e., variants of the many-objective optimization problems) is given as follows:

- MaSRV-1: In this variant the objective functions are: changed-history coupling (to maximize), changed-history cohesion (to maximize), changed-history MQ (to maximize), number of clusters (to maximize), and the difference between minimum and maximum number of modules in a cluster (to minimize).
- MaSRV-2: In this variant the objective functions are: lexical coupling (to maximize), lexical cohesion (to maximize), lexical MQ (to maximize), number of clusters (to maximize), and difference between minimum and maximum number of modules in a cluster (to minimize).
- MaSRV-3: In this variant the objective functions are: structural coupling (to maximize), structural cohesion (to maximize), structural MQ (to maximize), number of clusters (to maximize), and difference between the minimum and maximum number of modules in a cluster (to minimize).
- MaSRV-4: In this variant, the objective functions are: changed-history coupling + structural coupling (to maximize), changed-history coupling + structural cohesion (to maximize), changed-history coupling + structural MQ (to maximize), number of clusters (to maximize), and difference between minimum and maximum number of modules in a cluster (to minimize).
- MaSRV-5: In this variant, the objective functions are: lexical + structural coupling (to maximize), lexical + structural cohesion (to maximize), lexical + structural MQ (to maximize), number of clusters (to maximize), and difference between minimum and maximum number of modules in a cluster (to minimize).
- MaSRV-6: In this variant, the objective functions are: changed-history + lexical + structural coupling (to maximize), changed-history + lexical + structural cohesion (to maximize), changed-history + lexical + structural MQ (to maximize), number of clusters (to maximize), and difference between minimum and maximum number of modules in a cluster (to minimize).

The different set of many-objective software remodularization formulations (i.e., MaSRV-1, MaSRV-2, MaSRV-3, MaSRV-4, MaSRV-5, and MaSRV-6) represents the various aspects of SRPs. Therefore, the optimization of each aspect of SRP can produce different remodularization solutions.

Software remodularization solution encoding In the reformulation of any problem as a search-based optimization problem, the designing of objective functions and encoding of the candidate solution plays an important role. The generation of all possible solutions for the optimization problem is a challenging task, and for a real-world optimization problem, it becomes more difficult. In an optimization problem, the candidate solution is commonly defined in terms of the decision variable's set. The particular instance of each decision variable represents a specific solution of candidate. To generate all possible candidate solutions for a specific problem of optimisation, an effective encoding mechanism for the representation of candidate solution is required. For the software remodularization optimization problems, the integer vector-based encoding mechanism (Bavota et al. 2010, Praditwong et al. 2011; Prajapati and Chhabra 2017) is a widely accepted representation technique. Therefore, we have also used this encoding mechanism in this work.

3.2 MaABC

A variety of metaheuristic optimization approaches have been proposed to generate a set of a good representative sample of approximation of Pareto optimal solutions for many-objective optimization problems. These approaches are widely categorized as 1) Dimensionality reduction approach [39], 2) Relaxed dominance approach [40], 3) Diversity injection-based approach [41], 4) Aggregation-based approach [42], 5) Indicator-based approach [43], 6) Reference set-based approach [44], 7) Preference-based approach [45,46], 8) Decomposition based approach [47].

Even after huge development in designing various categories of many-objective algorithms, their applications to real-world optimization problems gained little attention. The divergent characteristics of real-world optimization problems make the application of many objective algorithms difficult and challenging. In the past few years, some researchers have tried to tailor the metaheuristic algorithms for the different real-world problems such as scheduling of various industrial tasks [48], calibration optimization of the automotive [49], optimization of hybrid car controllers [50]. The flexibility in the transformation of various software engineering tasks as a search-based optimization problem creates a huge opportunity for metaheuristic optimization algorithms. In the past two decades, under the umbrella of search-based software engineering (SBSE) [51], a large number of metaheuristic optimization algorithms have been developed to address various software engineering problems.

Most of the software engineering problems have been addressed using the conventional multi-objective metaheuristic optimization algorithms. However, an increase in the number of objective functions in many software engineering problems demanding more advanced multi-objective metaheuristic optimization algorithms, i.e., many-objective metaheuristic optimization algorithms. To address the many-objective software remodularization, the studies [10,11] proposed many-objective optimization algorithm by incorporating the various strategies in the artificial bee colony (ABC) algorithm [52]. Even though these approaches perform effectively, there are still various improvements corresponding to the generation of more effective remodularization solutions. In this work, we exploit the basic framework of MaABC [10,11] to optimize the software quality. The basic framework of MaABC remains similar to the original MaABC algorithm with some minor changes in the selection techniques.

The MaABC framework consists of four major components, and each component are responsible for performing

specialized activities. The components are: 1) initialization phase, 2) send employed bees, 3) send onlooker bees, and 4) send scout bees. The flowchart of the MaABC is presented in Fig. 3. The detailed operations involved in each of these phases are given in the following paragraphs.

Initialization phase The initialization phase of the MaABC performs many activities that set up a strong basis for the smooth working of the next phases. The first activity of this phase is the initialization of candidate solutions for the first generation's population. To perform this, we generate a set of candidate solutions of population size belonging to the search space of the remodularization problem. At the beginning of the algorithm, there requires a set of initial candidate solutions, i.e., a population that can proceed with the optimization process. To generate the initial candidate solution for the population, we use the random initialization approach, where the decision variables value for each of the candidate solutions is selected randomly. Apart from the population initialization, the various parameters of the algorithm also need to be initialized. Therefore, in this phase, the appropriate values for each of the parameters are assigned so that the optimization process can proceed toward the intended optimal direction.

To demonstrate the population initialization, let's consider the set of all possible remodularizationsolutions in the search space are t, i.e.,$s = \{s_1, s_2, ..., s_t\}$. The particular software remodularization solutions_i can be represented as a set of n decision variable, i.e., $s_i = \{s_i^1, s_i^2, ..., s_i^n\}$. The range of a particular decision variable, i.e.,$s_i^j \in [UB^j - LB^j]$, where UB^j and LB^j are the upper and lower bound of the jth decision variable of s_i candidate solution. For example, the remodularization solution demonstrated in Fig. 1 can be represented as s ={1, 2, 2, 1, 3, 1, 3, 3} where the range of each decision variable is between 1 to 8. In this example, the classes $\{s_1, s_4, s_6\}$ belong to package $\{m_1\}$, the classes $\{s_2, s_3\}$ belongs to package $\{m_2\}$, and classes $\{s_5, s_7, s_8\}$ belong to package $\{m_3\}$. In the context of the ABC algorithm, the candidate solution is viewed as food source, and the candidate solution's fitness is considered quality of the food source. After initialization of the population, the objectives functions and fitness function corresponding to each candidate solutions are computed. Based on the non-domination relations, the external archives CA and DA are updated according to their updation rules. The details of the updation rules for the CA and DA archives are given in the further paragraphs. The trial value, i.e., TR associated with each candidate solutions of the population is set to zero (i.e., $TR_1 = TR_2 = \cdots = TR_N = 0$).

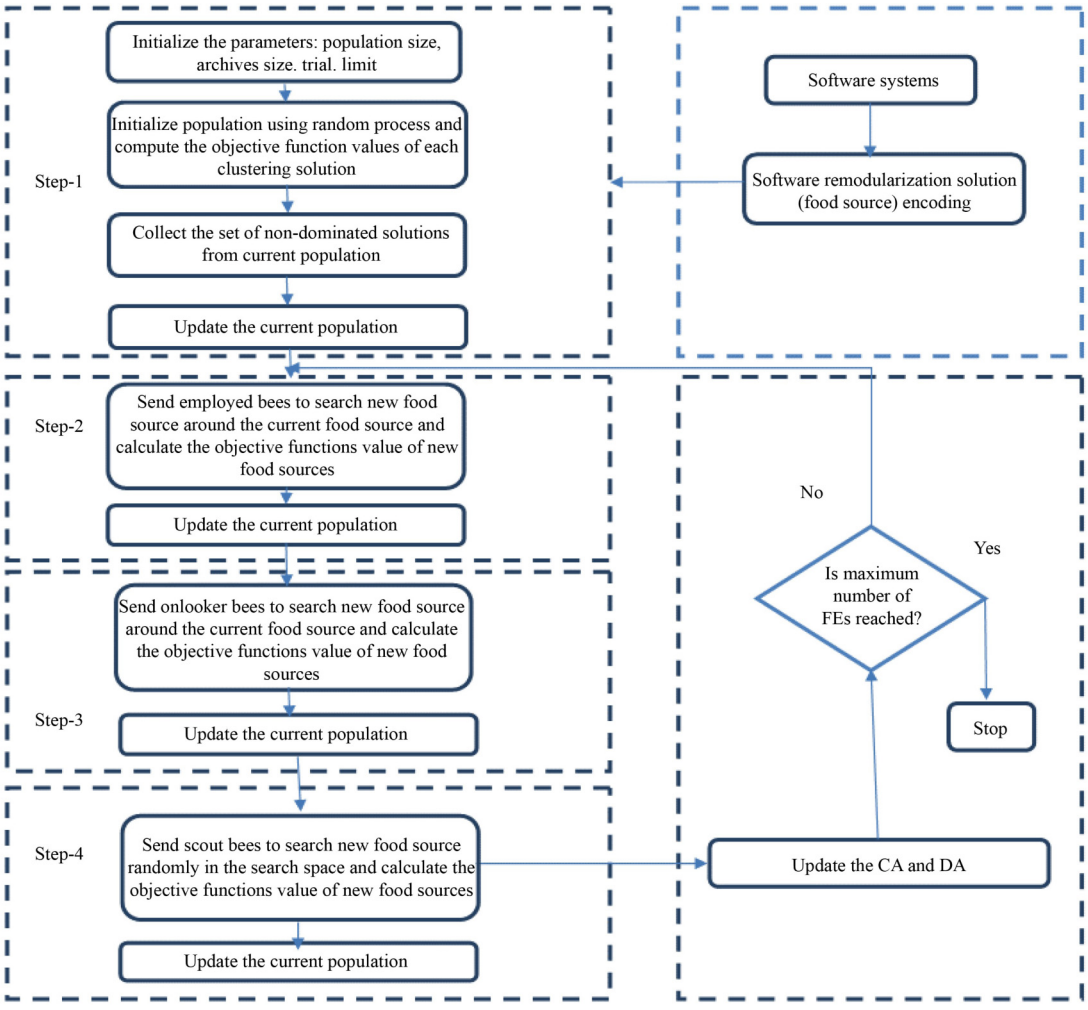

Fig. 3 Framework of the optimization process of the MaABC

Send employed bees After the initialization of the food sources/candidate solutions in the population, the employed bees staying in the hive fly towards the food sources. For each food source an individual employed bee is assigned. Each individual employed bee starts searching the new food sources based on the information about the food sources they have in the memory with the hope to find a better food source. In this study, to create the new solutions for the employed bees, we use the information of the candidate solutions of the current population and the candidate solutions of the CA and DA archives. The reason of considering the candidate solutions of CA and DA is that these solutions help in guiding the employed bees towards more promising candidate solutions.

To balance convergence and diversity in the resultant population, we fixed the consideration probability of the current population and the CA+DA candidate solutions 0.4 and 0.6, respectively. The objective functions and fitness of the newly generated candidate solution is computed. If the newly generated candidate solution's fitness is found better to the current solution, then the current solution of the population is replaced with the newly generated candidate solution; otherwise, the newly generated candidate solution is discarded and preserved the current solution for the next generation. The trial value of the unimproved candidate solutions of the population is incremented by 1. The detailed description of the procedure is given in Algorithm 1.

Algorithm 1 Send employed bees

Input: current population P, the candidate solutions of CA and DA,
Output: new population after employed bees
for i=1 to N
for j=1 to n
if r <=0.4 then
$s_{new}^{j} = RandInt(s_1^j, s_2^j, \ldots, s_N^j)$
else
$s_{new}^{j} = RandInt(s_1^j, s_2^j, \ldots, s_{|CA+DA|}^j)$
end if
end For
Compute the objective vectors and fitness
if the fitness (s_{new})>fitness (s_i) then
$s_i = s_{new}$
else
$TR_i = TR_i + 1$
end if
end for

Send onlooker bees After exploiting the search space around the current food source, each employed bee returns back to the hive and share the information of the food source/candidate solution to the onlooker bees waiting in the hive. These onlooker beesmake a further search around the food sources/candidate solutions shared by the employed bees. The employed bees share the food source information to the onlooker bees by performing the dance in the hive's dance area. The duration of the dance of each employed bee is the proportion to the quality of the food source they have exploited.

The number of onlooker bees flying towards the food sources having better quality is more. In another word, the food source having the better food source quality, will attract a more significant number of onlooker bees and food source having the poor food source quality may have a smaller number of onlooker bees. The movement probability of onlooker bee towards a particular food source is computed in terms of the candidate solutions' fitness values. The searching nature of the onlooker bees around the food source is the same as the employed bees' searching behaviour. The only difference between the employed bees and onlooker bees is the number of bees deploying on a particular food source. The pseudocode of the onlooker bees is given in Algorithm 2.

Algorithm 2 Send onlooker bees

Input: current population P, CA and DA, probability p_i of each solution, TR_i
Output: new population after onlooker bees
Set $i = 1, t = 1$
while $i \leqslant P$ do
if p_i> rand () then
Generate the value each s_{new}^{j} as generated in employed bees
end if
Compute the objective vectors and fitness
if fitness (s_{new})>fitness (s_i) then
$s_i = s_{new}$
else
$TR_i = TR_i + 1$
end if
$i = i + 1, t = t + 1$
if $t > N$ then
Set $i = 1$
end if
end while

Send scout bees Thesignificant contribution of the employed bees and the onlooker bees is the searching a better food source around a known food source. It means that the employed bees and the onlooker bees search a particular search space region based on their current and previous experiences. To explore a food source in the search space, the concept of scout bee is used. During the process of send employed bees and send onlooker bees, if the quality of any food does not improve after a certain attempt, then a scout bee corresponding to this food source and the scout bee searches a random food source in the search space. The random creation of the food source is the same as the random initialization of the candidate solutions in the population. The pseudocode of the scout beeis given in Algorithm 3.

Update the CA and DA The concept of applying two archives in balancing the diversity and convergence in the approximation of Pareto front is an effective method. The idea of two archives has been used in various metaheuristic search approaches to address the many-objective optimization problems [44,53,54]. The common purpose of the application of two external archives is to guide the optimization process as well as to store the approximation of the Pareto front. In the two archive-concept, the two different archives namely convergence-based archive (CA) and diversity-based archive (DA) are maintained to store the non-dominated collected during the optimization process. The size of CA and DA archives can be either fixed or variable. In this work, we keep the fixed and equal size of CA and DA archives. The major

Algorithm 3 Send scout bees

Input: size of the population P, number of decision variable n, TR_i , Limit.
Output: new population
for i=1 to P
if max $(TR_i) > Limit$ do
for j=1 to n
$s_{new}^j = RandInt(UB_i^j - LB_i^j)$
end for
Compute the objective vectors and fitness of newly generated solution
if fitness (s_{new})>fitness (s_i) then
$s_i = s_{new}$ and Set$TR_i = 0$
else
$s_i = s_i$
end if
end if
end for

idea for the updation of the CA and DA archives used in this work is borrowed from the literature [44,54].

The detailed description of the pseudocode used for implementing CA and DA updation rule is given in Algorithm 4. In this process, first of all, the non-dominated solutions from the population, after completion of each iteration, i.e.,send employed bees, send onlooker bees, and send scout bees, are collected. Next, the collected non-dominated solutions are compared with the candidate solutions stored in the CA and DA archives. Suppose the collected non-dominated solution dominates one or more candidate solutions of CA and DA archive. In that case, the dominated solutions from both CA and DA archives are deleted, and the collected non-dominated solution is placed in the CA archive. Suppose the collected non-dominated solution has the non-domination relation to all the CA and DA candidate solution. In that case, the collected non-dominated solution is placed in the DA archive. If the collected non-dominated solution is dominated solution the it is simply discarded.

Algorithm 4 Update CA and DA

Input: set of non-dominated solutions $\Theta = \{q_1, q_2, \ldots, q_k\}$, CA and DA archives.
Output: updated CA and DA
for i=1 to k do
if q_i can not be dominated by solutions of CA and DA
if q_i dominates any solution of CA and DA
Remove dominated solutions form CA and DA
Set $q_i.flag = 1$ // q_i with domination
else
Set $q_i.flag = 0$ // q_i without domination
end if
end if
end for
Add the solutions with $flag = 1$ to CA and solutions with $flag = 0$ to DA
if |CA|>maxSize of CA and |DA|>maxSize of DA then remove the solution based on deletion rules

Since the CA and DA archives' size is fixed, they can accommodate only a limited number of candidate solutions. If the number of candidate solutions increases to CA's size, then the extra candidate solutions from the CA are deleted based on their fitness (computed using Eq. (15)). In particular, the candidate solutions in CA have the smallest $I_{\varepsilon+}$ lost is deleted in every generation of ABC algorithm. Further, the fitness of the remaining candidate solutions of the CA are updated accordingly. For the case of DA archive, if the number of candidate solutions increases to the size archive, the candidate solutions located in the highly dense region are removed. Conversely, the candidate solutions located on the boundary or far distance are selected and kept in the DA. To compute the distance between the candidate solutions a suitable distance metric (Lp-norm-based distances) should be used. In this work, we use the L_2-norm-based distances (i.e., Euclidean distance) to compute the candidate solutions' similarity. The reason for selecting the L_2-norm-based distances to compute the similarity is that it works effectively for the vector having lengths2 to 7 as in our case.

Fitness computation The fitness computation of the candidate solutions to rank them for their selection is an important task in many-objective optimization. In this study, we use the fitness evaluation method as suggested in the Two_Arch2 [44]. In case of CA's overflows, the extra candidate solutions are removed. The CA's goal is to maintain the candidate solutions that have the high tendency of convergence. Therefore, we have to select a fitness evaluation method that can preserve this property of the CA. In this paper, we choose the $I_{\varepsilon+}$ quality indicator discussed in IBEA [43] as the fitness evaluation for the CA's candidate solutions. The $I_{\varepsilon+}$ is a quality indicator that compute the minimum distance that one candidate solution requires in order to dominate another candidate solution in the objective space. The fitness $F(s_1)$ of a particular candidate solution$'s_1'$ in term of $I_{\varepsilon+}$ corresponding to population $'P'$ can be computed as follows:

$$I_{\varepsilon+}(s_1, s_2) = \min_{\varepsilon}(f_i(s_1) - \varepsilon \leqslant f_i(s_2), 1 \leqslant i \leqslant m, \quad (17)$$

$$F(s_1) = \sum_{s_2 \in P \setminus \{s_1\}} -e^{-I_{\varepsilon+}(s_2, s_1)/0.05}, \quad (18)$$

where m is the number of objectives considered in the many-objective optimization problem. The quality indicator $I_{\varepsilon+}(s_1, s_2)$ is used to compute the individual fitness $F(s_1)$ in the population or set of candidate solutions "P". To remove the extra candidate solutions from the DA archive, we use selection approach govern by the Euclidean distance. In this approach, the boundary candidate solutions (solutions with minimal or maximal objective values) ofDA are firstly selected. Then, the candidate solutions with largest Euclidean distance are selected and placed in the DA.This is a repetitive process, and it is carried out until the DA is completely filled.

4 Experimentation setup

To evaluate the effectiveness of the proposed approach, we conducted an empirical analysis. To perform this, an experimentation setup is designed. The details of the experimental setup are given as follows.

System configuration-To implement the proposed approach; we use the Java programming language. The implemented proposed approach is executed on the personal computer

having Intel Core i7-1160G7 4.4 GHz CPU hardware configuration and Microsoft Windows 10 Home operating system.

Problem instances The selection of problem instances to evaluate the prosed approach is an important task as it helps to generalize the results. We select the sample of those problem instances that can represent of whole problem instances of the population. The major characteristics of the chosen problem sets are given in Table 1.

Table 1 Description of the selected problem instances

Software systems	Version	#Classes	#Relationships	#Modules
JFreeChart	0.9.21	401	1420	50
JHotDraw	6.0b1	398	2125	17
JavaCC	1.5	154	722	6
JUnit	3.81	100	276	6
DOM 4J	1.5.2	195	930	16

The selected problem instances are open-source software systems developed in Java programming language and easily available on the web. These software projects have also been by different researchers and academicians to validate similar approaches.

Results collection The proposed approach works in the randomization environment (i.e., metaheuristic optimization algorithm). Therefore, it is not guaranteed that the proposed work will produce the same results on different execution over the same problem instance. To validate the stochastic results, we collect the sample results of 31 run for each problem instance. The other challenge is that each of the sample results is a set of trade-off solutions (i.e., Pareto front) not a single solution due to the multi-objective nature of the problem. To select a single best solution from the Pareto front, we use the trade-off worthiness metric [45] approach.

Evaluation criteria To evaluates the performance of the proposed approach, we use the internal quality metrics and external quality metrics. The internal quality metric is used to evaluate the software remodularization solution from the design of the software structure perspective while the external quality metric is used to compare the software remodularization solution from the authoritative and developer's perspective. For the internal quality metrics, we use the modularization quality (MQ) [6] a highly used software quality metrics to evaluate the modularization quality. On the other hand, we use the MoJoFM [55] metric to compare the software remodularization solution with the authoritative and developer's perspective of software remodularization.

Competitive approaches To justify the proposed approach's supremacy, we compare the results with the existing remodularization approaches. The significant difference between our approach and the existing approaches [1,6,9,14,56] are related to the information used to define the objective functions and the customized metaheuristic optimization algorithm. The brief description of the existing software remodularization approaches is given in Table 2.

Authoritative modularization In the past three decades, many software remodularization approaches have been proposed. The different software remodularization approaches generally suggest different remodularization solution. The unavailability of benchmark or ground truth software modularization evaluates these remodularization approaches a challenging task. The limited availability of software engineers for the verification of remodularization solution makes it more challenging. All these things can lead 1) difficulty in assessing the modularization techniques, 2) trusting on a particular modularization technique can be a risk, 3) researchers may follow flawed strategies in improving the accuracy of the methods.

To evaluate the proposed approach, we create a ground truth software modularization of the selected software projects. In this study, the ground truth software modularization is the modularization solution verified by the software developers and system designers who have a good understanding of the project and problem domain. The creation of ground truth software modularization where the applications or software projects own developers not involved is known as authoritative modularization. It is easy for an academician and researchers to create authoritative modularization instead of ground truth software modularization. Even it is very challenging to create the authoritative modularization without the involvement of applications own developers, because there can be chance of omitting original design decision and inclusion of spurious one. But in this study, we try to overcome these challenges.

Creating an authoritative modularization with the help of software developers who are not a part of system development team is a time-consuming task. On the other hand, software engineers do not show interest in the system in which they have not worked. Moreover, such outsider software developers do not have the correct, complete, and idealized picture of the system's modular structure. To reduce the software developers' effort while ensuring the generation of accurate, authoritative modularization, we first extract the original modularization of the software system. Then we involve the software developers in completing the process. From the software systems we extract the bundle definition files for the original modularization. If the bundle definition files are not available, we extract the existing package organization as the original modularization.

Table 2 Description of the existing software remodularization approach

Approaches	Abbreviation	Metaheuristic	Information	Optimization approach
Praditwonget al. 2011 [6]	PRAD-2011	Two-Archive GA	Structural	Multi-objective
Mkaouer et al. 2015 [1]	MKAO-2015	NSGA-III	Structural + lexical + changed-history	Many-objective
Prajapati and Chhabra 2017 [9]	PRAJ-2017	NSGA-II	Structural + lexical	Multi-objective
Jalali et al. 2019 [56]	JALA-2018	Global and local search	Structural + lexical	Multi-objective
Prajapati et al. 2020 [14]	PRAJ-2020	NSGA-II	Structural + lexical + changed-history	Multi-objective

Developer's view for software remodularization Apart from the authoritative software remodularization, we also evaluated the proposed approach with the modularization solution suggested by the developers. In this case, we only select the 20 modules of each software project and show the developers and collect their suggestions. To conduct this task, we involve 21 software developers who are not the part of the system development. Because finding the original application's developers is not feasible for the academicians. The selected developers are four PhD students, five M.Tech students, and six B.Tech final year students, and six software developers working on similar projects. These students and software developers are selected based on their knowledge and understanding of the selected software projects.

5 Experimental results and discussion

To make the results more comprehensive, we broadly divided the experimentation results into two parts: 1) the results related to the internal quality metric (i.e., MQ results) and 2) the results related to the external quality metric (i.e., MoJoFM results). The details of the results and the discussion is provided in the following subsections.

5.1 MQ results

The mean MQ values of the remodularization solutions obtained through the different variants of the remodularization approaches are presented in Table 3. As the variant MaSRV-6 represents our proposed approach, we compared the our proposed MaSRV-6 with the other variants, i.e., MaSRV-1, MaSRV-2, MaSRV-3, MaSRV-4, and MaSRV-5. To this comparison, the Wilcoxon rank sum test (0.05 significance level)statistical test is applied. The symbols "−", "+", and "≈" are used to represent whether the MQ value of different variants is significantly worse than, better than, and not significant different than the proposed approach, i.e., MaSRV-6.

If we see the mean MQ values of each variant of the software remodularization, the mean MQ values of the variant MaSRV-6 is greater than the rest of the variants, MaSRV-1, MaSRV-2, MaSRV-3, MaSRV-4, and MaSRV-5 in all cases. If we rank the performance of each variant in terms of MQ values, the variant MaSRV-6 will get rank 1, and the variants MaSRV-1, MaSRV-2, MaSRV-3, MaSRV-4, and MaSRV-5 will get the rank 4, 6, 5, 3, and 2, respectively corresponding to the all cases. These results indicate that the proposed approach MaSRV-6 is the best performer, and the MaSRV-2 is the worst performer in generating the remodularization solution in terms of MQ values. If we see the Wilcoxon rank sum test results, the proposed MaSRV-6 variant performs significantly better to the MaSRV-1, MaSRV-2, and MaSRV-3 in all cases. There is only one case (i.e., JavaCC) where the MaSRV-6 is not significantly better to the MaSEV-4, and there are only two cases (i.e., JHotDraw and DOM4J) where the MaSRV-6 is not significantly better to the MaSEV-5.

The Wilcoxon rank-sum test results obtained by comparing the proposed approach and the existing approaches with respect to the MQ values are provided in the Table 4. The description of the statistical test results presented in Table 5 is as follows: The entries of first row and first column are the name of existing and proposed approach (i.e., MaSRV-6) and the rest of rows and columns are the comparison results in

Table 3 Mean MQ values of the different software remodularization variants

Software systems	MaSRV-1	MaSRV-2	MaSRV-3	MaSRV-4	MaSRV-5	MaSRV-6
JFreeChart	26.87 [−]	24.45 [−]	25.38 [−]	33.59 [−]	32.65 [−]	37.94
JHotDraw	14.39 [−]	13.53 [−]	12.48 [−]	19.43 [−]	20.43 [≈]	22.52
JavaCC	4.24 [−]	2.89 [−]	2.38 [−]	6.12 [≈]	5.52 [−]	6.48
JUnit	4.46 [−]	3.78 [−]	3.18 [−]	4.68 [−]	5.37 [−]	6.89
DOM 4J	8.29 [−]	6.48 [−]	5.34 [−]	11.30 [−]	12.48 [≈]	13.17

Table 4 Comparison of proposed approach to the existing approaches in terms of MQ values

	PRAJ-2020	PRAD-2011	JALA-2019	PRAJ-2017	MKAO-2015	Proposed
PRAJ-2020	NA	[− − − − −]	[− − − ≈ −]	[− ≈ − − −]	[− − − ≈ −]	[≈ + + ≈ +]
PRAD-2011	[+ + + + +]	NA	[− ≈ + ≈ −]	[+ + ≈ + +]	[+ + + + +]	[+ + + + +]
JALA-2019	[+ + + ≈ +]	[+ ≈ − ≈ +]	NA	[+ + + + +]	[+ + + ≈ +]	[+ + ≈ + +]
PRAJ-2017	[+≈ + + +]	[− − ≈ − −]	[− − − − −]	NA	[− − − − −]	[+ + + + +]
MKAO-2015	[+ + + ≈ +]	[− − − − −]	[− − − ≈ −]	[+ + + + +]	NA	[+ ≈ + + ≈]
Proposed	[≈ − − ≈ −]	[− − − − −]	[− − ≈ − −]	[− − − − −]	[− ≈ − − ≈]	NA

Table 5 Mean and Std of MoJoFM obtained through comparison of variants andauthoritative modularization

Software systems	MaSRV-1	MaSRV-2	MaSRV-3	MaSRV-4	MaSRV-5	MaSRV-6
JFreeChart	35.213 [2.426]	57.487 [5.353]	65.341 [6.732]	85.384 [6.845]	91.742 [8.328]	95.459 [7.462]
JHotDraw	27.348 [2.173]	52.485] [5.374]	67.395 [6.395]	82.394 [6.784]	90.304 [8.374]	93.754 [7.971]
JavaCC	38.384 [2.943]	49.948 [4.394]	61.485 [5.309]	80.406 [6.471]	89.418 [8.485]	96.803 [9.312]
JUnit	35.942 [3.294]	53.492 [4.582]	63.485 [5.394]	8.404 [6.482]	89.578 [7.405]	93.456 [8.348]
DOM 4J	29.994 [2.384]	51.394 [4.394]	66.320 [5.942]	82.482 [6.042]	90.493 [8.483]	92.403 [8.193]

terms of significance difference corresponding to all five problem instances. There are five symbols in each table entry, and each symbol is corresponding to each of the problem instances (i.e., JFreeChart, JHotDraw, JavaCC, Junit, and DOM4J). The "+" and "–" symbols are used to show the "significant" differences between the approaches and the "≈" symbol used to denote the "no significant" difference between the approaches. Specially, the "+" denotes that the approach depicted in row performs significantly better compared to the approach depicted in column over corresponding problem instance. Similarly, the "–" denotes that the approach depicted in column performs significantly better compared to the approach depicted in row corresponding to that position problem instance. The "≈" symbol indicates no significant difference between the row's approach and the column's approach corresponding to that position problem instance. For example, consider the symbols [≈ + + ≈ +] placed in second row and seventh column of Table 4. The first symbol, i.e., "≈" denotes no significant difference between the proposed approach and the PRAJ-2020corresponding to the JFreeChart problem instance. The second symbol i.e., "+" denotes that the proposed approach is performing significantly better to the PRAJ-2020 corresponding to the JHotDraw problem instance.

Even though the Table 4 presents the comparison results of each approach with other approaches, but here we are only focusing the results of the proposed approach and the other existing approaches. If we see the results of the proposed approach, it demonstrates that the proposed approach performs significantly better in most cases than the existing approaches. The proposed approach performs significantly better to the PRAD-2011 and PRAJ-2017 in all problem instances, while the proposed approach has few cases where it is not performing significantly better than the MKAO-2015 and PRAJ-2020. Overall, these results validate that the proposed approach has the good capability of generating the remodularization having the better MQ values.

5.2 MoJoFM results

The MoJoFM external quality metrics computes the similarity between the remodularization obtained through the proposed approach and the authoritative remodularization or developers' view of remodularization. First, we present the MoJoFM results of proposed and authoritative remodularization, then we present MoJoFM results of proposed and developers view of remodularization. Table 5 presentsthe mean and standard deviation of the MoJoFM after applying over the different variants of remodularization and the authoritative modularization. Here, the larger MoJoFM value indicates that the remodularization obtained through the different approaches is more similar or closer to each other. In comparison, the smaller MoJoFM value indicates that the remodularization obtained through the different approaches is more dissimilar to each other.If we see the MoJoFM results of each software remodularization variants, the variant MaSRV-6 (i.e., proposed approach) has achieved the better values than the rest of the software remodularization variants, i.e., MaSRV-1, MaSRV-2, MaSRV-3, MaSRV-4, and MaSRV-5 in most of the cases. On the other hand, the variant MaSRV-1 has the worst MoJoFM values than the rest of the variants in most cases.The such good value of the variant MaSRV-6 indicate that the proposed approach can generate more similar modularization corresponding to the authoritative modularization. Overall pattern of the MoJoFM values of each software remodularization variant is MaSRV-1<MaSRV-2< MaSRV-3< MaSRV-4< MaSRV-5< MaSRV-6 in all problem instances.

To test the similarity of the remodularization solution produced through the different variants of the remodularization approaches with the remodularization solution suggested by the developers, we also computed the MoJoFM values by applying over these two modularizations. Table 6 presents results of MoJoFM obtained through comparison of different software remodularization variants of the proposed approach and the modularization suggested by the developers. Similar to the case of authoritative modularization, the variant MaSEV-6 can achieve the better MoJoFM values than the rest of the variants, i.e., MaSRV-1, MaSRV-2, MaSRV-3, MaSRV-4, and MaSRV-5 in most of the cases. The variant MaSRV-1 has shown the lowest MoJoFM values in almost all cases and regarded as worst performer among all variants. The MoJoFM values of the MaSRV-5 are closer to the MoJoFM values of MaSRV-6 in most cases compared to the rest of the variants. Hence, the MaSRV-5 is the second-best performer in terms of similarity with the developers view of modularization. Overall, the order of the variants with respect to the MoJoFM values corresponding to all problem instances is MaSRV-1< MaSRV-2< MaSRV-3< MaSRV-4< MaSRV-5< MaSRV-6. These observations validate that the proposed MaSRV-6 has a strong capability to generate the remodularization solution that can be easily acceptable to the software developers.

Apart from comparing the remodularization results of the different software remodularization variants in terms of MoJoFM values with respect to the authoritative and

Table 6 Mean and Std of MoJoFM obtained through comparison of variants and developers' modularization

Software systems	MaSRV-1	MaSRV-2	MaSRV-3	MaSRV-4	MaSRV-5	MaSRV-6
JFreeChart	26.846 [2.394]	48.384 [4.294]	65.058 [5.395]	85.405 [6.495]	92.048 [6.495]	94.954 [7.584]
JHotDraw	35.485 [3.472]	53.223 [4.385]	67.645 [5.762]	83.565 [6.351]	90.475 [8.576]	93.475 [8.221]
JavaCC	38.875 [2.485]	47.575 [3.472]	64.567 [4.998]	81.332 [6.305]	89.412 [7.337]	94.392 [7.894]
JUnit	32.574 [2.495]	45.566 [4.001]	68.058 [5.485]	81.048 [7.049]	91.494 [7.595]	93.204 [7.434]
DOM 4J	36.048 [2.495]	46.954 [3.595]	66.495 [6.503]	83.204 [7.596]	89.496 [7.123]	95.312 [8.563]

developers view of software remodularization, we have also compared the best performing variants, i.e., MaSRV-6 (i.e., proposed approach) with the existing software remodularization approaches. Tables 7 and 8 present the Wilcoxon test results obtained by comparing the proposed and existing remodularization approaches. The meaning and description of the symbols used in Table 7 are the same as provided in the Table 4. Table 7 presents the Wilcoxon results obtained by comparing the all considered approaches in terms of the authoritative modularization. The results presented in Table 7 show that the proposed approach performs significantly better than existing approaches in most cases. The second-best performer among all approaches is the PRAJ-2020, and the third best performer is the MKAO-2015. The approach PRAD-2011 is the worst performer, and JALA-2019 and PRAJ-2017 are the average performer.

To validate the supremacy of the proposed approach (i.e., MaSRV-6) with respect to the existing approaches of software remodularization from the developers view, we compared the obtained remodularization results of the proposed with the remodularization of the existing approaches. Table 8 presents the MoJoFM results after applying it between the remodularization of the proposed approach and the existing approaches. These results are shown in terms of the Wilcoxon rank sum test (i.e., whether the proposed approach perform significantly better, worst, or no significant difference). Wilcoxon results obtained through the comparison of the all considered approaches in terms of the developers suggested modularization. The results show that the proposed approach performs significantly better than the existing approaches in most cases. If we see the rest of the approaches' results, we can find that the Amarjeet et al. [14] approach is second and MKAO-2015 is the third outperforming approach among all approaches. The results of the PRAD-2011 is worst in most cases, and the results of JALA-2019 and PRAJ- 2017 are the average.

5.3 Discussion

In the formulation of SRP as a many-objective optimization problem, the definitions of the objective functions and the designing of the metaheuristic optimization algorithm significantly contribute to finding the good remodularization solutions. Apart from the evaluation of the software remodularization solution quality, the objective functions help in guiding the remodularization process towards the expected solution. For the object-oriented software system, software package coupling and cohesion are the two main objective functions that are often used as objective functions and some other supportive objective functions in SBSR. The software package coupling and cohesion are generally defined in terms of class coupling of the software system. Therefore, the definition of class coupling has a major importance in SBSR.

To compute the coupling between the source code classes, various types of source code information such as structural, lexical, and changed-history information are used. The different types of information used in computing the class coupling for software package coupling and cohesion computation have different importance in generating the remodularization solution in the SBSR. For example, structural information help in guiding the remodularization process towards the remodularization solution which is better from the structural quality point of view. The use of lexical or textual information can lead the remodularization process towards the remodularization solution which is more semantically coherent with the developers perspective of software remodularization. Similarly, the changed-history information-based class coupling can lead remodularization process towards the remodularization solution, which is logically coherent from the developers perspective.

To generate the sophisticated software remodularization solution that can be more effective from the structural quality point of view and the developer's perspective of software remodularization, the use of different types of structural, lexical, and changed history information in computation of class coupling is highly suggested. Many previous researchers have used the various dimensions of structural, lexical, and changed history information to compute the class coupling for software remodularization. The common issue in all the approaches is that they give equal importance to the different dimensions of the structural, lexical, and changed history

Table 7 Comparison of the proposed approach and the existing approaches in termsMoJoFM with respect to the authoritativeness

	PRAJ-2020	PRAD-2011	JALA-2019	PRAJ-2017	MKAO-2015	Proposed
PRAJ-2020	NA	[−−−−−]	[−−−−−]	[−−−−−]	[−−−−−]	[≈++++]
PRAD-2011	[+++++]	NA	[−≈+≈−]	[+++++]	[+++++]	[+++++]
JALA-2019	[+++++]	[+≈−≈+]	NA	[+++++]	[+++≈+]	[+++++]
PRAJ-2017	[+++++]	[−−−−−]	[−−−−−]	NA	[−−−−−]	[+++++]
MKAO-2015	[+++++]	[−−−−−]	[−−−≈−]	[+++++]	NA	[+++++]
Proposed	[≈−−−−]	[−−−−−]	[−−−−−]	[−−−−−]	[−−−−−]	NA

Table 8 Comparison of the proposed approach and the existing approaches in terms MoJoFM with respect to the developer's view

	PRAJ-2020	PRAD-2011	JALA-2019	PRAJ-2017	MKAO-2015	Proposed
PRAJ-2020	NA	[−−−−−]	[−−−−−]	[−−−−−]	[−−−−−]	[+≈+++]
PRAD-2011	[+++++]	NA	[−−≈−+]	[+++≈+]	[≈−+≈+]	[+++++]
JALA-2019	[+++++]	[++≈+−]	NA	[+++++]	[+−+++]	[+++++]
PRAJ-2017	[+++++]	[−−−≈−]	[−−−−−]	NA	[−−≈≈+]	[+++++]
MKAO-2015	[+++++]	[≈+−≈−]	[−+−−−]	[++≈≈−]	NA	[+++++]
Proposed	[−≈−−−]	[−−−−−]	[−−−−−]	[−−−−−]	[−−−−−]	NA

information in class coupling computation. However, it well known fact that the different dimensions of the structural, lexical, and changed history information would not contribute equally in computing the class coupling, i.e., each dimension of structural, lexical, and changed history information have their own importance in the contribution of class coupling computation.

To reveal the importance of different types of source code information such as structural, lexical, and changed history information as well as their different dimensions in computation of class coupling for the purpose of software remodularization, an empirical study with various variants of set of objective functions have been conducted.Each variant of the many-objective software remodularization exploited the different types and combinations of source code information for the computation of class coupling. The results presented in Sections 5.1 and 5.2 demonstrate that the use of different structural, lexical, and changed history information with their relative importance helps produce remodularization solutions that are highly acceptable to the software developers. On the other hand, the individual information (i.e., either structural, lexical, or changed-history) used to compute the class coupling cannot generate the remodularization solution that can be acceptable to the software developers. So, the results presented in this study support the assumption that the use of different dimensions of structural, lexical, and changed history information with their relative importance will help produce remodularization solutions that will be highly acceptable to the software developers. The comparative results can also validate this assumption. Different existing approaches use different types and dimensions of structural, lexical, and changed-history information with their equal importance in the computation of class coupling for software remodularization.

6 Threats to validity

The designed experimental setup used to evaluate the proposed approach is producing good results. However, there are many factors associated with the experimentation that can affect the proposed approach's outcome. Therefore, it becomes necessary for the empirical study to identify the possible threats that can affect the validity of the results and the actions required to mitigate their impact. In this study, we have identified the following four categories of threats and applied a suitable action to mitigate those types of threats.

Conclusion validity In conclusion validity, there must exist a causal relationship between the experimentation and treatment outcome. Our study's major threats that can affect the conclusion validity are: 1) initialization of population-the random initialization of the population may bias the results if it is favourable initialization. To mitigate this threat, we executed the algorithm many times with different random initialization. 2) use of a statistical approach to evaluate and compare the results obtained through much execution the inappropriate use of the statistical method can mislead the conclusion. To mitigate this threat, we used the non-parametric statistical test, i.e., Wilcoxon rank-sum test. The Wilcoxon rank-sum test produces a good result with data of non-normal distribution.

Internal validity The internal validity is concerned with the validity of the results corresponding to the various treatment factors. In our approach, the different parameter settings of the algorithm can affect the results. To mitigate this threat, we use the trial-and-error method to determine the values of the parameters of the algorithms.

Construct validity Construct validity is concerned with the relationship between theory and outcome. The output of the approach must be synchronized with the theory. The major threat to this validity is the cost of the evaluation. The different metaheuristic approaches may require a different cost to a particular iteration. To provide the equal cost of the assessment to each algorithm, we assigned the equal number of fitness evaluations to each algorithm instead of an equal number of iterations. The other threat can be the appropriateness of quality measures. In this approach, we use the well-accepted quality measure to evaluate the quality of the solutions.

External quality This validity is concerned with the generalization of the approach's outcome in the broader perspective of the problem instances. In this approach, we have considered the problem instances having different characteristics ranging from small to large. These set of problem instances have already been used by different researchers in the area of software remodularization.

7 Conclusion and future works

In this paper, we introduced a multi-dimensional information-driven many-objective search-based software engineering approach for modularizing the software system. The approach aims at generating the remodularization solution that improves the software quality by optimizing the different version of coupling and cohesion measures such as structural-based coupling and cohesion, lexical-based coupling and cohesion, and changed-history-based coupling and cohesion. To optimize the different quality metrics to produce the remodularization solution, we used the MaABC algorithm, a many-objective search-based software engineering approach with some effective changes in selection strategies. Specifically, in this study, we addressed several issues of existing software remodularization approaches that are restricted to the use of particular types of structural or lexical information-based cohesion and coupling. We created multiple variants of many-objective SRPs using the different sets of objective functions in this continuation. An extensive experiment for remodularization of five object-oriented systems is performed to validate the supremacy of the proposed contributions. The obtained results show that the proposed contributions can generate a more effective remodularization solution than traditional SBSR approaches. In future work, the proposed work can be integrated with the IDEs and configuration management system to automate remodularization recommendation system.

References

1. Mkaouer W, Kessentini M, Shaout A, Koligheu P, Bechikh S, Deb K, Ouni A. Many-objective software remodularization using NSGA-III.

ACM Transactions on Software Engineering and Methodology, 2015, 24(3): 17

2. Mancoridis S, Mitchell B S, Rorres C, Chen Y F R, Gansner E R. Using automatic clustering to produce high-level system organizations of source code. In: Proceedings of the 6th International Workshop on Program Comprehension. 1998, 45−52
3. Parashar A, Chhabra J K. Mining software change data stream to predict changeability of classes of object-oriented software system. Evolving Systems, 2016, 7(2): 117–128
4. Parashar A, Chhabra J K. Package-restructuring based on software change history. National Academy Science Letters, 2017, 40(1): 21–27
5. Anquetil N, Lethbridge T C. Experiments with clustering as a software remodularization method. In: Proceedings of the 6th Working Conference on Reverse Engineering. 1999, 235−255
6. Praditwong K, Harman M, Yao X. Software module clustering as a multi-objective search problem. IEEE Transactions on Software Engineering, 2011, 37(2): 264–282
7. Abdeen H, Ducasse S, Sahraoui H A, Alloui I. Automatic package coupling and cycle minimization. In: Proceedings of the 16th Working Conference on Reverse Engineering. 2009, 103−112
8. Mahdavi K, Harman M, Hierons R M. A multiple hill climbing approach to software module clustering. In: Proceedings of the International Conference on Software Maintenance. 2003, 315−324
9. Amarjeet, Chhabra J K. Harmony search based remodularization for object-oriented software systems. Computer Languages, Systems & Structures, 2017, 47: 153–169
10. Amarjeet, Chhabra J K. FP-ABC: fuzzy-Pareto dominance driven artificial bee colony algorithm for many-objective software module clustering. Computer Languages, Systems & Structures, 2018, 51: 1−21
11. Amarjeet, Chhabra J K. Many-objective artificial bee colony algorithm for large-scale software module clustering problem. Soft Computing, 2018, 22(19): 6341–6361
12. Marcus A, Poshyvanyk D, Ferenc R. Using the conceptual cohesion of classes for fault prediction in object-oriented systems. IEEE Transactions on Software Engineering, 2008, 34(2): 287–300
13. Bavota G, De Lucia A, Marcus A, Oliveto R. Software re-modularization based on structural and semantic metrics. In: Proceedings of the 17th Working Conference on Reverse Engineering. 2010, 195−204
14. Prajapati A, Parashar A, Chhabra J K. Restructuring object-oriented software systems using various aspects of class information. Arabian Journal for Science and Engineering, 2020, 45(12): 10433–10457
15. Doval D, Mancoridis S, Mitchell B S. Automatic clustering of software systems using a genetic algorithm. In: Proceedings of the 9th International Workshop Software Technology and Engineering Practice. 1999, 73−81
16. de Oliveira Barros M. An analysis of the effects of composite objectives in multiobjective software module clustering. In: Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation. 2012, 1205−1212
17. Kumari A C, Srinivas K. Hyper-heuristic approach for multi-objective software module clustering. Journal of Systems and Software, 2016, 117: 384–401
18. Prajapati A, Geem Z W. Harmony search-based approach for multi-objective software architecture reconstruction. Mathematics, 2020, 8: 1906
19. Kargar M, Isazadeh A, Izadkhah H. Semantic-based software clustering using hill climbing. In: Proceedings of the International Symposium on Computer Science and Software Engineering Conference. 2017, 55−60
20. Rathee A, Chhabra J K. Clustering for software remodularization by using structural, conceptual and evolutionary features. Journal of Universal Computer Science, 2018, 24(12): 1731–1757
21. Li W, Henry S. Object-oriented metrics that predict maintainability. Journal of Systems and Software, 1993, 23(2): 111–122
22. Chidamber S R, Kemerer C F. A metrics suite for object oriented design. IEEE Transactions on Software Engineering, 1994, 20(6): 476–493
23. Martin R. OO design quality metrics-an analysis of dependencies. In: Proceedings of the Workshop on Pragmatic and Theoretical Directions in Object-Oriented Software Metrics. 1994
24. Briand L, Devanbu P, Melo W. An investigation into coupling measures for C++. In: Proceedings of the 19th International Conference on Software Engineering. 1997, 412−421
25. Briand L C, Daly J W, Wüst J K. A unified framework for coupling measurement in object-oriented systems. IEEE Transactions on Software Engineering, 1999, 25(1): 91–121
26. Hitz M, Montazeri B. Measuring coupling and cohesion in object-oriented systems. In: Proceedings of the International Symposium on Applied Corporate Computing. 1995
27. Eder J, Kappel G, Schreft M. Coupling and cohesion in object-oriented systems. Klagenfurt: University of Klagenfurt, 1994
28. Lee Y S, Liang B, Wu S, Wang F. Measuring the coupling and cohesion of an object-oriented program based on information flow. In: Proceedings of the International Conference on Software Quality. 1995
29. Savić M, Ivanović M, Radovanović M. Analysis of high structural class coupling in object-oriented software systems. Computing, 2017, 99(11): 1055–1079
30. Wu Z, Palmer M. Verbs semantics and lexical selection. In: Proceedings of the 32nd Annual Meeting on Association for Computational Linguistics. 1994, 133−138
31. McLachlan G J, Krishnan T. The EM Algorithm and Extensions. 2nd ed. Hoboken: John Wiley & Sons, Inc., 2008
32. Zimmermann T, Weißgerber P, Diehl S, Zeller A. Mining version histories to guide software changes. IEEE Transactions on Software Engineering, 2005, 31(6): 429–445
33. Beck F, Diehl S. Evaluating the impact of software evolution on software clustering. In: Proceedings of the 17th Working Conference on Reverse Engineering. 2010, 99−108
34. Oliva G A, Santana F W S, Gerosa M A, de Souza C R B. Towards a classification of logical dependencies origins: a case study. In: Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution. 2011, 31−40
35. Beyer D, Noack A. Clustering software artifacts based on frequent common changes. In: Proceedings of the 13th International Workshop on Program Comprehension. 2005, 259−268
36. Fluri B. Assessing changeability by investigating the propagation of change types. In: Proceedings of the 29th International Conference on Software Engineering. 2007, 97−98
37. D'Ambros M, Lanza M, Robbes R. On the relationship between change coupling and software defects. In: Proceedings of the 16th Working Conference on Reverse Engineering. 2009, 135−144
38. Sun X, Li B, Zhang Q. A change proposal driven approach for changeability assessment using FCA-based impact analysis. In: Proceedings of the 36th International Conference on Computer Software and Applications. 2012, 328−333
39. Saxena D K, Duro J A, Tiwari A, Deb K, Zhang Q. Objective reduction in many-objective optimization: linear and nonlinear algorithms. IEEE Transactions on Evolutionary Computation, 2013, 17(1): 77–99
40. Yuan Y, Xu H, Wang B, Yao X. A new dominance relation-based evolutionary algorithm for many-objective optimization. IEEE Transactions on Evolutionary Computation, 2016, 20(1): 16–37
41. Yang S, Li M, Liu X, Zheng J. A grid-based evolutionary algorithm for many-objective optimization. IEEE Transactions on Evolutionary

Computation, 2013, 17(5): 721–736
42. Díaz-Manríquez A, Toscano-Pulido G, Coello C A C, Landa-Becerra R. A ranking method based on the R2 indicator for many-objective optimization. In: Proceedings of the IEEE Congress on Evolutionary Computation. 2013, 1523−1530
43. Zitzler E, Künzli S. Indicator-based selection in multiobjective search. In: Proceedings of the 8th International Conference on Parallel Problem Solving from Nature. 2004, 832−842
44. Wang H, Jiao L, Yao X. Two_Arch2: an improved two-archive algorithm for many-objective optimization. IEEE Transactions on Evolutionary Computation, 2015, 19(4): 524–541
45. Rachmawati L, Srinivasan D. Multiobjective evolutionary algorithm with controllable focus on the knees of the Pareto front. IEEE Transactions on Evolutionary Computation, 2009, 13(4): 810–824
46. Rachmawati L, Srinivasan D. Preference incorporation in multi-objective evolutionary algorithms: a survey. In: Proceedings of the IEEE International Conference on Evolutionary Computation. 2006, 962−968
47. Zhang Q, Li H. MOEA/D: a multiobjective evolutionary algorithm based on decomposition. IEEE Transactions on Evolutionary Computation, 2007, 11(6): 712–731
48. Yuan Y, Xu H. Multiobjective flexible job shop scheduling using memetic algorithms. IEEE Transactions on Automation Science and Engineering, 2015, 12(1): 336–353
49. Lygoe R J, Cary M, Fleming P J. A real-world application of a many-objective optimisation complexity reduction process. In: Proceedings of the 7th International Conference on Evolutionary Multi-Criterion Optimization. 2013, 641−655
50. Narukawa K, Rodemann T. Examining the performance of evolutionary many-objective optimization algorithms on a real-world application. In: Proceedings of the 6th International Conference on Genetic and Evolutionary Computing. 2012, 316−319
51. Harman M, Jones B F. Search-based software engineering. Information and Software Technology, 2001, 43(14): 833–839
52. Karaboga D. An idea based on honey bee swarm for numerical optimization. Kayseri: Erciyes University, 2005
53. Li K, Chen R, Fu G, Yao Z. Two-archive evolutionary algorithm for constrained multiobjective optimization. IEEE Transactions on Evolutionary Computation, 2019, 23(2): 303–315
54. Prajapati A. Two-archive fuzzy-Pareto-dominance swarm optimization for many-objective software architecture reconstruction. Arabian Journal for Science and Engineering, 2021, 46(4): 3503–3518
55. Andritsos P, Tzerpos V. Information-theoretic software clustering. IEEE Transactions on Software Engineering, 2005, 31(2): 150–165
56. Jalali N S, Izadkhah H, Lotfi S. Multi-objective search-based software modularization: structural and non-structural features. Soft Computing, 2019, 23(21): 11141–11165

Amarjeet Prajapati is currently working as Assistant Professor in Department of Computer Science Engineering & Information Technology at Jaypee Institute of Information Technology, India. He received his PhD degree from National Institute of Technology, India. He has published many research papers in reputed journals and conferences. His area of interest includes software engineering, metaheuristic algorithms, machine learning, and soft computing.

Anshu Parashar is currently working as Assistant Professor in Department of Computer Science & Engineering at Thapar Institute of Engineering & Technology (TIET), India. He received his PhD degree from National Institute of Technology, India. He has published many research papers papers in reputed journals and various national and international conferences. His area of interest includes software engineering, machine learning, blockchain and cloud computing for sustainable development.

Amit Rathee is currently working as Assistant Professor in the Department of Computer Science, Government College Barota, India. He was awarded a PhD degree from National Institute of Technology, India in 2020. He has presented and published many research papers in reputed journals and various national and international conferences. His research interests include software engineering, human-computer interaction, soft computing, algorithm, and artificial intelligence.