DE GRUYTER J. Intell. Syst. 2018; 27(4): 619–641

Amarjeet* and Jitender Kumar Chhabra

TA-ABC: Two-Archive Artificial Bee Colony for Multi-objective Software Module Clustering Problem

DOI 10.1515/jisys-2016-0253
Received October 20, 2016; previously published online May 4, 2017.

Abstract: Multi-objective software module clustering problem (M-SMCP) aims to automatically produce clustering solutions that optimize multiple conflicting clustering criteria simultaneously. Multi-objective evolutionary algorithms (MOEAs) have been a most appropriate alternate for solving M-SMCPs. Recently, it has been observed that the performance of MOEAs based on Pareto dominance selection technique degrades with multi-objective optimization problem having more than three objective functions. To alleviate this issue for M-SMCPs containing more than three objective functions, we propose a two-archive based artificial bee colony (TA-ABC) algorithm. For this contribution, a two-archive concept has been incorporated in the TA-ABC algorithm. Additionally, an improved indicator-based selection method is used instead of Pareto dominance selection technique. To validate the performance of TA-ABC, an empirical study is conducted with two well-known M-SMCPs, i.e. equal-size cluster approach and maximizing cluster approach, each containing five objective functions. The clustering result produced by TA-ABC is compared with existing genetic based two-archive algorithm (TAA) and non-dominated sorting genetic algorithm II (NSGA-II) over seven un-weighted and 10 weighted practical problems. The comparison results show that the proposed TA-ABC outperforms significantly TAA and NSGA-II in terms of modularization quality, coupling, cohesion, Pareto optimality, inverted generational distance, hypervolume, and spread performance metrics.

Keywords: Artificial bee colony, multi-objective optimization, software module clustering, two-archive algorithm.

1 Introduction

The highly used commercial software systems frequently need to be modified in response to demand for change in customer, business, and technological requirements. These changes normally have to be performed in short deadlines and within limited budget. Hence, developers generally modify the systems without considering the design guidelines of the original software system. Such maintenance practices often degrade structural design of software systems [30]. For a software system, it becomes a very complicated task to make further changes in those whose structure quality deteriorated to the point where it is difficult to understand [34]. The modular structure with low cohesion and high coupling is one of the main reasons of structural design deterioration. The software module clustering technique, in which the software entities are organized into disjoint sets of cluster according to predefined criteria [43], is one of the successful methods to improve the modular structure of the complicated software systems.

The software module clustering process takes software modules with their dependency as an input and partitions the modules into several disjoint sets of clusters based on predefined rules so that the software become more understandable and maintainable [43]. The predefined rules can be various software structural

---

***Corresponding author: Amarjeet,** Department of Computer Engineering, NIT Kurukshetra, Haryana, India, e-mail: amarjeetnitkkr@gmail.com
Jitender Kumar Chhabra: Department of Computer Engineering, NIT Kurukshetra, Haryana, India

design criteria [7, 8, 20] such as minimum coupling, maximum cohesion, etc. The decomposition of software modules into clusters based on some structural design criteria is defined as the software module clustering problem (SMCP) [15, 32, 34, 35, 43]. Many software module clustering approaches in research literature have been proposed to address the SMCPs [15, 36, 43, 46].

These approaches can be broadly divided into two main groups: (1) search-based software module clustering and (2) non-search-based software module clustering approach. In search-based approaches, a problem is transformed as a search-based optimization problem and solved using search-based meta-heuristic algorithms (e.g. genetic algorithms, GA, USA) [16, 45], while in non-search-based approach the problem is solved using deterministic algorithms (e.g. hierarchical clustering). The software partitioning problem (i.e. SMCP) is a class of an non-deterministic polynomial-time-hard (NP-hard) problem [43]; hence, deterministic algorithm cannot be a good alternate because exponential time is needed to solve it. This observation provides the motivation for the use of search-based meta-heuristics in solving the SMCPs. The meta-heuristic approaches do not ensure the generation of optimal solution since these approaches evaluate only a part of the feasible search space, but try to search the different part in the search space in an effective way to get a near-optimal solution in reasonable computation time and cost [41].

Mostly, search-based approaches first transform the software systems into a module dependency graph (MDG) [35] and then solve the SMCPs as a graph partitioning problem [34, 38, 43]. MDGs are directed graphs in which vertex and edges represent modules and their relationships, respectively. Based on the number of optimization criteria, SMCPs can be designated as a single-objective software module clustering problems (S-SMCPs) or multi-objective software module clustering problems (M-SMCPs) and can be formulated as single- or multi-objective optimization problem. The S-SMCPs have a single solution which optimizes single software quality criteria, while M-SMCPs have many solutions which optimize simultaneously more than one software quality criteria.

Most researchers [1, 2, 15, 32, 35, 37, 38] have formulated the SMCPs as a single-objective optimization problem and solved using different single-objective meta-heuristic algorithms [e.g. hill-climbing (HC) and GA]. The main limitation of the single-objective optimization approach is that it optimizes a single criterion and generates only a single solution at each run. Thus, little information can be provided to the decision makers about different aspects of the quality criterion. So formulation of SMCP as a multi-objective optimization problem, and solving using multi-objective evolutionary approach has recently become more practically useful.

The current search-based multi-objective software module clustering approaches [5, 28, 29, 43] have been applied successfully to solve the M-SMCPs. Most of the multi-objective software modules clustering approaches for M-SMCP problems are genetic-based multi-objective evolutionary algorithms (MOEAs) (e.g. [4, 5, 29, 43]). Even though these approaches have shown many advantages in solving the M-SMCPs, still there are some challenges corresponding to characteristics of MOEAs such as uncertainty, conflicting attribute, large number of quality criteria, etc that need to be addressed. In this paper, we address the following two main challenges:

- The SMCP is naturally a multi-objective optimization problem; hence, MOEA has to solve it by optimizing multiple objective functions simultaneously. However, the performance of MOEAs, for instance, non-dominated sorting genetic algorithm II (NSGA-II) [13], gets deteriorated if the number of objectives functions increases by more than three.
- Most of the MOEAs contain many control parameters that require to be set by users according to domain knowledge/experience and problem characteristics to achieve a satisfactory performance. However, it is the most challenging task for SMCPs because there are diverse categories of software problems and identifying control parameters for every different problem is very difficult and time consuming.

To address the above challenges, there is growing need for MOEAs that can solve M-SMCPs efficiently with a large number of objective functions. Recent works [12, 19, 40, 47, 48] in optimization literature have proposed several algorithms to address such kind of optimization problems using, for instance, preference based [19], objective reduction [40], reference based [12], decomposition based [47], and indicator based [48] approaches.

However, to the best of our knowledge, these techniques have not yet been explored to solve the M-SMCPs. We propose a multi-objective software module clustering technique by integrating the two-archive [44] and indicator based selection [48] concepts into the original artificial bee colony (ABC) [26].

To assess the effectiveness of the proposed approach, we applied it over seven un-weighted and 10 weighted open software projects. We report the results of our proposed approach and compared it with existing MOEAs [two-archive algorithm (TAA) and NSGA-II] that have been used to solve the M-SMCPs by the previous researchers [5, 43]. The results indicate that our proposed approach significantly outperforms TAA and NSGA-II based approaches, in terms of modularization quality (MQ) [43], coupling [43], cohesion [43], Pareto optimality [43], inverted generational distance (IGD) [50], hypervolume (HV) [49], and spread performance [13] metrics.

The rest of this paper is organized as follows: Section 2 presents related research works. Section 3 briefly describes the SMCPs and problem formulation. Section 4 gives a short description of two-archive evolutionary algorithm and detailed description of two-archive based artificial bee colony (TA-ABC). Section 5 presents details of the experimental setup. Section 6 presents the results and compares them to the best performing algorithms from the existing literature to demonstrate the superiority of the TA-ABC algorithm. Section 7 discusses the implications of the results. Section 8 provides threats to validity. Finally, Section 9 gives the concluding remarks and future research directions.

2 Related Works

It is a commonly accepted fact that a software system comprising well-modularized structure is easier to design, develop, test, maintain, and evolve [6, 9]. However, maintaining a large software system becomes difficult, especially if their modular structure degrades and is not documented well [15, 24]. To address the SMCPs, various search-based and deterministic approaches have been proposed in the literature [3, 29, 36, 38, 43]. Our approach formulates the SMCPs as search-based M-SMCP and solves using multi-objective two-archive ABC algorithm; hence, related works are centered on search-based software module clustering literature.

Mancoridis et al. [35] were the first who proposed the search-based optimization approach to address the SMCPs. The work [35] formulated the characteristics of well-modularized software as objective functions; the evaluation of these objective functions directs the optimization process towards good clustering. Further, the works have developed an automated tool named Bunch for clustering the software systems. Following this search-based optimization concept for SMCPs, many other search-based optimization methods have been designed in previous works, such as GA, HC algorithm, simulated annealing (SA), and so forth [15, 21, 29, 32, 33, 37, 38, 43].

The work of Mancoridis et al. [34] applied the Bunch tool for the maintenance and architecture recovery of software systems. To guide the process of searching, the authors designed an objective function, namely, MQ. The MQ is a trade-off between interconnectivity and intraconnectivity and has been integrated into the Bunch tool [34]. Using the Bunch tool with a set of meta-heuristic clustering algorithms (GA, HC, and SA), the software system is partitioned as subsystems at a high level. The clustering results process resulted in software which possesses the best quality in terms of grouping and turns out to be effective in a medium as well as for large systems.

Several studies [15, 21, 38] have demonstrated that for module clustering the HC algorithm has outperformed the standard search techniques such as GA and SA in terms of both the execution time and solution quality. However, it is well known that the HC algorithm suffers from the problem of early convergence to local optima [32]. To overcome this problem, the authors [32] proposed a multiple HC approach to address the software module clustering.

Praditwong [42] proposed two evolutionary algorithms based on GA, namely, Grouping Genetic Algorithm (GGA) and Group Number Encoding (GNE), to solve SMCPs. The author also performed a comparative study of these two genetic-based algorithms over various real-world SMCPs in terms of the MQ quality measure. The results demonstrated that GGA produced high-quality solutions compared to the GNE approach. Further, the same authors [43] formulated the SMCP, a multi-objective search problem [namely, maximizing cluster approach (MCA) and equal-size cluster approach (ECA)] and use two-archive evolutionary algorithms [44].

Even after formulation of SMCP as a search-based optimization problem, some other widely used search-based algorithms (e.g. ABC [26] and Grey Wolf Algorithm [27]) have not gained much attention. The ABC algorithm has been demonstrated to be effective and well-situated to solve various optimization problems in the field of science and engineering [11, 23, 25, 31]. Recently, Dahiya et al. [11] demonstrated the applicability of ABC in software testing; however, the applicability and usefulness of the ABC algorithm have not been studied by any researcher till date to solve the SMCPs. This paper formulates SMCP as search-based multi-objective optimization problem and solves using ABC meta-heuristic algorithm.

3 Software Module Clustering Problems

SMCP is a problem of automatically grouping software modules into disjoint sets of clusters to improve software design structure [43]. The SMCPs is basically a graph partitioning problem which is a class of NP-hard problem [17, 43]. The SMCPs can be represented as a MDG which is defined as a graph $G=(V, E)$, where V represents the set of modules and E is the set of relationships between modules. All modules need to be partitioned into k non-overlapping clusters $C_1, C_2, \ldots, C_k$; that is, $C_1 \cup C_2 \cup \cdots \cup C_k = V$, $C_i \neq \emptyset$ and $C_i \cap C_j = \emptyset$, $i, j = 1, 2, \in k$, and $i \neq j$. A good partitioning of the MDG is regarded as a partition with minimum interconnection and maximum intraconnection. The number of ways to partition an MDG containing a set of n vertex into k nonempty clusters can be computed by using the Stirling numbers of the second kind, $S(n, k)$ [22]. The searching for an optimal partition from an MDG becomes problematic as the number of modules increases. To solve such class of problems using deterministic or exhaustive methods requires very high computing time; hence, formulation of SMCPs as a search-based optimization problem is the best alternative to find a near-optimal solution. The search-based SMCPs can be formulated as single-objective or multi-objective optimization problem. The brief description of multi-objective optimization formulation for SMCP is given in the following subsection.

3.1 Multi-objective Formulation

In multi-objective SMCP, more than one and less than or equal to three objectives are optimized. It determines a clustering solution x^* for which

$$f(x^*) = \begin{cases} \min \left[f_1(x), f_2(x), \ldots, f_M(x) \right]^T & M \geq 2 \\ \mathrm{g}_{\mathrm{j}}(x) \geq 0 & j = 1, \ldots, P \\ \mathrm{h}_{\mathrm{k}}(x) = 0 & k = 1, \ldots, Q \\ x_i^L \leq x_i \leq x_i^U & i = 1, \ldots, n \end{cases} \tag{1}$$

where M and f_i represent the number of objective functions and i^{th} objective function, respectively. Q is the number of equality constraints; P is the number of inequality constraints, and x_i^U and x_i^L represent the upper and lower bounds of the decision variable x_i.

3.2 Module Clustering Objective Functions

The main goal of software module clustering is to improve the quality of clustering by optimizing various conflicting software attributes. Praditwong et al. [43] have proposed two multi-objective formulations (i.e. ECA and MCA) that capture attributes of a well-clustered software system. Moreover, these formulations also help in guiding the optimization process towards good clustering. The objective functions defined under MCA and ECA formulations are as follows: (1) maximization of cohesion (i.e. sum of intracluster edges), (2) minimization of coupling (i.e. sum of intercluster edges), (3) minimization of number of clusters, (4) maximization of MQ, (5) minimization of the number of isolated clusters, and (6) minimization of the differences between maximum size cluster and minimum size cluster.

The MCA formulation includes the objective function numbered with 1, 2, 3, 4, and 5, and ECA formulation includes the objective function numbered with 1, 2, 3, 4, and 6. The computations of all identified objective functions except the MQ objective are straightforward. The computation of MQ is defined as follows:

$$\mathrm{MQ} = \sum_{k=1}^{m} MF_k \quad \text{where } MF_k = \begin{cases} 0, & \text{if } i = 0 \\ \dfrac{i}{i + \frac{1}{2} j}, & \text{if } i > 0 \end{cases} \tag{2}$$

where *i* is the number of intracluster edges and *j* is that of intercluster edges of cluster *k* for an un-weighted MDG, while for weighted MDG, *i* represents the total weight of intracluster edges and *j* represents total weight of intercluster edges of cluster *k*.

4 Two-Archive based Artificial Bee Colony

The basic ABC algorithm [26] was designed to solve the single-objective continuous optimization problem. However, the software module clustering is a natural multi-objective optimization problem where various conflicting quality criteria need to be optimized to obtain a good quality software structure. The M-SMCP can be designed as S-SMCP by aggregating all objective functions into a single objective function and further can be solved using the single-objective ABC algorithm. However, such formulation has the following shortcoming: as the population evolves, all individual solutions suffer earlier convergence to the local optima in very few generations. This may lead single-objective ABC algorithm towards production of the population with low diversity in successive generations [37]. Hence, for complex M-SMCPs, we propose TA-ABC algorithm adapting the concepts of two-archive approach which can produce a good clustering solution with good convergence, satisfactory diversity, and acceptable complexity.

4.1 The Basic Concept of ABC Algorithm

The ABC, a meta-heuristic algorithm based on the behavior of bees, has gained wide attention and has been demonstrated to be effective and well situated for solving the various types of optimization problems in science and engineering fields [11, 23, 25, 31]. The main steps of the basic ABC algorithm are as follows:

- **Population initialization phase:** The initial population of the basic ABC algorithm is generated by a random process. Let $v_i = \{v_{i1}, v_{i2}, \ldots, v_{in}\}$ represent the i^{th} food source in the population with *n* number of decision variables. To initialize the population, each food source is generated as follows:

  $$v_{ij} = v_{ij}^{\min} + (v_{ij}^{\max} - v_{ij}^{\min}) \times r, \ \ j = 1, \ldots, n; \ \ i = 1, \ldots, SN, \tag{3}$$

  where $v_{ij}^{\max}$ and $v_{ij}^{\min}$ represent the upper and lower bounds for the decision variable *j*, respectively, and *r* is used a uniform random number in [0, 1].
- **Employed bee phase:** In the employed bee phase, the i^{th} food source v_{ij} of the population is assigned to the i^{th} employed bee, which generates a new neighboring solution around the assigned food source as follows.

  $$v_{\text{new}\,j} = v_{ij} + U(-1,1) \times (v_{ij} - v_{kj}) \tag{4}$$

Algorithm 1: The ABC Algorithm [26].

```
Input- Parameters values;
    $N_{FS}$: Population size (i.e. number of food source);
    $N_{IC}$: Number of iterations;
    $N_{LMT}$: Maximum number of trials;
Output- Optimal solution;
begin
  for $i = 1$ to $N_{FS}$ do    //Generation of food sources for initial population;
      $FS_i \leftarrow$ generate food source $i$ using Eq. (7);
      $f_i \leftarrow f(FS_i)$    //calculate fitness function of food source $i$;
      $Tr(i) \leftarrow 0$    //initialize trial to zero;
  $Itr \leftarrow 1$;    //initialize iteration to one
while $Itr < N_{IC}$ do
  for $i = 1$ to $N_{FS}$ do    //Employee bee phase;
      $CS_i \leftarrow$ generate a candidate solution using Eq. (8)
      $f(CS_i) \leftarrow$ evaluate fitness function of candidate solution
      if $f(CS_i) < f(FS_i)$ then    //greedy selection;
          $FS_i \leftarrow CS_i$
          $f(FS_i) \leftarrow f(CS_i)$
          $Tr(i) \leftarrow 0$;
      else $Tr(i) \leftarrow Tr(i) + 1$;
Calculate each onlooker's bee probability using Eq. (9);
$i \leftarrow 1$, $t \leftarrow 0$;    //Onlooker bee phase;
while $t < N_{FS}$ do
r ← rand(0, 1);    //random generation;
if $r < pi$ then
    $t \leftarrow t + 1$;
    $CS_i \leftarrow$ a candidate solution by Eq. (3);
    $f(CS_i) \leftarrow$ evaluate candidate solution;
if $f(CS_i) < f(FS_i)$ then    //greedy selection;
    $FS_i \leftarrow CS_i$
    $f(FS_i) \leftarrow f(CS_i)$
    $Tr(i) \leftarrow 0$;
else
    $Tr(i) \leftarrow Tr(i) + 1$;
    $i \leftarrow (i + 1) mod\ N_{FS}$
//Scout bee phase;
$ind = \{i: Tr(i) = max(Tr)\}$;
if $Tr\ (ind) > N_{LMT}$ then
    $FS_{ind} \leftarrow$ random solution by Eq. (7);
    $f_{ind} \leftarrow f(FS_{ind})$;
    $Tr(ind) \leftarrow 0$
$Itr \leftarrow Itr + 1$
```

where $i \in \{1, \ldots, SN\}$, and $k \in \{1, \ldots, SN\} \wedge k \neq i$ is a randomly chosen food source. After generating new solution v_{new} it is evaluated and compared to v_i then the solution with the higher fitness value is selected.

- **Onlooker bee phase:** The onlooker bees make a decision on food sources whether to select or not the food source selected by the employed bees. To perform this, the onlooker bees use the probability values, calculated using Eq. (9), to select the food source for discovering promising regions in the search space.

$$p_i = \frac{\text{fit}_i}{\sum_{i=n}^{SN} \text{fit}_i} \tag{5}$$

where fit_i is the fitness value of the i^{th} food source.

- **Scout bee phase:** If a food source cannot be further improved through a limited iteration, then the food source is supposed to be abandoned and a randomly produced food source will be replaced with it.

The above basic ABC algorithm was designed to solve the single-objective optimization problems that have the continuous decision variables; hence, the original form of the algorithm cannot be directly used for solving the combinatorial/discrete multi-objective optimization problems. The M-SMCP is a discrete multi-objective optimization problem; therefore, in this work, some alterations to the basic ABC algorithm have been done for making it suitable for the M-SMCPs.

4.2 Proposed TA-ABC Algorithm

This section presents a TA-ABC approach to solve M-SMCPs. The proposed TA-ABC has the following main features:

- The approach can work efficiently for more than three objective functions.
- The approach provides a good balance between exploration and exploitation.
- The proposed approach can be easily implemented.

To impart the above features, the TA-ABC algorithm exploits the concepts of two-archive technique [44] and indicator based ranking [48]. The combination of both these concepts makes TA-ABC algorithm perform

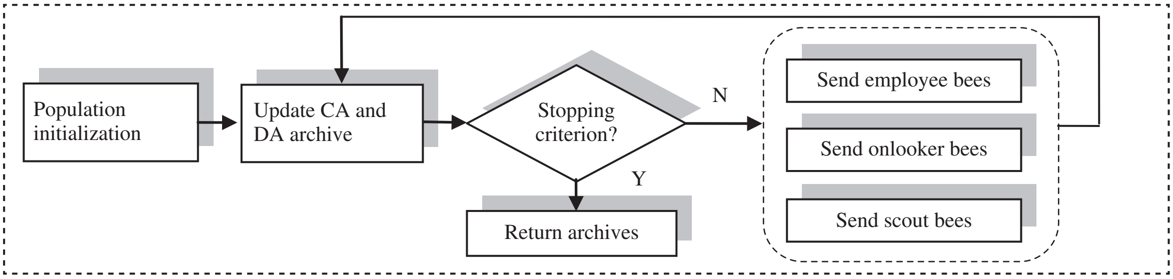

Figure 1: Flow Chart of Proposed TA-ABC Algorithm.

efficiently in case of more than three objective functions. On the other hand, use of ABC algorithm concepts makes the approach free from many parameters and perform good exploitation and exploration of the search space. The flow chart of TA-ABC is given in Figure 1.

The working of TA-ABC method is divided into six parts: food source representation, Population initialization, Send employed bees, Send onlooker bees, Send scout bees, and Update the archive. The detailed explanations of these parts are provided in the subsequent subsections.

4.2.1 Food Source Representation

To solve M-SMCPs with the TA-ABC algorithm, its solution requires to be modeled in a proper way, so as it can be solved efficiently. In the search-based techniques, the solution is encoded as a string of (typically binary) numbers. In our TA-ABC approach, each module clustering solution is encoded as a string of integer numbers instead of binary numbers. In the integer encoding, a single integer perturbation can separate a module clustering solution into two distinct module clustering solutions, while binary representation requires a large number of perturbations. Hence, in the integer encoding, individual module clustering solutions are a smaller distance from one another, which significantly increases the power of exploration and exploitation [10].

Let $\{m_1, m_2, \dots m_n\}$ be the set of n number of modules in the software system. Then the solution is represented as a vector of $n_m = n$ integers ($m = [m_1, m_2, \dots m_n]$). In this representation, the value $0 < m_i \leq n$ of the i^{th} module indicates the cluster to which the i^{th} module is assigned. A clustering solution with the same value for all the modules means that all modules are grouped in the same cluster, while a clustering solution containing all possible values (from 1 to n) denotes that each cluster holds only a single module. To demonstrate it, let us consider a hypothetical software system depicted in Figure 2.

In Figure 2 the clustering solution (i.e. food source) of software system contains eight modules (i.e. numbered with 1–8) distributed in three clusters, namely, C_1, C_2, and C_3. Hence, it can be represented as a vector $C = [1, 1, 2, 2, 2, 3, 3, 3]$, where modules 1 and 2 are in cluster C_1, modules 3, 4, and 5 are in cluster C_2, and modules 6, 7, and 8 are in cluster C_3.

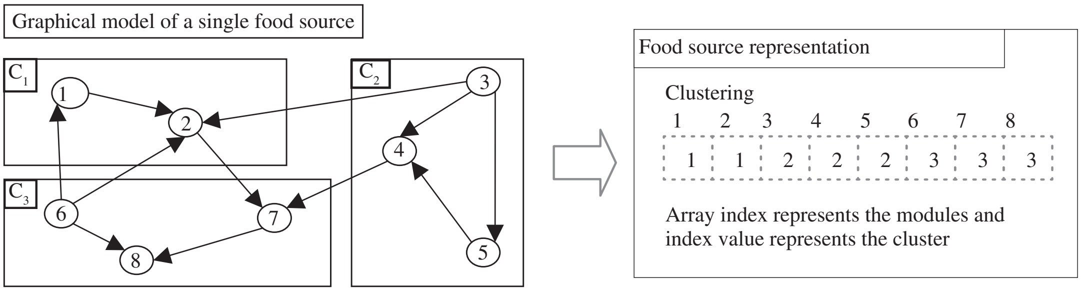
Graphical model of a single food source
C1
C2
C3
1
2
3
4
5
6
7
8
Food source representation
Clustering
1 2 3 4 5 6 7 8
1 1 2 2 2 3 3 3
Array index represents the modules and index value represents the cluster

Figure 2: Representation of a Simple Food Source (i.e. Software Clustering Solution).

Algorithm 2: Pseudo-Code of the Initialization of Food Sources.

```
TA-ABC (Dataset, CA, DA, FoodNumber, MaxTrial)
Generate food sources $c=(c_1, c_2, \ldots, c_{FoodNumber})$ randomly
  For $i=1$ to FoodNumber
      For $d=1$ to $D$       /* $D$ represents the dimension (i.e. total number of classes) */
        $c_i^d \leftarrow$ RandInt ($UB^d - LB^d$))       /* LB = 1 and UB = Total number of classes */
      End For
  End For
Calculate each objective function of $c_i$ food source based on considered multi-objective formulation
Initiate $Trial_1, Trial_2, \ldots, Trial_{FoodNumber}$ by 0
Update the External Archives CA and DA
```

4.2.2 Population Initialization

The TA-ABC algorithm receives the population size (PS), MaxTrial, the number of dimensions (D), the number of scouts (Scouts), and the two external archives, namely, convergence archive (CA) and divergence archive (DA), each with variable size and constant total size equal to PS. The number of food sources (clustering solutions) is set as equal to PS. After initialization of basic parameters, the initial food sources are generated randomly, and their nectar amount (clustering fitness function) is determined. In multi-objective approach, instead of finding a single solution, a set of non-dominated solutions are collected. For this, non-dominated food sources are collected and stored in the two external archives CA and DA according to their updating rules (details are given in subsection 4.2.6). Algorithm 2 provides pseudo-code of population initialization.

The RandInt ($UB^d - LB^d$)) generates a random number selected from a normal distribution in the range of 1 [i.e. Lower Bound (LB)] and number of classes [i.e. Upper Bound (UB)], and UB^d and LB^d are upper and lower bounds along the d^{th} dimension, respectively.

4.2.3 Send Employed Bees

Algorithm 3 presents the pseudo-code of Send Employed Bee module of the TA-ABC algorithm. After random initialization of the food source (Population initialization), the employed bees are sent to search new food sources. For this, the employed bees use the history information stored in combined $|CA+DA|$ archives. The main reason for using the external archive solutions is that it contains the best solutions found so far by the employed bees, and it may guide them towards better possible food sources. The main steps of the working process of employed bees are as follows: (1) Each of the employed bees searches a new food source with the help of food source stored in archives (Lines 1–4). (2) If the newly discovered food source is not the old food source, then the new food source is computed with the old food source using domination rank approach (Lines 5–6). (3) If the new food source dominates the old food source, then it replaces the old food source; otherwise, the old food source remains in the population, and its trial value is incremented by 1 (Lines 7–13).

4.2.4 Send Onlooker Bees

Algorithm 4 presents the pseudo-code of the Send Onlooker Bees module. In the send employed bee module, all the employed bees search optimal food source using the information provided by the CA and DA external archives. After searching the optimal food source, all employed bees come to the hive and share their information about the newly discovered food source with onlooker bees waiting in the hive. The onlooker bees collect the information provided by the employed bee regarding the food sources. Based on the collected information, each onlooker bee needs to make a decision process for the selection of food sources. To perform

Algorithm 3: Pseudo-Code of the Send Employed Bees.

```
For $i=1$ to FoodNumber
    Select a random component $d$, $d \in \{1, 2,\dots, D\}$ from food source $c_i$,
    Select a random food source $k$ from archive $|CA+DA|$, $k \neq i \in \{1, 2,\dots,(|CN+DA|)\}$,
        $v_i^d = x_k^d$,
        If $v_i \neq c_i$, then
            Calculate the objective functions of new food source: $v_i$
            If the new food source $v_i^d$ dominates old food source $c_i$
                Replace old food source $c_i$ with new food source $v_i$
            Else
                Increment $Trial_i$ by 1
            End If
    End If
End For
```

this, the onlooker bees compute the selection probability p_i of each food source c_i using Eq. (10) for each food source provided by the corresponding employed bee.

$$p_i = \frac{\mathrm{fit}(c_i)}{\sum_{m=1}^{\mathrm{Food\,Number}} \mathrm{fit}(c_m)} \tag{6}$$

The selection probability p_i is the probability of the food source provided by the employed bee i which is proportional to the fitness of food source. To calculate the fitness of a food source advertised by employed bees, we use the quality indicator $I_{\varepsilon+}$ given in IBEA [48]. $I_{\varepsilon+}$ is an indicator that calculates the minimum distance that one food source (i.e. solution) requires in order to dominate other food sources in the objective space. The value of $I_{\varepsilon+}$ between two solutions c_1 and c_2 is computed as follows:

$$I_{\varepsilon+}(c_1,c_2)=\min_{\varepsilon}(f_i(c_1))-\varepsilon \le f_i(c_2),\ 1\le i\le m) \tag{7}$$

where m is the number of objectives. Using Eq. (7), we assign the fitness to each solution according to the following equation.

Algorithm 4: Pseudo-Code of the Send Onlooker Bees.

```
Calculate probability value $p_i$ of each food source $C_i$ based On Eq. (10)
For $I=1$ to FoodNumber
    If rand$<p_i$, Then        /* Select $c_i$ employed bee to follow */
        Select a random component $d$, $d \in \{1, 2,\dots, D\}$ from food source $c_i$,
        Select a random food source $k$ from archive $|CA+DA|$, $k \neq i \in \{1, 2,\dots,(|CA+DA|)\}$,
            $v_i^d = x_k^d$,
            If $v_i \neq c_i$, Then
                Calculate the objective functions of new food source: $v_i$
                If the new food source $v_i^d$ dominates old food source $c_i$
                    Replace old food source $C_i$ with new food source $v_i$
                Else
                    Increment $Trial_i$ by 1
                End If
            End If
    End If
If $i>$FoodNumber, Then $i=1$        /* Reset the value of $i$ */
End For
```

$$\mathrm{fit}(c_i) = \sum_{c_j \in P \setminus \{x_i\}} -e^{-I_{\varepsilon+}(c_j, c_i)/0.05} \tag{8}$$

After computing the selection probability, the onlooker bees use the greedy technique to select a food source advertised by the employed bee. Further, each onlooker bee selects a food source from archive member randomly and performs the same steps as an employed bee has performed to update their current food source.

4.2.5 Send Scout Bees

At each cycle of the algorithm, the employed and onlooker bees search new food source around each old food source and evaluate them; if the old food source cannot be improved after a certain number of iterations, called *MaxTrial*, then the old food source is abandoned. In Send scout bee module, the algorithm sends scout bees for each abandoned food sources; the scout bees randomly search a new food source and replaces the abandoned food source if the newly generated food source dominates it. Otherwise, the old food source is kept in the population.

4.2.6 Update CA and DA Archives

To guide the employed and onlooker bees in a good direction, the TA-ABC algorithm uses the two external archives concepts inspired by the work presented in [44] to store the non-dominated solutions. These archives are convergence archive (CA) and diversity archive (DA) with variable size; however, the total size is fixed. Both CA and DA archives are updated as follows: (1) the algorithm first selects the non-dominated solutions from the population. (2) The selected non-dominated solutions are compared to the solutions stored in the CA and DA archives. (3) If the non-dominated solution is not dominated by any solution stored in CA or DA archive, then discard the solution (4). If the solution dominates any solution stored in CA or DA archives, then the dominated solution stored in CA and DA are removed. (5) If the solution is non-dominated with any solution stored in CA or DA archives, then add the solution in CA. (6) Finally, if the number of non-dominated solutions of both archive increases the total size of CA and DA, then delete the extra solutions from DA archive which have the minimal Euclidean distances to CA archive.

4.2.7 Termination

Each of the four modules (i.e. Send employed bee, Send onlooker bee, Send scout bee, and Update archive) of TA-ABC iterate cycle by cycle until the specified termination condition is reached. At the end of TA-ABC algorithm termination, the solutions stored in both CA and DA archives are returned as the output. In our implementation, the TA-ABC terminates after a predefined number of function evaluations same as in TAA and NSGA-II that have been used to solve M-SMCPs [43].

Algorithm 5: Pseudo-Code of the Send Scout Bees.

```
If there exists some $c_i$ | {$\mathrm{trial}_i > t$},
    Select one such $c_i$ randomly,
    For each component $d$, $d \in \{1, 2, \ldots, D\}$,
        $v_i^j = \leftarrow$RandInt ($\mathrm{UB}^d - \mathrm{LB}^d$))    /* LB = 1 and UB = Total number of classes */
    End For
    Calculate the objective functions of new food source: $v_i$
    Replace old food source $c_i$ with new food source $v_i$
    Set $\mathrm{trial}_i = 0$,
End If
```

Algorithm 6: Pseudo-Code of the Update the External Archive (CA and DA).

```
Collect $FS_{nd}$ the set of non-dominated food sources in current population        /* Addition Strategy */
for $i = 1$ to $|FS_{nd}|$ do
    if $FS_{nd}[i]$ is not dominated by any food source stored in either AC or DA archive, then
        if $FS_{nd}[i]$ dominates any food source stored in either AC or DA archive, then
            The dominated food sources stored in AC and DA archive are removed
            Add the $FS_{nd}[i]$ to archive AC
        else
            Add the $FS_{nd}[i]$ to archive DA
        end if
        end if
    end if
end for
if $|CA| + |DA| > limit$ then        /* Removal Strategy */
    Select a food source of DA with minimal Euclidean distances to CA archive.
    Delete the selected food source from DA archive.
end if
```

5 Experimental Setup

This section describes the experimental setup conducted to evaluate the proposed TA-ABC algorithm over 10 weighted and seven un-weighted MDGs with MCA and ECA multi-objective formulations. Further, an experiment is also performed to compare the results of the TA-ABC with the existing TAA algorithm and NSGA-II.

5.1 Test Problems

In this paper, varieties of MDGs of software systems with different characteristics are used. There are two types of MDGs (weighted and un-weighted) used to evaluate the proposed approach. Table 1 provides a brief description about the number of modules and links of MDGs of considered software systems. In un-weighted MDGs, each connection (link) represents the existence of a unidirectional variable or a method reference between two modules. In weighted MDGs each connection contains weights which are calculated according

Table 1: Descriptions of Testing Problems [43].

Systems name	Modules	Links
Un-weighted		
Mtunis	20	57
Ispell	24	103
Rcs	29	163
Bison	37	179
Grappa	86	295
Bunch	116	365
Incl	174	360
Weighted		
Icecast	60	650
gnupg	88	601
inn	90	624
bitchx	97	1653
xntp	111	729
exim	118	1225
Mod_ssl	135	1095
ncurses	138	682
lynx	148	1745
nmh	198	3262

to the number of the unidirectional variables and method references between modules. Larger connection weights specify more interconnection strength between modules and increase probability that it should be placed in the same cluster.

5.2 Research Questions

In our study, we evaluate the performance of our proposed TA-ABC approach for M-SMCPs by finding out whether it could generate the good modularization in terms of various structural quality metrics (i.e. MQ, coupling, and cohesion) compared to other existing algorithms. In addition to structural quality metrics, we also used the IGD [50], HV [49], spread performance metric [43], Pareto optimality [43], and execution time to compare the algorithms. The major goal of our study is to address the following research questions.

RQ1. MQ value as assessment criterion: How well does the proposed TA-ABC perform when compared against TAA and NSGA-II algorithms using the MQ as the assessment criterion?

RQ2. Coupling as assessment criterion: How well does the proposed TA-ABC perform when compared against TAA and NSGA-II algorithms in terms of coupling?

RQ3. Cohesion as assessment criterion: How well does the proposed TA-ABC perform when compared against TAA and NSGA-II algorithms in terms of cohesion?

RQ4. Pareto optimality as assessment criterion: How well does the TA-ABC algorithm perform at producing good approximations to the Pareto front compared to TAA and NSGA-II algorithms?

RQ5. IGD, hypervolume, and spread as assessment criterion: How well does the proposed TA-ABC perform when compared against TAA and NSGA-II algorithms in terms of IGD, HV, and spread as the assessment criterion?

Note that the IGD metric corresponds to the average Euclidean distance separating each reference solution (true Pareto front) from its closest non-dominated one (Pareto front obtained by the algorithm). For each studied software project, we use the set of Pareto optimal solutions produced by all algorithms over all runs as a true Pareto front.

5.3 Competitor Algorithms and Parameter Setup

This subsection provides a brief description about competitor algorithms with their parameter settings that have been used in this study. The TAA and NSGA-II are two popular algorithms which have been used to solve the M-SMCPs by the previous researchers [4, 5, 43]. In this paper, the results of the TA-ABC are compared with TAA and NSGA-II. The parameter settings of TAA and NSGA-II algorithms are the same as suggested in [5, 43]. Different search-based optimization approaches usually consume different amounts of fitness computations. To make a fair comparison between such meta-heuristic algorithms, an equal number of fitness function computations is allowed to each algorithm. The number of fitness evaluations (N_{FE}) for the TA-ABC approach is computed using the following method: $N_{\mathrm{FE}} \leq (\mathrm{SN}+\mathrm{SN}+1)^{*}\mathrm{MCN}+\mathrm{SN}$, where SN and MCN is the number of onlooker bees and maximum number of iterations, respectively. The parameter for NSGA-II is the same as TAA. The parameter values of the algorithms are assigned according to the number of modules (N) in the problem instances. The crossover and mutation operators are single-point crossover and single-point mutation, respectively. The mutation probability is set as $0.004^{*}\log_2(N)$. The crossover probability is set as 0.8 for population size less than 100, otherwise 1.0. The maximum number of generations, population size, and total archive size is $200N$, $10N$, and $10N$, respectively. The limit parameter for the TA-ABC is set as (D*PS)/2, where D and PS is the dimension of the problem and population size, respectively.

5.4 Collecting Results from Experiment

The search-based optimization algorithms are stochastic in nature; i.e. they can produce different values on each run. We collect the results of each algorithm on each MDG by executing 30 times, following the same approach as discussed in [5, 43].

6 Results and Analysis

This section illustrates the results obtained by TA-ABC for the solution of M-SMCPs and its comparison with current evolutionary multi-objective approaches (i.e. TAA and NSGA-II) that have already been used to solve the M-SMCPs. Each subsection addresses one of the four research questions given in Section 5.2.

6.1 The MQ Value as Assessment Criterion

This section presents the results of the experiments that answer the RQ1. To answer this research question, we compared the TA-ABC with TAA and NSGA-II algorithms over seven un-weighted and 10 weighted MDGs with MCA and ECA multi-objective formulations in terms of MQ values.

Table 2 presents the MQ values obtained by TA-ABC, TAA, and NSGA-II algorithms with MCA formulation.

The 8th and 9th columns in the table denote the *p*-values (*p*-value below 0.05 is considered statistically significant). The symbols [–] denote that the result is significantly in favor of TA-ABC compared to corresponding approach, symbol [+] denotes opposite, and symbol [≈] is used when there is not a significant favor to any of the approaches. First, if we compare the MQ results of the TA-ABC approach with the TAA approach on un-weighted MDGs, the results show that the TA-ABC approach outperforms TAA approach in five MDGs out of seven MDGs. There are four cases where TA-ABC performs significantly better than TAA approach. Hence, there is good evidence to suggest that for un-weighted MDGs, the TA-ABC approach outperforms the TAA approach. Similarly, for weighted MDGs, the results provide sufficient evidence that the TA-ABC out performs

Table 2: Comparison of MQ Values Obtained by TA-ABC, TAA, and NSGA-II Algorithm (with MCA Approach).

Systems	TA-ABC		TAA		NSGA-II		*p*-Values	*p*-Values
	Mean	STD	Mean	STD	Mean	STD	TA-ABC-TAA	TA-ABC-NSGA-II
Un-weighted								
Mtunis	2.352	0.012	2.294	0.013	2.134	0.087	0.012 [–]	<0.001 [–]
Ispell	2.258	0.068	2.269	0.043	2.075	0.046	0.168 [≈]	<0.001 [–]
Rcs	2.296	0.036	2.145	0.034	2.062	0.034	<0.001 [–]	<0.001 [–]
Bison	2.257	0.051	2.416	0.038	2.187	0.045	<0.001 [+]	<0.001 [–]
Grappa	12.851	0.235	11.586	0.106	10.487	0.214	<0.001 [–]	<0.001 [–]
Bunch	11.765	0.321	12.145	0.225	10.654	0.025	0.013 [+]	<0.001 [–]
Incl	12.869	0.356	11.811	0.351	10.598	0.342	<0.001 [–]	<0.001 [–]
Weighted								
Icecast	2.216	0.065	2.401	0.057	2.158	0.054	<0.001 [+]	<0.001 [–]
gnupg	6.418	0.087	6.259	0.072	5.864	0.044	<0.001 [–]	<0.001 [–]
inn	8.026	0.079	7.421	0.077	6.875	0.053	<0.001 [–]	<0.001 [–]
bitchx	3.602	0.038	3.572	0.055	3.254	0.028	0.086 [≈]	<0.001 [–]
xntp	6.869	0.061	6.482	0.110	6.157	0.089	0.034 [–]	<0.001 [–]
exim	5.458	0.104	5.316	0.132	5.024	0.067	<0.001 [–]	<0.001 [–]
Mod_ssl	9.854	0.254	8.832	0.097	8.798	0.154	<0.001 [–]	<0.001 [–]
ncurses	11.562	0.346	10.211	0.145	10.125	0.351	<0.001 [–]	<0.001 [–]
lynx	3.481	0.073	3.447	0.086	3.145	0.025	0.472 [≈]	<0.001 [–]
nmh	6.978	0.257	6.671	0.177	6.658	0.131	<0.001 [–]	<0.001 [–]

Table 3: Comparison of MQ Values Obtained by TA-ABC, TAA, and NSGA-II Algorithm (with ECA Approach).

Systems	**TA-ABC**		**TAA**		**NSGA-II**		***p*-Values**	***p*-Values**
	Mean	**STD**	**Mean**	**STD**	**Mean**	**STD**	**TA-ABC-TAA**	**TA-ABC-NSGA-II**
Un-weighted								
Mtunis	2.157	0.021	2.314	0.000	1.785	0.032	<0.001 [−]	<0.001 [−]
Ispell	2.342	0.036	2.339	0.022	1.981	0.084	0.127 [≈]	<0.001 [−]
Rcs	2.132	0.012	2.239	0.022	1.795	0.036	<0.001 [−]	<0.001 [−]
Bison	2.458	0.054	2.648	0.029	2.235	0.054	<0.001 [−]	<0.001 [−]
Grappa	13.687	0.148	12.578	0.053	12.521	0.052	<0.001 [−]	<0.001 [−]
Bunch	13.897	0.342	13.455	0.088	12.325	0.245	0.175 [≈]	<0.001 [−]
Incl	13.498	0.234	13.511	0.059	12.642	0.134	0.103 [≈]	<0.001 [−]
Weighted								
Icecast	2.842	0.036	2.654	0.039	2.561	0.061	<0.001 [−]	<0.001 [−]
gnupg	7.621	0.085	6.905	0.055	7.156	0.035	<0.001 [−]	<0.001 [−]
inn	7.837	0.062	7.876	0.046	7.264	0.052	0.121 [≈]	<0.001 [−]
bitchx	4.036	0.037	4.267	0.027	3.647	0.026	<0.001 [−]	<0.001 [−]
xntp	8.954	0.064	8.168	0.076	8.265	0.087	<0.001 [−]	<0.001 [−]
exim	6.351	0.076	6.361	0.084	5.867	0.068	0.108 [≈]	<0.001 [−]
Mod_ssl	9.258	0.057	9.749	0.071	8.871	0.102	<0.001 [+]	<0.001 [−]
ncurses	12.325	0.127	11.297	0.133	11.135	0.141	<0.001 [−]	<0.001 [−]
lynx	4.957	0.079	4.694	0.060	4.531	0.062	<0.001 [−]	<0.001 [−]
nmh	8.964	0.109	8.592	0.148	8.439	0.075	<0.001 [−]	<0.001 [−]

the TAA approach over most of the cases except one case. That is, TA-ABC approach beats TAA approach in nine weighted MDGs, including seven in which the results are statistically significant. Second, if we compare the results of the TA-ABC approach with NSGA-II approach, the results show that the TA-ABC approach outperforms the NSGA-II approach for both weighted and un-weighted MDGs.

Table 3 presents the results obtained by TA-ABC, TAA, and NSGA-II with ECA formulation on weighted and un-weighted datasets. The results provided in Table 3 clearly indicate that the TA-ABC approach outperforms TAA and NSGA-II in most of the cases. The MQ results obtained by the TA-ABC approach and TAA approach over un-weighted MDG show that the TA-ABC approach outperforms the TAA approach in six MDGs out of seven MDGs. There are four cases where the TA-ABC approach performs significantly higher compared to TAA approach. However, for weighted software applications, the results indicate that the TA-ABC approach outperforms the TAA approach over most of the cases except one case. That is, TA-ABC approach performs significantly better than TAA approach in all seven software applications. The comparison results of the TA-ABC with NSGA-II approach shows that the TA-ABC approach outperforms NSGA-II approach in all cases for both weighted and un-weighted software applications.

6.2 Coupling as an Assessment Criterion

The coupling values obtained by TA-ABC, TAA, and NSGA-II approaches on un-weighted and weighted MDGs with MCA formulation are shown in Table 4 and with ECA formulation in Table 5. From Table 4 for un-weighted MDGs it can be observed that TA-ABC approach obtained higher values of coupling than TAA approach in all seven cases, out of which five cases are significantly in favor of TA-ABC. For weighted MDGs, the TA-ABC outperforms TAA in all 10 cases in which six cases are statistically significant. However, TA-ABC performs significantly better than NSGA-II in all cases for un-weighted as well as weighted MDGs.

Table 5 presents the results comparing the performance of the TA-ABC approach and the TAA approach, and TA-ABC approach and NSGA-II approach with ECA formulation in terms of coupling measure. Coupling results shown in Table 5 for un-weighted MDGs indicate that the TA-ABC outperforms TAA in five cases, out

Table 4: Comparison of Coupling Values Obtained by TA-ABC, TAA, and NSGA-II Algorithm (with MCA Approach).

Systems	**TA-ABC**		**TAA**		**NSGA-II**		***p*-Values**	***p*-Values**
	Mean	**STD**	**Mean**	**STD**	**Mean**	**STD**	**TA-ABC-TAA**	**TA-ABC-NSGA-II**
Un-weighted								
Mtunis	63.391	3.256	64.733	4.185	66.561	3.419	0.123 [≈]	<0.001 [–]
Ispell	158.458	1.025	159.800	6.440	166.381	1.076	0.148 [≈]	<0.001 [–]
Rcs	217.391	12.361	235.733	30.669	228.261	12.979	0.014 [–]	<0.001 [–]
Bison	242.925	17.564	277.267	16.463	255.071	18.442	<0.001 [–]	<0.001 [–]
Grappa	385.125	19.261	420.467	22.380	404.381	20.224	<0.001 [–]	<0.001 [–]
Bunch	498.525	13.256	580.867	16.648	523.451	13.919	0.011 [–]	<0.001 [–]
Incl	519.125	22.153	536.467	28.048	545.081	23.261	<0.001 [–]	<0.001 [–]
Weighted								
Icecast	7484.858	368.256	7636.200	589.843	7859.101	386.669	0.125 [≈]	<0.001 [–]
gnupg	4191.188	412.357	5192.530	335.669	4400.747	432.975	<0.001 [–]	<0.001 [–]
inn	5375.388	389.235	6176.730	325.260	5644.157	408.697	<0.001 [–]	<0.001 [–]
bitchx	35837.36	278.365	35938.700	5406.697	37629.228	292.283	0.107 [≈]	<0.001 [–]
xntp	3559.058	356.127	4460.400	219.445	3737.011	373.933	0.014 [–]	<0.001 [–]
exim	11546.06	835.547	12347.400	1127.563	12123.363	877.324	<0.001 [–]	<0.001 [–]
Mod_ssl	11137.16	798.294	12138.500	621.962	11694.018	838.209	<0.001 [–]	<0.001 [–]
ncurses	2569.788	124.576	3071.130	188.785	2698.277	130.805	<0.001 [–]	<0.001 [–]
lynx	22149.56	1234.561	23150.900	1726.014	23257.038	1296.289	0.127 [≈]	<0.001 [–]
nmh	19620.16	562.371	19921.500	876.440	20601.168	590.490	0.162 [≈]	<0.001 [–]

Table 5: Comparison of Coupling Values Obtained by TA-ABC, TAA, and NSGA-II Algorithm (with ECA Approach).

Systems	**TA-ABC**		**TAA**		**NSGA-II**		***p*-Values**	***p*-Values**
	Mean	**STD**	**Mean**	**STD**	**Mean**	**STD**	**TA-ABC-TAA**	**TA-ABC-NSGA-II**
Un-weighted								
Mtunis	61.557	2.235	60.000	0.000	64.635	2.347	0.129 [≈]	<0.001 [–]
Ispell	157.624	1.234	145.933	5.595	165.505	1.296	<0.001 [+]	<0.001 [–]
Rcs	213.557	9.345	230.867	15.719	224.235	9.812	<0.001 [–]	<0.001 [–]
Bison	235.091	16.237	252.400	12.434	246.846	17.049	<0.001 [–]	<0.001 [–]
Grappa	376.291	12.894	387.667	16.601	395.106	13.539	0.122 [≈]	<0.001 [–]
Bunch	479.691	18.239	504.600	10.611	503.676	19.151	<0.001 [–]	<0.001 [–]
Incl	534.291	15.765	439.600	7.673	561.006	16.553	0.003 [+]	<0.001 [–]
Weighted								
Icecast	7434.024	325.128	7569.670	416.378	7805.725	341.384	<0.001 [–]	<0.001 [–]
gnupg	4090.354	384.265	4413.670	207.660	4294.872	403.478	<0.001 [–]	<0.001 [–]
inn	4874.554	368.864	5046.200	380.526	5118.282	387.307	<0.001 [–]	<0.001 [–]
bitchx	34936.52	356.195	35546.800	1266.136	36683.346	374.005	<0.001 [≈]	<0.001 [–]
xntp	4358.224	348.562	3692.070	109.004	4576.135	365.990	<0.001 [+]	<0.001 [–]
exim	12145.22	532.248	12612.900	1050.310	12752.481	558.860	0.108 [≈]	<0.001 [–]
Mod_ssl	10136.32	1123.561	11008.400	488.348	10643.136	1179.739	<0.001 [–]	<0.001 [–]
ncurses	2468.954	132.623	2607.270	115.030	2592.402	139.254	<0.001 [–]	<0.001 [–]
lynx	18148.72	1137.295	20546.700	956.032	19056.156	1194.160	<0.001 [–]	<0.001 [–]
nmh	17819.32	932.584	18576.800	473.564	18710.286	979.213	<0.001 [–]	<0.001 [–]

of which three cases are statistically significant in favor of TA-ABC. For weighted MDGs, the TA-ABC outperforms TAA in nine cases out of which seven cases are statistically significant in favor of TA-ABC. However, the coupling results of the TA-ABC and NSGA-II show that the TA-ABC approach performs significantly better than NSGA-II in all cases for un-weighted and weighted MDGs.

6.3 Cohesion as an Assessment Criterion

This section compares TA-ABC algorithm with TAA and NSGA-II approach, i.e. how each of the multi-objective approaches performs in terms of cohesion as an assessment criterion using MCA and ECA formulations. Table 6 presents the cohesion results of the TA-ABC approach, TAA, and NSGA-II with MCA formulation. The cohesion results obtained from TA-ABC and TAA approaches over un-weighted MDGs given in Table 6 show that the TA-ABC outperforms TAA in all cases out of which five cases are significantly better. For weighted MDGs, the TA-ABC outperforms TAA in all cases out of which eight cases are significantly better. The cohesion results for TA-ABC and NSGA-II clearly show that the TA-ABC performs NSGA-II algorithm significantly better in all cases of weighted and un-weighted MDGs.

Table 7 presents the results comparing the performance of {TA-ABC approach, TAA approach} and {TA-ABC approach, NSGA-II approach} in terms of cohesion measure using ECA formulation. First, if we compare the cohesion values of TA-ABC approach and TAA approach, the results shown in Table 7 indicate that the TA-ABC approach outperforms TAA in five cases out of the seven un-weighted MDGs, out of which three cases are significantly better. In weighted MDGs, the TA-ABC approach outperforms TAA in nine out of 10 cases, out of which seven cases are significantly better. Second, if we compare the TA-ABC approach and NSGA-II approach, the comparison results indicate that the TA-ABC approach significantly outperforms NSGA-II in all cases for weighted and un-weighted MDGs.

6.4 Pareto Optimality as Assessment Criterion

This section compares the TA-ABC algorithm with TAA and NSGA-II in terms of how well each performs at producing good approximations to the Pareto front. Table 8 presents the dominance relationship for the results obtained from TA-ABC and TAA with both MCA and ECA formulations. This dominance relationship is used to compare any two solutions in multi-objective space. In this table, A denotes the TA-ABC with MCA, B denotes the TA-ABC with ECA, C denotes the TAA with MCA, and D denotes the TAA with ECA. The heading

Table 6: Comparison of Cohesion Values Obtained by TA-ABC, TAA, and NSGA-II Algorithm (with MCA Approach).

Systems	TA-ABC		TAA		NSGA-II		*p*-Values	*p*-Values
	Mean	STD	Mean	STD	Mean	STD	TA-ABC-TAA	TA-ABC-NSGA-II
Un-weighted								
Mtunis	25.304	1.365	24.633	2.092	23.719	3.419	0.112 [≈]	<0.001 [–]
Ispell	23.771	2.156	23.100	3.220	19.8095	1.076	0.212 [≈]	<0.001 [–]
Rcs	54.304	12.354	45.133	15.335	48.869	12.979	0.013 [–]	<0.001 [–]
Bison	57.538	3.856	40.367	8.231	51.465	18.442	<0.001 [–]	<0.001 [–]
Grappa	102.438	6.389	84.767	11.190	92.81	20.224	<0.001 [–]	<0.001 [–]
Bunch	114.738	5.687	73.567	8.324	102.275	13.919	0.011 [–]	<0.001 [–]
Incl	100.438	12.568	91.767	14.024	87.46	23.261	<0.001 [–]	<0.001 [–]
Weighted								
Icecast	1685.571	158.366	1609.900	294.921	1498.45	386.669	0.089 [≈]	<0.001 [–]
gnupg	1605.404	135.854	1104.733	167.834	1500.625	432.975	<0.001 [–]	<0.001 [–]
inn	1172.304	201.361	771.633	162.630	1037.92	408.697	<0.001 [–]	<0.001 [–]
bitchx	7695.303	1532.563	7644.633	2703.349	6799.369	292.283	0.102 [≈]	<0.001 [–]
xntp	1184.471	88.356	733.800	109.722	1095.495	373.933	0.017 [–]	<0.001 [–]
exim	3679.97	512.364	3279.300	563.781	3391.319	877.324	<0.001 [–]	<0.001 [–]
Mod_ssl	3412.403	256.845	2911.733	310.981	3133.974	838.209	<0.001 [–]	<0.001 [–]
ncurses	825.104	69.325	574.433	94.392	760.8595	130.805	<0.001 [–]	<0.001 [–]
lynx	2929.237	632.862	2428.567	863.007	2375.498	1296.289	<0.001 [–]	<0.001 [–]
nmh	2182.937	363.581	2032.267	438.220	1692.433	590.490	<0.001 [–]	<0.001 [–]

Table 7: Comparison of Cohesion Values Obtained by TA-ABC, TAA, and NSGA-II Algorithm (with ECA Approach).

Systems	TA-ABC		TAA		NSGA-II		*p*-Values	*p*-Values
	Mean	STD	Mean	STD	Mean	STD	TA-ABC-TAA	TA-ABC-NSGA-II
Un-weighted								
Mtunis	26.221	0.361	27.000	0.000	24.682	2.347	0.106 [≈]	<0.001 [–]
Ispell	24.188	2.231	30.033	2.798	20.2475	1.296	<0.001 [+]	<0.001 [–]
Rcs	56.221	13.265	47.567	7.859	50.882	9.812	<0.001 [–]	<0.001 [–]
Bison	61.455	8.236	52.800	6.217	55.5775	17.049	<0.001 [–]	<0.001 [–]
Grappa	106.855	9.356	101.167	8.301	97.4475	13.539	0.078 [≈]	<0.001 [–]
Bunch	124.155	86.349	111.700	5.305	112.1625	19.151	0.007 [–]	<0.001 [–]
Incl	92.855	6.348	140.200	3.836	79.4975	16.553	0.002 [+]	<0.001 [–]
Weighted								
Icecast	1710.988	123.456	1643.167	208.189	1525.138	341.384	<0.001 [–]	<0.001 [–]
gnupg	1655.821	88.346	1494.167	103.830	1553.562	403.478	<0.001 [–]	<0.001 [–]
inn	1422.721	36.123	1336.900	190.263	1300.857	387.307	<0.001 [–]	<0.001 [–]
bitchx	8145.723	563.238	7840.600	633.068	7272.31	374.005	0.091 [≈]	<0.001 [–]
xntp	784.888	55.237	1117.967	54.502	675.9325	365.990	<0.001 [+]	<0.001 [–]
exim	3380.39	235.642	3146.567	525.155	3076.76	558.860	0.067 [≈]	<0.001 [–]
Mod_ssl	3912.823	145.326	3476.800	244.174	3659.415	1179.739	<0.001 [–]	<0.001 [–]
ncurses	875.521	12.365	806.367	57.515	813.797	139.254	<0.001 [–]	<0.001 [–]
lynx	4929.657	213.023	3730.633	478.016	4475.939	1194.160	<0.001 [–]	<0.001 [–]
nmh	3083.357	142.691	2704.600	236.782	2637.874	979.213	<0.001 [–]	<0.001 [–]

N_{XY} indicates the number of solutions generated by approach X that are dominated by solutions produced by Y. In comparison, the approach X is better than approach Y if N_{XY} is small and N_{YX} is large.

Table 8 shows that the number of solutions produced by {TA-ABC with ECA} outperform {TA-ABC with MCA} in all of the problems studied. The {TA-ABC with MCA} outperforms {TAA with MCA} for un-weighted problems (four out of seven problems), while, in weighted systems, the {TA-ABC with MCA} outperforms the {TAA with MCA} in only one problem. The {TA-ABC with MCA} outperforms {TAA with ECA} for un-weighted

Table 8: Results of Dominated Comparison of TA-ABC Algorithm and TAA.

	N_{AB}	N_{BA}	N_{AC}	N_{CA}	N_{AD}	N_{DA}	N_{BC}	N_{CB}	N_{BD}	N_{DB}
Un-weighted										
Mtunis	28	0	14	23	16	22	0	30	6	27
Ispell	30	0	25	26	24	21	2	24	5	25
Rcs	30	0	14	21	16	20	0	25	7	22
Bison	30	0	22	14	12	11	5	24	8	11
Grappa	27	0	21	18	17	16	0	30	4	27
Bunch	26	0	16	13	18	17	4	25	8	26
Incl	30	0	19	15	17	21	5	21	3	22
Weighted										
Icecast	30	0	11	18	14	15	8	22	7	17
gnupg	30	0	17	19	22	21	2	17	6	21
inn	30	0	18	25	21	27	1	29	5	18
bitchx	30	0	14	27	17	22	3	24	8	26
xntp	30	0	21	22	22	21	0	30	9	18
exim	30	0	19	17	16	18	8	24	11	17
Mod_ssl	30	0	22	25	23	25	1	27	7	27
ncurses	30	0	16	17	17	22	7	18	5	28
lynx	30	0	14	19	12	14	0	30	5	26
nmh	30	0	21	24	20	24	0	30	2	24

Table 9: Results of Dominated Comparison of TA-ABC Algorithm and NSGA-II.

	N_{PQ}	N_{QP}	N_{PR}	N_{RP}	N_{PS}	N_{SP}	N_{QR}	N_{RQ}	N_{QS}	N_{SQ}
Un-weighted										
Mtunis	28	0	26	21	15	23	1	28	7	25
Ispell	30	0	22	24	23	26	3	25	5	26
Rcs	30	0	15	24	15	22	0	25	7	28
Bison	30	0	23	15	10	13	5	23	5	16
Grappa	27	0	20	17	17	16	0	27	4	27
Bunch	26	0	16	13	18	17	4	25	8	22
Incl	30	0	18	14	17	21	4	21	6	24
Weighted										
Icecast	30	0	12	18	12	16	7	21	6	19
gnupg	30	0	17	19	22	21	2	17	7	26
inn	30	0	19	24	21	27	1	28	5	14
bitchx	30	0	14	27	18	23	5	24	8	26
xntp	30	0	22	18	24	22	0	30	10	17
exim	30	0	12	17	19	18	6	22	11	17
Mod_ssl	30	0	22	25	20	14	1	27	7	25
ncurses	30	0	19	17	23	22	7	22	6	29
lynx	30	0	14	19	17	14	1	30	5	22
nmh	30	0	21	24	24	13	0	30	3	21

problems (three out of seven problems), while in weighted systems, the {TA-ABC with ECA} outperforms the {TAA with MCA} in eight out of 10 problems. The {TA-ABC with ECA} outperforms {TAA with MCA} in all un-weighted and weighted problems. Similarly, the {TA-ABC with ECA} outperforms {TAA with ECA} in all un-weighted and weighted problems. These findings taken together indicate that the TA-ABC is better than TAA.

Table 9 presents the dominance relationship for the results obtained from TA-ABC and NSGA-II with both MCA and ECA formulations. In this table, P denotes the TA-ABC with MCA, Q denotes the TA-ABC with ECA, R denotes the NSGA-II with MCA, and S denotes the NSGA-II with ECA.

Table 9 shows that the number of solutions produced by {TA-ABC with ECA} outperforms {TA-ABC with MCA} in all of the problems studied. The {TA-ABC with MCA} outperforms {NSGA-II with MCA} for un-weighted problems (five out of seven problems), while in weighted systems, the {TA-ABC with MCA} outperforms {NSGA-II with MCA} in only two problems. The {TA-ABC with MCA} outperforms {NSGA-II with ECA} for un-weighted problems (two out of seven problems), while in weighted systems, the {TA-ABC with ECA} outperforms the {NSGA-II with MCA} in seven of 10 problems. The {TA-ABC with ECA} outperforms {NSGA-II with MCA} in all un-weighted and weighted problems. Similarly, the {TA-ABC with ECA} outperforms {NSGA-II with ECA} in all un-weighted and weighted problems. These findings taken together indicate that the TA-ABC is better than NSGA-II.

6.5 IGD, Hypervolume, and Spread as Assessment Criteria

In the previous sections, we compared our TA-ABC algorithm with other existing algorithms (i.e. TAA and NSGA-II) in terms of structural quality metrics (i.e. MQ, coupling, and cohesion) and Pareto optimality. In this section, we compare the TA-ABC algorithm with existing algorithms in terms of IGD, HV, and spread values for both MCA and ECA formulations. The symbol [–] denotes that the result is significantly in favor of TA-ABC compared to corresponding approach, symbol [+] denotes opposite, and symbol [≈] is used when there is not a significant favor to any of the approaches. Table 10 presents IGD values of the results obtained through TA-ABC, TAA, and NSGA-II over weighted and un-weighted software projects. Tables 11 and 12 report the statistics of the HV and spread of the results obtained through TA-ABC, TAA, and NSGA-II algorithms, respectively. For the MCA and ECA formulations, IGD statistics given in Table 10 indicates that TA-ABC outperforms other

Table 10: The Statistics of IGD Metric Values Obtained at 30 Runs of TA-ABC, TAA, and NSGA-II Algorithms with MCA and ECA.

Systems	MCA			ECA		
	TA-ABC	TAA	NSGA-II	TA-ABC	TAA	NSGA-II
Un-weighted						
Mtunis	2.737×10^{-4}	2.741×10^{-4} [≈]	3.261×10^{-4} [–]	2.534×10^{-4}	2.538×10^{-4} [≈]	4.652×10^{-4} [–]
Ispell	3.891×10^{-3}	3.958×10^{-3} [–]	4.184×10^{-3} [≈]	3.653×10^{-3}	3.738×10^{-3} [–]	4.142×10^{-3} [–]
Rcs	4.486×10^{-3}	4.493×10^{-3} [≈]	4.274×10^{-3} [+]	4.278×10^{-3}	4.376×10^{-3} [–]	4.678×10^{-3} [–]
Bison	4.103×10^{-4}	4.194×10^{-4} [–]	4.229×10^{-4} [–]	4.103×10^{-4}	4.104×10^{-4} [≈]	4.106×10^{-4} [≈]
Grappa	5.912×10^{-4}	6.052×10^{-4} [–]	5.232×10^{-4} [–]	5.768×10^{-4}	5.125×10^{-4} [–]	5.212×10^{-4} [–]
Bunch	6.192×10^{-4}	6.365×10^{-4} [–]	6.413×10^{-4} [–]	6.192×10^{-4}	6.365×10^{-3} [–]	6.324×10^{-3} [–]
Incl	5.987×10^{-3}	6.172×10^{-3} [–]	6.215×10^{-3} [–]	6.001×10^{-3}	6.218×10^{-3} [–]	6.187×10^{-3} [–]
Weighted						
Icecast	7.786×10^{-3}	7.945×10^{-3} [–]	7.776×10^{-3} [≈]	7.674×10^{-3}	7.743×10^{-3} [–]	8.242×10^{-3} [–]
gnupg	5.476×10^{-3}	5.489×10^{-3} [≈]	5.271×10^{-3} [+]	5.274×10^{-3}	5.386×10^{-3} [–]	5.671×10^{-3} [–]
inn	4.103×10^{-4}	4.194×10^{-4} [–]	4.229×10^{-4} [–]	5.526×10^{-3}	5.533×10^{-4} [≈]	6.261×10^{-4} [–]
bitchx	6.912×10^{-4}	7.062×10^{-4} [–]	7.212×10^{-4} [–]	7.683×10^{-3}	7.734×10^{-3} [–]	8.241×10^{-3} [–]
xntp	6.192×10^{-3}	6.365×10^{-3} [–]	6.413×10^{-3} [–]	5.276×10^{-3}	5.386×10^{-3} [–]	5.671×10^{-3} [–]
exim	7.987×10^{-3}	8.172×10^{-3} [–]	8.215×10^{-3} [–]	5.526×10^{-4}	5.533×10^{-4} [≈]	6.261×10^{-4} [–]
Mod_ssl	5.737×10^{-4}	5.740×10^{-4} [≈]	6.283×10^{-4} [–]	7.683×10^{-3}	7.734×10^{-3} [–]	8.241×10^{-3} [–]
ncurses	6.128×10^{-4}	6.736×10^{-4} [–]	7.128×10^{-4} [–]	6.122×10^{-4}	6.647×10^{-4} [–]	7.123×10^{-4} [–]
lynx	7.008×10^{-3}	7.692×10^{-3} [–]	8.314×10^{-3} [–]	8.123×10^{-3}	8.612×10^{-3} [–]	9.341×10^{-4} [–]
nmh	5.121×10^{-3}	5.734×10^{-3} [–]	6.502×10^{-3} [–]	5.012×10^{-3}	5.398×10^{-3} [–]	6.328×10^{-3} [–]

Table 11: The Statistics of HV Metric Values Obtained at 30 Runs of TA-ABC, TAA, and NSGA-II Algorithms with MCA and ECA.

Systems	MCA			ECA		
	TA-ABC	TAA	NSGA-II	TA-ABC	TAA	NSGA-II
Un-weighted						
Mtunis	0.2718	0.1515 [–]	0.2755 [≈]	0.5242	0.3286 [–]	0.5215 [≈]
Ispell	0.4335	0.3232 [–]	0.2115 [–]	0.4579	0.0821 [≈]	0.6846 [+]
Rcs	0.5381	0.3370 [–]	0.6701 [+]	0.7201	0.1429 [–]	0.5677 [–]
Bison	0.5122	0.0745 [–]	0.3821 [–]	0.3999	0.4939 [+]	0.1717 [–]
Grappa	0.5170	0.5155 [≈]	0.1524 [–]	0.4991	0.3267 [–]	0.3483 [–]
Bunch	0.2880	0.3168 [+]	0.0393 [–]	0.7616	0.6015 [–]	0.3533 [–]
Incl	0.5064	0.2263 [–]	0.3094 [–]	0.5065	0.3518 [–]	0.0794 [–]
Weighted						
Icecast	0.6211	0.5946 [–]	0.1373 [–]	0.4204	0.4212 [≈]	0.2488 [–]
gnupg	0.5302	0.6758 [+]	0.4217 [–]	0.3085	0.1274 [–]	0.0515 [–]
inn	0.9559	0.1015 [–]	0.0582 [–]	0.8767	0.2050 [–]	0.2077 [–]
bitchx	0.6459	0.2439 [–]	0.2925 [–]	0.5272	0.1598 [–]	0.1477 [–]
xntp	0.7492	0.0465 [–]	0.2305 [–]	0.6413	0.5502 [–]	0.7332 [+]
exim	0.1244	0.2125 [+]	0.2524 [+]	0.6543	0.1295 [–]	0.5680 [–]
Mod_ssl	0.0564	0.0519 [≈]	0.1041 [+]	0.6562	0.4352 [–]	0.2538 [–]
ncurses	0.7373	0.3349 [–]	0.3827 [–]	0.3598	0.4141 [+]	0.1107 [–]
lynx	0.3703	0.2205 [–]	0.1058 [–]	0.7116	0.3318 [–]	0.6337 [–]
nmh	0.6966	0.6898 [≈]	0.2562 [–]	0.5819	0.3474 [–]	0.3625 [–]

algorithms in most of the cases for weighted and un-weighted MDGs. Additionally, TAA seems to be better than NSGA-II in most of the cases. The results based on the HV metric show that TA-ABC performs better than A-TAA and NSGA-II in most of the cases. Results in Table 12 indicate that the spread values achieved with the Pareto front generated by the TA-ABC algorithm is lower than those of TAA and NSGA-II, and in most of the cases it has the significantly lower values.

Table 12: The Statistics of Spread Metric Values Obtained at 30 Runs of TA-ABC, TAA, AND NSGA-II Algorithms with MCA and ECA.

Systems	**MCA**			**ECA**		
	TA-ABC	**TAA**	**NSGA-II**	**TA-ABC**	**TAA**	**NSGA-II**
Un-weighted						
Mtunis	0.0129	0.0646 [–]	0.0939 [–]	0.0332	0.1000 [–]	0.3728 [–]
Ispell	0.1081	0.2548 [–]	0.2822 [–]	0.0802	0.0778 [≈]	0.1734 [–]
Rcs	0.0249	0.0257 [≈]	0.1897 [–]	0.0510	0.0866 [–]	0.1515 [–]
Bison	0.1836	0.3951 [–]	0.2836 [–]	0.1521	0.8278 [–]	0.6633 [–]
Grappa	0.0258	0.0128 [+]	0.1523 [–]	0.3558	0.4608 [–]	0.1117 [+]
Bunch	0.0858	0.1044 [–]	0.0145 [+]	0.0428	0.1778 [–]	0.1601 [–]
Incl	0.3451	0.2906 [–]	0.2747 [–]	0.0874	0.3231 [–]	0.1544 [–]
Weighted						
Icecast	0.1677	0.2452 [–]	0.2873 [–]	0.2167	0.7026 [–]	0.3508
gnupg	0.0159	0.0763 [–]	0.0860 [–]	0.0783	0.2372 [–]	0.3109 [–]
inn	0.1242	0.2371 [–]	0.4045 [–]	0.4104	0.7794 [–]	0.8934 [–]
bitchx	0.1827	0.2756 [–]	0.1841 [≈]	0.2540	0.6059 [–]	0.4622 [–]
xntp	0.2971	0.5802 [–]	0.6537 [–]	0.0530	0.4223 [–]	0.1450 [–]
exim	0.1920	0.5939 [–]	0.3607 [–]	0.1017	0.4426 [–]	0.3881 [–]
Mod_ssl	0.3569	0.7684 [–]	0.4207 [–]	0.0441	0.8191 [–]	0.3010 [–]
ncurses	0.1781	0.2960 [–]	0.0386 [+]	0.0292	0.6537 [–]	0.4755 [–]
lynx	0.0646	0.0613 [≈]	0.0925 [–]	0.1533	0.1994 [–]	0.6796 [–]
nmh	0.1044	0.2977 [–]	0.6147 [–]	0.2078	0.2594 [–]	0.3948 [–]

7 Discussions

This section discusses the contributions and implications of our TA-ABC for M-SMCPs. The main contribution of the proposed TA-ABC approach with respect to the existing approaches on software module clustering (TAA and NSGA-II) is that this paper integrates the external archive concepts of TAA algorithm into the ABC algorithm so that balanced exploration and exploitation is achieved for more than three objective functions. The experimental results showed that the proposed TA-ABC approach performed better compared to existing approaches in terms of MQ, coupling, cohesion, Pareto optimality, and IGD values in most of the cases. In this study, we observed that the following helped in improving the quality of software systems in terms of MQ, coupling, cohesion, Pareto optimality, and IGD:

- The original TAA fails to maintain the diversity in case of all the solutions in CA are on the true Pareto front and the size of the CA has reached the limit of the union of DA and CA. In such situation there is no space available for any additional member of DA. The reason is that the CA does not maintain the diversity; it only maintains the convergence. However, to achieve a good balance between the diversity and convergence, TA-ABC algorithm maintains the diversity in CA in the case when the CA has reached the limit.
- Similarly, the updating strategy for CA and DA in TA-ABC algorithm generates good Pareto optimal solutions compared to TAA algorithm. The main reason for generating such good Pareto optimal solutions by the TA-ABC is that the approach is designed to produce good approximations to the Pareto front.

To conclude, we found that our approach produces good software clustering in terms of MQ, coupling, cohesion, and Pareto optimality in most of the cases compared to existing algorithms.

8 Threats to Validity

To explain the limitations and strengths of our proposed approach, we explore the factors that could affect the validity of the results obtained by TA-ABC. In this paper, we considered two major categories of threats (i.e. external validity and internal validity) that could affect the validation of results. External validity (or selection validity) concerns the degree to which the findings (i.e. results sample) of the approach can be generalized to the wider classes of problems. In search-based software engineering, this is a very important threat to the validity of findings because of a large number of diverse software systems available to any study. In our experimentation, this threat to validity has been mitigated by the fact that the proposed approach is concerned with MDG, an abstract representation of software systems. Since there is a many to one relation between the software systems and MDG (i.e. many individual software systems can map into a single MDG), the findings of a set of MDGs of a particular size is relevant to wider MDGs. In order to mitigate the possible external threats to validity, the experimentation uses the various size of MDGs, both un-weighted and weighted.

Internal validity is the degree to which conclusions can be drawn about the causal effect of independent variables on the dependent variables [18]. In this empirical study, the choice of statistical test (i.e. two-tailed t-test) was made to support the comparability with other existing studies [28, 43]. The t-test is more appropriate to data with normal distribution. However, studies [4, 14, 39] suggest that the t test is robust, even in the presence of non-normal distributed and significantly skewed data, if the sample sizes are sufficiently large as our empirical study.

9 Conclusions and Future Works

This paper presented a TA-ABC approach to address M-SMCPs. For this, the original ABC algorithm has been redesigned as multi-objective ABC algorithm by integrating the concept of external archives. The TA-ABC has been applied to solve M-SMCPs with two well-known multi-objective formulations of software clustering domain (ECA and MCA). The performance of the TA-ABC has been evaluated on two datasets obtained through different alternates: weighted MDGs and un-weighted MDGs. The results of the TA-ABC have been compared with the results reported in the literature. The five main quality criteria (i.e. MQ, coupling, cohesion, Pareto optimality, IGD, HV, and spread performance metric) have been used to assess the quality of the obtained clustering solutions. The results clearly reveal that TA-ABC is able to obtain better clustering solutions in terms of MQ, coupling, cohesion, Pareto optimality, IGD, HV, and spread performance metric. Hence, TA-ABC approach can be very useful to solve clustering problem of software and thus can help software managers in the better management of the software. In a future study, we will customize other meta-heuristic algorithms such as MOPSO, MODE, MOABC, MOSOS, etc. to address the M-SMCPs.

Bibliography

[1] H. Abdeen, S. Ducasse, H. A. Sahraoui and I. Alloui, Automatic package coupling and cycle minimization, in: *Proceedings of the 16th Working Conference on Reverse Engineering*, France, pp. 103–112, 2009.

[2] P. Amarjeet and J. K. Chhabra, Harmony search based remodularization for object-oriented software systems, *Comput. Lang. Syst. Struct*. **47** (2017), 153–169.

[3] P. Amarjeet and J. K. Chhabra, Improving modular structure of software system using lexical and structural dependencies. *Inform. Software Tech*. **82** (2017), 96–120.

[4] P. Amarjeet and J. K. Chhabra, Improving package structure of object-oriented software using multi-objective optimization and weighted class connections, *J. King Saud U. Comput. Inf. Sci*., in press. Available online 2 November 2015.

[5] M. Barros, An analysis of the effects of composite objectives in multi-objective software module clustering, in: *Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation*, USA, pp. 1205–1212, 2012.

[6] V. R. Basil and A. J. Turner, Iterative enhancement: a practical technique for software development, *IEEE T. Software Eng*. **1** (1975), 390–396.

[7] J. K. Chhabra, K. K. Aggarwal and Y. Singh, Code and data spatial complexity: two important software understandability measures, *Inform. Software Tech*. **45** (2003), 539–546.

[8] J. K. Chhabra, K. K. Aggarwal and Y. Singh, Measurement of object-oriented software spatial complexity, *Inform. Software Tech*. **46** (2004), 689–699.

[9] L. L. Constantine and E. Yourdon, *Structured Design*, Prentice Hall, USA, 1979.

[10] E. Cotilla-Sanchez, P. D. H. Hines, C. Barrows, S. Blumsack and M. Patel, Multi-attribute partitioning of power networks based on electrical distance, *IEEE T. Power Syst.* **28** (2013), 4979–4987.

[11] S. S. Dahiya, J. K. Chhabra and S. Kumar, Application of artificial bee colony algorithm to software testing. in: *2010 21st Australian Software Engineering Conference*, Auckland, pp. 149–154, 2010.

[12] K. Deb and H. Jain, An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints, *IEEE T. Evolut. Comput.* **18** (2004), 577–601.

[13] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, A fast and elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II, *IEEE T. Evolut. Comput.* **6** (2002), 182–197.

[14] L. Devroye, *Non-Uniform Random Variate Generation*. Springer-Verlag, USA, 1986.

[15] D. Doval, S. Mancoridis, B. S. Mitchell, Automatic clustering of software systems using a genetic algorithm, in: *Proceedings of IEEE Conference on Software Technology and Engineering Practice*, USA, pp. 73–81, 1999.

[16] A. E. Ezugwu, N. A. Okoroafor, S. M. Buhari, M. E. Frincu, S. B. Junaidu, Grid resource allocation with genetic algorithm using population based on multisets, *J. Intell. Syst.* **26** (2017), 169–184.

[17] A. Farrugia, Vertex-partitioning into fixed additive induced hereditary properties is NP-hard, *Electron. J. Combin.* **11** (2004), 1–9.

[18] M. Genero, J. Olivas, M. Piattini and F. Romero, Using metrics to predict OO information systems maintainability, in: *Proceedings of the 13th International Conference on Advanced Information Systems Engineering* (CAiSE'01), Springer-Verlag, London, UK, pp. 388–401, 2001.

[19] D. Gong, J. Sun and X. Ji, Evolutionary algorithms with preference polyhedron for interval multi-objective optimization problems. *Inform. Sciences* **233** (2013), 141–161.

[20] V. Gupta and J. K. Chhabra, Package level cohesion measurement in object-oriented software, *J. Braz. Comput. Soc.* **18** (2011), 251–266.

[21] M. Harman, R. Hierons and M. Proctor, A new representation and crossover operator for search-based optimization of software modularization, in: *Proceedings of the Genetic and Evolutionary Computation Conference*, USA, pp. 1351–1358, 2002.

[22] J. Harris, J. Hirst and M. Mossinghoff, *Combinatorics and Graph Theory*, Springer, New York, pp. 212–237, 2000.

[23] H. A. Hashim, B. O. Ayinde and M. A. Abido, Optimal placement of relay nodes in wireless sensor network using artificial bee colony algorithm, *J. Netw. Comput. Appl.* **64** (2016), 239–248.

[24] S. D. Hester, D. L. Parnas and D. F. Utter, Using documentation as a software design medium. *Bell Syst. Technol. J.* **60** (1981), 1941–1977.

[25] H. T. Jadhav and P. D. Bamane, Temperature dependent optimal power flow using g-best guided artificial bee colony algorithm, *Int. J. Elec. Power Energy Syst.* **77** (2016), 77–90.

[26] D. Karaboga, *An Idea Based on Honey Bee Swarm for Numerical Optimization*, Technical Report-TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.

[27] V. Kumar, J. K. Chhabra and D. Kumar, Grey wolf algorithm-based clustering technique, *J. Intell. Syst.* **26** (2017), 153–168.

[28] A. C. Kumari and K. Srinivas, Hyper-heuristic approach for multi-objective software module clustering, *J. Syst. Software* **117** (2016), 384–401.

[29] A. C. Kumari, K. Srinivas and M. P. Gupta, Software module clustering using a hyper-heuristic based multi-objective genetic algorithm. *Advance Computing Conference (IACC)*, *2013 IEEE 3rd International*, Ghaziabad, pp. 813–818, 2013.

[30] M. M. Lehman, On understanding laws, evolution, and conservation in the large-program life cycle, *J. Syst. Softw.* **1** (1980), 213–22.

[31] X. Li and G. Yang, Artificial bee colony algorithm with memory, *Appl. Soft Comput.* **41** (2016), 362–372.

[32] K. Mahdavi, M. Harman and R. M. Hierons, A multiple hill climbing approach to software module clustering, in: *Proceedings of the International Conference on Software Maintenance*, Netherlands, pp. 315–324, 2003.

[33] A. S. Mamaghani and M. R. Meybodi, Clustering of software systems using new hybrid algorithms, in: *Proceedings of the Ninth IEEE International Conference on Computer and Information Technology* (CIT'09), vol. 1, Bangladesh, 2009.

[34] S. Mancoridis, B. S. Mitchell, Y.-F. Chen and E. R. Gansner, Bunch: a clustering tool for the recovery and maintenance of software system structures, in: *Proceedings of the IEEE International Conference on Software Maintenance*, UK, pp. 50–59, 1999.

[35] S. Mancoridis, B. S. Mitchell, C. Rorres, Y. F. Chen and E. R. Gansner, Using automatic clustering to produce high-level system organizations of source code, in: *Proceedings of the International Workshop on Program Comprehension*, Italy, pp. 45–53, 1998.

[36] O. Maqbool and H. A. Babri, Hierarchical clustering for software architecture recovery, *IEEE T. Software Eng.* **33** (2007), 759–780.

[37] B. S. Mitchell, A heuristic search approach to solving the software clustering problem. PhD. dissertation, Drexel University, USA, 2002.

[38] B. S. Mitchell and S. Mancoridis, Using heuristic search techniques to extract design abstractions from source code, in: *Proceedings of the Genetic and Evolutionary Computation Conference*, USA, pp. 1375–1382, 2002.

[39] L. E. Moses, *Think and Explain with Statistics*, Addison-Wesley, USA, 1986.

[40] M. Ó Cinnéide, L. Tratt, M. Harman, S. Counsell and I.-H. Moghadam, Experimental assessment of software metrics using automated refactoring, in: *ESEM'12*, Sweden, pp.49–58, 2012.
[41] V. Plevris and M. Papadrakakis, A hybrid particle swarm – gradient algorithm for global structural optimization, *Comput. Aided Civ. Inf. Eng*. **26** (2011), 48–68.
[42] K. Praditwong, Solving software module clustering problem by evolutionary algorithms, in: *2011 Eighth International Joint Conference on Computer Science and Software Engineering (JCSSE)*, Nakhon Pathom, pp. 154–159, 2011.
[43] K. Praditwong, M. Harman and X. Yao, Software module clustering as a multi-objective search problem, *IEEE T. Software Eng*. **37** (2011), 264–282.
[44] K. Praditwong and X. Yao, A new multi-objective evolutionary optimization algorithm: the two-archive algorithm, in: *Proceedings of the International Conference on Computational Intelligence and Security*, vol 1, Hong Kong, pp. 286–291, 2006.
[45] P. Prashanth, K. K Pattanaik and P. Singh, BAT and hybrid BAT meta-heuristic for quality of service-based web service selection. *J. Intell. Syst*. **26** (2017), 123–137.
[46] A. Ramírez, J. R. Romero and S. Ventura, An approach for the evolutionary discovery of software architectures, *Inform. Sciences* **305** (2015), 234–255.
[47] Q. Zhang and H. Li, MOEA/D: a multiobjective evolutionary algorithm based on decomposition. *IEEE T. Evolut. Comput*. **11** (2007), 712–731.
[48] E. Zitzler and S. Künzli, *Indicator-Based Selection in Multi-objective Search*, *Parallel Problem Solving from Nature – PPSN VIII*, pp. 832–842, Springer, Berlin, Germany, 2004.
[49] E. Zitzler and L. Thiele, Multiobjective optimization using evolutionary algorithms – a comparative case study, in: *Conference on Parallel Problem Solving from Nature (PPSN V)*, pp. 292–301, 1998.
[50] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca and V. G. da Fonseca, Performance assessment of multi-objective optimizers: an analysis and review, *IEEE T. Evolut. Comput*. **7** (2003), 117–132.