Contents lists available at ScienceDirect

The Journal of Systems and Software

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

# Hyper-heuristic approach for multi-objective software module clustering

A. Charan Kumari[a,*], K. Srinivas[b]

[a] THE NORTHCAP UNIVERSITY, Gurgaon, India
[b] Dayalbagh Educational Institute, Dayalbagh, Agra, India

ARTICLE INFO

*Article history:*
Received 17 July 2015
Revised 4 April 2016
Accepted 5 April 2016
Available online 8 April 2016

*Keywords:*
Software maintenance
Software module clustering
Multi-objective optimization
Hyper-heuristics

ABSTRACT

In the software maintenance phase of software development life cycle, one of the main concerns of software engineers is to group the modules into clusters with maximum cohesion and minimum coupling.

To analyze the efficacy of Multi-objective Hyper-heuristic Evolutionary Algorithm (MHypEA) in solving real-world clustering problems and to compare the results with the reported results in the literature for single as well as multi-objective formulations of the problem and also to present a CASE tool that assists software engineers in software module clustering process.

The paper reports on empirical evaluation of the performance of MHypEA with the reported results in the literature. The comparison is mainly based on two factors – quality of the obtained solutions and the computational effort.

On all the attempted problems, MHypEA reported good results in comparison to all the studies that were reported on multi-objective formulation of the problem, with a computational effort of nearly one-twentieth of the computational effort required by the other multi-objective algorithms.

The hyper-heuristic approach is able to produce high quality clustered systems with less computational effort.

## 1. Introduction

Software evolution is a natural phenomenon in the software development life cycle. As the software evolves, the modular structure of software degrades, and at one point it becomes a challenging task to maintain the software further. Hence software module clustering is an important activity during software maintenance whose main goal is to obtain good modular structures. Software engineers greatly emphasize on good modular structures as it is easier to comprehend, develop and maintain such software systems. Due to the increase in the complexity and size of the software systems, maintenance has become a crucial issue for software engineers. The situation becomes awful, when the software lacks proper documentation on the changes that are performed during system evolution. As a consequence, it becomes practically difficult to modify the software in future. The replacement of such software with a new one is not a feasible option, as it does not guarantee the full functionality. The practices such as reverse engineering and reengineering have emerged to handle such software systems. As the source code is the only exact duplication of the system available to software developers and maintainers, the reverse engineering community is working towards the development of methods to extract the high-level structural information from the source code directly. Such methods are inclined to focus on design recovery through software clustering, program slicing, source code analysis etc.

Mancoridis et al. (1998) were the first to develop methods to facilitate the automatic recovery of the modular structure of a software system from its source code using Hill Climbing and Genetic Algorithms. They reported that their experimentation has shown good results for many of the investigated systems. They have also developed a clustering tool called Bunch (Mancoridis et al., 1999) that creates system decomposition automatically by treating clustering as an optimization problem and incorporated the integration of designer knowledge about the system structure into an otherwise fully automatic clustering process. By considering the practical significance of the problem, Praditwong et al. (2011) formulated two novel multi-objective formulations of the software module clustering with different objectives and evaluated the effectiveness of the multi-objective approach on real-world module clustering problems. Through their empirical study they claimed that the multi-objective approach produces significantly better solutions than the existing single-objective approach. Since then many researchers experimented using various metaheuristic search

* Corresponding author.
*E-mail addresses:* charankumari@yahoo.co.in (A.C. Kumari), ksri12@gmail.com (K. Srinivas).

http://dx.doi.org/10.1016/j.jss.2016.04.007
0164-1212/© 2016 Elsevier Inc. All rights reserved.

techniques (as will be presented in the last section on Related work) to find good modular structures.

Much of the earlier work has been focused on development of formulations of the problem and their implementations using well experimented metaheuristic search techniques and their variations. The rising interest of the researchers towards multi-objective optimization not only visualizes a transformation from the traditional techniques to more and more sophisticated and automated techniques but also accentuates the need for experimentation with new technologies. Also from the literature it has been observed that though the multi-objective search techniques were able to improve the quality of clustering; they did take two orders of magnitude longer than the single-objective algorithms, so there is clearly still a scope for improvement in the space of quality of results versus computational effort required. To achieve these objectives the present study has experimented with hyper-heuristics. Many researchers also experimented on hyper-heuristics for different optimization problems in different fields (Cowling et al., 2001, 2002a, 2002b; Bai and Kendall, 2005; Kendall and Mohamad, 2004; Burke et al., 2003; Cowling and Chakhlevitch, 2003; Storer et al., 1995) and reported better results. Some initial experimental results with the hyper-heuristic approach on software module clustering have been presented in Charan Kumari et al. (2013) and Charan Kumari and Srinivas (2013) and this paper is a promising extension of the previous work and is quite comprehensive in the following ways.

- Additional test problems have been added for further experimentation to examine the efficacy of hyper-heuristics in module clustering.
- The effectiveness of MHypEA in solving real-world clustering problems has been compared with the reported results in the literature for single as well as multi-objective formulations of the problem.
- All the results have been statistically analyzed using *t*-test with a confidence level of 95%.
- An effect size is reported to facilitate the interpretation of substantive significance of the results.
- An interactive automated CASE tool (ModGroup) has been designed and developed with good features to assist the software engineers in automatically clustering the modules.
- To validate the developed tool, a real-world problem from a software industry has been clustered using the developed tool.

The rest of the paper is structured as follows. Section 2 describes the software module clustering problem as a multi-objective optimization problem. The basic concepts of hyper-heuristics are presented in Section 3. A detailed explanation of Multi-objective Hyper-heuristic Evolutionary Algorithm (MHypEA) is provided in Section 4. Section 5 presents the experimental setup and Section 6 provides an in-depth analysis of the obtained results. Section 7 describes the framework of the automated CASE tool – ModGroup and its validation on a real-world application (non-open source) is presented in Section 8. Related work is presented in Section 9 and Section 10 concludes.

## 2. Problem description

Software module clustering is the process of grouping software modules into clusters in such a way that the highly dependent modules are grouped into the same cluster. The clustering assists in better comprehension of the system and provides easy maintenance in the future. This decomposition is based on the relationships among the modules. Generally these relationships are represented in the form of Module Dependency Graphs (MDGs), in which modules are represented as nodes and their inter relationships are represented by means of edges connecting the nodes. The input to the search algorithm is the MDG; where the nodes represent the modules and the edges represent the relationships among the modules. MDGs can be either weighted or unweighted. In a weighted MDG, the edges are assigned weights representing the strength of relationship between the modules and an unweighted MDG depicts the relationship among modules by the presence or absence of an edge. The primary objective of software module clustering is to produce software clusters that are of high quality with maximum intra-connectivity (cohesion) and minimum inter-connectivity (coupling). Intra-connectivity is the measure of density of connections among the modules in a single cluster. High intra-connectivity indicates a good clustering; as the largely dependent modules are grouped into the same cluster. Inter-connectivity is the measure of density of connections among modules in different clusters. Low inter-connectivity indicates a good clustering; as the clusters are highly independent of each other.

Praditwong et al. (2011) formulated the software module clustering as a multi-objective problem with two multi-objective approaches – Maximizing Cluster Approach (MCA) and Equal-size Cluster Approach (ECA). The objectives identified under MCA approach are

- Maximization of sum of intra-edges of all clusters.
- Minimization of sum of inter-edges of all clusters.
- Maximization of number of clusters.
- Maximization of Modularization Quality (MQ).
- Minimization of the number of isolated clusters.

The main aim of the MCA approach is to achieve good partitions having high cohesion and low coupling; while maximizing the number of clusters and minimizing the number of isolated clusters.

The ECA approach encourages producing clusters of nearly equal size, and promotes the decomposition of the software system into approximately equal size clusters. The objectives identified in this approach are

- Maximization of sum of intra-edges of all clusters.
- Minimization of sum of inter-edges of all clusters.
- Maximization of number of clusters.
- Maximization of Modularization Quality (MQ).
- Minimizing the difference between minimum and maximum number of modules in a cluster.

Modularization Quality (MQ) introduced by Mancoridis et al. (1998) is the sole objective in the single-objective formulation of software module clustering problems. The goal of MQ is to limit excessive coupling. The MQ measure attempts to find a balance between coupling and cohesion by combining them into a single measurement. The aim is to reward increased cohesion with a higher MQ score and to punish increased coupling with a lower MQ score. The Modularization Quality (MQ) is defined as (Praditwong et al., 2011)

$$MQ = \sum_{k=1}^{n} MF_k, \tag{1}$$

where $MF_k$ is the Modularization Factor of the $k$th cluster and $n$ is the total number of clusters identified by the algorithm. Modularization Factor (MF) is the ratio of intra-edges and inter-edges in each cluster and is given by

$$MF_k = \begin{cases} 0 & if \quad i = 0 \\ \dfrac{i}{i + j/2} & if \quad i > 0 \end{cases}, \tag{2}$$

$i$ is the sum of all the weights of intra-edges which has their both ends in the same cluster and $j$ is the sum of all the weights of inter-edges which has their ends in two different clusters. For an unweighted MDG, the weight is considered as 1.

**Algorithm 1** Hyper-heuristic algorithm (Burke et al., 2003).

```
1: Start with a set $H$ of heuristics, each of which is applicable to a problem state and transforms it to a new problem state.
2: Let the initial problem state be $S_0$
3: If the problem state is $S_i$ then find the heuristic that is most suitable to transform the problem state to $S_{i+1}$
4: If the problem is solved, stop. Otherwise go to step 3
```

**Table 1**
Set of low-level heuristics designed for MHypEA.

| Group with ***copy*** mutation | Group with ***exchange*** mutation |
|---|---|
| h1 : EA/rand/uniform/copy | h7 : EA/rand/uniform/exchange |
| h2 : EA/rand-to-best/uniform/copy | h8 : EA/rand-to-best/uniform/ exchange |
| h3 : EA/rand/hc1/copy | h9 : EA/rand/hc1/ exchange |
| h4 : EA/rand-to-best/hc1/copy | h10 : EA/rand-to-best/hc1/ exchange |
| h5 : EA/rand/hc2/copy | h11 : EA/rand/hc2/ exchange |
| h6 : EA/rand-to-best/hc2/copy | h12 : EA/rand-to-best/hc2/ exchange |

## 3. Hyper-heuristics

The main objective of hyper-heuristics is to develop more general systems that are able to handle a wide range of problem domains rather than current metaheuristic technology which tends to be customized to a particular problem or a narrow class of problems. One of the main motivation of hyper-heuristic approach is that they should be cheaper to implement and easier to use than problem specific special purpose methods and the goal is to produce good quality solutions in a more general framework.

Undoubtedly, the results of problem-specific solvers incorporating domain-specific knowledge, fine tuned and tailor-made will outperform those of a general framework algorithm. The hyper-heuristic approach will require little tuning and will reduce the need for more problem-specific knowledge, thereby significantly reduces the deployment time and use. Here the trade-off is between the problem-specific approaches that can yield quality of solutions and more generic, adaptive and applicable approaches that compromise a little on the quality of solutions.

Hyper-heuristics are often defined as "*heuristics to choose heuristics*" (Burke et al., 2003). A heuristic is considered as a rule-of-thumb that reduces the search effort required to find a solution. Metaheuristic operates directly on the problem search space with the goal of finding optimal or near-optimal solutions; whereas the hyper-heuristic operates on the heuristics search space which consists of all the heuristics that can be used to solve a given problem. Thus, hyper-heuristics are search algorithms that do not directly solve problems, but, instead, search the space of heuristics that can then solve the problem. Therefore Hyper-heuristics are an approach that operates at a higher level of abstraction than a metaheuristic.

The term Hyper-heuristics was introduced by Cowling et al. and described it as "*The hyper-heuristics manage the choice of which lower-level heuristic method should be applied at any given time, depending upon the characteristics of the heuristics and the region of the solution space currently under exploration*" (Cowling et al., 2001). So, they are broadly concerned with intelligently choosing a right heuristic. A general framework of a hyper-heuristic is presented in Algorithm 1 (Burke et al., 2003).

## 4. Multi-objective Hyper-heuristic Evolutionary Algorithm (MHypEA)

This section describes the Multi-objective Hyper-heuristic Evolutionary Algorithm (MHypEA) (Charan Kumari and Srinivas, 2013) which has the following features:

- General framework which provides solutions of good quality for a wider range of optimization problems.
- Generic design of the low-level heuristics without incorporating any domain specific knowledge.
- Selection of low-level heuristics based on reinforcement learning coupled with roulette- wheel selection.
- Dynamic adaptation of the weights assigned to low-level heuristics based on the performance of the heuristics in directing the search towards good regions.
- Good exploration and exploitation of the search space.

The general form of the low-level heuristics is EA/selection/crossover/mutation. Selection refers to the way in which the parents are selected for generating the offspring. Two types of selection are employed – *rand* and *rand-to-best*. In *rand*, both the parents are selected randomly from the population, whereas in *rand-to-best*, one parent is selected randomly from the population and the other parent is elite (the best one).

Three types of crossover operators are used to generate the offspring. Each of these crossover operators produces two children at a time. The first operator is uniform crossover; where the two children are generated by randomly selecting each gene from either of the parents. The second operator is a hybrid crossover1 (*hc1*) that is defined by hybridizing the single-point crossover with uniform crossover. First a crossover point is selected and the first child is generated by copying the genes from the first parent and the second child is generated by copying the genes from the second parent till the crossover point. Thereafter, the remaining genes of the two children are taken from either of the parents randomly. The third operator is a hybrid crossover2 (*hc2*) that is framed by the hybridization of two-point crossover with uniform crossover. Two crossover points are selected and the first child is generated by copying the genes from the first parent and the second child is generated by copying the genes from the second parent till the first crossover point. Subsequently the genes of the two children are taken randomly from either of the parents till the second crossover point. Thereafter the remaining genes are copied from the respective parent to the respective child.

Two types of mutation are defined – *copy* and *exchange*. In the first mutation operator (copy), two genes are selected randomly and the second gene is copied into the first one. In the second mutation operator (exchange), two randomly selected genes exchange their positions. Based on the discussed selection, crossover and mutation operators, twelve low-level heuristics are defined for the hyper-heuristic and are shown in Table 1.

The hyper-heuristic selects a promising low-level heuristic at the beginning of every iteration based on the information about the effectiveness of each low-level heuristic accumulated in previous runs. This is implemented through the principle of reinforcement learning (Kaelbling et al., 1996). The key idea is to "reward" improving low-level heuristics during each iteration of the search by increasing its weight and "punish" poorly performing ones by decreasing its weight. Initially all the low-level heuristics have been assigned with a value of 1 as their weights. After each generation, the weight of the selected heuristic has been adaptively increased/decreased based on the capability of the heuristic in guiding the search towards an optimal solution. That is, if the implementation of a selected heuristic on the parent population generates an offspring with a better Modularization Quality (MQ) value, then the weight of that particular heuristic is increased by one; otherwise it is decreased by one. Thus the weights of low-level heuristics are adaptively changed as the search progresses and reflect the effectiveness of low-level heuristics. The roulette-wheel approach is used to select a heuristic randomly with the probability proportional to its weight.

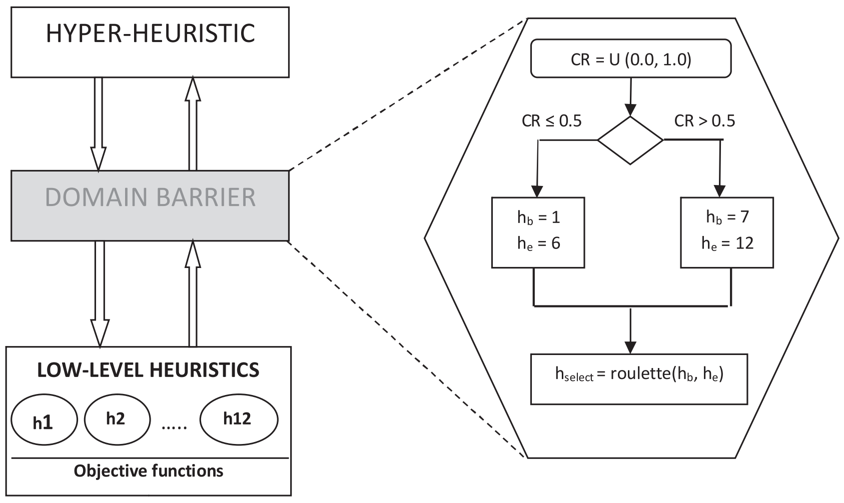


**Fig. 1.** Framework of the hyper-heuristic.

**Algorithm 2** Multi-objective Hyper-heuristic Evolutionary Algorithm (MHypEA) (Charan Kumari and Srinivas, 2013).

```
1: Initialize parent population
2: Evaluate the fitness of parent population
3: While (not termination-condition) do
4:     Select a low-level heuristic based on the selection mechanism
5:     Apply the selected low-level heuristic on the parent population and
         obtain offspring population
6:     Evaluate the fitness of offspring population
7:     Combine parent and offspring populations
8:     Perform non-dominated sorting on the combined population and select
         the individuals (for the next generation) starting from the first front
         by taking crowding distance into consideration
9: end while
```

The framework of the hyper-heuristic is shown in Fig. 1. It consists of two phases. The first phase selects the type of mutation to be adopted. Here, CR is a random number drawn from the uniform distribution on a unit interval, to select an EA model either with *copy* or with *exchange* mutation with equal probability. The values of $h_b$ and $h_e$ denotes the beginning and ending of the subscripts of low-level hyper-heuristics selected. The second phase selects a specific low-level heuristic within the selected group. This phase uses the reinforcement learning strategy with adaptive weights using roulette-wheel to select a specific model of EA. The pseudo code of the MHypEA is given in Algorithm 2.

In the beginning, the parent population is initialized and its fitness is evaluated. A low-level heuristic is selected as described above and is applied on the parent population to generate the offspring. After evaluating the fitness of offspring, the parent and offspring populations are combined. The combined population is sorted into different fronts based on the concept of nondominated sorting procedure as described by Deb et al. (2002). In addition, to maintain a good spread among the solutions within the same front crowding distance is calculated to estimate the density of solutions surrounding a particular solution in the population by taking the average distance of two solutions on either side of the solution along each of the objectives (Deb et al., 2002). The new population for the next generation is selected starting from the first front, followed by the second front and so on. While doing so, in case if the inclusion of entire front exceeds the population size (as the combined population is twice the population size), then the solutions with large crowding distance value are selected to maintain diversity. The process is continued till the termination criterion is met.

## 5. Experimental setup

This section presents the research questions, details of MDGs used for experimentation, solution encoding method, experimental settings and the methodology adopted for obtaining the results.

### 5.1. Research questions

This paper aims to provide experimental evidence to answer the following research questions.

**RQ1:** Is the hyper-heuristic approach able to identify clusters better than other examined algorithms?
**RQ2:** Does the hyper-heuristic approach require a lower number of function evaluations than the other multi-objective algorithms (Two-Archive and NSGA-II) to cluster the modules?

### 5.2. Data sets

The performance of the MHypEA for the solution of Multi-Objective Software Module Clustering has been studied on two sets of MDGs. The first set with six unweighted MDGs has been provided by Spiros Mancoridis and Mark Harman, as referenced in the literature (Praditwong et al., 2011). This set contains MDGs with the number of modules varying from 20 to 174 and the links among the modules from 57 to 360. The second set of MDGs (Barros, 2012) has been made available by Barros with the number of modules in the range of 26 to 119 and the interconnections among the modules from 61 to 276. The descriptions of these two sets of MDGs are given in Tables 2 and 3, respectively. Due to non-availability of data despite many efforts, the weighted MDGs as referenced in Praditwong et al., 2011) were not experimented.

### 5.3. Solution encoding

The solution to the problem consists of grouping the modules into clusters. Each solution is encoded as an integer array whose size is equal to the number of modules, with the array indices representing the modules and the contents of the array denoting the cluster to which the module is being mapped. The process involved is shown below diagrammatically.

| 2 | 3 | 3 | 1 | 2 | 1 | 3 | 2 |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

**Table 2**
Details of MDGs – Data set I.

| Name | Modules | Links | Description |
|---|---|---|---|
| mtunis | 20 | 57 | An operating system for educational purpose written in the Turing language |
| ispell | 24 | 103 | Software for spelling and typographical error correction in files. |
| rcs | 29 | 163 | Revision Control System used to manage multiple revisions of files |
| bison | 37 | 179 | General-purpose parser generator for converting grammar descriptions into C programs |
| grappa | 86 | 295 | Genome Rearrangements Analyzer under Parsimony and other Phylogenetic Algorithms |
| incl | 174 | 360 | Graph drawing tool |

**Table 3**
Details of MDGs – Data set II.

| Name | Modules | Links | Description |
|---|---|---|---|
| JODAMONEY | 26 | 102 | Money management library |
| JXLSREADER | 27 | 73 | Library for reading excel files |
| SEEMP | 31 | 61 | Small information system |
| JAVAOCR | 59 | 155 | Written text recognition library |
| SERVLET | 100 | 276 | Java servlets API |
| JUNIT | 119 | 209 | Unit testing library |

The above instance denotes clustering of eight modules into three clusters. The modules numbered 4 and 6 are grouped together and are mapped to cluster 1 and similarly the modules 1, 5 and 8 to cluster 2 and the modules 2, 3 and 7 to cluster 3.

### 5.4. Experimental settings

The algorithmic parameters of population size and number of generations are based on the number of modules (N) in the MDG to cope up with the increase in the complexity involved in clustering more number of modules. In the studies reported in the literature, the population size has been fixed at 10 N and the number of generations has been fixed at 200 N. In MHypEA, population size as well as number of generations has been fixed at 10 N. The main reason for these settings is to reduce the computational effort required to cluster the modules. Also from the literature it has been observed that though the multi-objective search techniques were able to improve the quality of clustering (Praditwong et al., 2011); they did take two orders of magnitude longer than the single-objective algorithms, so there is clearly a scope for further improvement in the space of quality of results versus required computational effort. By considering these issues, the number of generations has been set to 10 N in MHypEA without compromising on the quality of the solutions. The comparison could be considered as fair comparison, as the same test problems have been attempted and the number of generations is rather decreased in comparison to the other reported algorithms in the literature. All the algorithms have been implemented in MATLAB 7.6.0, on an Intel® Core$^{TM}$ 2 Duo CPU T6600 @2.20 GHz processor, 3GB RAM and Windows 7 platform.

### 5.5. Methodology

Multi-objective Hyper-heuristic Evolutionary Algorithm has been proposed and implemented for software module clustering problem and two multi-objective approaches (MCA and ECA), each with five objectives have been optimized simultaneously to identify the clusters. The algorithm, MHypEA has been executed for 30 independent runs on each test problem with both the approaches (ECA and MCA). In each run, the solution with the highest MQ has been chosen to be the best solution. The mean and standard deviation of the three main objectives - MQ, intra-edges and inter-edges of the best solutions have been calculated. Though it is a multi-objective optimization algorithm and provides a set of non-dominated solutions rather than a single solution, there are two reasons to select the solution with the highest MQ value as the best solution. All the previous algorithms using multi-objective approaches (Praditwong et al., 2011; Barros, 2012) have been collected the results in the same way. To facilitate the comparison of the hyper-heuristic approach with other algorithms reported in the literature the same procedure has been adopted.

The obtained results have been compared with the reported results of Two-Archive Multi-objective Evolutionary Algorithm (Praditwong et al., 2011), NSGA-II (Barros, 2012), Hill Climbing (Praditwong et al., 2011; Barros, 2012) and Genetic Algorithms (Praditwong, 2011). The comparison is based on two factors – quality of the obtained solutions and the computational effort. As the quality of clustering depends on cohesion and coupling, the three main objectives of MQ, intra-edges and inter-edges have been considered for examining the quality of the solutions. The number of function evaluations has been measured as an estimate of computational effort rather than the wall clock time, to facilitate comparison with most of the reported results in the literature. Intuitively, the wall clock times are dependent on the speeds of the systems (that may vary based on many factors) executing the algorithm and does not provide an equal ground for comparison, unless all the algorithms are executed on the same system configuration. All the results have been statistically analyzed using two tailed *t*-test with 58 degrees of freedom with a confidence level of 95%. A parametric *t*-test has been used for statistical analysis rather than a nonparametric one; as the only data available from the literature are the mean and standard deviation. The highlighted figures in the tables denote comparisons where there is a statistically significant difference at 95% confidence level. In addition, an effect size is also calculated to facilitate the interpretation of substantive significance of the results. In this study, Cohen's *d*, Standardized Mean Difference (SMD) statistic has been reported. As a general rule of thumb, Cohen interpreted an effect size of 0.20 as small in magnitude; those around 0.50 are medium and those around or above 0.80 are large. Intuitively, large effects are always more important than small or medium ones and are highlighted in the tables.

## 6. Results and analysis

This section presents the results obtained by MHypEA for the solution of software module clustering and its comparison with the reported results in the literature. For the sake of convenience, the reported results with single and multi-objective formulations of the problem are presented in different sections.

### 6.1. Single-objective formulation

The results obtained by MHypEA are compared with the reported results of single-objective formulations solved with Hill Climbing (Praditwong et al., 2011; Barros, 2012) and Genetic Algorithms (Praditwong, 2011). The comparison is possible as Modularization Quality (MQ) has been considered as one of the objectives to optimize in multi-objective formulation; whereas it is the sole objective for optimization in single-objective formulation.

#### 6.1.1. Comparison with HILL Climbing

The values of MQ, intra-edges and inter-edges obtained by Hill Climbing (Praditwong et al., 2011) and MHypEA on the Data set – I with MCA approach are shown in Tables 4–6, and with ECA approach are shown in Tables 7–9, respectively. The 6th column in the table denotes the *p*-Value (a number of *p*-Value below 0.05

**Table 4**
Comparison of MQ values obtained by Hill Climbing and MHypEA (with MCA approach) on Data set – I.

| Test problem | Hill Climbing (Praditwong et al., 2011) | | MHypEA (MCA approach) | | *p*-Value | Cohen's *d* index |
|---|---|---|---|---|---|---|
| | Mean | Std. D | Mean | Std. D | | |
| mtunis | 2.249 | 0.060 | 2.310 | 0.011 | 2.149e−001 | −1.41 |
| ispell | 2.337 | 0.022 | 2.334 | 0.027 | 9.410e−001 | 0.12 |
| rcs | 2.218 | 0.020 | 2.204 | 0.028 | 7.276e−001 | 0.57 |
| bison | 2.639 | 0.041 | 2.653 | 0.032 | 7.775e−001 | −0.38 |
| grappa | 12.676 | 0.017 | 12.476 | 0.101 | 2.303e−003 | 2.76 |
| incl | 13.568 | 0.035 | 13.492 | 0.295 | 4.715e−001 | 0.36 |

**Table 5**
Comparison of Intra-edges obtained by Hill Climbing and MHypEA (with MCA approach) on Data set – I.

| Test problem | Hill Climbing (Praditwong et al., 2011) | | MHypEA (MCA approach) | | *p*-Value | Cohen's *d* index |
|---|---|---|---|---|---|---|
| | Mean | Std. D | Mean | Std. D | | |
| mtunis | 22.643 | 3.489 | 26.333 | 1.516 | 1.164e−012 | −1.37 |
| ispell | 25.356 | 3.030 | 29.667 | 2.581 | 3.491e−014 | −1.53 |
| rcs | 36.810 | 6.742 | 49.500 | 9.306 | 1.245e−024 | −1.56 |
| bison | 40.225 | 5.994 | 49.933 | 4.593 | 2.248e−023 | −1.81 |
| grappa | 82.167 | 3.175 | 98.625 | 9.023 | 1.737e−033 | −2.43 |
| incl | 140.510 | 9.862 | 138.213 | 11.985 | 9.274e−003 | 0.20 |

**Table 6**
Comparison of Inter-edges obtained by Hill Climbing and MHypEA (with MCA approach) on Data set – I.

| Test problem | Hill Climbing (Praditwong et al., 2011) | | MHypEA (MCA approach) | | *p*-Value | Cohen's *d* index |
|---|---|---|---|---|---|---|
| | Mean | Std. D | Mean | Std. D | | |
| mtunis | 69.724 | 6.914 | 61.333 | 3.032 | 4.800e−021 | 1.57 |
| ispell | 156.299 | 5.984 | 146.667 | 5.163 | 1.112e−022 | 1.72 |
| rcs | 253.372 | 13.446 | 227.000 | 18.612 | 3.234e−033 | 1.62 |
| bison | 278.546 | 11.855 | 258.133 | 9.187 | 3.651e−032 | 1.92 |
| grappa | 426.675 | 6.277 | 392.750 | 16.853 | 4.366e−043 | 2.66 |
| incl | 439.992 | 19.710 | 444.834 | 23.071 | 1.515e−004 | −0.22 |

**Table 7**
Comparison of MQ values obtained by Hill Climbing and MHypEA (with ECA approach) on Data set – I.

| Test problem | Hill Climbing (Praditwong et al., 2011) | | MHypEA (ECA approach) | | *p*-Value | Cohen's *d* index |
|---|---|---|---|---|---|---|
| | Mean | Std. D | Mean | Std. D | | |
| mtunis | 2.249 | 0.060 | 2.314 | 0.000 | 1.514e−001 | −1.53 |
| ispell | 2.337 | 0.022 | 2.341 | 0.021 | 9.162e−001 | −0.18 |
| rcs | 2.218 | 0.020 | 2.242 | 0.019 | 5.082e−001 | −1.23 |
| bison | 2.639 | 0.041 | 2.651 | 0.026 | 8.004e−001 | −0.34 |
| grappa | 12.676 | 0.017 | 12.584 | 0.050 | 5.641e−002 | 2.46 |
| incl | 13.568 | 0.035 | 13.518 | 0.052 | 3.570e-001 | 1.12 |

**Table 8**
Comparison of Intra-edges obtained by Hill Climbing and MHypEA (with ECA approach) on Data set – I.

| Test problem | Hill Climbing (Praditwong et al., 2011) | | MHypEA (ECA approach) | | *p*-Value | Cohen's *d* index |
|---|---|---|---|---|---|---|
| | Mean | Std. D | Mean | Std. D | | |
| mtunis | 22.643 | 3.489 | 27.000 | 0.000 | 1.662e−018 | −1.76 |
| ispell | 25.356 | 3.030 | 31.066 | 2.503 | 2.936e−019 | −2.05 |
| rcs | 36.810 | 6.742 | 51.600 | 6.538 | 4.585e−030 | −2.22 |
| bison | 40.225 | 5.994 | 54.100 | 4.279 | 1.566e−031 | −2.66 |
| grappa | 82.167 | 3.175 | 105.600 | 7.861 | 4.371e−043 | −3.90 |
| incl | 140.510 | 9.862 | 148.453 | 3.002 | 1.506e−017 | −1.08 |

**Table 9**
Comparison of Inter-edges obtained by Hill Climbing and MHypEA (with ECA approach) on Data set – I.

| Test problem | Hill Climbing (Praditwong et al., 2011) | | MHypEA (ECA approach) | | *p*-Value | Cohen's *d* index |
|---|---|---|---|---|---|---|
| | Mean | Std. D | Mean | Std. D | | |
| mtunis | 69.724 | 6.914 | 60.000 | 0.000 | 5.505e-028 | 1.98 |
| ispell | 156.299 | 5.984 | 140.867 | 4.006 | 2.596e-034 | 3.03 |
| rcs | 253.372 | 13.446 | 222.800 | 12.076 | 2.166e-039 | 2.39 |
| bison | 278.546 | 11.855 | 249.800 | 10.559 | 1.802e-039 | 2.56 |
| grappa | 426.675 | 6.277 | 381.800 | 15.722 | 1.499e-050 | 3.74 |
| incl | 439.992 | 19.710 | 428.870 | 7.098 | 5.344e-017 | 0.75 |

**Table 10**
Comparison of number of function evaluations spent by Hill Climbing and MHypEA on Data set – I.

| Test problem | Hill Climbing (Praditwong et al., 2011) | | MHypEA |
|---|---|---|---|
| | Mean | Std. D | |
| mtunis | 934.367 | 204.558 | 40,200 |
| ispell | 1823.333 | 509.483 | 57,840 |
| rcs | 3291.000 | 884.664 | 84,390 |
| bison | 5388.267 | 1453.368 | 137,270 |
| grappa | 68,429.867 | 27,558.047 | 740,460 |
| incl | 125,424.433 | 45,517.068 | 3,029,340 |

**Table 11**
Comparison of MQ values obtained by Hill Climbing and MHypEA (with MCA approach) on Data set – II.

| Test problem | Hill Climbing (Barros, 2012) | | MHypEA (MCA approach) | | *p*-Value |
|---|---|---|---|---|---|
| | Mean | Std. D | Mean | Std. D | |
| JODAMONEY | 1.69 | 0.08 | 2.74 | 0.01 | 1.547e−026 |
| JXLSREADER | 2.13 | 0.11 | 3.59 | 0.00 | 5.131e−031 |
| SEEMP | 2.27 | 0.17 | 4.65 | 0.00 | 2.912e−038 |
| JAVAOCR | 2.43 | 0.13 | 8.92 | 0.00 | 2.068e−065 |
| SERVLET | 2.54 | 0.20 | 11.05 | 0.08 | 3.710e−063 |
| JUNIT | 2.00 | 0.15 | 10.77 | 0.00 | 6.011e−072 |

**Table 12**
Comparison of MQ values obtained by Hill Climbing and MHypEA (with ECA approach) on Data set – II.

| Test problem | Hill Climbing (Barros, 2012) | | MHypEA (ECA approach) | | *p*-Value |
|---|---|---|---|---|---|
| | Mean | Std. D | Mean | Std. D | |
| JODAMONEY | 1.69 | 0.08 | 2.74 | 0.00 | 5.681e−027 |
| JXLSREADER | 2.13 | 0.11 | 3.59 | 0.00 | 2.043e−031 |
| SEEMP | 2.27 | 0.17 | 4.65 | 0.00 | 2.912e−038 |
| JAVAOCR | 2.43 | 0.13 | 8.98 | 0.06 | 1.777e−061 |
| SERVLET | 2.54 | 0.20 | 10.98 | 0.09 | 9.811e−063 |
| JUNIT | 2.00 | 0.15 | 10.69 | 0.05 | 3.865e−068 |

**Table 13**
Comparison of number of function evaluations spent by Hill Climbing and MHypEA on Data set –II.

| Test problem | Hill Climbing (Barros, 2012) | MHypEA |
|---|---|---|
| JODAMONEY | 1,352,000 | 67,860 |
| JXLSREADER | 1,458,000 | 73,170 |
| SEEMP | 1,922,000 | 96,410 |
| JAVAOCR | 6,962,000 | 348,690 |
| SERVLET | 20,000,000 | 1,001,000 |
| JUNIT | 28,322,000 | 1,417,290 |

is considered statistically significant) and the shaded figures denote comparisons where there is significant difference at 95% confidence level. From the Table 4 it is observable that hill-climbing algorithm obtained higher values of MQ than MHypEA with MCA approach in 4 out of 6 problems, in which the results are significant in one problem and with ECA approach the results are not significant. In Table 4, effect size is large in two instances pointing that the differences in the means of MQ of the two approaches are consistent. An effect size of 2.0 indicates a nonoverlap of 81.1% in the two approaches. But there is a strong evidence to suggest that MHypEA outperformed hill-climbing algorithm in obtaining higher number of intra-edges and lower number of inter-edges on all the test problems (except in ‘incl’ with MCA approach) with statistically significant differences in the means. In Tables 5 and 6 reporting the intra-edges and inter-edges, the effect sizes are large, except in the case of ‘incl’. The comparison of ECA approach reveals that the effect size is large for intra-edges and inter-edges in all the test instances, justifying that the MHypEA results are significant.

Table 10 reports the number of function evaluations spent by Hill Climbing and MHypEA. The number of evaluations of the Hill Climbing approach is averaged over 30 independent runs. The Hill Climbing approach has not been executed for a fixed number of evaluations and it has been terminated when it can no longer find a better MQ (Praditwong et al., 2011); while MHypEA has been executed for a fixed number of function evaluations. Therefore the comparison is not fair in this case.

The values of MQ obtained by Hill Climbing (Barros, 2012) and MHypEA with MCA approach on Data set – II are shown in Table 11 and MHypEA with ECA approach on Data set – II are shown in Table 12. The performance of MHypEA is found to be excellent on Data set – II with both approaches and as is apparent from the tabulated results that on all the test problems the values of MQ attained by MHypEA are far higher than those obtained by Hill Climbing. On large problem instances, MQ found by MHypEA is nearly 4 to 5 times more than MQ found by Hill Climbing. Furthermore, the lower values of standard deviation point towards the capability of MHypEA in producing consistent results. The results are statistically significant at 95% confidence level on all the test problems.

Table 13 reports the number of function evaluations spent by Hill Climbing and MHypEA on data set – II. It is perceptible form the table that the number of function evaluations spent by the Hill Climbing approach is nearly 20 times more than that of MHypEA.

### *6.1.2. Comparison with Genetic Algorithms*

The values of MQ obtained by Genetic Algorithms with Group Number Encoding (GNE) (Praditwong, 2011) and MHypEA with MCA approach are shown in Table 14 and MHypEA with ECA approach are shown in Table 15. GNE obtained higher values of MQ than MHypEA in two test problems wherein the results are not statistically significant. MHypEA obtained higher values of MQ than GNE (Praditwong, 2011) with both the approaches (MCA and ECA) in four test problems out of six examined, in which three results are statistically significant. The reported effect sizes in Tables 14 and 15 presenting the comparison of GNE and MHypEA (with MCA approach) reveals that in almost all the cases the effect size is large (except in the case of ‘ispell’).

The values of MQ obtained by Grouping Genetic Algorithm (GGA) (Praditwong, 2011) and MHypEA with MCA approach are shown in Table 16 and MHypEA with ECA approach are shown in Table 17. From Table 16 it is observable that MHypEA with MCA approach is able to obtain better values for MQ in four instances, in which the results are significant in two instances. MHypEA with ECA approach is able to obtain higher values of MQ in all the test problems when compared with GGA, in which 3 results are statistically significant. The number of function evaluations is reported in Table 18.

## *6.2. Multi-objective formulation*

In this section the results of MHypEA are compared with the reported results of Two-Archive Multi-objective Evolutionary Algorithm (Praditwong et al., 2011) on Data set – I and NSGA-II (Barros, 2012) on Data set – II. In this case the comparison can be considered as a fair comparison as all the algorithms have optimized the same set of objectives and also the results obtained after experimentation have been collected in the same fashion.

**Table 14**
Comparison of MQ values obtained by GNE and MHypEA (with MCA approach).

| Test problem | GNE (Praditwong, 2011) | | MHypEA (MCA approach) | | *p*-Value | Cohen's *d* index |
|---|---|---|---|---|---|---|
| | Mean | Std. D | Mean | Std. D | | |
| mtunis | 2.289 | 0.026 | 2.310 | 0.011 | 5.521e−001 | −1.05 |
| ispell | 2.346 | 0.014 | 2.334 | 0.027 | 7.466e−001 | 0.55 |
| rcs | 2.263 | 0.020 | 2.204 | 0.028 | 1.456e−001 | 2.42 |
| bison | 2.289 | 0.026 | 2.653 | 0.032 | 2.094e−011 | −12.48 |
| grappa | 10.828 | 0.097 | 12.476 | 0.101 | 5.104e−028 | −16.64 |
| incl | 8.143 | 0.089 | 13.492 | 0.295 | 5.126e−048 | −24.54 |

**Table 15**
Comparison of MQ values obtained by GNE and MHypEA (with ECA approach).

| Test problem | GNE (Praditwong, 2011) | | MHypEA (ECA approach) | | *p*-Value | Cohen's *d* index |
|---|---|---|---|---|---|---|
| | Mean | Std. D | Mean | Std. D | | |
| mtunis | 2.289 | 0.026 | 2.314 | 0.000 | 3.992e−001 | −1.35 |
| ispell | 2.346 | 0.014 | 2.341 | 0.021 | 8.841e−001 | 0.28 |
| rcs | 2.263 | 0.020 | 2.242 | 0.019 | 5.625e−001 | 1.07 |
| bison | 2.289 | 0.026 | 2.651 | 0.026 | 4.240e−012 | −13.92 |
| grappa | 10.828 | 0.097 | 12.584 | 0.050 | 7.927e−033 | −22.75 |
| incl | 8.143 | 0.089 | 13.518 | 0.052 | 1.480e−060 | −73.74 |

**Table 16**
Comparison of MQ values obtained by GGA and MHypEA (with MCA approach).

| Test problem | GGA (Praditwong, 2011) | | MHypEA (MCA approach) | | *p*-Value | Cohen's *d* index |
|---|---|---|---|---|---|---|
| | Mean | Std. D | Mean | Std. D | | |
| mtunis | 2.231 | 0.051 | 2.310 | 0.011 | 8.755e−002 | −2.14 |
| ispell | 2.338 | 0.016 | 2.334 | 0.027 | 9.162e−001 | 0.18 |
| rcs | 2.232 | 0.029 | 2.204 | 0.028 | 5.231e−001 | 0.98 |
| bison | 2.231 | 0.051 | 2.653 | 0.032 | 5.605e−011 | −9.91 |
| grappa | 12.454 | 0.144 | 12.476 | 0.101 | 8.085e−001 | −0.17 |
| incl | 13.139 | 0.170 | 13.492 | 0.295 | 6.291e−003 | −1.46 |

**Table 17**
Comparison of MQ values obtained by GGA and MHypEA (with ECA approach).

| Test problem | GGA (Praditwong, 2011) | | MHypEA (ECA approach) | | *p*-Value | Cohen's *d* index |
|---|---|---|---|---|---|---|
| | Mean | Std. D | Mean | Std. D | | |
| mtunis | 2.231 | 0.051 | 2.314 | 0.000 | 4.875e−002 | −2.30 |
| ispell | 2.338 | 0.016 | 2.341 | 0.021 | 9.322e−001 | −0.16 |
| rcs | 2.232 | 0.029 | 2.242 | 0.019 | 8.034e−001 | −0.40 |
| bison | 2.231 | 0.051 | 2.651 | 0.026 | 2.002e−011 | −10.37 |
| grappa | 12.454 | 0.144 | 12.584 | 0.050 | 1.113e−001 | −1.20 |
| incl | 13.139 | 0.170 | 13.518 | 0.052 | 4.606e−005 | −3.01 |

**Table 18**
Comparison of number of function evaluations spent by GGA, GNE and MHypEA.

| Test problem | GGA (Praditwong, 2011) | GNE (Praditwong, 2011) | MHypEA |
|---|---|---|---|
| mtunis | 800,000 | 800,000 | 40,200 |
| ispell | 1,152,000 | 1,152,000 | 57,840 |
| rcs | 1,682,000 | 1,682,000 | 84,390 |
| bison | 2,738,000 | 2,738,000 | 137,270 |
| grappa | 14,792,000 | 14,792,000 | 740,460 |
| incl | 60,552,000 | 60,552,000 | 3,029,340 |

### 6.2.1. Comparison with Two-Archive Multi-objective Evolutionary Algorithm

The values of MQ, intra-edges and inter-edges obtained by Two-Archive Multi-objective Evolutionary Algorithm (Praditwong et al., 2011) and MHypEA, with MCA approach are shown in Tables 19, 20 and 21, respectively. From the Table 19 it can be observed that MHypEA outperformed Two-Archive Algorithm in all the test problems, in which three results are statistically significant. Interestingly all the three results are pertaining to the large test problems, highlighting the efficacy of MHypEA in clustering complex systems with large number of nodes and edges. Moreover, the intra-edges and inter-edges reported by MHypEA is excellent in all the test problems with significant results. The comparison of Two-Archive algorithm and MHypEA with MCA approach based on Cohen's *d* effect size indicator substantiates that in most of the test problems, the effect sizes are large enough and validates the efficacy of MHypEA in identifying clusters with minimum coupling and maximum cohesion.

Table 22 reports the MQ values attained by both the algorithms with ECA approach. Though the results are not significant, MHypEA is able to achieve better values in comparison. Tables 23 and 24 present the intra and inter-edges obtained by the algorithms. In all the test problems MHypEA is able to achieve higher values for the intra-edges (high cohesion) and lower values for the inter-edges (low coupling) signifying its ability to suggest good modular structures. The comparison of Two-Archive algorithm and MHypEA with ECA approach based on Cohen's d effect size indicator reports that the effect sizes are small with respect to MQ and medium in most instances and large enough in some of the instances with respect to intra-edges and inter-edges.

**Table 19**
Comparison of MQ values of the best solutions obtained by Two-Archive Algorithm and MHypEA with MCA approach.

| Test problem | Two-Archive Algorithm (Praditwong et al., 2011) | | MHypEA | | *p*-Value | Cohen's *d* index |
|---|---|---|---|---|---|---|
| | Mean | Std. D | Mean | Std. D | | |
| mtunis | 2.294 | 0.013 | 2.310 | 0.011 | 5.737e−001 | −1.32 |
| ispell | 2.269 | 0.043 | 2.334 | 0.027 | 1.836e−001 | −1.81 |
| rcs | 2.145 | 0.034 | 2.204 | 0.028 | 1.994e−001 | −1.89 |
| bison | 2.416 | 0.038 | 2.653 | 0.032 | 7.877e−006 | −6.74 |
| grappa | 11.586 | 0.106 | 12.476 | 0.101 | 2.261e−015 | −8.59 |
| incl | 11.811 | 0.351 | 13.492 | 0.295 | 1.590e−016 | −5.18 |

**Table 20**
Comparison of Intra-edges of the best solutions obtained by Two-Archive Algorithm and MHypEA with MCA approach.

| Test problem | Two-Archive Algorithm (Praditwong et al., 2011) | | MHypEA | | *p*-Value | Cohen's *d* index |
|---|---|---|---|---|---|---|
| | Mean | Std. D | Mean | Std. D | | |
| mtunis | 24.633 | 2.092 | 26.333 | 1.516 | 8.001e−006 | −0.93 |
| ispell | 23.100 | 3.220 | 29.667 | 2.581 | 1.557e−021 | −2.25 |
| rcs | 45.133 | 15.335 | 49.500 | 9.306 | 1.079e−005 | −0.34 |
| bison | 40.367 | 8.231 | 49.933 | 4.593 | 4.000e−021 | −1.43 |
| grappa | 84.767 | 11.190 | 98.625 | 9.023 | 4.697e−024 | −1.36 |
| incl | 91.767 | 14.024 | 138.213 | 11.985 | 2.463e−049 | −3.56 |

**Table 21**
Comparison of inter-edges of the best solutions obtained by Two-Archive Algorithm and MHypEA with MCA approach.

| Test problem | Two-Archive Algorithm (Praditwong et al., 2011) | | MHypEA | | *p*-Value | Cohen's *d* index |
|---|---|---|---|---|---|---|
| | Mean | Std. D | Mean | Std. D | | |
| mtunis | 64.733 | 4.185 | 61.333 | 3.032 | 3.811e−009 | 0.93 |
| ispell | 159.800 | 6.440 | 146.667 | 5.163 | 6.521e−029 | 2.25 |
| rcs | 235.733 | 30.669 | 227.000 | 18.612 | 6.020e−009 | 0.34 |
| bison | 277.267 | 16.463 | 258.133 | 9.187 | 1.848e−028 | 1.43 |
| grappa | 420.467 | 22.380 | 392.750 | 16.853 | 4.917e−032 | 1.39 |
| incl | 536.467 | 28.048 | 444.834 | 23.071 | 8.437e−058 | 3.56 |

**Table 22**
Comparison of MQ values of the best solutions obtained by Two-Archive Algorithm and MHypEA with ECA approach.

| Test problem | Two-Archive Algorithm (Praditwong et al., 2011) | | MHypEA | | *p*-Value | Cohen's *d* index |
|---|---|---|---|---|---|---|
| | Mean | Std. D | Mean | Std. D | | |
| mtunis | 2.314 | 0.000 | 2.314 | 0.000 | – | – |
| ispell | 2.339 | 0.022 | 2.341 | 0.021 | 9.580e−001 | −0.09 |
| rcs | 2.239 | 0.022 | 2.242 | 0.019 | 9.356e−001 | −0.14 |
| bison | 2.648 | 0.029 | 2.651 | 0.026 | 9.443e−001 | −0.10 |
| grappa | 12.578 | 0.053 | 12.584 | 0.050 | 9.187e−001 | −0.11 |
| incl | 13.511 | 0.059 | 13.518 | 0.052 | 9.087e−001 | −0.12 |

**Table 23**
Comparison of Intra-edges of the best solutions obtained by Two-Archive Algorithm and MHypEA with ECA approach.

| Test problem | Two-Archive Algorithm (Praditwong et al., 2011) | | MHypEA | | *p*-Value | Cohen's *d* index |
|---|---|---|---|---|---|---|
| | Mean | Std. D | Mean | Std. D | | |
| mtunis | 27.000 | 0.000 | 27.000 | 0.000 | – | – |
| ispell | 30.033 | 2.798 | 31.066 | 2.503 | 1.700e−002 | −0.38 |
| rcs | 47.567 | 7.859 | 51.600 | 6.538 | 2.676e−007 | −0.55 |
| bison | 52.800 | 6.217 | 54.100 | 4.279 | 3.196e−002 | −0.24 |
| grappa | 101.167 | 8.301 | 105.600 | 7.861 | 1.172e−007 | −0.54 |
| incl | 140.200 | 3.836 | 148.453 | 3.002 | 1.491e−024 | −2.39 |

**Table 24**
Comparison of Inter-edges of the best solutions obtained by Two-Archive Algorithm and MHypEA with ECA approach.

| Test problem | Two-Archive Algorithm (Praditwong et al., 2011) | | MHypEA | | *p*-Value | Cohen's *d* index |
|---|---|---|---|---|---|---|
| | Mean | Std. D | Mean | Std. D | | |
| mtunis | 60.000 | 0.000 | 60.000 | 0.000 | – | – |
| ispell | 145.933 | 5.595 | 140.867 | 4.006 | 1.573e−012 | 1.04 |
| rcs | 230.867 | 15.719 | 222.800 | 12.076 | 1.412e−011 | 0.57 |
| bison | 252.400 | 12.434 | 249.800 | 10.559 | 4.327e−003 | 0.22 |
| grappa | 387.667 | 16.601 | 381.800 | 15.722 | 5.064e−007 | 0.36 |
| incl | 439.600 | 7.673 | 428.870 | 7.098 | 5.189e−022 | 1.45 |

**Table 25**
Comparison of number of function evaluations spent by Two-Archive Algorithm and MHypEA.

| Test problem | Two-Archive Algorithm (Praditwong et al., 2011) | MHypEA |
|---|---|---|
| mtunis | 800,000 | 40,200 |
| ispell | 1,152,000 | 57,840 |
| rcs | 1,682,000 | 84,390 |
| bison | 2,738,000 | 137,270 |
| grappa | 14,792,000 | 740,460 |
| incl | 60,552,000 | 3,029,340 |

**Table 28**
Comparison of number of function evaluations spent by NSGA-II and MHypEA.

| Test problem | NSGA-II (Barros, 2012) | MHypEA |
|---|---|---|
| JODAMONEY | 1,352,000 | 67,860 |
| JXLSREADER | 1,458,000 | 73,170 |
| SEEMP | 1,922,000 | 96,410 |
| JAVAOCR | 6,962,000 | 348,690 |
| SERVLET | 20,000,000 | 1,001,000 |
| JUNIT | 28,322,000 | 1,417,290 |

Table 25 depicts the number of function evaluations spent by both the algorithms. Remarkably, MHypEA is able to obtain high quality solutions with a computational effort of nearly one-twentieth of the computational effort required by the Two-Archive Multi-objective Evolutionary algorithm.

### 6.2.2. Comparison with NSGA-II

This section presents the comparison of performance of MHypEA with the reported results of NSGA-II (Barros, 2012) on Data set – II.

The values of MQ achieved by NSGA-II and MHypEA (with MCA approach) are shown in Table 26. The efficacy of MHypEA in solving Multi-Objective Software Module Clustering problem is indicated by the obtained results. There is a perceptible variation in the values of MQ obtained by the solver algorithms on all the test problems with a statistical difference. Table 27 reports the MQ values found by NSGA-II and MHypEA (with ECA approach). It is very interesting to observe that MHypEA is effective in suggesting good modular structures. The significance of the results at 95% confidence level along with the reported effect sizes validates the efficacy of MHypEA over NSGA-II.

The computational effort of the solver algorithms expressed in terms of number of function evaluations is shown in Table 28. From the results it is also evident that the computational effort of MHypEA is nearly one-twentieth of that of NSGA-II.

## 6.3. Revisiting research questions

**RQ1:** *Is the hyper-heuristic approach able to identify clusters better than other examined algorithms?*

To answer RQ1, a Multi-objective Hyper-heuristic Evolutionary Algorithms has been developed with a generic framework and has been implemented on two data sets as referenced in the literature. The effectiveness of the hyper-heuristic approach has been compared with the other reported results of single and multi-objective optimization algorithms. The quality of clustering has been measured based on the values of Modularization Quality (MQ), intra-edges (a measure of cohesion) and inter-edges (a measure of coupling). From the results of the study it is observable that MHypEA is competitive in obtaining better values for MQ (in many instances) in comparison to the single-objective approaches whose sole objective for optimization is MQ. The comparison with the multi-objective approaches reveals that the hyper-heuristic approach is successful in on all the three parameters chosen for testing the quality of clustering. From the comparison there is a strong evidence to claim that the hyper-heuristic approach is effective identifying clusters better than other examined algorithms.

**RQ2:** *Does the hyper-heuristic approach require a lower number of function evaluations than the other multi-objective algorithms (Two-Archive and NSGA-II) to cluster the modules?*

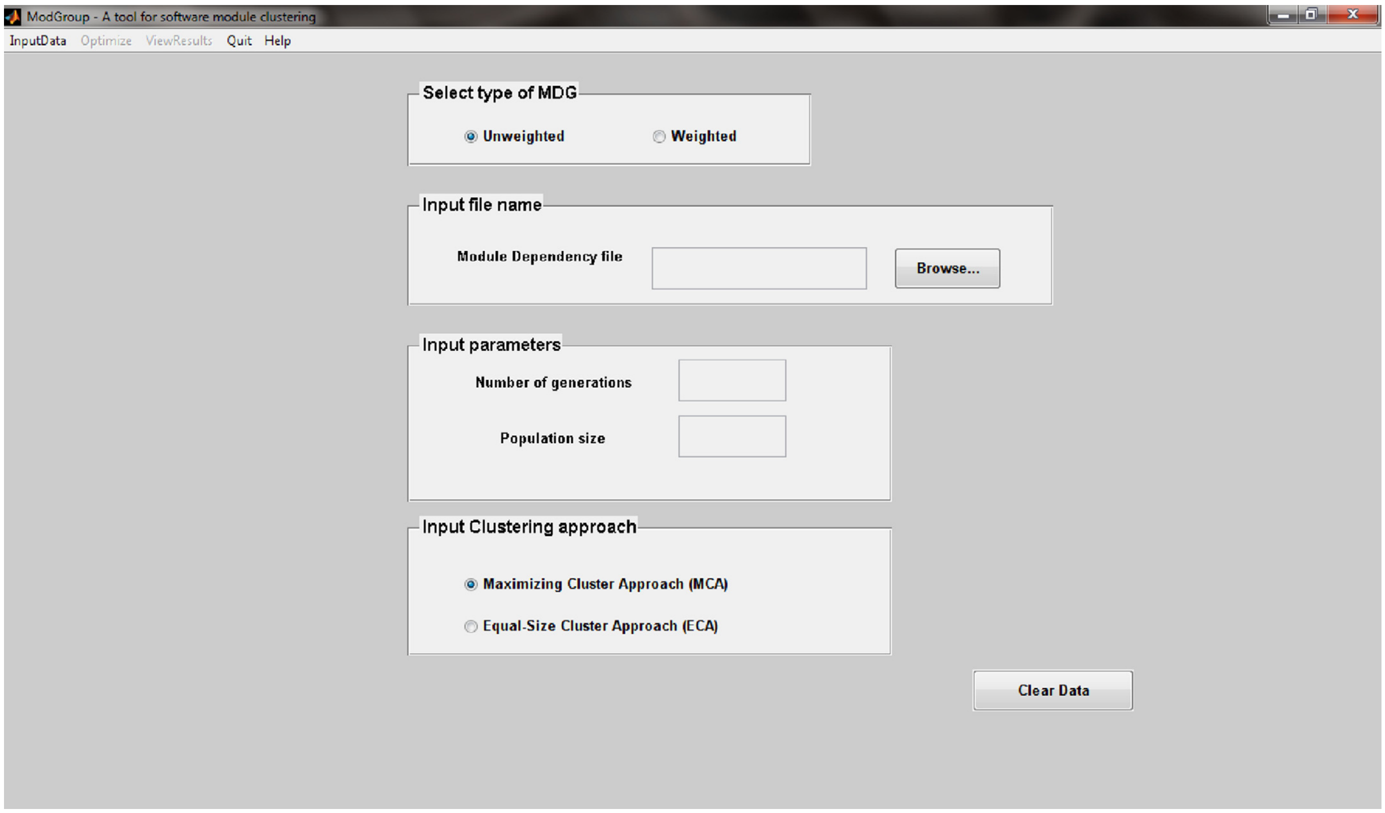
To answer RQ2, the number of function evaluations is computed as a measure of computational effort. From the Table 25 (comparison with Two-Archive algorithm) and Table 28 (comparison with NSGA-II) it is perceptible that the hyper-heuristic approach is able to identify the clusters with a computational effort of nearly one-twentieth of the computational effort required by other two multi-objective evolutionary algorithms. These results lead us to conclude that the hyper-heuristic approach requires lower number of function evaluations than the other multi-objective algorithms and satisfactorily answers RQ2.

**Table 26**
Comparison of MQ values of the best solutions obtained by NSGA-II and MHypEA (with MCA approach).

| Test problem | NSGA-II (Barros, 2012) | | MHypEA (MCA approach) | | *p*-Value | Cohen's *d* index |
|---|---|---|---|---|---|---|
| | Mean | Std. D | Mean | Std. D | | |
| JODAMONEY | 2.26 | 0.04 | 2.74 | 0.01 | 1.225e−016 | −16.25 |
| JXLSREADER | 2.87 | 0.12 | 3.59 | 0.00 | 8.670e−016 | −8.46 |
| SEEMP | 4.18 | 0.09 | 4.65 | 0.00 | 6.557e−012 | −7.38 |
| JAVAOCR | 7.37 | 0.40 | 8.92 | 0.00 | 3.215e−019 | −5.47 |
| SERVLET | 6.92 | 0.44 | 11.05 | 0.08 | 6.458e−038 | −13.02 |
| JUNIT | 8.22 | 0.46 | 10.77 | 0.00 | 2.516e−028 | −7.83 |

**Table 27**
Comparison of MQ values of the best solutions obtained by NSGA-II and MHypEA (with ECA approach).

| Test problem | NSGA-II (Barros, 2012) | | MHypEA (ECA approach) | | *p*-Value | Cohen's *d* index |
|---|---|---|---|---|---|---|
| | Mean | Std. D | Mean | Std. D | | |
| JODAMONEY | 2.26 | 0.04 | 2.74 | 0.00 | 2.806e−017 | −16.60 |
| JXLSREADER | 2.87 | 0.12 | 3.59 | 0.00 | 4.568e−016 | −8.47 |
| SEEMP | 4.18 | 0.09 | 4.65 | 0.00 | 6.557e−012 | −7.38 |
| JAVAOCR | 7.37 | 0.40 | 8.98 | 0.06 | 9.628e−019 | −5.61 |
| SERVLET | 6.92 | 0.44 | 10.98 | 0.09 | 2.130e−037 | −12.77 |
| JUNIT | 8.22 | 0.46 | 10.69 | 0.05 | 1.699e−026 | −7.54 |


ModGroup - A tool for software module clustering
InputData Optimize ViewResults Quit Help
Browse...
Clear Data

Select type of MDG: [✓ Unweighted] Weighted

Input file name
Module Dependency file

Input parameters
Number of generations
Population size

Input Clustering approach
[✓ Maximizing Cluster Approach (MCA)]
Equal-Size Cluster Approach (ECA)

**Fig. 2.** Interface of ModGroup tool.

### 6.4. Threats to validity and limitations of the study

This section discusses threats to validity and limitations of the results presented. There are primarily two potential threats as discussed below.

#### 6.4.1. External validity

External validity is concerned with generalization. Threats to external validity are conditions that limit our ability to generalize the results of our experiment to industrial practice (Wohlin et al., 2000).

In this work, two data sets of unweighted MDGs (each set consisting of six systems) and one system with a weighted MDG (will be discussed in Section 8) are experimented. The hyper-heuristic approach is found to be effective in finding clusters with high cohesion and low coupling on all the systems studied. Experimentation with more number of weighted MDGs is to be performed to further strengthen the capabilities of hyper-heuristic approach in the module clustering process. As the results of the experiments are mostly based on the MDGs of open-source programs; the alignment of the behavior of these programs with the non-open source programs is to be studied further.

#### 6.4.2. Internal validity

Threats to internal validity are influences that can affect the dependent variables with respect to causality, without the researcher's knowledge. Thus they threat the conclusion about a possible casual relationship between treatment and outcome (Wohlin et al., 2000).

In this paper, the efficacy of hyper-heuristics in module clustering has been analyzed and the results are compared against the reported results in the literature. To facilitate a degree of comparability, the MDGs studied were obtained from the researchers in the field and the key variables used for measuring the performance are in terms of quality of solutions and computational effort. To have a fair comparison, the same procedure as described in the literature (Praditwong et al., 2011) has been adopted to measure the performance of MHypEA and also the algorithm has been executed for 30 independent runs on each system to account for its inherent randomness.

## 7. ModGroup – a CASE tool for software module clustering

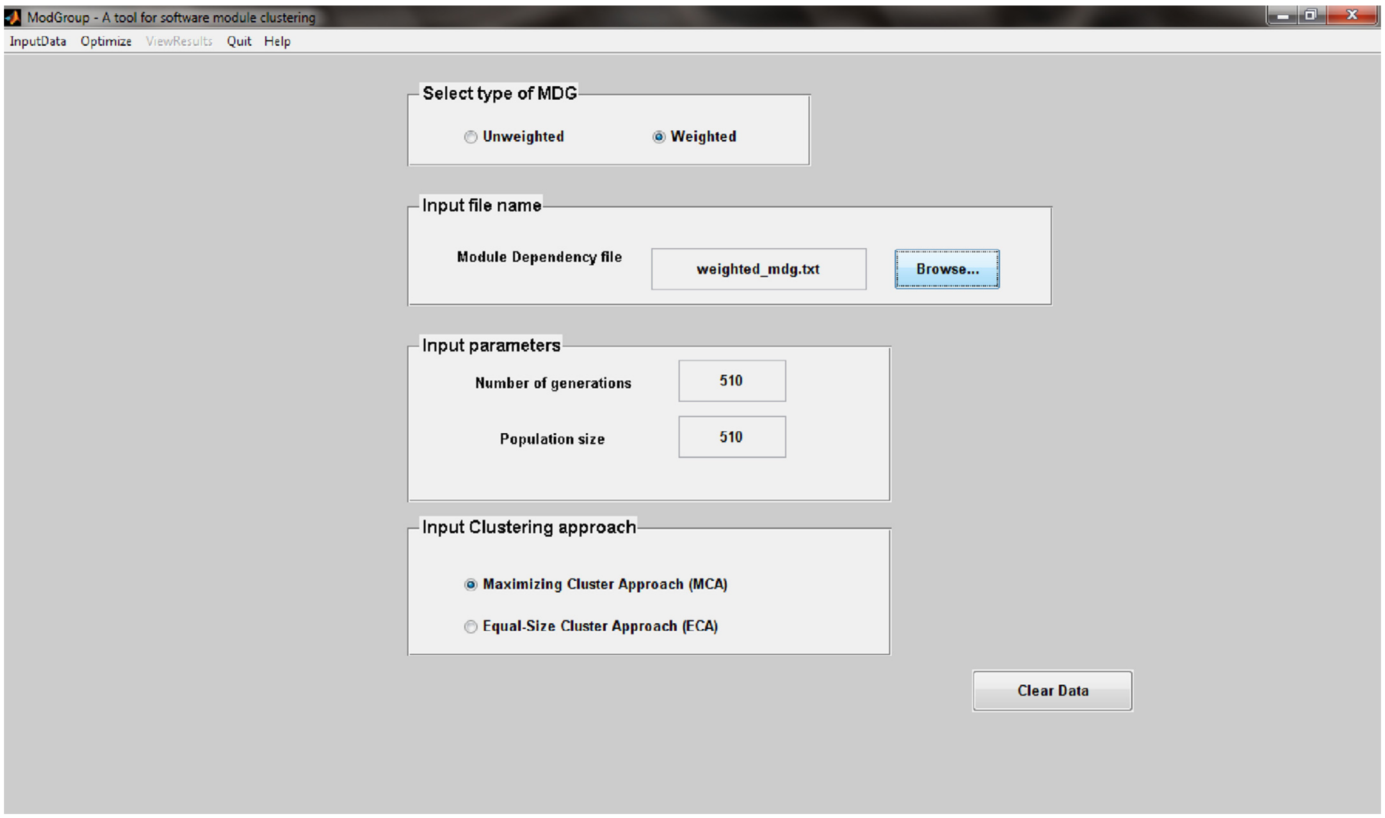
Based on the experimental results and the efficacy of hyper-heuristics in module clustering, efforts have been made to translate the developed algorithm into a CASE tool to assist the software engineers in identifying high quality clusters in less computational effort with the following features.

- It is user-interactive.
- It can cluster weighted and unweighted MDGs.
- It facilitates clustering with MCA and ECA approaches.
- It uses MHypEA, the most effective metaheuristic for clustering that has been developed and experimented during the study.
- It is equipped with a progress bar to indicate the status of evolution.
- It provides a facility to pause/resume/stop the clustering process.
- It can also save/print the results obtained.

This section presents the framework of ModGroup along with its implementation details.

### 7.1. User interface

Fig. 2 shows the user interface of the tool with five options – InputData, Optimize, ViewResults, Quit and Help.


ModGroup - A tool for software module clustering

Select type of MDG: Unweighted [✓ Weighted]

Input file name — Module Dependency file: weighted_mdg.txt

Input parameters — Number of generations: 510; Population size: 510

Input Clustering approach: [✓ Maximizing Cluster Approach (MCA)] Equal-Size Cluster Approach (ECA)

**Fig. 3.** InputData window of ModGroup.

*7.1. InputData*

The InputData option enables the user to input the details pertaining to software module clustering problem. The user can specify the type of MDG either as weighted or unweighted using a radio button. By default, the tool considers the input MDG as an unweighted one. The Module dependency file is a text file containing the data regarding the connections or edges among the modules in the MDG. For an unweighted MDG, it contains two fields with the names of the modules that are connected together and in case of weighted MDG the third field contains the weight or strength of the relationship between the two modules that are connected together. The algorithmic parameters of population size and number of generations have been taken by default as 10 N, N being the number of modules in the MDG. The user can opt for either MCA or ECA approach for clustering the modules. A snapshot of Inputdata window of the tool is shown in Fig. 3.

*7.2. Optimize*

The optimization process has been equipped with a progress bar to indicate the status of optimization with a rough estimate on the remaining time for completion. The tool also updates the user throughout the optimization process with the display of objective function values. The user has been provided with a facility to pause the optimization process in-between the execution to view the detailed clustering results obtained so far. Thereafter either the user can resume the execution or can stop the execution process. An instance of optimization window is shown in Fig. 4.

*7.3. ViewResults*

The ViewResults option provides a detailed listing of all the solutions obtained by the algorithm. The tool selects a solution with the highest MQ value as the recommended solution to maintain uniformity with the prevailing practices as reported in the literature (Praditwong et al., 2011; Barros, 2012). All other solutions are listed in the decreasing order of their MQ values. For each solution, it lists the identified clusters along with the objective function values. This option has been provided with *Save* and *Print* facilities.

*7.4. Quit*

It enables the user to exit from the tool.

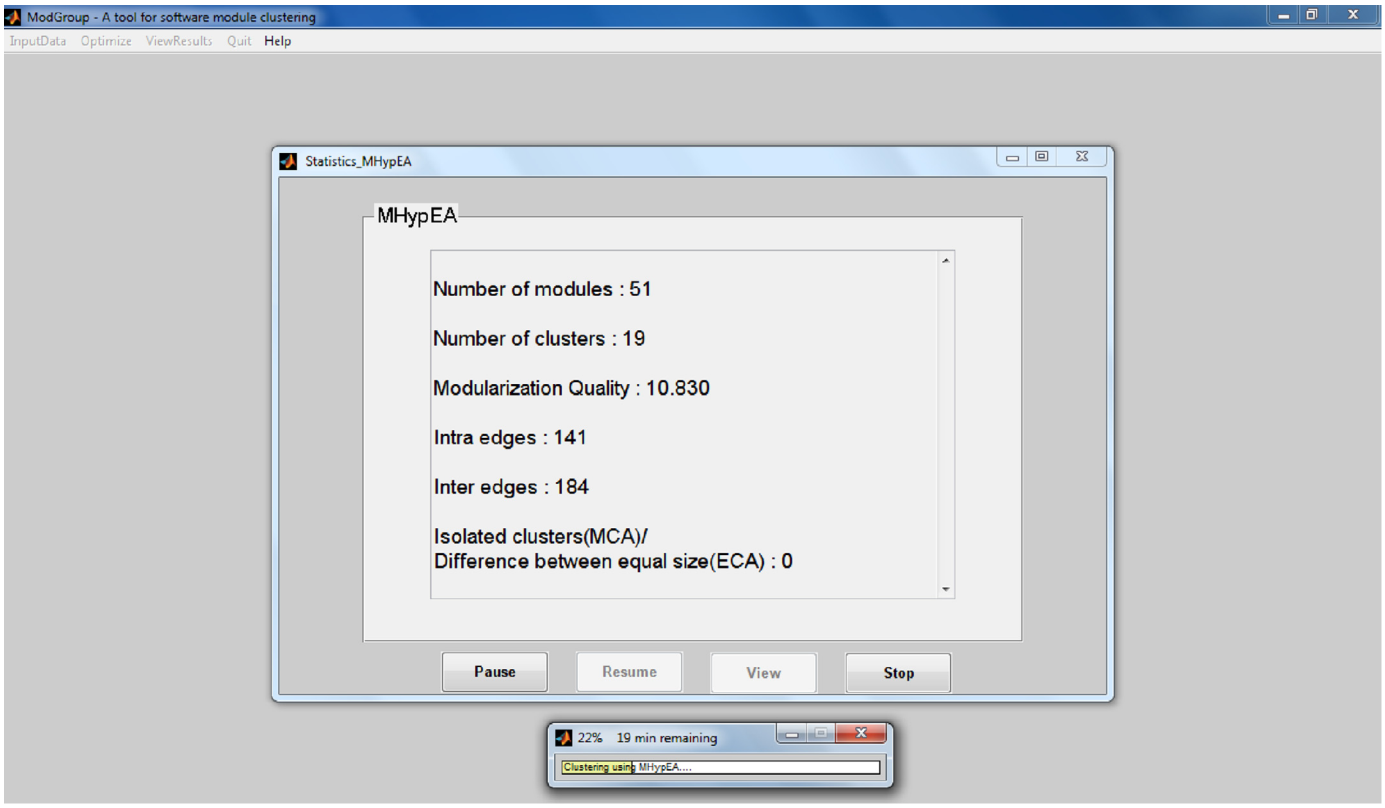
*7.5. Help*

The Help option provides the user with a detailed description on the topics – how to use the tool, about the problem and description of MHypEA algorithm.

## 8. Validation of ModGroup on a real-world application

The main purpose of this section is to demonstrate the utility of the CASE tool on a real-world application (weighted MDG) for software module clustering. The application is related to ‘Address Verification and Fraud Detection’ kindly made available by a reputed multi-national software development company. The data that has been provided for experimentation is a Software component consisting of 51 modules with 119 inter-connections. The strength of these inter-connections has been weighted between 1 and 6 (1 indicating the lowest and 6 indicating the highest). The provided data has been visualized to show its structure as MDG and is presented in Fig. 5.

The statistics of the results obtained by the ModGroup with MCA approach is shown in Fig. 6. The tool is able to identify 19 clusters with 141 intra-edges, 184 inter-edges and a MQ of 10.830.


ModGroup - A tool for software module clustering
InputData Optimize ViewResults Quit Help
Statistics_MHypEA
MHypEA
Pause
Resume
View
Stop
22% 19 min remaining
Clustering using MHypEA....

Number of modules : 51

Number of clusters : 19

Modularization Quality : 10.830

Intra edges : 141

Inter edges : 184

Isolated clusters(MCA)/
Difference between equal size(ECA) : 0

**Fig. 4.** Optimization window of the tool – ModGroup.

The first 5 solutions provided by ModGroup with MCA approach are included in Appendix for reference.

## 9. Related work

The key driving factor behind the need for semi-automated or automated techniques is scalability. Due to the rapid increase in the size and complexities of the systems, it is becoming a challenge for the software engineers to maintain the systems. Software maintenance is the process of enhancing and optimizing deployed software, as well as remedying defects. It involves changes to the software in order to correct defects and deficiencies found during field usage as well as the addition of new functionality to improve the software's usability and applicability. The main area of research that has been widely reported in this phase is modularization.

The purpose of module clustering is to obtain good modular structures with high degree of cohesion and low degree of coupling. Mancoridis et al. (1998) were the first to develop methods to facilitate the automatic recovery of the modular structure of a software system from its source code using Hill Climbing and Genetic Algorithms. Doval et al. (1999) described a technique for finding good Module Dependency Graph (MDG) partitions with relatively independent subsystems that contain modules which are highly inter-dependent. They applied Genetic Algorithm on a medium sized software system. Lutz (2001) devised a complexity measure based on minimum description length principle to compare hierarchical decompositions and used it as a fitness function for a Genetic Algorithm to successfully explore the space of all possible hierarchical decompositions of a system.

Harman et al. (2002) introduced a normalized representation for software modularization which reduces the size of the search space and also suggested a new crossover operator to promote the formation and retention of building blocks. They reported that the new crossover operator is better suited to genetic approaches than standard crossover. Mitchell (2002a, 2002b) and Mitchell and Mancoridis (2002) addressed several aspects of the software clustering problem. They implemented several heuristic search algorithms that automatically cluster the source code into subsystems. They developed a tool for automatic decomposition of software structures from its source code and conducted extensive evaluations via case studies and experiments. They also developed a tool called CRAFT that derives a 'reference decomposition' automatically by exploiting similarities in the results produced by several different clustering algorithms for evaluating software clustering results in the absence of benchmark decompositions.

Mahdavi et al. (2003) demonstrated a multiple hill climbing approach to software module clustering problem. They experimented on a total of 19 systems and found that the proposed approach has produced good results. Mitchell and Mancoridis (2003) and Mitchell et al. (2004) demonstrated that modeling the search landscape of metaheuristic search algorithms is a good technique for gaining insight into the algorithms. Harman et al. (2005) presented empirical results which compare the robustness of two fitness functions used for software module clustering : one Modularization Quality (MQ) used for module clustering, and the other is EVM, a clustering fitness function previously applied to time series and gene expression data. They claimed that both fitness functions degrade smoothly as noise increases, but the EVM fitness function appears to be more robust for real systems.

Parsa and Bushehrian (2005) designed and implemented a flexible software environment called Dynamic Assembly of Genetic Clustering components (DAGC), to assemble and experiment with genetic clustering algorithms for software modularization. Mitchell and Mancoridis (2007, 2006) emphasised the use of Bunch to the software engineers for program understanding and maintenance activities. They also illustrated the evaluation of the results produced by Bunch for the confidence in the tool. Khan et al. (2008) presented a self-adaptive Evolution Strategies based approach that

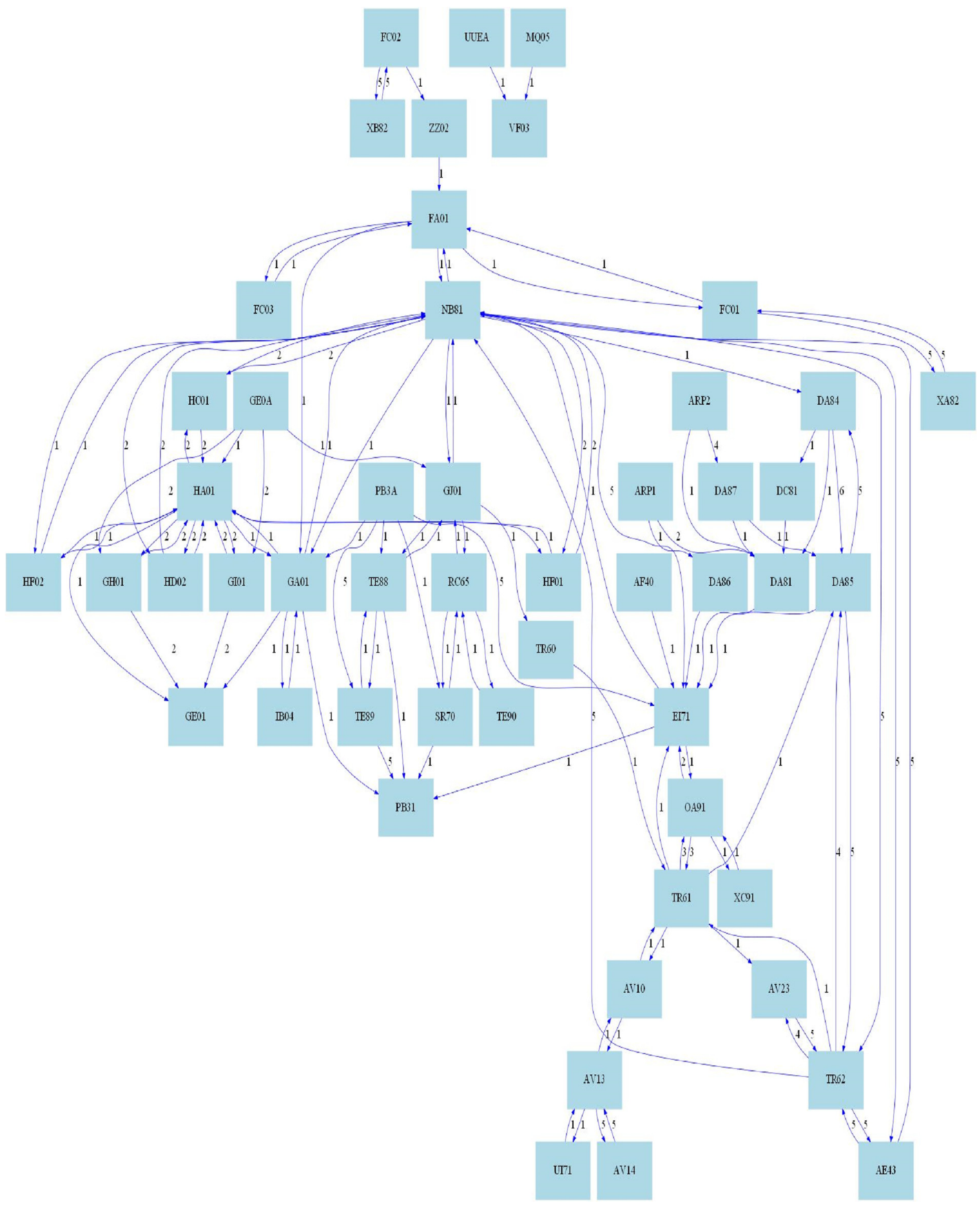
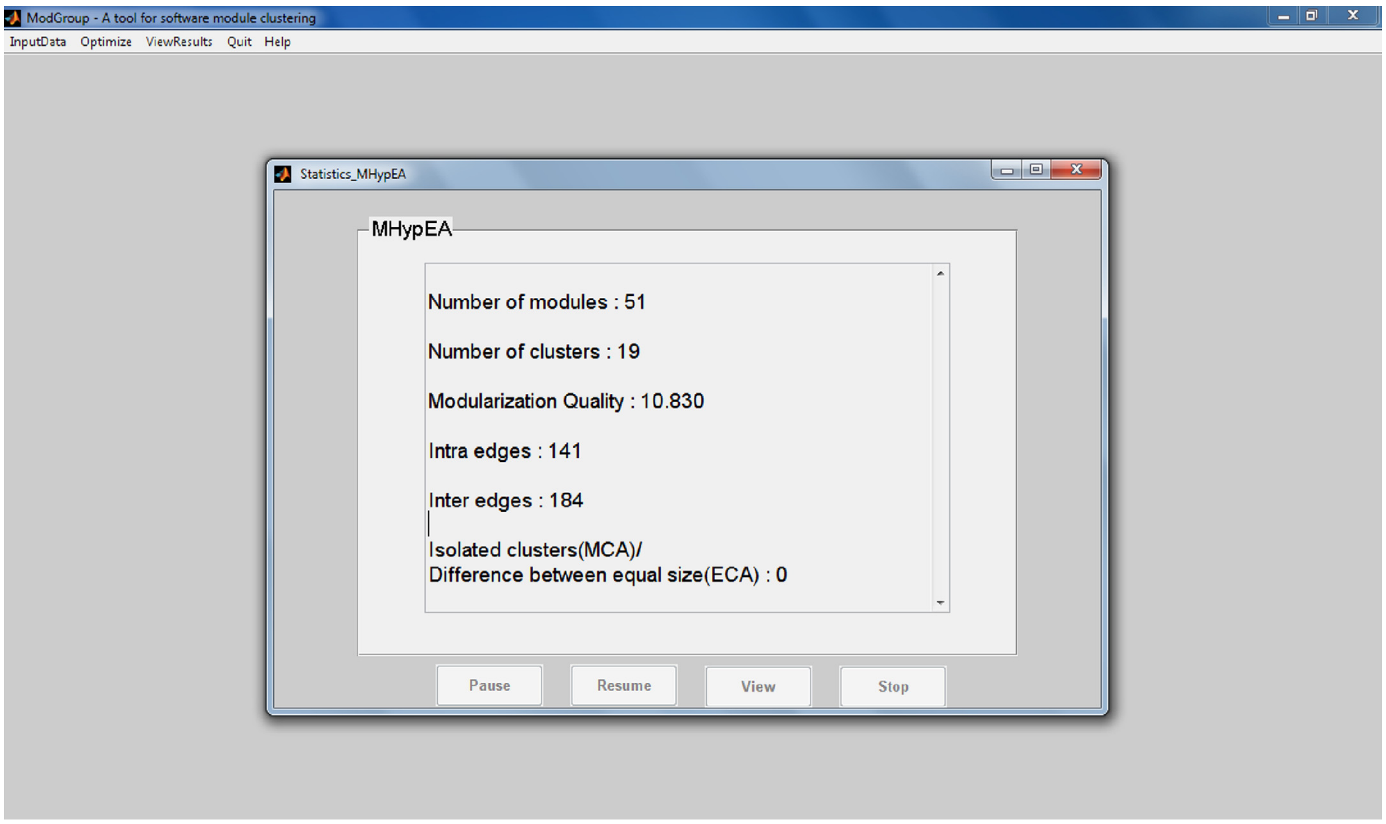


**Fig. 5.** The structure of the real-application data.

explores the large solution space to find an effective decomposition of a system. They compared the proposed approach against GA based approach using industrial systems of different sizes and claimed that the approach is effective in finding quality solutions. Praditwong (2011) introduced the Grouping Genetic Algorithm (GGA) for the problem of software module clustering.

Praditwong et al. (2011) proposed two novel multi-objective formulations of the software module clustering with different objectives. They evaluated the effectiveness of multi-objective approach on real-world module clustering problems. Through their empirical study they claimed that the multi-objective approach produces significantly better solutions than the existing single-objective approach. Barros (2012) addressed the efficiency and effectiveness of using two composite objectives while searching solutions for the software clustering problem. It is observable that the researchers are more centric towards the development of tailor-


ModGroup - A tool for software module clustering
InputData Optimize ViewResults Quit Help
Statistics_MHypEA
MHypEA
Pause
Resume
View
Stop

Number of modules : 51

Number of clusters : 19

Modularization Quality : 10.830

Intra edges : 141

Inter edges : 184

Isolated clusters(MCA)/
Difference between equal size(ECA) : 0

**Fig. 6.** Results obtained by ModGroup with MCA approach.

made and more problem specific algorithms and undoubtedly such algorithms produce high quality solutions only for a particular kind of problem. To achieve a more generic and holistic optimization and to connect diverse software engineering activities, Harman et al. (2012) were the first to suggest hyper-heuristic search as a methodology for selecting or generating heuristics. They argued that hyper-heuristic search will improve the applicability and generality of Search Based Software Engineering (SBSE) techniques and thus addresses the problem of adaptability.

Despite the benefits of hyper-heuristic approach, a very few works have been reported in the literature with hyper-heuristic search. Zhang et al. (2014) applied hyper-heuristics in release planning, Guizzo et al. (2015) in integration testing and Jia et al. (2015) in combinatorial interaction testing. Charan Kumari et al. (2013) and Charan Kumari and Srinivas (2013) were the first to apply hyper-heuristic approach on software module clustering problem. They experimented on six test problems with multiple objectives and reported that the hyper-heuristic approach requires less computational effort in identifying good modular structures.

The distribution of the metaheuristics applied in the literature for module clustering problem is shown in Fig. 7. The analysis of the pie chart reveals that though a variety of techniques have been applied for the purpose, Genetic Algorithms and Hill Climbing were widely experimented.

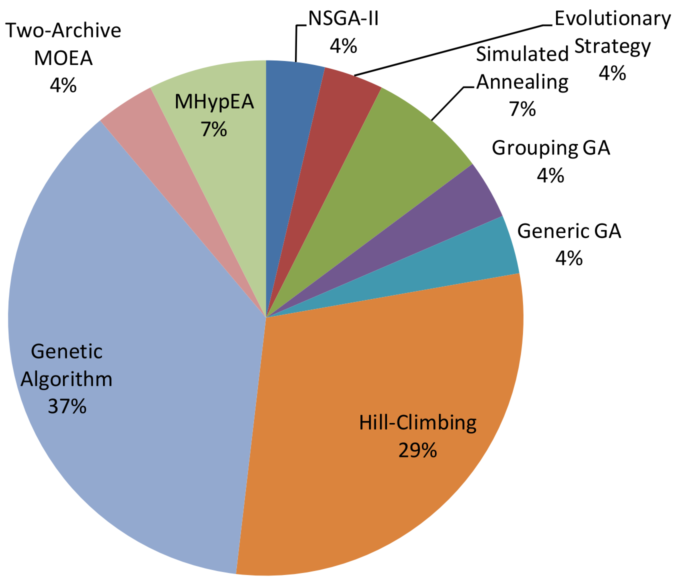


**Fig. 7.** Metaheuristics applied for solving the module clustering problem.

## 10. Conclusions and future work

This paper presented the solution of Multi-Objective Software Module Clustering problem using hyper-heuristic approach. The problem has been studied using two multi-objective approaches to clustering – Maximizing Cluster Approach (MCA) and Equal-size Cluster Approach (ECA), each of them with five objectives. Multi-objective Hyper-heuristic Evolutionary Algorithm (MHypEA) has been implemented for the solution of software module clustering. The performance of MHypEA has been evaluated on two data sets obtained from two different sources each containing six unweighted Module Dependency Graphs (MDGs). The performance of MHypEA has been compared with the results reported in the literature. The comparison is mainly based on the quality of the obtained solutions and computational effort. The three main objectives of intra-edges, inter-edges and Modularization Quality (MQ) have been taken to assess the quality of the solutions and the number of function evaluations has been taken as the criteria to gauge the computational effort.

The main findings of the comparison are:

i In multi-objective formulation, the comparison of results obtained by MHypEA with Two-Archive algorithm and NSGA-II on different data sets reveal that MHypEA is able to obtain better solutions with a computational effort of nearly one-twentieth of the computational effort required by the Two-Archive algorithm and NSGA-II.
ii The efficacy of MHypEA is found to be equally good in suggesting quality clusters with high cohesion and low coupling irrespective of the approach (MCA or ECA) chosen for clustering.
iii MHypEA is found to be competitive in obtaining better values of MQ (in other words good modular structures) and in some instances obtained even much higher values of MQ than those obtained by single-objective formulations of Hill Climbing and Genetic Algorithms.

Also a user-friendly automated CASE tool *ModGroup* has been developed for multi-objective software module clustering with MCA and ECA approaches. The tool has been featured with an option to interrupt the clustering process in-between to view the intermediate results and thereafter either resume the process or stop the execution. The performance of the tool has been validated on a weighted MDG taken from a real-world application with 51 modules and 119 edges. MHypEA is also able to suggest good modular structures for weighted MDGs.

The future work will consider making the developed CASE tool available to the researchers and practitioners for further experimentation. It would be also interesting to examine the effectiveness of other hyper-heuristic approaches on software module clustering problem.

## Acknowledgment

The authors are extremely grateful to Prof. P.S.Satsangi, Chairman, Advisory Committee on Education, Dayalbagh, for continued guidance and support. The authors are also thankful to Prof. Spiros Mancoridis, Prof. Mark Harman and Prof. Márcio Barros for providing the MDGs referenced in the paper.

## APPENDIX

### Pareto optimal solutions provided by ModGroup with MCA approach using MHypEA

| | |
|---|---|
| Software Module clustering Recommended Solution : | |
| Modularization Quality : | 10.830,329 |
| Intra edges : | 141 |
| Inter edges : | 184 |
| Isolated clusters(MCA)/ Difference between equal clusters (ECA) : | 0 |
| Number of clusters : | 19 |
| Cluster# | Modules selected |
| 1 | FC02 XB82 |
| 2 | ZZ02 FA01 FC03 |
| 3 | NB81 HF01 AV23 TR62 AE43 |
| 4 | FC01 XA82 |
| 5 | GA01 IB04 |
| 6 | HA01 HC01 HD02 HF02 |
| 7 | PB31 TE89 |
| 8 | PB3A EI71 AF40 |
| 9 | GE01 GI01 |
| 10 | GE0A GH01 |
| 11 | GJ01 TE88 TR60 |
| 12 | RC65 SR70 TE90 |
| 13 | TR61 OA91 XC91 AV10 |
| 14 | AV13 UI71 AV14 |
| 15 | DA85 DA84 |
| 16 | ARP1 DA86 |
| 17 | DA81 DC81 |
| 18 | ARP2 DA87 |
| 19 | UUEA VF03 MQ05 |
| Other Solutions : | |
| Solution # : | 2 |
| Modularization Quality : | 10.830,329 |
| Intra edges : | 145 |
| Inter edges : | 176 |
| Isolated clusters(MCA)/ Difference between equal clusters (ECA) : | 0 |
| Number of clusters : | 18 |
| Cluster# | Modules selected |
| 1 | FC02 XB82 |
| 2 | ZZ02 FA01 FC03 |
| 3 | NB81 HF01 AV23 TR62 AE43 |
| 4 | FC01 XA82 |
| 5 | GA01 IB04 |
| 6 | HA01 HC01 HD02 HF02 |
| 7 | PB31 TE89 |
| 8 | PB3A EI71 AF40 |
| 9 | GE01 GE0A GI01 GH01 |
| 10 | GJ01 TE88 TR60 |
| 11 | RC65 SR70 TE90 |
| 12 | TR61 OA91 XC91 AV10 |
| 13 | AV13 UI71 AV14 |
| 14 | DA85 DA84 |
| 15 | ARP1 DA86 |
| 16 | DA81 DC81 |
| 17 | ARP2 DA87 |
| 18 | UUEA VF03 MQ05 |
| Solution # : | 3 |
| Modularization Quality : | 10.797,824 |
| Intra edges : | 137 |
| Inter edges : | 192 |
| Isolated clusters(MCA)/ Difference between equal clusters (ECA) : | 1 |
| Number of clusters : | 20 |
| Cluster# | Modules selected |
| 1 | FC02 XB82 |
| 2 | ZZ02 FA01 FC03 |
| 3 | NB81 AV23 TR62 AE43 |
| 4 | FC01 XA82 |
| 5 | GA01 IB04 |
| 6 | HA01 HC01 HD02 HF02 |
| 7 | PB31 TE89PB31 TE89 |
| 8 | PB3A EI71 AF40 |
| 9 | GE01 GI01 |
| 10 | GE0A GH01 |
| 11 | HF01 |
| 12 | GJ01 TE88 TR60 |
| 13 | RC65 SR70 TE90 |
| 14 | TR61 OA91 XC91 AV10 |
| 15 | AV13 UI71 AV14 |
| 16 | DA85 DA84 |
| 17 | ARP1 DA86 |
| 18 | DA81 DC81 |
| 19 | ARP2 DA87 |
| 20 | UUEA VF03 MQ05 |
| Solution # : | 4 |
| Modularization Quality : | 10.780,418 |
| Intra edges : | 140 |
| Inter edges : | 186 |
| Isolated clusters(MCA)/ Difference between equal clusters (ECA) : | 1 |
| Number of clusters : | 20 |

(continued)

(*continued on next page*)

| Cluster# | Modules selected |
|---|---|
| 1 | FC02 XB82 |
| 2 | ZZ02 FA01 FC03 |
| 3 | NB81 HF01 AV23 TR62 AE43 |
| 4 | FC01 XA82 |
| 5 | GA01 IB04 |
| 6 | HA01 HC01 HD02 HF02 |
| 7 | PB31 TE89 |
| 8 | PB3A EI71 |
| 9 | GE01 GI01 |
| 10 | GE0A GH01 |
| 11 | GJ01 TE88 TR60 |
| 12 | RC65 SR70 TE90 |
| 13 | TR61 OA91 XC91 AV10 |
| 14 | AV13 UI71 AV14 |
| 15 | DA85 DA84 |
| 16 | AF40 |
| 17 | ARP1 DA86 |
| 18 | DA81 DC81 |
| 19 | ARP2 DA87 |
| 20 | UUEA VF03 MQ05 |

| | |
|---|---|
| Solution # : | 5 |
| Modularization Quality : | 10.771,731 |
| Intra edges : | 146 |
| Inter edges : | 174 |
| Isolated clusters(MCA)/ Difference between equal clusters (ECA) : | 0 |
| Number of clusters : | 18 |

| Cluster# | Modules selected |
|---|---|
| 1 | FC02 XB82 |
| 2 | ZZ02 FA01 FC03 |
| 3 | NB81 HF01 AV23 TR62 AE43 |
| 4 | FC01 XA82 |
| 5 | GA01 IB04 |
| 6 | HA01 HC01 HD02 HF02 |
| 7 | PB31 TE88 TE89 |
| 8 | PB3A EI71 AF40 |
| 9 | GE01 GE0A GI01 GH01 |
| 10 | GJ01 TR60 |
| 11 | RC65 SR70 TE90 |
| 12 | TR61 OA91 XC91 AV10 |
| 13 | AV13 UI71 AV14 |
| 14 | DA85 DA84 |
| 15 | ARP1 DA86 |
| 16 | DA81 DC81 |
| 17 | ARP2 DA87 |
| 18 | UUEA VF03 MQ05 |

## References

Bai, R, Kendall, G., 2005. An investigation of automated planograms using a simulated annealing based hyper-heuristic. In: Ibaraki, T., Nonobe, K., Yagiura, M. (Eds.), Metaheuristics: Progress as Real Problem Solvers, 32. Springer, US, pp. 87–108.

Burke, E, Kendall, G., Newall, J., Hart, E., Ross, P., Schulenburg, S., 2003. Hyper--heuristics: an emerging direction in modern search technology. In: Glover, F., Kochenberger, G.A. (Eds.), Handbook of Metaheuristics. Kluwer Academic Publishers, pp. 457–474.

Burke, E.K., Silva, J.D.L., Soubeiga, E., 2003. Multi-objective hyper-heuristic approaches for space allocation and timetabling. In: Proceedings of 5th International Conference on Metaheuristics, pp. 129–158.

Charan Kumari, A, Srinivas, K., 2013. Software module clustering using a fast multi-objective hyper-heuristic evolutionary algorithm. Int. J. Appl. Inf. Syst. 5 (6), 12–18.

Charan Kumari, A, Srinivas, K., 2013. Scheduling and inspection planning in software development projects using multi-objective hyper-heuristic evolutionary algorithm. Int. J. Softw. Eng. Appl. 4 (3), 45–57.

Charan Kumari, A, Srinivas, K., Gupta, M.P., 2013. Software module clustering using a hyper-heuristic based Genetic Algorithm. In: Proceedings of IEEE 3rd International Advance Computing Conference, pp. 813–818.

Cowling, P, Chakhlevitch, K., 2003. Hyperheuristics for managing a large collection of low level heuristic to schedule personnel. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 1214–1221.

Cowling, P., Kendall, G., Soubeiga, E., 2001. Hyperheuristic approach to scheduling a sales summit. In: Proceedings of the Third International Conference of Practice and Theory of Automated Timetabling, pp. 176–190.

Cowling, P, Kendall, G., Soubeiga, E., 2002. Hyperheuristics: a tool for rapid prototyping in scheduling and optimisation. In: Proceedings of the Applications of Evolutionary Computing on Evoworkshops, pp. 1–10.

Cowling, P, Kendall, G., Soubeiga, E., 2002. Hyperheuristics: a robust optimisation method applied to nurse scheduling. In: Proceedings of the 7th International Conference on Parallel Problem Solving from Nature, pp. 851–860.

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

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

Doval, D, Mancoridis, S., Mitchell, B.S., 1999. Automatic clustering of software systems using a genetic algorithm. In: Proceedings of International Conference on Software Tools and Engineering Practice, pp. 73–81.

Guizzo, G, Fritsche, G.M., Vergilio, S.R., Pozo, A.T.R., 2015. A hyper-heuristic for the multi-objective integration and test order problem. In: Proceedings of Annual Conference on Genetic and Evolutionary Computation, pp. 1343–1350.

Harman, M., Hierons, R., Proctor, M., 2002. A new representation and crossover operator for search based optimization of software modularization. In: Proceedings of the 2002 Conference on Genetic and Evolutionary Computation, pp. 1351–1358.

Harman, M, Swift, S., Mahdavi, K., 2005. An empirical study of the robustness of two module clustering fitness functions. In: Proceedings of the Conference on Genetic and Evolutionary Computation, pp. 1029–1036.

Harman, M, Burke, E., Clark, J., Yao, X., 2012. Dynamic adaptive search based software engineering. In: IEEE International symposium on empirical software engineering and measurement, pp. 1–8.

Jia, Y, Cohen, M.B., Harman, M., Petke, J., 2015. Learning combinatorial interaction test generation strategies using hyperheuristic search. In: IEEE International Conference on Software Engineering, pp. 540–550.

Kaelbling, L.P., Littman, M.L., Moore, A.W., 1996. Reinforcement learning: a survey. J. Artif. Intell. Res. 4 (1), 237–285.

Kendall, G, Mohamad, M., 2004. Channel assignment optimisation using a hyper-heuristic. In: Proceedings of IEEE Conference on Cybernetics and Intelligent Systems, pp. 791–796.

Khan, B., Sohail, S., Javed, M.Y., 2008. Evolution strategy based automated software clustering approach. In: Proceedings of the 2008 Advanced Software Engineering and its Applications, pp. 27–34.

Lutz, R., 2001. Evolving good hierarchical decompositions of complex systems. J. Syst. Archit. 47 (7), 613–634.

Mahdavi, K, Harman, M., Hierons, R.M., 2003. A multiple hill climbing approach to software module clustering. In: Proceedings of the International Conference on Software Maintenance, pp. 315–324.

Mancoridis, S, Mitchell, B.S., Rorres, C., Chen, Y., Gansner, E.R., 1998. Using automatic clustering to produce high-level system organizations of source code. In: Proceedings of the 6th International Workshop on Program Comprehension, pp. 45–52.

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

Mitchell, B.S., Mancoridis, S., 2002. Using heuristic search techniques to extract design abstractions from source code. In: Proceedings of the Conference on Genetic and Evolutionary Computation, pp. 1375–1382.

Mitchell, B.S., Mancoridis, S., 2003. Modeling the search landscape of metaheuristic software clustering algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 2499–2510.

Mitchell, B.S., Mancoridis, S., 2007. On the evaluation of the bunch search-based software modularization algorithm. Soft Comput. Fusion Found. Methodol. Appl. 12 (1), 77–93.

Mitchell, B.S., Mancoridis, S., 2006. On the automatic modularization of software systems using the bunch tool. IEEE Trans. Softw. Eng. 32 (3), 193–208.

Mitchell, B.S., Mancoridis, S., Traverso, M., 2002. Search based reverse engineering. In: Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, pp. 431–438.

Mitchell, B.S., Mancoridis, S., Traverso, M., 2004. Using interconnection style rules to infer software architecture relations. In: Proceedings of the Conference on Genetic and Evolutionary Computation, pp. 1375–1387.

Mitchell, B.S., 2002. A Heuristic Search Approach to Solving the Software Clustering Problem (Ph.D. thesis). Drexel University, Philadelphia.

Parsa, S., Bushehrian, O., 2005. The design and implementation of a framework for automatic modularization of software systems. J. Supercomput. 32 (1), 71–94.

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

Praditwong, K., 2011. Solving software module clustering problem by evolutionary algorithms. In: Eighth International Joint Conference on Computer Science and Software Engineering, pp. 154–159.

Storer, R.H., Wu, S.D., Vaccari, R., 1995. Problem and heuristic search space strategies for job shop scheduling. ORSA J. Comput. 7, 453–467.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A., 2000. Experimentation in Software Engineering: An Introduction, 6. Springer International Series in Software Engineering.

Zhang, Y, Harman, M., Ochoa, G., Ruhe, G., Brinkkemper, S., 2014. An Empirical Study of Meta- and Hyper-Heuristic Search for MultiObjective Release Planning Research Note, RN/14/07. University College, London.

**A. Charan Kumari** received her Ph.D. from Dayalbagh Educational Institute, in collaboration with Indian Institute of Technology, Delhi, India, under their MOU. Currently she is associated with THE NORTHCAP UNIVERSITY, India, as an Associate Professor in the department of Computer Science and Engineering. She has an excellent teaching experience of 15 years in various esteemed institutions and received various accolades including the best teacher award. Her current research interests include Search-based Software engineering, Evolutionary computation and soft computing techniques. She has published papers in journals of national and international repute. She has delivered an invited talk at 43rd CREST open workshop on Hyper-heuristics at UCL, London. She is a member of IEEE, Computer Society of India (CSI) and Systems Society of India (SSI).

**K. Srinivas** received B.E. in Computer Science & Technology, M.Tech. in Engineering Systems and Ph.D. in Evolutionary Algorithms. He is currently working as Assistant Professor in Electrical Engineering Department, Faculty of Engineering, Dayalbagh Educational Institute, Agra, India. His research interests include Soft Computing, Optimization using Metaheuristic Search Techniques, Search Based Software Engineering, Mobile Telecommunication Networks, Systems Engineering, and E-learning Technologies. He received Director's medal for securing highest marks in M.Tech. Programme at Dayalbagh Educational Institute in 1999. He is an active researcher, guiding research, published papers in journals of national and international repute, and is also involved in R&D projects. He is a member of IEEE and a life member of Systems Society of India (SSI). He is also the Publication Manager of Literary Paritantra (Systems) – An International Journal on Literature and Theory.