Search-Based Object-Oriented Software Re-Structuring with Structural Coupling Strength

Amarjeet∗ and Jitender Kumar Chhabra

Department of Computer Engineering, NIT Kurukshetra, Haryana 136 119, India

Abstract

In real world, the software systems often need to be continuously modified to satisfy the ever changing requirements and environment. Mostly, it is carried out without following the original design principles of the system. Over a period of time, such a continuous modification deteriorates the structural quality, hence increases the system complexity. To improve the structural quality of whole system, the software clustering seems more feasible technique. Recently, the search – based approach gain more attention to solve the software clustering problem. In this paper, we propose a search – based multi – objective optimization to re-structure the object – oriented software system using different coupling strength scheme such as binary coupling, absolute coupling and relative coupling scheme. The approach is evaluated over four real – world and three random software applications. The experimentation results show that how the use of absolute and relative coupling strength scheme leads to generate more effective solutions compared binary coupling strength.

Keywords: Multi-objective; Modularization; Optimization; Re-structuring; Search-based.

1. Introduction

A successful and heavily used software system often requires constant maintenance that is triggered due to evolving technologies, changing requirements and stakeholder1. It has been demonstrated that the maintenance of software system, cost up to 75% of total software development cost2. In order to satisfy the new requirements, maintainers modify the software system. Most of the time developers do not follow the original design rules of system during the modification that leads system structure erosion3. There can be many reasons not to follow the original design rules, for instance short deadlines, lack of proper developer training, and absence of long – term commitment developers to the project4. Such maintenance practices deteriorate the system structure quality and leads system more complex for future maintenance5. Hence, to make system more flexible it requires improving the system structure. To improve the system structure, software refactoring plays an important role, where the system structure is modified without changing the external behavior6. Applying software refactoring to re-structure the whole system is often challenging and costly7. Software clustering is a most feasible technique that can be used to obtain refactoring of whole system more efficiently. Hence, software clustering is widely used as an activity of software refactoring for whole system. Software

∗Corresponding author. Tel.: +91-8950775885.
E-mail address: amarjeetnitkk@gmail.com
clustering techniques has been an active research for more than twenty years. In literature, the software clustering has been performed at various levels of software granularity, with different clustering approaches. A basic concept often used by the existing approaches is that software clustering is strongly influenced by software structural features of software entities, i.e., the manner in which entities are coupled and organized within a system. The different structural characteristics (e.g., different dimensions of coupling) of the software entities can be considered for software clustering. In software engineering literature, there is enough works associating the structural features of systems with their maintainability. Mostly, software clustering approaches have been used successfully for small size software systems. But for complex and large size software system this approach does not perform efficiently. The emergence of Search Based Software Engineering (SBSE) concepts has made software clustering problem more efficient by formulating it as a search based optimization problem.

Based on the search based software engineering concepts, some clustering problems have been formulated using this concept in the literature. These techniques first model the software system into graph and then partition it. In graph the software entities (e.g., classes) are represented by nodes and their relationships (e.g., function calls, inheritance etc.) by edges. Majority of search based clustering approaches perform clustering by analyzing the different dimensions of coupling between software entities. Each dimensions of coupling is defined with a unique category (e.g., structural, dynamic, semantic and historical etc.) Researchers working in the area of software clustering based on structural category has addressed the different structural relationships with attribute such as extends, reference and calls. They have used these structural relationships without giving any detail about their relative importance and have just considered presence and absence of relationships to compute the coupling strength. In this paper, we consider the weights of relationship according to their importance while performing software clustering for object-oriented systems. This paper uses eight type of structural relationships (e.g., extents, has parameter, contains, method call, reference, implements, is of type, returns and throws).

In this paper, we analyze the different form of structural relationships, where coupling strength is determined in terms of relationships. Three methods have been proposed to calculate the coupling strength between the classes. We use NSGA II as multi-objective evolutionary algorithm to evaluate the different coupling strength scheme and perform three experiments using it. In first experiment, we use binary coupling strength scheme, in second absolute coupling scheme and in third relative coupling strength scheme to evaluate.

The remainder of this paper is organized as follows: Section 2 provides background. Section 3 presents the multiobjective concepts. Section 4 presents the way of measuring dependency strength Section 5 provides method of clustering solution evaluation. Section 6 describes the experimental methods. Section 7 presents proposed methodology. Section 8 presents the findings of the experimental study. Section 9 concludes.

2. Software Clustering: Background

In software clustering, a set of software entities based on some criteria is organized into a set of clusters. It can be performed at various level of software granularity; hence the definition of entities and cluster changes accordingly. For example in object-oriented software system, we can consider the classes as entities and packages as cluster. Software clustering is useful to solve different domain of software engineering problem and according to domain it is known by different names such as, software module clustering, module extraction, logical component extraction etc. The usefulness of software clustering in many software domains makes it a hot topic in the research field. It is widely used in the domain of refactoring, reverse engineering and component based software engineering. Software clustering research can be categorized in several ways, such as by clustering approach (search based, consensus based, hierarchical and partitioned based), type of entities (variable, method, class, source file), types of information (conceptual, semantic, static, dynamic) and type of user interaction (automatic or semi-automatic).

Majority of the software clustering approaches perform clustering by analyzing the similarity between software entities. Wiggerts first has given the theoretical background for cluster analysis in software re-modularization and also classified the clustering approaches into graph theoretical approach, construction approach, optimization approach and hierarchical approach. The usefulness of these clustering algorithms and their parameters into the software clustering was studied by Anquetil et al. To evaluate the usefulness of the clustering algorithms Tzerpos and Holt proposed MOJO metric that measure the distance between two clustering, it is also helpful for evaluation.
of the stability of clustering algorithms. Later same authors\textsuperscript{23} performed an experimentation with various clustering algorithms to evaluate its stability using MOJO metric.

Emergence of search based software engineering techniques has made many software engineering problems efficiently solvable\textsuperscript{24}. The software clustering problem contains many characteristics that help to formulate it as a search based software engineering problem. Mancordis \textit{et al.}\textsuperscript{14} first formulated the software clustering problem as search based optimization problem and evaluated over many real world problem instances. They formulated the module clustering problem as single objective search problem (based on Hill-Climbing) to find good quality module structure for a system. Similar to the above approach, Doval \textit{et al.}\textsuperscript{25} also formulated the software clustering problem as single objective optimization problem using MQ as fitness function and proposed a genetic algorithm to address problem. The representations of software engineering problem as search problem directly affect the size of search space. To address this issue Harman \textit{et al.}\textsuperscript{26} proposed a normalized representation for a software clustering problem, which reduced the size of the search space and helped to improve the outcome of Genetic Algorithms. The simple Hill Climbing algorithm\textsuperscript{14} faced the local minima and efficiency problem. To address these problems, Kiarash Mahdavi \textit{et al.}\textsuperscript{16} introduced a Multiple Hill Climbing algorithm. Bilal Khan \textit{et al.}\textsuperscript{27} proposed a new software clustering approach based on evolution strategy and their results showed that it provide better results in most of the cases.

Most of the search based software clustering approaches used single objective to optimize the clustering. Recently, multi – objective based software clustering gained more attention. Praditwong \textit{et al.}\textsuperscript{17} proposed multi – objective optimization approach to solve the software clustering problem. The authors used two composite objective formulations i.e., Equal Cluster – size Approach (ECA) and Maximizing Cluster Approach (MCA). Later M. Barros\textsuperscript{28} analyzed the multi – objective software clustering with NSGA-II algorithm for evaluation of efficiency and effectiveness of composite objectives.

The search based software re-structuring approaches perform the clustering by searching and optimizing the some quality criteria such as coupling between software entities. The aforementioned approaches evaluate the coupling strength in terms of the relationships present between the software entities. They consider the coupling strength as binary value, i.e., presence or absence of relationships between classes or integer value i.e., total number of relationships present between the classes. As for different quality measurement goal such as external quality factor, the different type of relationships has different importance (i.e., weights)\textsuperscript{29}. Hence, weights of relationships need to be assigned according to their relative importance not just a binary value. So, in order to produce precise quality system through clustering, it requires identifying the various types of relationships and their relative importance (i.e., weights) to evaluate the dependency strength between entities. In this paper, we consider the relative weight of relationships for evaluation of coupling strength. To the best of our knowledge, the relative weights for the relationships in the literatures have not been considered in multi – objective software optimization for re-structuring object – oriented software system.

3. Multi – Objective Optimization

Normally the search based software clustering problem can be roughly classified as single objective and multi – objective optimization problem. In single objective optimization, the clustering is determined by optimizing a single quality criteria, whereas in multi – objective optimization the clustering is determined by optimizing multiple conflicting quality criteria. To solve these search based optimization problem the various heuristic techniques are used for navigation over the search space. The detailed description of single and multi – objective software clustering are as follows.

3.1 Single objective software clustering

In single objective software clustering, only the single objective is optimized. It determines a clustering $M_*$ for which

$$F(M_*) = \min / \max F(M) | M \in \Psi$$
where $\Psi$ is the set of all feasible clustering. $M$ is the software clustering solutions such as $F: \Psi \to R$ is an objective function. Here function $F$ can be minimization function or maximization function. Most of the software clustering problems is based on this single – objective optimization problem. Different single objective clustering approaches varies with optimization function $F$ and optimization method. Even though single objective clustering methods have been widely applied, still they have some weakness. (1) These single objective methods attempt to optimize just one objective function and this may restrict the clustering solution to a particular software structure property. (2) A single fixed clustering solution returned by single objective approach may not be suitable for the software structure with multiple potential structures.

3.2 Multi – objective software clustering

In multi – objective software clustering, more than one conflicting objective function is optimized. It determines clustering solutions $M^*$ for which

$$F(M^*) = \min(F_1(M), F_2(M), \ldots, F_m(M)) \mid M \in \Psi$$

where $m$ is the number of objective functions and $F_i$ represents the $i^{th}$ objective function. In multi – objective software clustering, there is usually no single best solution, but there can be more than one non-dominated clustering solution. For two clustering solutions $M_1, M_2 \in \Psi$, solution $M_1$ is said to dominate solution $M_2$ (denoted as $M_1 \leq M_2$) if and only if

$$\forall i \in (1, \ldots, m) F_i(M_1) \leq F_i(M_2)$$

and

$$\exists i \in (1, \ldots, m) F_i(M_1) < F_i(M_2)$$

Otherwise $M_1$ and $M_2$ are said to be non-dominated solutions. The set of all non-dominated solutions in objective space is called Pareto front. The multi – objective clustering techniques provide flexible clustering solutions where developer has more options for selection of best solution based on their requirements.

4. Coupling Strength and Quality Criteria

In order to evaluate the system structural quality more precisely, while applying software module clustering techniques, it requires identifying the various types of structural feature which incorporate coupling between software entities. These different types of coupling derived from the relationships collectively used to determine the coupling strength between entities. For a particular structural quality measurement goal, the different types of relationships have different importance. Hence, for any two types of relationship between classes the weight can be defined as if one is stronger than the other, or if both have equal weight. For example, between two classes, the “method invocation” relationship has more importance than “attribute reference” relationship for control flow based software testability while the “method invocation” relationship has equal importance as “attribute reference” relationship for software comprehension. The definition of such partial order on the set of relationship types is to some degree subjective. So, in order to determination of precise quality measure requires to determine the relative importance of relationships. The determination of actual relationship weight and their partial order is a challenging problem and it is out of scope for this paper.

4.1 Structural relationships between classes

In object oriented software system each structural relationship between classes is characterized with their feature. For example a relationship can be characterized with extends call, reference etc. So it needs to classify the different types of relationships according to their importance. In software module clustering, it is necessary to collect all possible relationships from the source code. If there is more than one relationship between the classes then we merge all relationship weights to evaluate coupling strength. According to relationship importance, there may be many types of structural relationships between classes, but for sake of simplicity we have considered only eight types of relevant relationships. The type of relationships between class A and class B is given as follows:
• Extends (EX): Class A extends other general Class B.
• Has Parameter (HP): Class A has a method with parameter of Class B.
• Reference (RE): Class A invokes the attribute or method of Class B.
• Calls (CA): Method of Class A calls the method of Class B.
• Implement (IM): Class A implement, or realize, the behaviour specified by Interface B.
• Is of Type (IT): In Class A has an attribute that is type of Class B.
• Return (RN): Class A has a method that return an object of Class B.
• Contains (CO): Class A contains attribute that is type of class B (also called an aggregation relationship).
• Throws (TH): Class A throws an exception to Class B.

4.2 Coupling strength schemes

Each type of relationships presents between two classes exhibit different forms of coupling and contribute different weight in overall coupling strength between two classes. To calculate the coupling strength between the classes, the paper presents three coupling strength scheme i.e., binary coupling strength scheme, absolute coupling strength scheme and relative coupling strength scheme. The detailed mathematical description of the coupling strength calculation between two classes is as follows:

Let $G = (V_c, E_c)$ represent a weighted software structure graph, where $V_c$ is set of edges which represents set of classes and denoted as $V_c = \{v_1, v_2, \ldots, v_n\}$, $E_c$ is the set of edges denoted as $E_c = \{e_1, e_2, \ldots, e_m\}$. Each edge $e_i$ is set of relationships $r = \{\text{extend, has parameter, reference, call, implement, is of type, return, contain, throws}\}$. Each relationship is denoted as a triple $r_i = \{v_a, v_b, w_i\}$ where $v_a, v_b \in V_c, w_i \in R$ represents a relationship between classes $c_a$ and $c_b$ with weight $w_i$.

4.2.1 Binary coupling strength (BCS)

In binary coupling strength scheme, the coupling strength is considered as binary value. It widely used method in software clustering and can be calculated as follows:

$$CS_{BCS}(c_a, c_b) = \begin{cases} 1 & \text{if } \sum_{r=EX}^TH p_r(c_a, c_b) > 0 \\ 0 & \text{otherwise} \end{cases}$$

(1)

where $p_r(c_a, c_b)$ is a predicate, its value can be zero or one according to presence or absence of relationships between the classes.

4.2.2 Absolute coupling strength (ACS)

In binary coupling strength, the coupling strength between the classes is either zero or one and the importance of individual relationships is not considered. In absolute coupling strength, each individual relationships weight between the classes is also considered and they collectively form the coupling strength between the classes. The coupling strength between two classes can be determined by three aspects:

• The relationship category.
• The number of instances of a particular category.
• The weight of relationship of each category.

The different types of relationships may have different weights. But in this coupling strength scheme all relationships are given a binary weight. The absolute coupling strength between classes $c_a$ and $c_b$ can be calculated as follows:

$$CS_{ACS}(c_a, c_b) = \begin{cases} \text{Udefined} & \text{if}(i = j) \\ \sum_{r=EX}^T H w_r N_r(c_a, c_b) & \text{otherwise} \end{cases}$$

where, $w_r$ is binary weight associated with individual relationship between classes $a$ and $b$ and $N_r$ is the number of instances of particular relationship type, between classes.
4.2.3 Relative coupling strength (RCS)

Allowing different weight to different relationship in various quality measurement goals brings flexibility for precise quality evaluation, but it also raises a new problem: determining the weight $w_r$ of various types of relationships. Computation of actual weight of each relationship type is subjective matter and according to requirement goal the relative weight of each relationship can be determined. In this paper we determine the relative weight of each relationship as follows:

$$w_r = \frac{\text{Total number of } r\text{-type relation in system}}{\text{Total number of all relations}} \times \frac{\text{Total number of } r\text{-type relation within all packages}}{\text{Total number of } r\text{-type relations in system}}$$

The rationale behind using such relative weight of relationships is that it force to guide the optimization process towards original design rule perceived by system developers. Now using the following formula, the relative coupling strength between the two classes can be calculated.

$$CS_{RCS}(c_a, c_b) = \begin{cases} 
\text{Undefined if } (i = j) \\
\sum_{r=E}^{T} w_r N_r(c_a, c_b) \text{ otherwise}
\end{cases}$$

where, $w_r$ is positive non-null weight associated with individual relationship between classes $a$ and $b$ and $N_r$ is the number of instances of $r$-type relationship between classes.

4.3 Re-structuring quality criteria

Software re-structuring using search based technique is a typical multi-objective problem where two or more conflicting criteria must be optimized to provide a good module structure of a software system. In this paper, we consider four conflicting criteria as objective function to improving the structure of packages: 1) cohesion (maximize); coupling (minimize); modularity quality (MQ) (maximize); Difference between Max and Min module size (minimize). The first three objective functions are based on coupling strength between classes and the next one is base on the module size.

5. Proposed Approach

Each of the existing software clustering techniques presented in section 2 operates on different types of entities and their relationship at various level of software granularity. In this paper, we consider classes as entities and packages as module. The general process of proposed approach is given in Fig. 1. The approach has three main phases: (1) Extracting classes and their relationships, (2) assigning weight to relationship and calculating coupling strength, and (3) applying clustering techniques.
5.1 Extracting classes and their relationships

In this phase, the classes and their relationships are extracted from the source code of object oriented system. There are many object oriented source code analyzers existing which can be used for this purpose. Some of these are PF-CDA, Stan4J and Structure 101. The PF-CDA is used for analysis of java classes and their dependencies. The Stan4J and structure 101 is structural analysis tool at various level of granularity. In this paper, we use the PF-CDA and Structure 101 to extract the classes and their relationships. Generally, only those relationships from the source code are considered, that affect the quality measurement goal. In this paper, we consider the eight important relationships that contribute in most of the quality measurement goal. The descriptions of these relationships are given in section 3.

5.2 Assigning weight to relationship and calculating coupling strength

In order to cluster a set of classes into package precisely, it requires assigning weight to relationships according to their importance in quality measurement goal. The detailed description about weight assignment is given in section 3. After assigning weight to the relationships we combine the all relationship weight that exists between any two classes to calculate dependency strength.

5.3 Applying clustering algorithm

After assigning weight to all relationships and finding the dependency strength among classes, the classes are grouped now into the packages. Software clustering algorithms are used to group the similar software classes into same package and dissimilar classes into different package. According to number of criteria, the clustering algorithms can be single or multi – objective clustering algorithms. But in this paper, we consider multiple criteria as described in section 4. Hence, to perform the clustering, any suitable multi – objective evolutionary algorithms can be used. In this paper, we use NSGA-II as an evolutionary based multi – objective algorithm in software clustering problem because it has performed well in similar software engineering problems.

6. Experiment Setup

This section provides details about the experimentation to perform software module clustering. The section A describes the multi – objective algorithms and their parameter used. Section B describes the problem instances used. Section C describes techniques for result collection.

6.1 Multi – objective algorithms and parameters

This paper uses the Non-Dominated Sorting Algorithms (NSGA-II), to perform the experiments. It is a genetic based meta – heuristic algorithm and applied as multi – objective evolutionary algorithm. To generate the best results, it requires proper parameter settings. In literature, same algorithm used to perform software clustering. So, we also follow the same configuration setting. The algorithm uses single point crossover operator for crossover, uniform mutation operator for mutation and binary tournament strategy for selection. The crossover probability is set 80% for the problem instance having less than 100 classes and is set 100% otherwise. For the mutation probability it is set $0.04 \times \log_2(N)$, with $N$ number of classes for all types of problem instance.

6.2 Problem instance selection

The experiment was executed with four real – worlds and three random instance problems. The real – worlds problem instances are based on the java programming language and are open – source or free – software projects. The relationships with their characteristics are collected using the PF-CDA, open – source static analysis tool. The detail about the selected problem instances are given in the Table 1. These problem instances are selected covering different size and various characteristics.
Table 1. Information about of the original software applications.

<table>
<thead>
<tr>
<th>Original systems</th>
<th># packages</th>
<th># classes</th>
<th>Binary weighted</th>
<th>Absolute weighted</th>
<th>Relative weighted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real-world Systems</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JavaCC</td>
<td>6</td>
<td>154</td>
<td>2.28</td>
<td>3.10</td>
<td>2.83</td>
</tr>
<tr>
<td>JUnit</td>
<td>6</td>
<td>100</td>
<td>3.03</td>
<td>2.95</td>
<td>2.57</td>
</tr>
<tr>
<td>Java Servlet API</td>
<td>4</td>
<td>63</td>
<td>2.55</td>
<td>2.68</td>
<td>2.43</td>
</tr>
<tr>
<td>XML API DOM</td>
<td>9</td>
<td>119</td>
<td>5.58</td>
<td>4.84</td>
<td>4.29</td>
</tr>
<tr>
<td>Random Systems</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random50</td>
<td>7</td>
<td>50</td>
<td>2.74</td>
<td>2.71</td>
<td>2.65</td>
</tr>
<tr>
<td>Random100</td>
<td>12</td>
<td>100</td>
<td>3.88</td>
<td>3.62</td>
<td>3.53</td>
</tr>
<tr>
<td>Random150</td>
<td>18</td>
<td>150</td>
<td>5.96</td>
<td>5.84</td>
<td>5.71</td>
</tr>
</tbody>
</table>

Table 2. Mean and standard deviation of MQ.

<table>
<thead>
<tr>
<th>Re-structured Systems</th>
<th>Mean</th>
<th>Binary Weighted</th>
<th>SD</th>
<th>%-Dev</th>
<th>Mean</th>
<th>Absolute Weighted</th>
<th>SD</th>
<th>%-Dev</th>
<th>Mean</th>
<th>Relative Weighted</th>
<th>SD</th>
<th>%-Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real problems</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JavaCC</td>
<td>3.16</td>
<td>0.116</td>
<td>38.60</td>
<td>3.85</td>
<td>0.146</td>
<td>24.19</td>
<td>3.89</td>
<td>0.100</td>
<td>37.46</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JUnit</td>
<td>3.97</td>
<td>0.130</td>
<td>31.02</td>
<td>3.98</td>
<td>0.121</td>
<td>34.92</td>
<td>3.98</td>
<td>0.167</td>
<td>54.86</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Java Servlet</td>
<td>3.18</td>
<td>0.145</td>
<td>24.71</td>
<td>3.30</td>
<td>0.149</td>
<td>23.13</td>
<td>3.40</td>
<td>0.083</td>
<td>39.92</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XML API</td>
<td>6.05</td>
<td>0.248</td>
<td>8.42</td>
<td>5.96</td>
<td>0.591</td>
<td>23.14</td>
<td>6.75</td>
<td>0.293</td>
<td>57.34</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random Problems</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random50</td>
<td>2.93</td>
<td>0.048</td>
<td>6.93</td>
<td>3.57</td>
<td>0.091</td>
<td>31.73</td>
<td>3.79</td>
<td>0.072</td>
<td>43.02</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random100</td>
<td>4.76</td>
<td>0.118</td>
<td>22.68</td>
<td>4.76</td>
<td>0.155</td>
<td>31.49</td>
<td>4.97</td>
<td>0.196</td>
<td>40.79</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random150</td>
<td>6.67</td>
<td>0.133</td>
<td>11.91</td>
<td>10.35</td>
<td>0.221</td>
<td>77.23</td>
<td>10.51</td>
<td>0.186</td>
<td>84.06</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6.3 Collecting results from experiment

To collect the results, the NSGA-II algorithm with given objective functions and configuration was executed 30 times for all eight original systems. The results for each original systems and each running cycle yielded a set of Pareto front ($PF_i$). After running all cycles for a given problem instance, the Pareto front with the highest MQ value is chosen to be the best solution in each running cycle. Then the mean and standard deviation of MQ is estimated using the result from 30 running cycle.

7. Results and Analysis

This section illustrates the proposed method through experiments on four real-world and three random problems instances. In this paper, first we performed three experiments using NSGA II algorithm with given objective functions and configuration. In first experiment, all seven problem instances are evaluated by considering coupling strengths between classes as binary value (i.e., un-weighted) and mean and standard deviation of modularity quality (MQ) is evaluated. In second experiment, the same thing is performed using absolute weighted scheme on the same problem instances are. In third experiment, again the same problem instances are evaluated by considering the coupling strength as relative weight scheme. As the output of multi-objective NSGA-II is a set of solutions (Pareto front), therefore we calculate the mean and standard deviation of modularity quality and it makes easier to analyze them. The overall results of these experiments are given in Table 2.

In Table 2 the vertical columns shows the mean, standard deviation and percentage of improvement of mean MQ values in re-structured object-oriented systems of the three coupling scheme. The horizontal columns show the problem instances. The Table shows the mean of MQ values of all re-structured problem instances of each coupling strength scheme improves drastically. For example, in binary coupling strength the minimum percentage improvement in mean MQ value is 6.93% and maximum improvement is 38.6%. If we compare the three coupling strength scheme, the absolute coupling strength scheme and relative coupling strength scheme perform better than the binary coupling strength in most of the problem instances. Now, if we compare the absolute coupling strength scheme and relative coupling strength scheme, the Table shows that the relative coupling strength scheme perform better than the absolute coupling strength scheme with all problem instance.
The Fig. 2 summarises the results of binary coupling, absolute coupling and relative coupling scheme. Basically, it shows the percentage improvement in MQ values of all three coupling strength scheme. The horizontal axis shows the problem instances and the vertical axis shows the percentage improvement in the MQ values.

8. Conclusion and Future Work

In this paper, we have proposed a multi – objective optimization approach to improve the object – oriented software module structure using various coupling strength scheme namely binary coupling, absolute coupling and relative coupling strength scheme. The approach is evaluated over the four real – world and three random problem instance. The experimentation results show that the proposed approach with each coupling strength scheme, improves the MQ value of each problem instance. The comparative study shows that the MQ value of absolute coupling strength and relative coupling strength scheme improves better in comparison of binary coupling strength scheme over many problem instances, while the MQ value of relative coupling strength improves better in comparison of absolute coupling strength scheme over all the problem instances. Hence, the results conclude that the performance of the multi – objective optimization approach can be improved with the suitable coupling strength scheme. The future works includes the formulation and evaluation of more coupling strength schemes with multi – objective optimization techniques for improving the object – oriented software structures.

References


[31] http://www.stan4j.com