Indonesian Journal of Electrical Engineering and Computer Science
Vol. 4, No. 2, November 2016, pp. 465 ~ 472
DOI: 10.11591/ijeecs.v4.i2.pp465-472

Clustering Techniques for Software Engineering

Shohag Barman*[1], HiraLal Gope[2], M M Manjurul Islam[3], MdMehedi Hasan[4], Umme Salma[5]
[1,4,5]Department of Computer Science & Engineering, University of Chittagong, Chittagong-4331, Bangladesh
[2]Department of Computer Science & Engineering, Sylhet Agricultural University, Sylhet-3100, Bangladesh
[3]School of Electrical, Electronics, and Computer Engineering, University of Ulsan, South Korea
*Corresponding author, e-mail:shohag3340csecu@gmail.com

Abstract

Software industries face a common problem which is the maintenance cost of industrial software systems. There are lots of reasons behind this problem. One of thepossible reasons is the high maintenance cost due to lack of knowledge about understanding the software systems that are too large, and complex. Software clustering is an efficient technique to deal with such kind of problems that arise from the sheer size and complexity of large software systems. Day by day the size and complexity of industrial software systems are rapidly increasing. So, it will be a challenging task for managing software systems. Software clustering can be very helpful to understand the larger software system, decompose them into smaller and easy to maintenance. In this paper, we want to give research direction in the area of software clustering in order to develop efficient clustering techniques for software engineering.Besides, we want to describe the most recent clustering techniques and their strength as well as weakness. In addition, we propose genetic algorithm based software modularizationclustering method. The result section demonstrated that proposed method can effectively produce good module structure and it outperforms the state of the art methods.

Keywords*: hierarchical clustering; graph-theoretic clustering; optimization clustering, information-theoretic clustering; genetic algorithm.*

1. Introduction

Most of the software systems are very large and complex. That's why it is very difficult to understand the complex structure of those software systems. Software clustering is an important research area in software engineering. This research area has been carried out for a long time. During this time, several clustering techniques have been developed. In this research area, there is less work on selecting efficient methods for software clustering. There are many problems in software clustering. Many software systems need to modify according to varieties of demand such as improve program structure, fixing bugs, an extension to the new platform, easy maintenance and addition of the new capabilities [1]. When a software system is large then it must need to partition into a small system. Otherwise, it will be difficult to distribute larger system among the development team. Modular dependency graph (MDG) is useful to partition the large software system into smaller subsystem. It converts the source codes into language independent graph. Basically, clustering techniques partitions the MDG into the clusters and each cluster represents the subsystem. In this work, we want to give a research guideline for only software clustering algorithms that decompose larger software systems into asmaller subsystem, thus leading to understanding the software system. This is because understanding the structure of a software system is valuable for maintainers. Software clustering algorithms can be defined as groups of entities, such as classes or source files. On the other hand, we can say the main objective of software clustering is to break down large software systems into smaller and convenient subsystems that are easier to understand. Several papers [2-7] presented many distinct algorithms that can automatically create software decompositions. There are many essential difficulties and challenges for implementing clustering algorithms. In this work, we include adescription of four efficient clustering techniques that illustrate how software modules are partitioning into clusters according to some pre-specified criteria. We alsopropose a genetic algorithm based software modularization clustering (GASMC) method to partition larger software system (module) into smaller software system (clusters). We compare

Received August 28, 2016; Revised September 30, 2016; Accepted October 16, 2016

our result with state of the art method and proposed method outperformed the state of the art. The result section demonstrated that our proposed method can effectively partition the module into clusters.

We organize the rest of the paper as follows: Section 2 introduces the issues and challenges for software clustering. Section 3 presents four mostly important software clustering techniques. Section 4 includes proposed genetic algorithm based software modularization clustering. Section 5 contains the result and discussion section. Finally, Section 6 includes conclusion and add references.

2. Issuesand Challenges of Clustering Algorithm

In software clustering, attributes and relationships to be clustered. Attributes may be variables or procedures; Otherwise, attributes can be considered as modules or classes. Here, two important things are necessary to consider:

a. What types of relationships exists between attributes?
b. Are the relationships weighted or not?

2.1. Similarity

The degree of similarity between attributes is another important issue. There are many kinds of similarity measures:

a. Association coefficient: Some common features exist or not between attributes.
b. Distance measures: The degree of dissimilarity between attributes.

2.2. Similarity measurement

We can define the similarity measurement in terms of feature for each attribute as follows:

a. Number of common features in attribute $p$and attribute$q$.
b. Number of features itself p.
c. Number of features itself q.
d. Number of features misses in both attribute $p$and attribute q.

2.3. Association coefficient

Association coefficients are defined in the following which is based on some values:

Simple Matching Coefficient= $\frac{a+d}{a+b+c+d} = \frac{a+d}{a+b+c+d}$

Jaccard Coefficient= $\frac{a}{a+b+c} = \frac{a}{a+b+c}$

Sorensen Coefficient= $\frac{2a}{2a+b+c} = \frac{2a}{2a+b+c}$

3. Software Clustering Techniques

Clustering is the action of organizing data into groups of similar objects based only on information found in the data that describes the objects and their relationships. The most discussed clustering algorithms for software engineering include graph-theoretical algorithms, hierarchical algorithms, optimization algorithms, Information-theoretic algorithm.

3.1. Graph Theoretical based Software Clustering

Graph theoretic algorithms always try to find a subgraph that can be used for clustering. There are many kinds of subgraphs such as connected components, maximum cliques, and spanning trees are used for this purpose. Aggregation algorithms and minimal spanning tree algorithms are two most familiar graph-theoretical clustering algorithms in software engineering.

In Aggregation algorithms, the number of nodes is reduced by converting them into few nodes or aggregate nodes. Then the aggregate nodes can be used as clusters or another level of aggregate nodes. Minimal spanning tree (MST) algorithms start with searching an MST in a graph. After that two or more neighbor nodes join in the cluster or the graph is juncture into

clusters by considering no “long” edges. Basically, the classical MST algorithm is not well enough for software clustering because of the algorithm tries to create large clusters. To solve this problem, Bauer and Trifu [8] suggest a two-pass MST algorithm. In the first pass, two neighbor nodes are iteratively joined in the cluster whereas the second pass accredits the left unknown clusters entities to the cluster who are the “closest” to. As an example, four-group clustering after removal of three successive longest edges. Figure 1 and 2 illustrates the graph-theoretical algorithm.

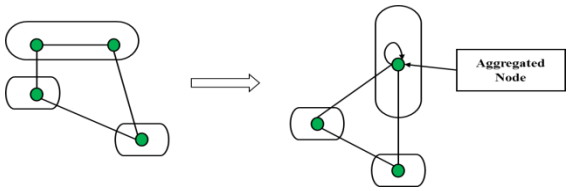

Figure 1. Aggregate Nodes

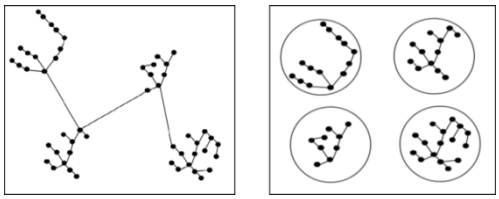

Figure 2. Minimal Spanning Tree

3.2. Hierarchical ClusteringAlgorithms

Hierarchical Clustering algorithms can be categorized as either agglomerative algorithm or divisive algorithm. The agglomerative algorithm works in a bottom-up fashion while divisive is formed in atop-down fashion. The agglomerative algorithm begins by placing each object in its own cluster and then merges these clusters into larger. After that, it continues until all of the objects are in a single cluster or until certain termination conditions are satisfied. Actually, most of the hierarchical clustering algorithms work in this way. The divisive algorithm works just the reverse of agglomerative hierarchical clustering algorithm by beginning with all objects in one cluster. Then, it subdivides the clusters into smaller parts. After that, it continues until each object forms a cluster on its own or until it satisfies certain termination conditions. Figure 3 illustrates that hierarchy of clustering algorithms for four entities.

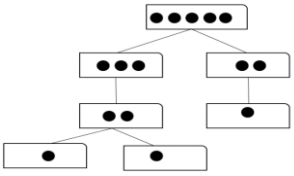

Figure 3. Hierarchical Clustering

Algorithm: agglomerative hierarchical clustering
Assign each object to a cluster
Calculate the similarity distance matrix
repeat
Merge the two closest clusters
Update the distance matrix to reflect the distance between the new cluster and the original clusters
until only a single cluster remains

3.3. Optimization Algorithms

An optimization algorithm starts with an initial solution and uses some heuristic to improve initial solution by iterating adaptions. For both hierarchical and non-hierarchical clustering, this algorithm can be used. The most common optimization algorithms are called 'partitioning techniques' or simply 'iterative partitioning' [9]. Iterative partitioning algorithms start with an initial partition in which entities are moved to other clusters in order to improve the partition according to some criterion. This relocating goes on until no further improvement of this criterion takes place. In such situation, aheuristic algorithm can be applied for improvement. Other optimization algorithms can be considered like genetic, hill-climbing, spectral, and clumping techniques. Genetic algorithm is the best optimization algorithm of them. So, we are interested in sketching the genetic algorithm in this work. Genetic algorithms are optimization search algorithms based on the concept of evolution observed in nature. Genetic algorithms use selection, crossover, and mutation operators for finding the best solution to a specific problem. These algorithms always try to search an optimal solution until a specific termination condition is satisfied. Genetic algorithms work on a population of potential solutions applying the principle of survival of the fittest to produce better and better approximations to a solution.

Genetic algorithm work as follows:

a. A GA search begins with an initial population of feasible solutions.
b. Crossover is the basic search operator in a GA. Crossover is used to combine pairs of population members to create offspring.
c. After the creation of new offspring, another operator called mutation is applied on the results. Mutation is a random change of parts of an offspring, which is more commonly based on a heuristic neighborhood search. The offspring are used to create a new generation of solutions. This can range from the new population consisting of just offspring to a heuristic to replace varying degrees of the old population members with the offspring.
d. Repeat 3 until aproper termination condition is met.

Selection:

Selection is the process of choosing two parents from population pool. Theselection pressure is very important for GA to improve the fitness of population through successive generations. Superior individuals should be more attractive. But, it is needed to control the fitness values between superior and inferior individuals. In this work, we use theRoullete wheel for selecting two parents.

Crossover and mutation are two basic operators of genetic algorithm. The performance of genetic algorithm is depending on two operators.

Crossover:

a. An operation to generate a new chromosome by combining partial characteristics of parent chromosomes
b. Selected mates may have good properties to survive in next generations
c. So, we can expect that exchanging features may produce other good individuals

There are mostly common three types of crossover.

1. Single point crossover
2. Two-point crossover

Single Crossover: Single point crossover – choose one crossover point, copy the binary string from beginning of chromosome to the crossover from one parent, and rest of the binary string of the chromosome is copied from the second parent. Single Crossover shown in Figure 4.

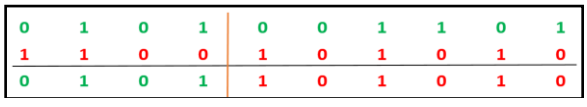

Figure 4. Single Crossover

Two-point crossover:

Choose two crossover points, copy binary string from beginning of chromosome to the first crossover point from one parent, the part from the first to the second crossover point is copied from the second parent and the rest is copied from the first parent (Figure 5).

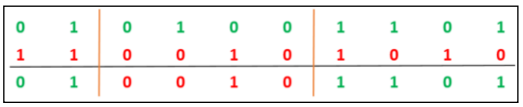

Figure 5. Two-Point Crossover

Mutation:

Mutation is an operation to partially modify a parent chromosome with a low probability. Mutation as shown in Figure 6.

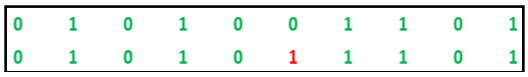

Figure 6. Mutation

3.4. Mutual Information based Software Clustering

Many studies for software clustering have shown how to decompose structural information into large software systems [10]. Mostly, the existing algorithm concentrates on all attributes of the software artifacts while we want suggest to use mutual information approach to concentrate on theimportance of a particular attribute for software clustering purposes. As a result, people can keep in mind several attributes are more important than others for the assurance of the clusters. Feature selection by using information theory like mutual information is an important step for software clustering. In fact, Software artifacts to be clustered and the corresponding features explain the artifacts. Mutual information measures the dependencies between software artifacts and features. It is important to introduce entropy for the knowledge of mutual information.

Entropy is the measure of uncertainty of a random variable. The higher the entropy, the lower the certainty for predicting the random variable. Let XXbe a discrete random variable taking its values from a set of artifacts. In this example, XX is the set $\{x_1, x_2, x_3, \dots\dots, x_n\}\{x_1, x_2, x_3, \dots\dots, x_n\}$. If probability mass function is $p(x)p(x)$ then the entropy $H(X)H(X)$ can be defined as follows:

$$H(X) = -\sum_{x \in X} P(x) \log P(x) \tag{1}$$

Now, let Y denote a second discrete random variable taking values from the set $YY$of all the features in the software system. In this is an example, YY is the set $\{y_1, y_2, y_3, \dots, y_n\}\{y_1, y_2, y_3, \dots, y_n\}$. The conditional entropy can be defined as follows:

$$\begin{aligned} H(Y \mid X) &= \sum_{x \in X} p(x) H(Y \mid X = x) \\ &= \sum_{x \in X} p(x) \left[-\sum_{y \in Y} p(y \mid x) \log p(y \mid x) \right] \\ &= -\sum_{x \in X} \sum_{y \in Y} p(x, y) \log p(x, y) \end{aligned} \tag{2}$$

Basically, the mutual information of two discrete variables measures the amount of information that one variable contains the information of another one. In another word, we can define mutual information as the reduction in the uncertainty of one random variable due to the knowledge of the other. The mutual information $I(X;Y)I(X;Y)$ of random variables can be defined as follows:

$$I(X;Y) = H(X) - H(X \mid Y) = H(Y) - H(Y \mid X) = I(Y;X) \tag{3}$$

4. Proposed Genetic Algorithm based Software Modularization Clustering (GASMC)

Clustering techniques group the software modules into different clusters that make well program structure. We consider modular dependency graph (MDG) [11] to represent the problem. MDG is a directed graph consist of nodes and edges; where the node represents the modules (e.g. source files, functions) and edges represent the relationship between modules. MDG can be classified into weighted and unweighted. In the weighted MDG, edges of MDG assign a weight that denotes the strong relationships between the modules. Otherwise, it is called the unweighted MDG.

To find the best solution of modularization of software must be encoded in the chromosome. We use integer representation to encode a chromosome. The following figure illustrates that an MDG and its chromosome representation shown in Figure 7.

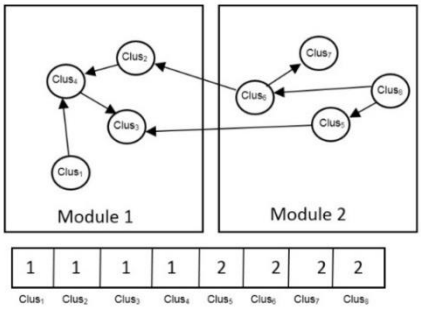

Figure 7. Modular Dependency Graph and its Chromosome Representation in GASMC

In this work, modularization quality (MQ) is appliedtoachieve better fitness scores for clusterings in the genetic algorithm [11]. MQ is the sum of all Modularization Factors (MF). The MF is sum of the ratio of inner edges and outer edges in each cluster, where theinner edge is connected between modules in the same cluster and the outer edge is connected between a module in a cluster and a module in another cluster. MF is demonstrated in the following:

$$MF = 0; if\ i = 0$$

$$MF = \frac{i}{i + \frac{1}{2}j}$$

Where i is the sum of inner edge and j is the sum of theouter edge.
The overall fitness MQ is calculated by:

$$MQ = \sum_{i=1}^{n} MF_i$$

Where i is a cluster and n is the total number of clusters.
We use the fitness proportionate selection that is called roulette wheel selection [11] operator to choose parents from the population pool and reproduce based on the fitness function. Calculate the cumulative fitness of the total population over the sum of the fitness of all individuals. Then, the probability of each individual is calculated for selection. The probability of an individual in the population is being selected as follows:

$$Prob_i = \frac{fitness_i}{\sum_{i=1}^{N} fitness_i}$$

Where, N is the number of individual in the population pool.
We applied one-point crossover and uniform mutation to parents to find the better solution. The pseducode of GASMC is as follows:

Generate n initial chromosomes(solutions); // population size $= n$
repeat {

for $i \leftarrow$ 1 to$k$ {
select two chromosomes p_1, p_2 ;
$offspring_i \leftarrow$ crossover(p_1, p_2);
$offspring_i \leftarrow$ Mutation($offspring_i$);
}
replace k chromosomes in the population with $offspring_1$, …, $offspring_k$;
} **until** (stop condition is satisfied);
return the best chromosome in the population;

Table 1. Test Problems

Name	Modules	Edges	Description
mtunis	20	57	An operating system developed in Turing language for educational purposes
ispell	24	103	Software for spelling and typographical errors correction
rcs	29	163	Revision control system used to manage multiple revisions of files
bison	37	179	A Parser that converts grammar description into C programs
grappa	86	295	Genome rearrangements analyzer
bunch	116	365	Software clustering tool
incl	174	360	Graph drawing tool

Table 2. Comparisons of MQ Values of the Best Solutions Obtained by the Proposed GASMC, Hill Climbing, and GGA Method

MDG Name	GASMC		Hill Climbing		GGA	
	Mean	Std	Mean	Std	Mean	Std
mtunis	2.451	0.027	2.132	0.181	2.308	0.091
ispell	2.547	0.085	2.245	0.119	2.369	0.110
rcs	2.728	0.025	2.256	0.140	2.558	0.088
bison	2.4978	0.089	2.387	0.102	2.241	0.061
grappa	13.486	0.048	12.436	0.130	12.556	0.122
bunch	12.390	0.067	10.031	0.202	11.853	0.126
incl	13.528	0.016	13.011	0.150	13.273	0.143

Table 3. GASMC Parameters and Values

GASMC Parameters	Values
Population Size	100
Probability of crossover	0.5
Probability of mutation	0.01
Number of Generations	No improvement in several successive generations
Fitness Function	MQ
Selection Method	Roulette Wheel
Types of crossover	One-point
Types of mutation	Uniform

5. Results and Discussion

In this section, the result is obtained by proposed method are explained. In order to evaluate the effectiveness of proposed Genetic Algorithmbased software modularization clustering (GASMC) to software clustering problem, we applied it to seven real-world software module clustering problems [12], and compared its output to that of other state-of-art-algorithms software clustering algorithms such as, Hill Climbing [13], and group genetic clustering (GGA) [14]. Table 1 illustrate that the test problem sand the parameters that are selected for performance evaluation. The number of modules varies from 20 to 174 and their edges differ from 57 to 365. In this work, we have considered only unweightedmoduledepengy graphs (MDGs). The description of the test problems is given table 1.We applied our proposed GA to each of the 7 test problems independently and performed 30 runs respectively. In each run, we also calculated the mean and standard deviation of MQ and the achieved highest MQ value represent the best solutions. Table 2 illustrates that comparison of MQ Values of the best

solutions obtained by the proposed method and state of the art method. Compared with hill climbing, the mean valuesof MQ obtained by GGA are better than that of hill climbing, and the standard deviations obtained by GGA are also better, which indicates that GGA is more stable than hill climbing. On the other hand, the mean values obtained by GASMC are better than those of hill climbing and GGA, and the standard deviations are obtained by proposed GASMC are very low than those of hill climbing and GGA. The GASMC produced higher MQ value that indicates that it can find a better partition of modules into clusters and standard deviation also indicate the consistency performance. Table 3 illustrates the GASMC parameters and values that are used in the simulation results.

6. Conclusion

To improve the program structure and maintenance of larger software system is a difficult task in software engineering. Efficient clustering techniques are abetter solution for decomposing a larger software system into a smaller system. This work presented most recent clustering techniques and demonstrated how it can be applied to software engineering. This paper also proposed a genetic algorithm based software modularization method for software module clustering. The result section strengthen that proposed method is able to produce better module p and it outperforms the state of the art. In future, we will focus on designing more effective software modularization clustering using hybrid algorithms.

References

[1] KramerHH, Uchoa E, Fampa M, Köhler V, Vanderbeck F. Column generation approaches for the software clustering problem. *Computational Optimization and Applications.* 2016: 1-22.
[2] Andritsos P, Tzerpos V. Software clustering based on information loss minimization*. IEEE*. 2003: 334.
[3] Choi SC, Scacchi W. Extracting and restructuring the design of large systems. *IEEE Software.* 1990; 7(1): 66-71.
[4] Hutchens DH, Basili VR. System structure analysis: Clustering with data bindings*. IEEE Transactions on Software Engineering.* 1985; 11(8): 749–757.
[5] MullerHA, UhlJ.S. *Composing Subsystem Structures using (k, 2)-partite Graphs.*Conference on Software Maintenance. November 1990: 12–19.
[6] Tzerpos V, Holt RC. *ACDC: An algorithm for comprehension driven clustering.* Proceedings of the Seventh Working Conference on Reverse Engineering. 2000: 258–267.
[7] Everitt B. Cluster Analysis. London: Heineman Educational Books. 1974.
[8] Bauer M, Trifu M. *Architecture-aware adaptive clustering of OO systems.* Proceedings of the 8th European Conference on Software Maintainance and Reengineering (CSMR '04).Tampere. Finland. 2004: 3–14.
[9] Tommikarkkainen S. Introduction to partitioning-based clustering methods with a robust example. 2006.
[10] Andritsos P, Tzerpos V. Information-theoretic software clustering*. IEEE Transactions on Software Engineering.* 2005; 31 (2): 150-165.
[11] Mancoridis S, Mitchell BS, Rorres C, Chen, YF, Gansner ER. Using Automatic Clustering to Produce High-Level System Organizations of Source Code. *IWPC.* 1998; 98: 45-52.
[12] Mitchell BS. A heuristic search approach to solving the software clustering problem. Doctoral dissertation. Drexel University.
[13] MahdaviK, Harman M, Hierons RM. *A multiple hill climbing approach to software module clustering.* Proceedings of the international Conference onSoftware Maintenance. *IEEE.* 2003; 315-324.
[14] Praditwong K. Solving software module clustering problem by evolutionary algorithms. *Computer Science and Software Engineering (JCSSE)*. IEEE Eighth International Joint Conference on. 2011: 154-159.