A Modified Genetic Algorithm for Software Clustering Problem

Ali Asghar Pourhaji Kazem
Computer Department
Islamic Azad University of Tabriz
Tabriz, Iran

Shahriar Lotfi
Computer Science Department
Islamic Azad University of Tabriz
Tabriz, Iran

Abstract: Structure of most software systems is large and complex. Therefore, understanding of these software systems structures is difficult. The reason for this complexity is dependency of many modules of them to each other. Software clustering is the process that divides software systems into meaningful partitions. Software clustering algorithms try to find near optimal partitions from the extraordinarily large space of possible partitions. According to large space of possible partitions software clustering problem is NP-Hard. Genetic algorithms can be use for this kind of problems. In this paper, a modified genetic algorithm, proposed for software clustering. Results of proposed algorithm show that it works better than other genetic software clustering algorithms.

Key-Words: Reverse Engineering, Software Clustering, Genetic Algorithm, Genetic Clustering Algorithm

1 Introduction

Software maintenance is one of the important steps in software development process [10]. Total cost of software maintenance is estimated to comprise at least 50% of total life cycle. Also understanding the software system is very important for maintenance phase. However, when a software system becomes large, understanding of it will be difficult [7, 5].

Reverse engineering can used for solving this problem. Reverse engineering is the process of analyzing software with the objective of recovering its design and specification [1]. The software itself is unchanged by the reverse engineering process [11]. The software source code is usually available as the input of the reverse engineering process. The objective of reverse engineering is to derive the design or specification of a software system from its source code. The process of reverse engineering starts with an analysis phase. During this phase, the system is analyzed using automated tools to discover its structure [10, 11]. Discovered structure of software can viewed as a directed graph called Module Dependency Graph (MDG). The software Modules are represented as nodes and relationships between them (i.e., function calls, inheritance) as directed edges in MDG [3, 7].

One way making MDGs more accessible and understandable is to partition them. Partitioning of MDG is called software clustering. The main objective of software clustering process is partitioning the MDG of software to meaningful partitions. In fact, partitioning of MDG is separating its nodes (i.e., modules) into clusters (i.e., subprograms).

Software clustering algorithms try to find good partition from the extraordinarily large space of possible partitions [7, 14]. Therefore, software-clustering problem is NP-Hard and genetic algorithms generally are used for this kind of problems [7, 12, 13].

Genetic algorithms formally introduced in the United States in the 1970s by John Holland at University of Michigan [4, 9]. Genetic algorithms are ubiquitous nowadays having been successfully applied to numerous problems from different domains, including optimization, automatic programming, machine learning, economics, operations research, ecology, population genetics, studies of evolution and learning and social systems [4]. Genetic algorithms mimic the process of natural selection and can be used as a technique for solving complex optimization problems, which have very large search spaces. Genetic algorithms have been found to overcome some of the problems of traditional search methods such as hill climbing.

A GA is an algorithm that operates on a similar principle. When applied to a problem the standard genetic algorithm proceeds as follows: an initial population of individuals is generated at random. An individual is a solution for the given problem represented by chromosomes. Each chromosome consists of genes (i.e., bits), with each gene being an instance of a particular allele. The random generated set of individuals called population. At every evolutionary step, known as a generation, the individuals in the current population decoded and evaluated according to some predefined quality criterion, referred to as the fitness function. The next
generation formed by selecting the individuals from current generation according to their fitness value [9]. There are many selection techniques that currently in use. One of the simplest selection procedures is Holland's original selection, where individuals selected with a probability proportional to their fitness values. Thus, individuals with high fitness values have a better chance of reproducing, while individuals with low fitness values are more likely to disappear [4, 9].

The next step in the GA is crossover operation. Crossover exchange subparts of two chromosomes, roughly mimicking biological recombination between two single chromosome organisms. After crossover operation, the mutation operator applied to some chromosomes in the population. Mutation randomly changes the allele values of some locations in the chromosome. The above actions repeated until some condition is satisfied [4].

If the GA has been designed well, the population will converge to an optimal solution to the problem. However, GAs are stochastic iterative processes and are not guaranteed to converge; hence the termination condition may be considered as producing fixed number of generations or reaching a acceptable fitness value.

Many clustering algorithms are used for clustering of software systems. Anquetil designed a hierarchical clustering algorithm suite, which offers a selection of association and distance coefficients as well as update rules [2]. Mancoridis and Mitchell provide a suite of algorithms for software clustering that include Hill Climbing, Exhaustive, and Genetic Algorithms [3, 5, 6, 7]. In this paper, a modified genetic algorithm is proposed for software clustering that produce best results compared to Mancoridis and Mitchell genetic algorithm.

The rest of this paper is organized as follows: Section 2, provides proposed genetic algorithm for software clustering. Section 3, proposed a discussion about action of proposed algorithm. Experimental results of proposed algorithm are shown in Section 4 and finally Section 5 concludes this paper.

2 Genetic Clustering Algorithm

In this section, proposed genetic algorithm for software clustering problem that includes encoding, creation of initial population, fitness functions, crossover operator and mutation operator.

2.1 Encoding

Integer strings are used for proposed genetic algorithm. It is assumed that nodes in MDG have unique identifiers from zero to \( N - 1 \). Each chromosome is a string with length \( N \), consists of non-negative integer numbers less than \( K \). \( K \) is the number of clusters in the chromosome. Integer numbers in the string indicates the cluster labels that contains nodes. For example, the MDG in Fig.1 is encoded as following string \( S \):

\[
S: 1 0 0 2 1 3 3 \quad (K = 4)
\]

Formally, each chromosome is a string that is defined as:

\[
S: l_0 l_1 \ldots l_{k-1}
\]

where \( K \) is the number of clusters in the chromosome and \( l_i \) is the cluster label that contains node with identifier \( i \).

![Fig 1. A sample partition](image)

2.2 Creation of Initial Population

Encoding and creation of initial population are very important for efficiency of genetic algorithm [4]. If initial population is created such a way that its chromosomes have better talent for reaching effective answer, genetic Algorithm converges quickly. Therefore, two different methods are used for creating initial population and the above criterion is considered.

2.2.1 Method 1

Maximum number of clusters for a MDG is \( N \), where \( N \) is the number of nodes in MDG. For creating one chromosome in this method, at first, an
integer random number named K (K ≤ N) is
produced and then the number of clusters in the
chromosome is considered K. After identifying K
different genes are selected randomly and the
selected genes are setting with integer numbers 0 to
K - 1. This guarantees that number of clusters in the
chromosome is K. Finally, remained genes from
chromosome set with random non-negative integer
numbers less than K. Fig.2 show the algorithm used
in method 1 for creating initial population.

**Algorithm 1:**

1. K = Random (N)
2. Select K different Genes from chromosome randomly.
3. Set selected genes with integer numbers 0 to K – 1.
4. Set remained genes from chromosome with random non-
negative integer numbers less than K.

**Fig.2 Algorithm of method 1 for initial population creation**

### 2.2.2 Method 2

At first Graph Center Node (GCN) is defined as one
node in graph, that its degree is greater than the
average degree of all nodes in graph. According to
software clustering concepts, it is efficient to place
MDG center nodes in different clusters. A new
heuristic method is introduced, that expresses the
number of clusters in the last result is closer to the
number of GCNs in MDG.

In this method, number of GCNs (GN) in MDG
is found and then method 1 is used for creating
chromosomes. The main difference between this
method and method 1 is in creating random numbers
for cluster count in the chromosome. In this method,
normal distribution with mean equal GN is used for
creating random numbers for cluster counts. This
guarantees that the major number of created random
numbers be equal to number of center nodes in
MDG. Fig.3 show the algorithm used in method 2
for creating initial population.

**Algorithm 2:**

1. Find number of GCNs in MDG.
2. Produce a random non-negative integer number for K with
normal distribution.
3. Set selected genes with integer numbers 0 to K - 1.
4. Set remained genes from chromosome with random non-
negative integer numbers less than K.

**Fig.3 Algorithm of method 2 for initial population creation**

### 2.3 Fitness Function

Two different fitness functions are used in new
genetic algorithm. The BasicMQ\(^1\) measurement [3,6,7] is the first fitness function and TurboMQ
measurement [7] is the second one that is used in
new genetic algorithm.

The BasicMQ measures inter-connectivity (i.e.,
connections between the components of two distinct
clusters) and intra-connectivity (i.e., connections
between the components of the same cluster)
independently. The BasicMQ fitness function is
designed to produce higher values as the intra-
connectivity increases and the inter-connectivity
decreases. The intra-connectivity measurement \(A_i\) of
cluster \(i\) consisting of \(N_i\) components and \(\mu_i\)
 intra-edge relations, is defined as:

\[
A_i = \frac{\mu_i}{N_i^2}
\]  

(1)

The value of \(A_i\) is between 0 and 1. \(A_i\) is 0
when modules in a cluster do not share any
software-level resources; \(A_i\) is 1 when every module
in a cluster uses a software resource from all of the
other modules in its cluster (i.e., a complete graph).

Inter-Connectivity (E) measures the degree of
connectivity between two distinct clusters. A high
degree of inter-connectivity indicates a poor
subsystem partition. The inter-connectivity \(E_{ij}\)
between clusters \(i\) and \(j\) consisting of \(N_i\) and \(N_j\)
components, respectively, and with \(e_{ij}\) inter-edge
dependencies is defined as:

\[
E_{i,j} = \begin{cases} 
0 & \text{if } i = j \\
\frac{e_{ij}}{2N_iN_j} & \text{if } i \neq j 
\end{cases}
\]  

(2)

The value of \(E_{ij}\) is between 0 and 1. \(E_{ij}\) is 0 when
there are no module-level relations between cluster \(i\)
and cluster \(j\); \(E_{ij}\) is 1 when each module in cluster \(i\)
depends on all of the modules in cluster \(j\) and vice-
versa. After formal definition of inter-connectivity
and intra-connectivity, the BasicMQ measurement
is defined for a MDG partitioned into \(k\) clusters as:

\[
BasicMQ = \begin{cases} 
\frac{1}{K} \sum_{i=1}^{K} A_i & \text{if } k > 1 \\
A_i & \text{if } k = 1 
\end{cases}
\]  

(3)

\(^1\) Modularization Quality
The value of the BasicMQ measurement is bounded between -1 (no cohesion within the subsystems) and 1 (no coupling between the subsystems).

TurboMQ is another fitness function that is introduced in [7]. Formally, the TurboMQ measurement for an MDG partitioned into $k$ clusters is calculated by summing the Cluster Factor (CF) for each cluster of the partitioned MDG. The Cluster Factor, $CF_i$, for cluster $i$ ($1 \leq i \leq K$) is defined as a normalized ratio between the total weight of the internal edges (edges within the cluster) and half of the total weight of external edges (edges that exit or enter the cluster) [7]. Similar to BasicMQ fitness function, $\mu_i$ refer to the internal edges of a cluster as intra-edges, and the edges between two distinct clusters $i$ and $j$ as inter-edges ($\varepsilon_{i,j}$ and $\varepsilon_{j,i}$ respectively). Below, the TurboMQ is defined:

$$\text{TurboMQ} = \sum_{i=1}^{k} CF_i$$  \hspace{1cm} (4)

$$CF_i = \begin{cases} 0 & \mu_i = 0 \\ \frac{2\mu_i}{2\mu_i + \sum_{j=1}^{k} (\varepsilon_{i,j} + \varepsilon_{j,i})} & \text{otherwise} \end{cases}$$  \hspace{1cm} (5)

The maximum value for $CF_i$ is one and therefore, the value of the TurboMQ measurement is bounded between zero and $K$.

2.4 Penalty

As stated in section 2.2.2, in method 2 for initial population creation, number of GCNs (GN) in MDG is identified and then normal distribution is used for producing random numbers for $K$ (number of clusters in the chromosome). Mean ($\mu$) and standard deviation ($\sigma$) of normal distribution are defined as:

$$\mu = GN$$

$$\sigma = \text{Min} \left(N - GN, GN\right)$$  \hspace{1cm} (6)

Above definitions for $\mu$ and $\sigma$, lead to producing random integer numbers for $K$, near to GN and between 0 and N-1. It is guessed that the number of clusters in the near optimal result is near to GN. Therefore, penalty is used to fine the chromosomes that produced cluster number for them is not equal to GN. Penalty for chromosomes is defined as:

$$\text{Penalty} = \left| \frac{K - GN}{N} \right|$$  \hspace{1cm} (7)

According to defined penalty, Chromosomes that their cluster numbers are equal to GN ($K = GN$), have zero penalty value, But chromosomes that their cluster number is not equal to GN ($k \neq GN$), have positive penalty value and this value is subtracted from their fitness value. Using penalty values, increases the chance of selection of chromosomes that have cluster numbers equal to GN.

2.5 Selection

Roulette wheel selection is the common method for selection in the genetic algorithms. Therefore, roulette wheel method is used for selection in the proposed algorithm. In addition, elitism is applied to selection process in the proposed algorithm. Elitism is an addition to selection method that forces the genetic algorithm to retain some number of the best individuals at each generation [4, 9]. Such individuals can be lost if they are not selected to reproduce or if they are destroyed by crossover or mutation.

2.6 Crossover Operator

Three different crossover operators that are used in the proposed algorithm are one point, two point and uniform crossover. After many experiments on different MDGs, it is determined that two-point crossover operator is better than others are. Crossover operator combines two individuals with a probability named crossover rate. Crossover rate in the proposed algorithm is considered 75% for populations of 100 individuals or fewer and 100% for populations of greater than 100 individuals.

2.7 Mutation Operator

The mutation operator randomly selects two genes of a chromosome and exchanges the value of them. Typical mutation rate for binary encoding is 0.004. However, because decimal encoding is used in this algorithm, the number of bits that would have been used if the encoding were binary must multiply stated mutation rate. Therefore, mutation rate is considered $0.004 \log_2$ [3, 6, 7].
3 Discussion

In this section, the proposed genetic algorithm for software clustering is compared with genetic software clustering algorithm in [3, 7].

Suppose, one MDG that is complete graph. Complete graph is one, that each node of it has one edge to each other nodes in graph. Best answer for clustering this MDG based on BasicMQ fitness function is that the all nodes of MDG placed in the same cluster. If the algorithm in [3, 7] and BasicMQ fitness function is used for clustering this MDG, fitness values for chromosomes in the population will be 0 or 1. Chromosomes that indicate one cluster have the fitness value 1 and other chromosomes have the fitness value 0. With considering that the initial population created randomly, the number of chromosomes with one cluster is near to zero. Therefore, the number of chromosomes with zero fitness value in the initial population is very high and this makes convergence of genetic algorithm difficult.

If the proposed genetic algorithm is used for clustering the complete MDG and method 1 is used for creating initial population, algorithm convergences quickly at primary populations. The reason for quick convergence of proposed genetic algorithm is that the chance for creation of chromosomes with one cluster in the initial population is high.

For comparing the proposed genetic clustering algorithm with genetic clustering algorithm in [3, 7], both of them is implemented with Delphi programming language. Both of algorithms are tested with different random MDGs and the proposed algorithm produced efficient results. Differences in efficiency of algorithms, is stated in next subsection.

4 Experimental Results

For comparing the proposed genetic clustering algorithm with genetic clustering algorithm in [3, 7], both of them is implemented in Delphi environment. Different experiments are performed and the results of these experiments stated in the rest of this section.

At first three different random MDGs with different number of nodes, are clustered with new algorithm and Fig.4 shows the maximum fitness value (MQ) of each generation. This experiment shows that new algorithm how convergence in some point.

Three different random MDGs with different number of nodes are clustered with proposed algorithm ten times, and Fig.5 shows the stability of proposed algorithm.

For comparing the efficiency of proposed algorithm and algorithm introduced in [3, 7], three different random MDGs with different number of nodes are clustered with both of algorithms and final fitness values of each algorithm is shown in Fig.6. TurboMQ fitness function is used in both of algorithms and method 2 is used for creating initial population in the proposed algorithm. Results of this experiment explicitly show that proposed algorithm work better than the other algorithm and produce best partitions for MDGs.
Fig. 7 shows the final fitness values produced by both of algorithms applying to different MDGs that are complete graphs. BasicMQ fitness function is used for both of algorithms and method 1 is used for creating initial population in the proposed algorithm. Results show that the proposed algorithm found best fitness value for those MDGs, because the maximum value of BasicMQ is one, but the other algorithm only found best results for small number of nodes and not for MDGs with great number of nodes.

![Comparing proposed algorithm with previous algorithm for complete MDGs](image)

5 Conclusion

Structure of most software systems is large and complex and therefore, understanding of these software systems structures is difficult. Software clustering algorithms are used for dividing these software systems into meaningful partitions. According to large space of possible partitions, software-clustering problem is NP-Hard. Therefore, genetic algorithms can be use for software clustering problem.

In this paper, a modified genetic algorithm proposed for software clustering problem. Proposed algorithm has implemented in Delphi environment. Proposed algorithm runs on different MDGs and experimental results showed that stability of proposed algorithm is well. Also, results of experiments showed that proposed algorithm found best results compared to genetic software clustering algorithm in [3, 7].

References: