DE GRUYTER
OPEN

JACSM 2016, Vol. 8, No. 2, pp. 99 - 113
10.1515/jacsm-2016-0007

SOFTWARE SYSTEMS CLUSTERING USING ESTIMATION OF DISTRIBUTION APPROACH

Mahjoubeh Tajgardan, Habib Izadkhah, Shahriar Lotfi

Department of Computer Science, Faculty of Mathematical Science, University of Tabriz, Tabriz, Iran
mahjoubeh_tajgardan@yahoo.com; izadkhah@tabrizu.ac.ir; shahriar_lotfi@tabrizu.ac.ir

Abstract

Software clustering is usually used for program understanding. Since the software clustering is a NP-complete problem, a number of Genetic Algorithms (GAs) are proposed for solving this problem. In literature, there are two well-known GAs for software clustering, namely, Bunch and DAGC, that use the genetic operators such as crossover and mutation to better search the solution space and generating better solutions during genetic algorithm evolutionary process. The major drawbacks of these operators are (1) the difficulty of defining operators, (2) the difficulty of determining the probability rate of these operators, and (3) do not guarantee to maintain building blocks. Estimation of Distribution (EDA) based approaches, by removing crossover and mutation operators and maintaining building blocks, can be used to solve the problems of genetic algorithms. This approach creates the probabilistic models from individuals to generate new population during evolutionary process, aiming to achieve more success in solving the problems. The aim of this paper is to recast EDA for software clustering problems, which can overcome the existing genetic operators' limitations. For achieving this aim, we propose a new distribution probability function and a new EDA based algorithm for software clustering. To the best knowledge of the authors, EDA has not been investigated to solve the software clustering problem. The proposed EDA has been compared with two well-known genetic algorithms on twelve benchmarks. Experimental results show that the proposed approach provides more accurate results, improves the speed of convergence and provides better stability when compared against existing genetic algorithms such as Bunch and DAGC.

Key words: Software System, Clustering, Genetic Algorithm, Estimation of Distribution Algorithm (EDA), Probability Model

1 Introduction

In large software systems, program comprehension is an important factor for its development and maintenance [1]. Clustering is presented as a key activity in reverse engineering to extract software architecture (structure) to improve

understanding of the program [2]. The aim of the software clustering methods is to automatically group the similar artifacts of a software system together into clusters and discover the software structure based on relationships between artifacts in a software system, in which the relationships between the artifacts of different clusters are minimized, and the relationships between the artifacts of the same cluster are maximized (maximum cohesion and minimum coupling) [4]. In general, low coupling and high cohesion are characteristics for well-designed software systems [3]. The first stage in the software clustering is to extract a Call Dependency Graph (CDG) from the program to improve the comprehensibility of complex software systems [4]. CDG is usually used in search-based clustering algorithms for modeling the calls between artifacts. Figure 1 shows a sample of the clustered call dependency graph of a program. In this sample the relationship between artifacts in clusters is high and the coupling between clusters is low (well-designed).

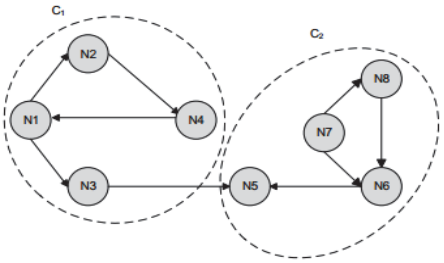

Figure 1. Clustered call dependency graph

Considering huge search space, the problem of finding the best clustering for a software system is a non-deterministic polynomial complete (NP-Complete) problem, hence, the necessity of the use of evolutionary algorithms to achieve a proper clustering is known [4]. Some genetic algorithms are provided in the context of software clustering in which communication and information exchange between individuals is done through the selection and recombination of the individual in a generation. This information movement causes partial solutions to combine with each other, and then higher quality solutions are obtained possibly. With all positive features that the standard genetic algorithm has, the major drawback of this algorithm is that the behavior of genetic algorithm depends on parameters like how to define the crossover and mutation operators and their probabilities, etc. [5]. Therefore, researcher requires experiments in order to choose the suitable values for these parameters [5]. Crossover is a process of taking the pairs of selected parents and producing new offspring from them. The aim of mutation operator is to avoid 'getting stuck' at local optimum points, maintain genetic diversity and discover new areas of the search space. These operators are executed serially. Crossover and mutation operators have a fixed rate of happening (i.e., the operators are applied with a fixed probability) that varies across problems. In

problems that require certain crossover and mutation operators, defining these operators is very difficult and complex, because genetic operators must be defined in a way that can produce valid individuals. For example, in a permutation-based encoding, we need operators that can be maintained as well as the permutation property of individuals [6]. Moreover, in some problems in which the genetic operators do not guarantee the building blocks protection, GA shows a poor performance [5].

In order to overcome the disadvantages of the genetic algorithm, a class of evolutionary algorithms called estimation of distribution algorithms (EDAs) is provided that has advantages in comparison with genetic algorithm. These advantages include [5]: (1) generating new individuals using the probability distribution of all virtualization solutions of previous generation, instead of using the genetic operators; (2) maintaining building blocks in successive generations by giving more chances to partial solutions; and (3) improving the speed of progress towards optimal solution by maintaining building blocks.

In this paper, we recast the estimation of distribution algorithm for software systems clustering, aiming to overcome the genetic algorithm problems and achieving the better clustering by keeping the building blocks during evolutionary process. We propose a probability distribution function to generate a new population using the features of clustering problem, which can solve the problem using genetic operators such as crossover and mutation. The results of our experiments showed that our estimation of distribution algorithm can provide acceptable clustering from the perspective of a domain expert, and as a result contribute to the understanding of software system.

The structure of the rest of this paper is as follows: Section 2 provides some background about EDA and addresses the limitations of the existing works. Section 3 explains the proposed algorithm for software clustering using EDA. Section 4 gives the results of applying our clustering algorithm and some known evolutionary algorithms and discusses on results. Finally, Section 5 concludes the paper.

2 Background and Related Works

In this section, we provide the basic information required for software systems clustering using the estimation of distribution algorithms and some related works in the field of software systems clustering.

2.1 Estimation of distribution algorithms

EDAs are population-based search algorithms based on probabilistic modeling of promising solutions [5]. In EDAs the new population is generated using a probability distribution estimated from the selected individuals of the previous

generation [5]. Figure 2 illustrates the EDA approaches in the optimization process. The EDA follow the following steps in the optimization process:

1. Firstly, the initial individuals of size μ as initial population are generated. The generation of these μ individuals is usually carried out by assuming a uniform distribution on each variable. Then, each individual using fitness function is evaluated.
2. Secondly, λ individuals (where $\lambda \leq \mu$) based on specified criteria are selected for calculating the n–dimensional probabilistic model that better represents the interdependencies. The aim is to calculate joint probability distribution of selected individuals. This step is also known as the learning procedure, and it is the most crucial one, since representing appropriately the dependencies between the variables is essential for a proper evolution towards fitter individuals.
3. New population is generated according to calculated probability distribution.
4. Finally, new population is replaced into previous population.

Steps 2, 3 and 4 are repeated until a stopping condition is verified. Examples of stopping conditions are: achieving a fixed number of populations or a fixed number of different evaluated individuals, uniformity in the generated population, and the fact of not obtaining an individual with a better fitness value after a certain number of generations.

2.2 Software clustering algorithms

Generally, in literature, software clustering algorithms can be generally categorized into the following groups:

1. Clustering algorithms based on concept analysis [7]: In such algorithms, the goal is to classify procedures and variables into clusters. Clustering algorithms of this group are merely used for extracting software architecture form the respective procedural codes and are not conclusive for large systems, as quoted by the author [7].
2. Hierarchical clustering algorithms [8-11]: In these algorithms, each entity is initially considered in a separate cluster, and then; these clusters are gradually combined with each other creating larger clusters. These algorithms provide hierarchical structure from system architecture [8]. The pitfall of hierarchical methods is their failure to benefit from software engineering criteria for determining clusters or code clusters. The hierarchical approaches seem to be useful for program understanding and knowledge discovery, because in general they allow the original problem to be studied at different levels of detail by navigating up and down the hierarchy. However, it is a difficult problem to find the appropriate

height at which to prune a hierarchy of clusters to obtain optimal partitioning.

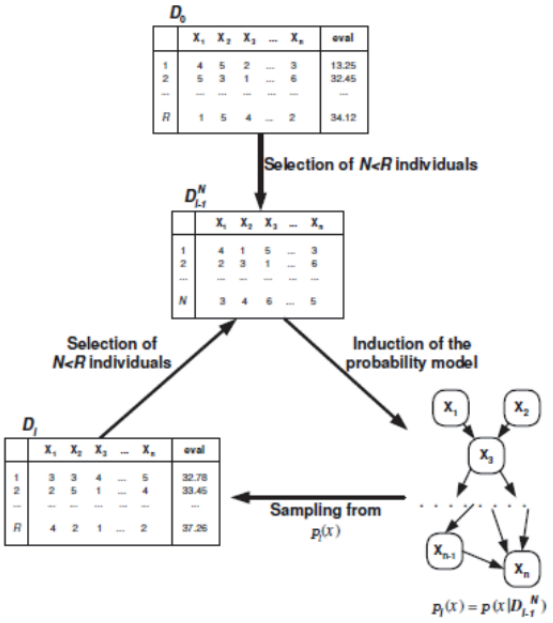

Figure 2. Illustration of EDA approaches in the optimization process

3. Search-based clustering algorithms: clustering problem is treated as a search task in these algorithms. Since searching the complete state space turns the situation into a NP-Complete problem [11], heuristic search techniques such as genetic algorithm are deployed for finding the optimal or near optimal answer during a reasonable time. Search operation is carried out using criteria of maximal cohesion and minimal coupling of clusters. These criteria are particularly suitable in object-based systems for identifying sub-systems or clusters. These methods are divided into two categories: global search (Such as Bunch [12, 13] and DAGC [6]), local search (Such as SAHC [13] and NAHC [13]) and combining local and global search (Such as HC+Bunch [14]) methods. The main drawback of local search methods is that they have the risk of getting stuck in local maximum values, but, global search methods are able to escape from these local maximums [12]. Search-based algorithms have been able to achieve better results than the hierarchical techniques.

Genetic algorithms are widely and effectively used for NP-Hard optimization problems. They can produce acceptably near-optimal answers in reasonable time [6]. Genetic clustering algorithms are very subjective [15]; well-known tools such as DAGC, Bunch use genetic algorithm for clustering soft-

ware systems. In the Bunch [8], each individual is an array that the number of its genes is equal to the number of nodes in the call dependency graph (CDG) and the content of each gene identifies a cluster that contains the corresponding node. In the DAGC [6], each array (individuals) is a permutation of the nodes of N integer. An individual can be decoded into a clustering by the following process: m^{th} cell of the individual represents the node number 'm' of the CDG. Its content includes number of another node of graph like 'p' ($1 \le p \le N$) and if 'p' is equal or greater than 'm', then 'm' is placed in a new cluster otherwise 'm' belongs to the same cluster as 'p'.

Objective function used in Bunch and DAGC and our algorithm is TurboMQ [11, 12]. If the internal edges of cluster and edges between two clusters are respectively represented by μ_i and $\varepsilon_{i,j}$, TurboMQ value will be then computed as follows:

$$CF_i = \begin{cases} 0 & \mu_i = 0 \\ \dfrac{2\mu_i}{2\mu_i + \sum_{j=1}^{k} (\varepsilon_{i,j} + \varepsilon_{j,i})} & otherwise \end{cases} \quad (1)$$

$$TurboMQ = \sum_{i=1}^{k} CF_i \quad (2)$$

3 The proposed algorithm

Search-based software clustering methods such as the Bunch and DAGC are clearly superior to the hierarchical methods. However, they have a particular drawback that it was explained before. We try to address it by introducing a new algorithm in this section. This new algorithm is based on probabilistic model and does not use genetic operators such as crossover and mutation, instead keep the building blocks. In other words, we present a probabilistic model to generate a new population. This section explains our proposed probabilistic model (subsection 3.1) and software clustering using EDA (subsection 3.2).

3.1 Probabilistic model

To obtain the probability model, let M be an n×n square matrix (where n is the number of software artifacts), and initially the values of all elements except the main diagonal are 1/n. The value of M[i, j] represents the probability that two artifacts will be placed in the same cluster on individuals in the next generation. For example, if we have software system containing 5 artifacts, initially, the probability will be as shown in Table (1).

Table 1. The initial probability matrix

	F1	F2	F3	F4	F5
F1	0	0.2	0.2	0.2	0.2
F2	0.2	0	0.2	0.2	0.2
F3	0.2	0.2	0	0.2	0.2
F4	0.2	0.2	0.2	0	0.2
F5	0.2	0.2	0.2	0.2	0

$$M[i,j] = \frac{1}{n} \qquad i,j = 1,2,\dots,n \tag{3}$$

Then, we find the best and worst individuals of the population in each generation and change the values of probability matrix as follows:

1. If two artifacts i and j are placed in the same cluster in the best individual but are not placed in the same cluster in the worst individual (suppose t is the number of iterations for the evolutionary algorithm):

$$\text{if } M[i,j] < 1 - \frac{5}{t} \qquad M[i,j] = M[i,j] + \frac{5}{t} \tag{4}$$

2. If two artifacts i and j are placed in the same cluster in the worst individual but are not placed in the same cluster in the best individual:

$$\text{if } M[i,j] > \frac{5}{t} \qquad M[i,j] = M[i,j] - \frac{5}{t} \tag{5}$$

3. If two artifacts i and j are placed in the same cluster in the best and the worst individuals, the value of probability in the probability matrix does not change.

After changing the probability model, the new population is produced using the new possibilities. For example, Suppose in the Table (2), (a) and (b) are the best and the worst individuals respectively, then new probabilities assuming t=100 are given in Table (3).

It is obvious that in this model the probability of any two artifacts is not equal to 0 and 1. We consider this condition for maintaining the diversity of our population and preventing premature convergence.

Table 2. The best and worst individual

	F1	F2	F3	F4	F5
(a)	3	3	1	1	2
(b)	3	1	2	2	3

Table 3. The new probability matrix

	F1	F2	F3	F4	F5
F1	0	0.25	0.2	0.2	0.2
F2	0.25	0	0.2	0.2	0.2
F3	0.2	0.2	0	0.2	0.2
F4	0.2	0.2	0.2	0	0.2
F5	0.2	0.2	0.2	0.2	0

3.2 Clustering using EDA

In our proposed algorithm each solution is shown as an individual. To represent individuals, we use Bunch algorithm's chromosome representation [11, 12], but with limited number of clusters. In the Bunch, each individual is an array that the number of its genes is equal to the number of nodes in the call dependency graph (CDG) and the content of each gene identifies a cluster that contains the corresponding node and its numeric value is between one to N that N is the number of nodes in the CDG. Formally, an encoding on a string S is defined as:

$$S = s1\ s2\ s3\ s4\ \ldots\ sN \tag{6}$$

Where, N is the number of nodes in the CDG and s_i identifiers the cluster that contains the i^{th} node of the graph. For example, the graph in Figure 3 is encoded as the following string S:

$$S = 2\ 2\ 3\ 3\ 1\ 1\ 1$$

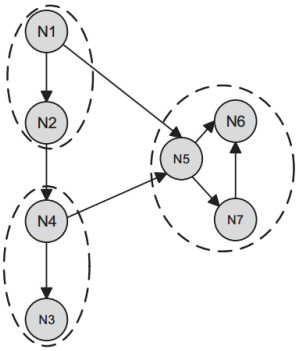

Figure 3. A sample clustering

In contrast with existing genetic based algorithm for software clustering, in our method, we use the probability model instead of the using crossover and mutation. In the proposed algorithm, first an initial population of individuals is generated randomly and the individuals are evaluated using TurboMQ fit-

ness function. Then until the termination condition is established, individuals are selected using the selection operator and then offspring is generated according to calculated probability distribution. The previous population is replaced by the new population. Figure 4 shows two first generation of our proposed algorithm. In this figure initial population indicate the number of individuals and the corresponding fitness.

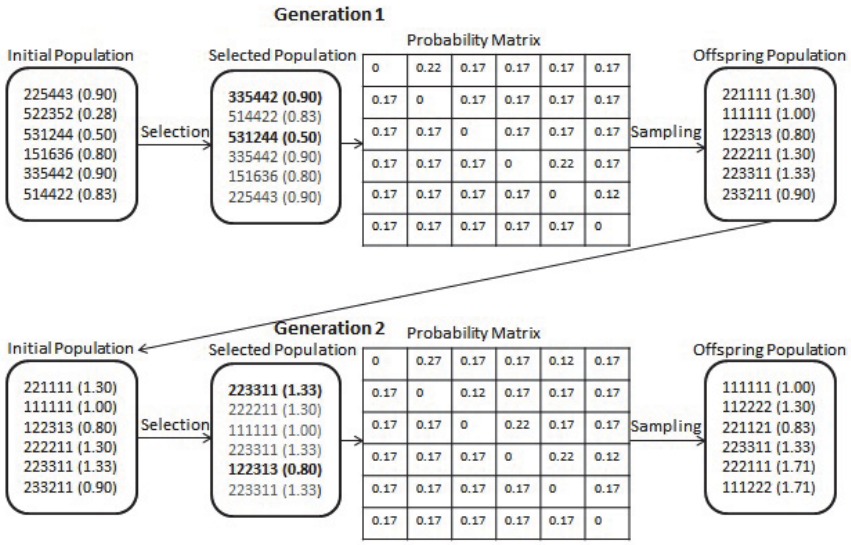

Figure 4. Two first generation of our proposed algorithm

We use the top of triangle probability matrix for the generation of each individual in new population as Algorithm 1. In any iteration, the old population is replaced by new produced population. Briefly, the EDAs process is as Algorithm 2.

Algorithm 1: New population generation algorithm

- Until for each artifact, the cluster is not determined, the following steps are repeated:
  1. Select a row from the probability matrix, randomly.
  2. For each two artifacts generate a random number from [0, 1]. If the value of the random number is smaller or equal to their probability value, two artifacts are placed in the same cluster.

Algorithm 2: EDA based software clustering algorithm

BEGIN
Generate initial population of size μ, randomly.
Evaluate each individual using TurboMQ (Eq. 2)

While (a fixed number of generation is not achieved)

1. Select λ=2 promising and worse individuals (where $\lambda \leq \mu$);
2. Calculate the probability distribution matrix using selected individuals (Section 3.1);
3. Generate new population according to calculated probability distribution;
4. Evaluate each new offspring using TurboMQ;
5. Replace offspring into main population;

END

4 Experimental Results

In this section, we compare the results obtained by proposed EDA and five well-known algorithms such as Bunch, DAGC, NAHC (Next Ascent Hill Climbing), SAHC (Steepest Ascent Hill Climbing), HC+Bunch. For evaluating the obtained clustering, internal and external metrics are used. External one is used to compare results of obtained clustering algorithm by the clustering provided by a domain expert. In fact, the external metrics are used for assessing the reliability of an algorithm. Mojo [16], edgeMojo [17], Precision/Recall [11], and F_m as harmonic mean of Precision/Recall are of external metrics. Mtunis is an academic operating system and since the clustering of this operating system is available so we've used it to evaluate the reliability of proposed algorithm. When the clustering produced by the expert is not available, internal metrics can be used. Table 4 shows the comparison of the proposed algorithm with some existing clustering algorithms. What is clear in this table is that proposed algorithm is able to provide clustering similar to clustering of an expert (When amount of Mojo, edgeMojo is lower, it represents more similarity between clustering produced algorithm with the one produced by an expert, while the larger F_m indicates more similarity).

In Table 5, the proposed algorithm is compared with known evolutionary algorithms on twelve benchmarks in terms of TurboMQ and the average value in twenty runs. In all these cases, it's obvious that the proposed algorithm was able to separate the clusters equal or better than Bunch and DAGC. The results of the EDA are equal to Bunch in seven and two cases and better than Bunch in five and nine cases in terms of TurboMQ and average, respectively. These results are also equal to DAGC in two cases, better than DAGC in ten cases in terms of TurboMQ and better than DAGC in terms of average in all cases.

Table 4. Comparisons of proposed algorithm with two well-known GA

	Mojo	Edge Mojo	F_m
Bunch	**5**	**7.47**	**0.57**
DAGC	**7**	**10.33**	**0.48**
HC+Bunch	**9**	**11.14**	**0.25**
NAHC	**5**	**13.14**	**0.53**
SAHC	**5**	**10.81**	**0.55**
EDA	**5**	**7.47**	**0.57**

Table 5. Comparisons of proposed algorithm with two well-known GA

	BUNCH			DAGC			EDA		
Software systems	**# of clusters**	**TurboMQ**	**Average**	**# of clusters**	**TurboMQ**	**Average**	**# of clusters**	**TurboMQ**	**Average**
compiler	**4**	**1.506**	**1.506**	**4**	**1.506**	**1.455**	**4**	**1.506**	**1.506**
boxer	**7**	**3.101**	**3.101**	**7**	**3.101**	**2.910**	**7**	**3.101**	**3.091**
mtunis	**5**	**2.314**	**2.286**	**6**	**2.241**	**2.048**	**5**	**2.314**	**2.314**
ispell	**7**	**2.177**	**2.140**	**8**	**1.997**	**1.872**	**6**	**2. 190**	**2.180**
bison	**13**	**2.606**	**2.539**	**15**	**1.763**	**1.679**	**12**	**2.664**	**2.633**
cia	**14**	**2.706**	**2.627**	**19**	**1.833**	**1.691**	**12**	**2.787**	**2.740**
ciald	**8**	**2.851**	**2.834**	**12**	**2.463**	**2.275**	**8**	**2.851**	**2.849**
modulizer	**7**	**2.648**	**2.608**	**9**	**2.112**	**1.915**	**7**	**2.648**	**2.628**
nos	**5**	**1.636**	**1.625**	**5**	**1.606**	**1.508**	**5**	**1.636**	**1.635**
rcs	**9**	**2.175**	**2.115**	**11**	**1.894**	**1.766**	**8**	**2.194**	**2.161**
spdb	**6**	**5.741**	**5.741**	**8**	**5.314**	**5.076**	**6**	**5.741**	**5.741**
star	**10**	**3.809**	**3.673**	**16**	**2.831**	**2.524**	**9**	**3. 832**	**3.766**

In Table 6, the speed of convergence in proposed algorithm and Bunch is compared. We run the algorithms ten times and considered 1000 for number of iterations in each run. In cases that the obtained TurboMQ by our algorithm is equal to Bunch, the advantage of our method is that the speed of convergence to the solution is more and algorithm finds the solution in lower reps; for illustration Figure 5 and Figure 6 show convergence diagram of compiler benchmark for Bunch and EOD respectively. Obviously, EOD is converged in lower number of iterations.

Table 6. Comparisons of proposed algorithm with Bunch in terms of the average of iterations for the convergence to the solution

	compiler	**spdb**	**boxer**	**mtunis**	**ciald**	**modulizer**	**nos**
BUNCH	258	325	270	396	577	505	345
EOD	70	98	114	141	314	329	113

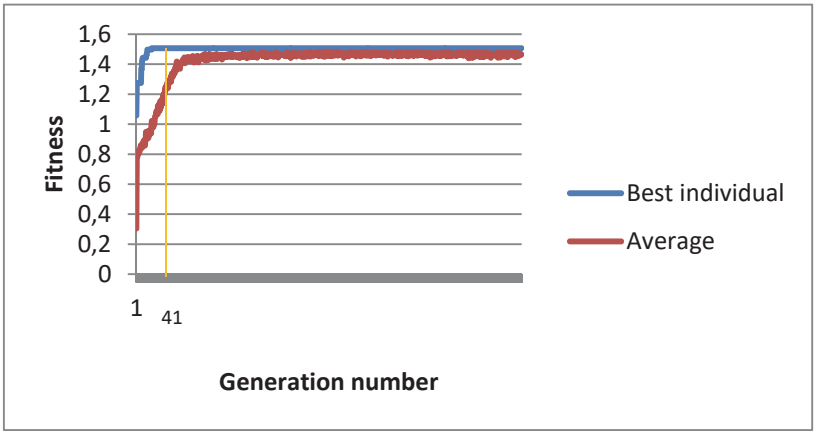

Figure 5. Convergence diagram (compiler) for EDA

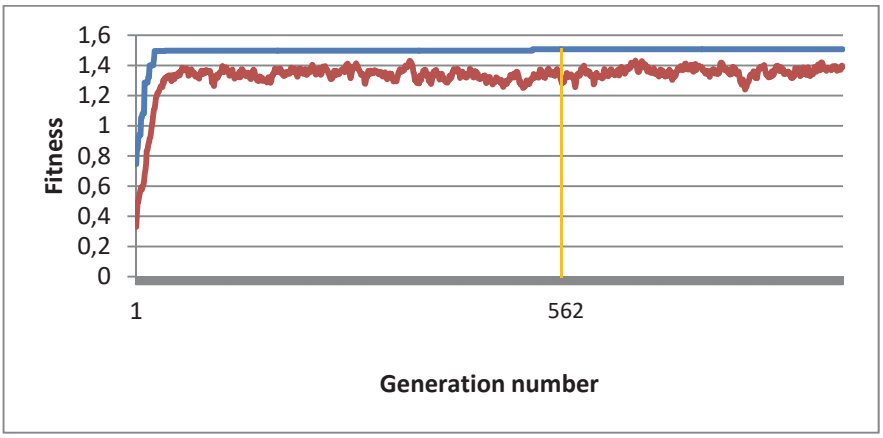

Figure 6. Convergence diagram (compiler) for Bunch

In Table 7, our algorithm is compared with Bunch and DAGC in terms of the standard deviation of the results of 20 runs (the lower Std. Deviation indicates more stability). The results of this table show that stability of proposed algorithm is higher than Bunch and DAGC in most and all cases, respectively. So, we can say our algorithm has higher stability. For example, stability diagram of proposed algorithm for one of the benchmarks (compiler) is presented in Figure 7.

Table 7. Comparisons of proposed algorithm with two well-known GA in terms of Std. Deviation

	Std. Deviation											
Software systems	compiler	boxer	mtunis	ispell	bison	cia	modulizer	nos	rcs	spdb	star	ciald
EDA	0.002	0.018	0	0.009	0.024	0.034	0.038	0.002	0.028	0	0.075	0.003
BUNCH	0	0	0.028	0.023	0.047	0.053	0.039	0.015	0.030	0	0.080	0.010
DAGC	0.042	0.102	0.096	0.073	0.076	0.064	0.104	0.058	0.109	0.187	0.149	0.106

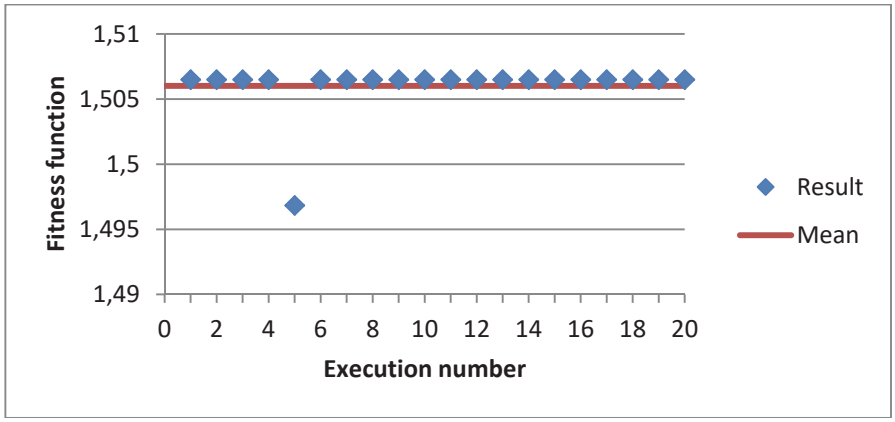

Figure 7. Stability diagram

5 Conclusion

In this paper, we have used estimation of distribution algorithm for software clustering problem. A probability model was presented using features of clustering problem. Results of initial tests showed that the proposed algorithm is very promising. For future work we are planning to do the following work:

1. In future work, we will try to test our algorithm on many software systems.
2. One of the important issues related to Bunch encoded is largeness of search space due to presence of some repetitive answers, i.e., although some generated encodes have apparently different representations, but in reality, they represent the same clustering. For example, though two chromosomes S_1= 2 2 4 4 1 and S_2=1 1 5 5 3 have different appearances but they are actually representative of the same clustering. Because, in both, there are three clusters so that nodes of C_1 and C_2 are in same cluster, nodes of C_3 and C_4 are in same cluster and node node C_5, located in distinct cluster. Search space in Bunch algorithm is n^n; this large search space decelerates speed of this algorithm to find appropriate structure. The state space of n^n is the worst state for a problem and search in this space is impossible in a rational time. Such state space would cause doubt in finding a

good structure for software by Bunch. We believe that we can reduce it using limited number of clusters. It is well known fact that the number of clusters are much less than the number of classes in a program. Considering the number of classes as *n*, if we limit the number of clusters to maximum n/3 of classes (it is usually much less than n/3.); therefore, the state space of Bunch can be reduced to $(\frac{n}{3})^n$. The upper bound of this state space is O(n!). This significant reduction may have a significant effect on improvement of the quality of achieved structure.

References

1. Zhang Q., Qiu D., Tian Q., Sun L., 2010, *Object-oriented software architecture recovery using a new hybrid clustering algorithm*. Fuzzy Systems and Knowledge Discovery (FSKD), Seventh International Conference on. Vol. 6. IEEE, 2010.
2. Bittencourt R. A., and Dalton D. G., 2009, *Comparison of graph clustering algorithms for recovering software architecture module views*. Software Maintenance and Reengineering, CSMR'09. 13th European Conference on. IEEE, 2009.
3. Poshyvanyk D., Marcus A., Ferenc R., *Using information retrieval based coupling measures for impact analysis*. Empirical software engineering 14.1: 5-32, 2009
4. Izadkhah H., Elgedawy I., and Isazadeh A., *E-CDGM: An Evolutionary Call-Dependency Graph Modularization Approach for Software Systems*. Cybernetics and Information Technologies 16.3: 70-90, 2016.
5. Larranaga P., and Lozano J., *Estimation of distribution algorithms: A new tool for evolutionary computation*. Vol. 2. Springer Science & Business Media, 2002.
6. Parsa S., and Bushehrian O., *A new encoding scheme and a framework to investigate genetic clustering algorithms*. Journal of Research and Practice in Information Technology 37.1: 127, 2005.
7. Lindig C., and Snelting G., *Assessing Modular Structure of Legacy Code based on Mathematical Concept Analysis*. Proceedings of the International Conference on Software Engineering, 1997.
8. Lindig C., and Snelting G., *Assessing Modular Structure of Legacy Code based on Mathematical Concept Analysis*. Proceedings of the International Conference on Software Engineering, 1997.
9. Cui J. F., and Chae H. S., *Applying Agglomerative Hierarchical Clustering Algorithms to Component Identification for Legacy Systems*. Information and Software Technology, Volume 53, Issue 6, Pages 601-614, 2011.

10. Maqbool O., and Babri H., *Hierarchical clustering for software architecture recovery*. IEEE Transactions on Software Engineering 33.11: 759-780, 2007.
11. Andritsos P., and Tzerpos V., *Information-theoretic software clustering*. IEEE Transactions on Software Engineering 31.2: 150-165, 2005.
12. Mitchell Brian S., *A heuristic search approach to solving the software clustering problem*. Diss. Drexel University, 2002.
13. Mitchell, Brian S., and Mancoridis S., *On the automatic modularization of software systems using the bunch tool*. IEEE Transactions on Software Engineering 32.3: 193-208, 2006.
14. Mahdavi K., Harman M., and Hierons R. M., *A multiple hill climbing approach to software module clustering*. *Software Maintenance, ICSM 2003. Proceedings. International Conference on*. IEEE, 2003.
15. Praditwong K., Harman M., and Yao X., *Software module clustering as a multi-objective search problem*. *IEEE Transactions on Software Engineering* 37.2: 264-282, 2011.
16. Tzerpos V., and Holt R. C., MoJo: *A distance metric for software clusterings*. Reverse Engineering, Proceedings. Sixth Working Conference on. IEEE, 1999.
17. Wen Z., and Tzerpos V., An *effectiveness measure for software clustering algorithms*. Program Comprehension, Proceedings. 12th IEEE International Workshop on. IEEE, 2004.