ESBASCA: A Novel Software Clustering Approach

Bilal Khan, Shaleeza Sohail, and M. Younus Javed

Department of Computer Engineering, College of E & ME, National University of Sciences and Technology (NUST), Pakistan
shaleeza@ceme.edu.pk

Abstract

Maintenance is one of the key phases of software development life cycle, for long term effective use of any software. It can become very lengthy and costly for large software systems, especially when subsystem boundaries are not clearly defined. System evolution, lack of up to date documentation and high turn over rate of software professionals (leading to non availability of original designers of the software systems) can complicate the system structure many folds by making the subsystem boundaries ambiguous. Automated software module clustering helps software professionals to recover high-level structure of the system by decomposing the system into smaller manageable subsystems, containing interdependent modules. We treat software clustering as an optimization problem and propose a technique to get near optimal decompositions of relatively independent subsystems, containing interdependent modules. We propose the use of self adaptive Evolution Strategies to search a large solution space consisting of modules and their relationships. We compare our proposed approach with a widely used genetic algorithm based approach on a number of test systems. Our proposed approach shows considerable improvement in terms of quality and effectiveness and consistency of the solutions for all tests cases.

Keywords: ESBASCA, Software Clustering, ES Algorithm

1. Introduction

The last and ever going phase of system development life cycle is maintenance which plays a vital role in extending the life of any system. Maintenance of medium and large sized software systems can be a difficult task especially in the absence of the original designer and up to date documentation. The structure of software system suffers deterioration along the course of maintenance activity [1]. Hence, it becomes important to re-identify the subsystem boundaries in order to facilitate effective maintenance in future.

Software clustering is aimed at categorizing large systems into smaller manageable subsystems containing modules of similar features. The clustering facilitates better comprehension of the system. The decomposition is based on the relationships among the modules. These relationships are usually represented in the form of module dependency graph where modules are represented as nodes and the relationships as the edges between these nodes. The software clustering problem can be seen as partitioning of this graph into clusters containing interdependent modules. However, the number of possible partitions can be very large even for a small number of nodes [2]. Moreover, the fact, that even small differences between two partitions can generate quite different results, enhances the problem domain.
Hence finding the best clustering for a given set of modules has been proved to be a NP-hard problem [3].

In this paper, we propose Evolution Strategy Based Automated Software Clustering Approach (ESBASCA) that treats the clustering problem as an optimization problem with the goal of finding near optimal partitions. We define the criteria for near optimal partitioning in Section 4. Our approach searches the large solution space that consists of all the possible partitions and after a number of iterations finds the near optimal partitioning for the given system. The inherent quality of Evolution Strategies (ES) [4, 5] is the self adaptability which makes sure that as the number of iterations increase, ESBASCA always gets same or better result than before and never loose local optimal value during the execution. To show the effectiveness of our approach we have compared it with Genetic Algorithm (GA) [6, 7] based clustering approach and results show considerable improvement by ESBASCA. The improvement is due to two main factors that GA suffers from when compared to ES:

- Reproduction can eliminate good solutions in GA while good solutions always survive into the next generation in ES.
- In GA the strategy parameters (e.g., mutation strength) remain constant so it may remain stuck at local optima. Self adaptive ES on the other hand promises better results because self adaptation helps faster convergence and fine tuning of the search along the fitness landscape.

For any optimization algorithms we cannot achieve exactly similar result in different executions of the algorithm on same data. However, the results should be close to each other without any major variations. Consistency in results is one of the most desirable properties of any working algorithm. Hence, an important measure while evaluating the performance of any clustering algorithm is the consistency of the results produced by it. Keeping this in mind we conducted a comparative study on the consistency in results produced by ESBASCA and GA based approaches. We found that ESBASCA gives far more consistent results as compared to the GA based approach.

The structure of the rest of the paper is as follows: Section 2 reviews the related work in this field of research. In Section 3 we give a brief introduction of ES and related concepts. Section 4 outlines our application of the self adaptive ES for software clustering problem. Section 5 discusses the results of our technique on four average size industrial systems and compares with the results collected using GA based approach. Section 6 concludes the paper and presents future research directions.

2. Literature Survey

In this section, we discuss some of the approaches used for software clustering related to our work. We also present some work in Software Engineering literature that highlights the importance of consistency.

Belady and Evangelisti [8] identified that clustering the modules of software system can reduce its complexity for the programmer. They presented an approach that extracted the information form the documentation of the software system rather than its source code. But their work was restricted to particular systems. They also defined a measure for system's complexity but did not validate it.
Muller et al. [9] and Schwanke [10] presented software clustering approaches relying on the information from source code to extract system structure. These bottom-up approaches are semi automatic and require significant user interaction. Muller et al. [9] presented the heuristics based on the strength of interfaces and defined the principles of using small and few interfaces. A main contribution of Schwanke [10] was that he introduced the notion of shared neighbors that referred to resources providing similar functionality e.g. methods of a drawing library. Another significant feature of his work was maverick analysis that identifies components placed in wrong subsystems and assigns them to correct ones.

Hutchens and Basili [11] adopted a completely automatic approach based on data bindings, where data binding is an interaction between procedures involving the variables in the static scope of the related procedures. This technique works on a lower granularity level as it clusters related procedures into modules. Whereas our work is targeted at clustering related modules into subsystems.

Anquetil and Lethbridge [12] proposed an approach based on resource names instead of conventional reliance on relationships between components. They presented some good results but their technique highly relies on how much consistent the developer has been while naming the resources, which is not the measure one can always depend on.

Tzerpose and Holt [13] presented a comprehension driven approach. Their technique discovers clusters that follow the patterns extracted from the manual decompositions provided by the original designers of the systems. The proposed approach produces mixed results as the performance mainly depends on the competence of the original designer.

The work on software clustering more related to ours is by Doval et al. [14]. Their technique treats software clustering as an optimization problem and uses Genetic Algorithms to explore the solution space. As discussed in the previous section GA have few inherent problems when compared with ES, hence our approach works better in this regard.

We were hardly able to find any research work in Software Clustering literature that has formally compared the consistency of the results generated by different software clustering approaches. In this manner, our effort is one of the very first ones in this domain that compares consistency in results, an important and desirable property of any algorithm. To show the importance of our work we present some earlier work from Software Engineering field that highlights the significance of consistency in results. In addition, we discuss some of the approaches used for software clustering related to our proposed approach ESBASCA.

Consistency is an important step towards stability of the clustering algorithm. Tzerpos and Holt [15] defined a stable clustering algorithm as one whose output does not change significantly when its input software system is slightly modified. From this definition it is clear that an inconsistent algorithm, that is showing large variations even for the same input in different runs, can not be stable; hence, consistency is important for stability. The results presented in this paper have shown the consistency of our approach, ESBASCA.

Olson and Wolfson [16] explained the importance of consistency in Information Architecture. They presented an approach to indexing that selects names and topic in manner that gives consistent and effective retrieval.

Monge, Marco and Cervigón [17] discuss the significance of consistency in context of Software Measurement Methods. They have defined a homogeneous statistic that indicates how consistent a software measurement method is. They also provided a statistical analysis that compares given measurement methods and tells which one is more consistent.
3. Evolution Strategies

Evolution Strategies is a specialization of evolution algorithms. They are nature inspired optimization methods that apply selection and genetic operators to a population of individuals to evolve better solutions in an iterative manner. Each individual in the search space represents a potential solution. Each iteration is called a generation and in each generation a new population is created using the fittest individuals in the preceding generation. The operators, the idea of self adaptation and the generic ES Algorithm is presented in the following subsections.

3.1. Objective Function

The quality of solution is calculated using a problem dependent objective function that defines the fitness value (quality) of each member of the population. The function is designed in such a manner that an individual with higher fitness represents a better solution than an individual with a lower fitness.

3.2. Operators

ES typically uses selection, mutation and recombination operators to guide the search. Following sections have brief description of each:

3.2.1. Selection: ES uses truncation selection where only individuals with promising properties, i.e. high fitness values (objective function values), get a chance of reproduction. That is, the new generation is obtained by a deterministic process guaranteeing that only the best individuals from the selection pool of previous generation are transferred to the new generation [5].

3.2.2. Mutation: The mutation operator is the primary variation operator. Mutation must respect the reachability condition i.e. each (finite distant) point of the search space should be reachable in a finite number of steps. Moreover, it should be scalable in order to allow for self-adaptation [4]. This operator helps in ensuring that the search is not stuck at local optima. For that, it adds variations which facilitate exploring new possibilities in the search space without destroying the current high fitness values.

3.2.3. Recombination: In contrast to mutation that applies to individuals, the recombination operator shares information from multiple (standard is two) parents to produce a new offspring. This is unlike the cross-over operator in genetic algorithms where two offspring are produced.

3.2.4. Self Adaptation: By definition, self adaptation means that the control of strategy parameters of the computation is delegated to the computation itself. In ES, mutation strength is an important parameter that controls the spread of population on the fitness landscape. This parameter needs to be varied during the search process to fine tune the search e.g. larger mutation strength suitable at the start may not be suitable at later stages, as it may skip the optima. Self adaptation comes handy here and the adaptation of mutation strength is embedded in the algorithm by making it a part of an individual's gene [18]. So mutation strength itself too goes through the recombination and mutation process.
3.3. The ES Algorithm

ES applies the above defined operators to a population in an iterative process. The generic algorithm is outlined here:

1. Take an initial population of \( x \) individuals.
2. Generate \( y \) offspring, where each offspring is generated in the following manner:
   a. Select \( z \) parents from \( x \) (\( z \) is a subset of \( x \)).
   b. Recombine the \( z \) selected parents to form a new individual \( i \).
   c. Mutate the strategy parameter (adaptation).
   d. Mutate the individual \( i \) using the mutated strategy parameter.
3. Select new parent population consisting of \( x \) best individuals (based on objective function) from the pool of \( x \) and \( y \).
4. Go to 2, until termination condition occurs.

4. ES for Software Clustering

In this section we present the instantiation of ES Algorithm for the software clustering problem.

4.1. Variable Selection

In software clustering problem we have three types of variables which affect resolution of problem.

First, the entities involved, which in this case are modules of the system. We represent these modules with indices from 0 to \( n-1 \).

Second variable is the set of relationships among these modules. We used a third party fact extractor [19] that provided us with these relationships among modules and their weights. The relationships taken into account are those based on inheritance, containment, genericity and member access.

Third variable is the subsystems (clusters) which comprise of these modules. These subsystems are represented by 0 based indices. Therefore a system can have minimum one cluster and maximum \( n \) (equal to total number of modules in the system) subsystems.

4.2. Population Representation

Every individual solution of the software clustering problem is represented by an encoded string of integers. This encodes string is generated by assigning a cluster number to each entity, e.g. consider 5 modules to be grouped in 3 clusters. These can be encoded as \([1 0 2 2 1]\) representing cluster 0 contains module 1, cluster 1 contains modules 0 and 4, and cluster 2 contains modules 2 and 3.

4.3. Fitness Function
Fitness function is derived using the variables involved in the system. We are proposing the use of operators and algorithm presented in Section 3 on an objective function based on software engineering concepts of coupling and cohesion.

Cohesion is a measure of how strongly-related and focused the various responsibilities of a software subsystem are. Subsystems with high cohesion are considered preferable because high cohesion is associated with several desirable features of software including robustness, reliability, reusability, and understandability. However, low cohesion is associated with undesirable qualities such as being difficult to maintain, difficult to test, difficult to reuse, and even difficult to understand.

Coupling is the degree to which each subsystem relies on the other subsystems. Coupling is usually contrasted with cohesion. Low coupling often correlates with high cohesion, and vice versa.

It is believed that subsystems exhibiting high cohesion and low coupling form well designed systems. Hence, the resulting decompositions should have more intra-cluster relationships and less number of inter-cluster relationships. To achieve this property we use the objective function "Turbo MQ", used and defined by Mitchell [3].

For each cluster we calculate two quantities: intra-connectivity and inter-connectivity. \( \mu_i \) which refers to the Intra-connectivity of a cluster i is the weighted sum of all relationships (provided by the fact extractor) that exist between modules in that cluster i. A higher value of Intra-connectivity corresponds to high cohesion. \( \epsilon_{ij} \), that refers to inter-connectivity is the weighted sum of all relationships (provided by the fact extractor) that exist between modules in two distinct clusters i and j. This quantity can have values between 0 (when there are no subsystem level relations between subsystem i and subsystem j) and 1 (when all modules in subsystem i are related to all modules in subsystem j and vice-versa). A low value for inter-connectivity means low coupling.

Using these two quantities, a cluster factor \( CF_i \) is calculated for each cluster i and total fitness of the system is given by the sum of \( CF \) for all clusters. The cluster factor is calculated as:

\[
CF_i = \begin{cases} 
0 & \text{if } \mu_i = 0 \\
\frac{2\mu_i}{2\mu_i + \sum_{j=1}^n \epsilon_{ij} + \epsilon_{ji}} & \text{otherwise}
\end{cases}
\]

Total Fitness is given by:

\[ TurboMQ = \sum_{i=1}^n CF_i \]

Table 1. Test System Description

<table>
<thead>
<tr>
<th>Test System ID</th>
<th>Lines of Code</th>
<th>Header Files</th>
<th>Source Files</th>
<th>No. of Modules</th>
<th>No. of Relationships</th>
</tr>
</thead>
<tbody>
<tr>
<td>TS-1</td>
<td>45582</td>
<td>53</td>
<td>39</td>
<td>36</td>
<td>817</td>
</tr>
<tr>
<td>TS-2</td>
<td>16360</td>
<td>31</td>
<td>27</td>
<td>41</td>
<td>473</td>
</tr>
<tr>
<td>TS-3</td>
<td>51768</td>
<td>27</td>
<td>27</td>
<td>60</td>
<td>4973</td>
</tr>
<tr>
<td>TS-4</td>
<td>82877</td>
<td>74</td>
<td>68</td>
<td>80</td>
<td>4886</td>
</tr>
</tbody>
</table>
5. Results and Discussion

For the comparison of our approach with the widely used GA based approach, we have implemented both the GA and ES based software module clustering algorithms in C++. We wanted to use Bunch tool [3] for GA based approach but neither we could get hold of Acacia (from AT&T Research Labs), the tool that is needed to generate input for Bunch, nor we were able to find any helpful documentation regarding the input format to generate the input for Bunch by ourselves.

We used four medium sized industrial software systems in our study. These are object oriented systems implemented in C++. Implementation of these systems also involves software libraries MFC, ATL and STL. Each test system is introduced briefly in the following paragraphs followed by a statistical summary in Table 1. Details of the test systems and their module relationships can be found in [19]. The relationships taken into account are those based on inheritance, containment, genericity and member access. Doval et al. [14] in their paper on GA based software clustering used a single test system of 20 modules whereas we are using four test systems of different sizes ranging from 36 to 80 modules.

<table>
<thead>
<tr>
<th>Table 2. Common Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>-----------------------------</td>
</tr>
<tr>
<td>Initial Population Size</td>
</tr>
<tr>
<td>No. of Clusters</td>
</tr>
<tr>
<td>Termination Condition</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 3. GA Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>-------------------------</td>
</tr>
<tr>
<td>Selection Method</td>
</tr>
<tr>
<td>Cross-over Probability</td>
</tr>
<tr>
<td>Mutation Probability</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 4. ES Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>-------------------------</td>
</tr>
<tr>
<td>Mutation Type</td>
</tr>
<tr>
<td>Exponent for the Geometric Distribution</td>
</tr>
<tr>
<td>Recombination Operator Type</td>
</tr>
</tbody>
</table>

We found a few strange things about the stats presented in Table 1, e.g. 31 files for 41 modules and 27 files for 69 modules and so on. Upon collaborations with the fact extractor provider and the test system owners it was found that this was because of the coding standards adopted by the coders. Multiple modules had been defined in same files.

**TS-1** is a component of a large software system. It provides conversion support from intermediate data structures to a well known document format.

**TS-2** software system solves economic power dispatch problem using conventional and evolutionary computing techniques. It uses MFC document view architecture and implements conventional and genetic algorithms.
TS-3 is a component of a large software system. It provides conversion support from intermediate data structures to a well known printer language.

TS-4 is a software system for design document layout and composition. It provides visual support to define document layout and complete saving and loading mechanism for designed applications.

We performed our testing on Win-XP platform on a machine with 3GHz Intel Pentium IV processor and 2GB RAM. The specific settings for both algorithms are shown in the Tables 2 to 4. Table 2 shows the parameters common to both ES and GA.

<table>
<thead>
<tr>
<th>Test Systems</th>
<th>GA</th>
<th>ES</th>
<th>Percent Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>TS-1</td>
<td>4.32</td>
<td>5.30</td>
<td>23</td>
</tr>
<tr>
<td>TS-2</td>
<td>2.30</td>
<td>2.82</td>
<td>22.5</td>
</tr>
<tr>
<td>TS-3</td>
<td>1.72</td>
<td>2.51</td>
<td>46</td>
</tr>
<tr>
<td>TS-4</td>
<td>1.56</td>
<td>2.35</td>
<td>50</td>
</tr>
</tbody>
</table>

Here, we find it important to discuss the common features i.e. initial population size, the number of clusters and the termination criteria.

The larger the initial population size the better is the chance of finding a near optimal solution. But due to computation intensive nature of these approaches we have to make trade-off between the initial population size and execution performance. So for our test systems we empirically found out 300 to be a good option.

It is not feasible to check all decompositions containing 1 to n clusters where n is the number of modules in the test system. So we adopted a strategy based on the checking the range of ±2 number of clusters proposed by the benchmark decompositions provided by the designers of the test system. So we have five decompositions in all and we select the decomposition with the highest fitness as the final solution.

Another important decision is to chalk out an efficient termination criteria where again a trade-off has to be made between a good solution and execution performance. This also depends on the number of modules in the system and their relationships. We empirically found out that for the test systems used in this study, 3000 iterations is a good criterion as both ESBASCA and GA based approach converged within this limit. Rather ESBASCA converged well before this limit but we wanted to match our approach to the best possible results of GA based approach so we adopted this limit that favors GA based approach. The second criterion is simply to stop the process when no improvement has been made for a long time.

It should be noted that all these parameters that guide our search can be changed by the user of our application. Table 2 shows the values that we empirically found after experimentation with the test systems under study.

Parameters specific to GA used for our tests are presented in Table 3. The available options and details for these GA specific parameters are in [6, 7]. Parameters specific to ES used in our tests are presented in Table 4. The options and details of these parameters are in [4, 5].

We have compared the fitness value of the resulting decomposition of each test system by both ESBASCA and GA based approach. The collected results are also compared with reference decompositions provided by the original designers of the systems. The following sections present and discuss each category of these results.

5.1. Quality
Fitness value gives us the idea of how good is the decomposition according to a predefined objective function. Using the cohesion and coupling criteria given in Section 4, the Fitness values of the best decomposition found by both GA based approach and ESBASCA for each test system averaged over ten runs is presented in Figure 1. It can be seen that ESBASCA yields much better results for all test systems; the improvement is in the range of 20-50%. The improvement in fitness value by ESBASCA as compared to GA based approach calculated for each test system is given in Table 5.

This improvement in quality of results through ESBASCA is due to the absence of two inherent features of the GA based approach as mentioned in Section 1.

Reproduction can eliminate good solutions in GA, while good solutions always survive into the next generation in ES. The design of GA is such that parents do not survive in to the next generation and are replaced by the offspring, irrelevant of the fitness values. The result of such design is that the fitness value may suffer degradation if the offspring resulting from the cross over operator have less fitness than the parents. Hence, not only the convergence speed is affected but the solution may remain get stuck at local optima, if such situation continues to prevail through generations. A technique called Elitism [20] has been proposed that tries to minimize this loss over a number of generations.

This is not the case in ES where both parents and offspring compete to survive into the next generation and only the fittest survive; details in Section 3. This means that fitness value can either remain unchanged or improve in ES.

To show this quality of ESBASCA, we have monitored and recorded the fitness values of each test system over 500 generations for both ESBASCA and GA based approach. From the results it is clear that the fitness value either increases or remains constant over the generations in case of ESBASCA. However, it may suffer degradation in case of GA based approach. Due to space limitations we have presented the results of only one test system in graphical form in Figure 2 and Figure 3. For other test systems we have just shown the generations over which the fitness values decreased for GA, in tabular form in Figure 4.

Self adaptation of strategy parameters is the second feature that resulted in improved results for ESBASCA. GA may remain stuck at local optima due to the fixed mutation rate throughout the evolution. Self adaptive ES, on the other hand, adapts the mutation rate
along the course of evolution that helps in fine tuning the search.

Figure 2. Fitness Values: Generation 1-250

Figure 3. Fitness Values: Generation 251-500

For this, mutation rate is also evolved by applying the mutation operator in the same way as it is applied to the individual solutions. The evolution process keeps monitoring whether or not the change of mutation rate was advantageous according to its impact on the fitness of the individual solutions, and based on this information the mutation strength is modified.
5.2. Effectiveness

Similarity Measure gives us the idea of how good (effective) the resultant decomposition is, by comparing the decomposition produced by the clustering algorithm against the benchmark/expert decomposition. For obtaining the expert decompositions we approached the original designers of the test systems used in our study. Based on their knowledge of the system, source code, class listings and partial documentation of their corresponding systems, the designers provided us with the expert decompositions.
We have used the Precision and Recall [21, 12] similarity measure. Precision and Recall checks the correctness of our results on the basis of inter and intra cluster relations. Two entities in the same cluster are termed as Intra pair while two entities in different clusters are termed as Inter pair. Precision gives the percentage of intra pairs proposed by the clustering algorithm which are also intra in the expert decomposition. Recall gives the percentage of intra pairs in the expert decomposition which were found by the clustering algorithm.

Precision and Recall measures give us the idea that how close a decomposition is to that of the original designer. The higher the value of these percentages, the better is the decomposition produced by the clustering algorithm. Figure 5 and Figure 6 compare these resulting precision and recall percentages of the decompositions produced by GA based approach and ESBASCA averaged over ten runs for each test system. Again we can see that ESBASCA significantly outperforms GA based approach as it shows better precision and recall for all test systems. The percentage improvement in the precision and recall values by our approach as compared to GA based approach for each test system is provided in Table 6.
Figure 8. System Decomposition by GA based Approach

As it is customary to present the decomposition visually, we also present the visual comparison of the decompositions achieved by ESBASCA and GA based approaches with the actual decomposition of the system by original designer, in Figures 7 to 9 for TS-3. The figures were taken using DOT [22] and instead of module names, module numbers are shown in the figures. Representative data relating to only one test system is shown here due to space limitations. It can be seen from the figures that both ESBASCA and GA based approaches produced four clusters in comparison to three clusters in the original decomposition.

Figure 9. System Decomposition by ESBASCA
Table 6. Improvement in Effectiveness through ESBASCA

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th></th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GA</td>
<td>ES</td>
<td>Percent Improvement</td>
<td>GA</td>
<td>ES</td>
</tr>
<tr>
<td>TS-1</td>
<td>28.53</td>
<td>39.4</td>
<td>38</td>
<td>24.406</td>
<td>33.334</td>
</tr>
<tr>
<td>TS-2</td>
<td>26.17</td>
<td>32.67</td>
<td>25</td>
<td>26.96</td>
<td>42.646</td>
</tr>
<tr>
<td>TS-3</td>
<td>20.44</td>
<td>34.16</td>
<td>29</td>
<td>34.006</td>
<td>41.116</td>
</tr>
<tr>
<td>TS-4</td>
<td>27.94</td>
<td>35.87</td>
<td>39</td>
<td>42.208</td>
<td>57.136</td>
</tr>
</tbody>
</table>

ESBASCA based decomposition reduces the inter cluster edges to a small number, hence, achieving low coupling and high cohesion which is a desirable quality.

The results were then shown to the original designers of the systems for validation. The designers of two test systems could spare time for this purpose. For this validation, architectures extracted through both techniques were given to different coders of same caliber who previously had no knowledge about these test systems. The coders were then asked to fix a problem in the code based on their understanding of the architecture. The coders acknowledged that the architecture extracted by ESBASCA was relatively more meaningful and it easily mapped to the source code. Here it must be made clear that IDs were assigned to the architectures and it was not know to the persons validating the results that which architecture was obtained using what technique.

5.3. Consistency

Consistency is a vital criterion to judge an approach. We have compared the consistency of our approach and GA based approach in terms of the fitness values of the resulting decompositions produced by both these approaches. Fitness values of the best decomposition found by both GA based approach and ESBASCA for each test system in ten runs is presented in Figures 10 to 13. From these figures it is clear that
ESBASCA yields much better results than the GA based approach. The improvement in fitness value by ES as compared to GA calculated for each test system over ten runs is given in Table 5.

ESBASCA produces much consistent decompositions as compared to the GA based approach.

Figure 11. Comparison: Fitness Values for Test System 2

Figure 12. Comparison: Fitness Values for Test System 3
To highlight the second point mentioned above, the standard deviation of the fitness values using ESBASCA and GA over 10 runs for each test system was computed. The results are shown in Figure 14. The figure shows that GA based approach have a standard deviation in the range of 0.29 to 0.59 for the four test systems while the results with ESBASCA have a standard deviation in the range of 0.11 to 0.28 for the same test systems. This means that even the maximum deviation in ESBASCA's results is less than the minimum deviation of GA based results. This much less deviation by ESBASCA as compared to the GA based approach clearly indicates that our approach performs consistently without any major variations in results.
We would like to elaborate that the main reason for this difference in the consistency of ESBASCA and GA based approaches can be attributed to the primary operators involved in the two schemes. Mutation is the basic operator that provides genetic variation in ES. This operator helps in ensuring that the search is not stuck at local optima by adding variations in a manner that helps in exploring new possibilities in the search space without destroying the current high fitness values. Each individual has a probability of going through a small change when mutation is applied. For example,

\[ 10001000 \rightarrow 10101000 \]

The main operator in GA is the cross over operator. In contrast to mutation, cross over works with two individuals. Cross over operator combines parents (the individuals selected using the selection operator) to create offspring, in a bid to find individuals that have higher fitness values than either of the parents. In a single point cross over (the commonly used type of cross over) each individual is split at a point:

\[ 1 \leq j \leq L \text{ where } L \text{ is the length of the individual.} \]

By swapping the parts of the parents between \( j+1 \) and \( L \), two new individuals are created:

\[
\begin{align*}
100 & 110 \\
\rightarrow & \\
001 & 001
\end{align*}
\]

Hence, the variation achieved by the cross over operator is higher than that of mutation. While this variation is the main driving force of GA, it brings inconsistency for a non uniform population like the one in software clustering problem.

6. Conclusion and Future Work

Maintaining and understanding large software systems from source code or module dependency graph is a difficult task. Partitioning the graph can help but the number of possible partitions is quite large even for small systems. We have presented our self adaptive Evolution Strategies based approach that explores this large solution space to find an effective decomposition of the system. To study the effectiveness of our proposed approach, we have compared it against GA based approach using industrial systems of different sizes. Our approach yielded encouraging results. More specifically, our proposed approach, ESBASCA, produced better decompositions for all test cases with higher fitness values and higher recall percentages as compared to those produced by the GA based approach. In addition, the standard deviation among the achieved results by ESBASCA is much less than that of the GA based approach, highlighting the consistency in results of our approach. The encouraging consistent results showing the effectiveness of our approach makes it more stable as well.

In future we want to establish the stability of our approach using the stability measure defined in [15]. Also we want to develop a new similarity measure to remove a flaw in EdgeSim [23]. EdgeSim gives same results for two decompositions if all edges in both decompositions are of same type. It is possible that a module moves form one cluster to another cluster in a manner that edge types remain the same. EdgeSim will not point out this difference. Our similarity measure will incorporate this movement of modules between clusters.

References


