Clustering of Software Systems using New Hybrid Algorithms

Ali Safari Mamaghani
Computer Engineering Department
Islamic Azad University, Bonab Branch
Bonab, Iran
Safari_m_61@yahoo.com

Mohammad Reza Meybodi
Computer Engineering and Information Technology Department
AmirKabir University of Technology
Tehran, Iran
mmeybodi@aut.ac.ir

***Abstract*—Software clustering is a method for increasing software system understanding and maintenance. Software designers, first use MDG graph to model the structure of software system. In MDG, system modules (e.g. files ,classes) are represented as nodes and their relationships (e.g. function calls , inheritance relationships) as directed edges that connect the nodes. Once the MDG created, clustering algorithms are applied and create a partitioned MDG. Graph partitioning is a NP-Complete problem, So many algorithms for solving it has been reported in the literatures. In this paper two approximate algorithms have been proposed. The first algorithm is based on object migration learning automata and the second algorithm is a hybrid evolutionary algorithm obtained from combining object migration learning automata and genetic algorithm. The second algorithm by using learning automata and genetic algorithm accelerates the searching process and also prevents the algorithm from getting stuck in local optimal. Another positive point of the proposed algorithm is high stability. The proposed algorithms have been compared with some of the existing algorithms. Results show that the second algorithm has superiority over the existing methods.**

Keywords-Learning Automata, Genetic Algorithm, Software Clustering, Module Dependency Graph.

I. INTRODUCTION

Software supports many of this country's business, government, and social institutions. As the processes of these institutions change, so must the software that supports them. Changing software systems that support complex processes can be quite difficult, as these systems can be large (e.g. thousands or even millions of lines of code) and dynamic. Software Clustering is one of the suitable ways to increase system understanding and maintenance. Software Designers, first use MDG (Module Dependency Graph) to model the structure of software system. In MDG, system modules (e.g. files, classes) are represented as nodes and their relationships (e.g. Function calls, inheritance relationships) as directed edges that connect the nodes. Once the MDG created, clustering algorithms are applied to create a partitioned MDG. The goal of our software modularization process is to partition the components of a system into clusters (subsystems) so that the resultant organization concurrently minimizes inter-connectivity (i.e., connections between the components of two distinct clusters) while maximizing intra-connectivity (i.e., connections between the components of the same cluster). We do this task by treating clustering as an optimization problem where our goal is to maximize an objective function based on a formal characterization of the trade-off between inter- and intra-connectivity.

Creating a meaningful partition of an MDG, however, is difficult, because the number of possible partitions is very large even for a small graph [3]. Also, small differences between two partitions can yield very different qualitative results. Graph partitioning is a NP Problem and using exhaustive search algorithms are not suitable for this problem. So many approximate algorithms have been used. For example, Brian Mitchell used a search strategy, based on traditional hill climbing optimization techniques [1], that quickly discovers an acceptable sub-optimal clustering result. This algorithm starts with a random partition and repeatedly finds better neighboring partitions until no neighboring partition can be found with a higher MQ. MQ objective function will illustrate later. Getting stuck in local optimal is one problem of hill climbing algorithm. The evolutionary genetic algorithms were used to overcome the limitations of hill climbing algorithm. First genetic algorithm for this problem have been used by Doval [2, 3] in Bunch clustering tool [5]. Large search space is one of the problem of this algorithm so many repeated solutions have in search space. In order to overcome to this problem, DAGC genetic algorithm used by Parsa and his colleagues [7]. In clustering large software systems, the number of clusters found in a partition may be large. In this case it makes sense to cluster the clusters, thus creating a hierarchy of subsystems[1].

In this paper, two approximate algorithms based on learning automata for solving software clustering problem have been proposed. The first algorithm is based on learning automata and the second algorithm is a hybrid evolutionary algorithm obtained from combining object migrating learning automata and genetic algorithm. We compared these new algorithms with hill climbing method and Bunch and DAGC Genetic algorithms. Hybrid evolutionary algorithm by using learning automata and genetic algorithm accelerates the searching process and also prevents the algorithm from getting stuck in local optimal. Another positive point of the proposed algorithm is high stability.

The structure of the reminder of this paper is as follows: In Section 2, we introduce Modularization Quality (MQ) measurements. Section 3 describes new algorithm based on object migration learning automata for solving software clustering problem .In section 4, we explain new hybrid

978-0-7695-3836-5/09 $26.00 © 2009 IEEE
DOI 10.1109/CIT.2009.111

IEEE computer society

algorithm. Section 5 and 6 is dedicated to describe experimental results and paper conclusion respectively.

II. MEASURING MODULARIZATION QUALITY

In this section we return to discussing our Modularization Quality (MQ) measurements. MQ is the objective function of our searching process, and is therefore defined as a measurement of the quality or fitness of a particular system modularization. Two known modularization criteria have been introduced.

A. Basic MQ

The Basic MQ measurement [3] is the first objective function we defined. It 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 Basic MQ objective function is designed to produce higher values as the intra-connectivity increases and the inter-connectivity decreases.

Intra-Connectivity (cohesion): Intra-Connectivity (A) measures the degree of connectivity between the components that are grouped in the same cluster. A high degree of intra-connectivity indicates good subsystem partitioning because the modules grouped within a common subsystem share many software-level components. A low degree of intra-connectivity indicates poor subsystem partitioning because the modules assigned to a particular subsystem share few software-level components. We define the intra-connectivity measurement A_i of cluster i consisting of N_i components and μ_i intra-edge dependencies as:

$$A_i = \frac{\mu_i}{N_i^2} \qquad (1)$$

This measurement is a fraction of the maximum number of intra-edge dependencies that can exist for cluster i, which is N_i^2. The value of A_i is bounded between the values of 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., the modules and dependencies within a subsystem form a complete graph).

Inter-Connectivity (coupling): Inter-Connectivity (E) measures the degree of connectivity between two distinct clusters. A high degree of inter-connectivity indicates a poor subsystem partitioning. Having a large number of inter-dependencies complicates software maintenance because changes to a module may affect many other parts of the system due to the subsystem relationships. We define the inter-connectivity Eij between clusters i and j consisting of Ni and Nj components, respectively, with ε_{ij} inter-edge dependencies as:

$$E_{ij} = \begin{cases} 0 & if\ i = j \\ \dfrac{\varepsilon_{ij}}{2N_iN_j} & if\ i \neq j \end{cases} \qquad (2)$$

Inter-connectivity measurement is a fraction of the maximum number of inter-edge dependencies between clusters i and j ($2N_iN_j$). This measurement is bound between the values of 0 and 1. E_{ij} is 0 when there are no module-level dependencies between sub system i and subsystem j; E_{ij} is 1 when each module in subsystem i depends on all of the modules in subsystem j and vice-versa. Now that inter- and intra-connectivity have been described formally, we define the Basic MQ measurement for a MDG partitioned into k clusters, where A_i is the intra connectivity of the i^{th} cluster and E_{ij} is the inter-connectivity between the i^{th} and j^{th} clusters as:

$$BasicMQ = \begin{cases} \dfrac{1}{k}\sum_{i=1}^{k} A_i - \dfrac{1}{\frac{k(k-1)}{2}}\sum_{i,j=1}^{k} E_{i,j} & if\ k > 1 \\ A & if\ \ k = 1 \end{cases} \qquad (3)$$

The Basic MQ measurement demonstrates the trade off between inter-connectivity and intra-connectivity by rewarding the creation of highly cohesive clusters, while penalizing the creation of too many inter-edges. The Basic MQ measurement is bounded between -1 (no cohesion within the subsystems) and 1 (no coupling between the subsystems).

However, this measurement has two significant drawbacks [1]. First, the performance of this measurement is poor (computational complexity is $O(V^3)$) , which limits its usage to small systems (i.e., fewer than 75 modules). The second problem with the Basic MQ measurement is that its design cannot support MDGs that have edge weights.

B. Turbo MQ

The Turbo MQ [4] measurement was designed to overcome the two limitations of Basic MQ. Specifically, Turbo MQ supports MDGs that have edge weights, and is much faster than Basic MQ (computational complexity is O(V)). Formally, the Turbo MQ measurement for an MDG partitioned into k clusters is calculated by summing the Cluster Factor (CF) for each cluster of the partitioned MDG

$$CF_i = \begin{cases} 0 & \mu_i = 0 \\ \dfrac{2\mu_i}{2\mu_i + \sum_{\substack{j=1 \\ i \neq j}}^{k} (\varepsilon_{i,j} + \varepsilon_{j,i})} & otherwise \end{cases} \qquad (4)$$

$$TurboMQ = \sum_{i=1}^{k} CF_i \qquad (5)$$

III. LEARNING AUTOMAT BASED ALGORITHM

For a MDG graph with n nodes, there are n! partitions with permutation based encoding scheme[7]. If we use

learning automata for solving software clustering problem, automata will have n! actions and the large number of actions reduces speed of convergence in automata[9]. for this reason, we can use object migration learning automata for this problem [8]. To solving the problem, we show object migration automata as $< V, \alpha, \phi, \beta, F, G >$.

In this Automaton, $\alpha = \{\alpha_1, \alpha_2, .., \alpha_k\}$ is set of allowed actions of automaton. This automaton have k actions (the number of actions equals with the number of MDG nodes).

$V = \{V_1, V_2, ...V_n\}$ is set of objects. These objects, instead of holding a partition number holds a Node number (value 1=<p<=k). The objects move on different states of automata and create new permutations.

$\phi = \{\phi_1, \phi_2, ..., \phi_{KN}\}$ is set of states and N is memory depth for automata. The set of automata states are divided to k subsets $\{\phi_1, \phi_2, ..., \phi_N\}$, $\{\phi_{N+1}, \phi_{N+2}, ..., \phi_{2N}\}$, ..., $\{\phi_{(k-1)N+1}, \phi_{(k-1)N+2}, ..., \phi_{kN}\}$, so objects are classified in terms of their states. If object u is situated in the set of states $\{\phi_{(j-1)N+1}, \phi_{(j-1)N+2}, ..., \phi_{jN}\}$, node associated to object u will be j. In the set of states of action j, states $\phi_{(j-1)N+1}$ and ϕ_{jN} are referred as internal state and boundary states respectively. The objects lying in $\phi_{(j-1)N+1}$ and ϕ_{jN} are referred as more and less certainty objects.

$B = \{0,1\}$ is the set of inputs of automata. In this set, 1 and 0 stand for failure and success respectively.

$F : \phi \times \beta \rightarrow \phi$ stands for state mapping function. This function produces the next state in terms of current state and the input of automata. This function determines the movement of objects in states of automata. The function F is different for diverse automatons.

$G : \phi \rightarrow \alpha$ is output mapping function. This function decides what action to do in exchange for any automata state. If object is in the set of states $\{\phi_{(j-1)N+1}, \phi_{(j-1)N+2}, ..., \phi_{jN}\}$, action j is selected (therefore, the node associated to object u will be j).

Encoding: The objects lying on automata actions are permutation of graph nodes. m^{th} action from automata represents node number m from graph. It dose not have the number of related cluster in spite of Bunch Genetic Method [5], but The number of another node from graph like p is in it. In fact, arrangement of objects on the actions of automata is defined as same clustering relations. This permutation is mapped with a decoding algorithm to a clustering. If $p \geq m$, so m will be in the new cluster, otherwise m will be in the same cluster in which p was. In order to have a better understanding of this encoding, note figure 1. for example, object number 5 which is bigger than 2 is located in action number 2, , therefore node number 2 will be located in a new cluster. Node number 1 was already located in cluster number 1. Therefore node number 2 will be located in cluster number 2.

Automata includes 6 actions $\{\alpha_1, \alpha_2, ..., \alpha_6\}$ (numbered by nodes of graph) and depth of 5. The set of states {1,6,11,16,21,26} and {5,10,15,20,25,30} are referred as internal and boundary states set of the automata respectively. First, every object is located in the boundary state of related action. While, links of automata are like links of Tsetline automata. Algorithm in figure 2 represents decoding of such a permutation based Encoding

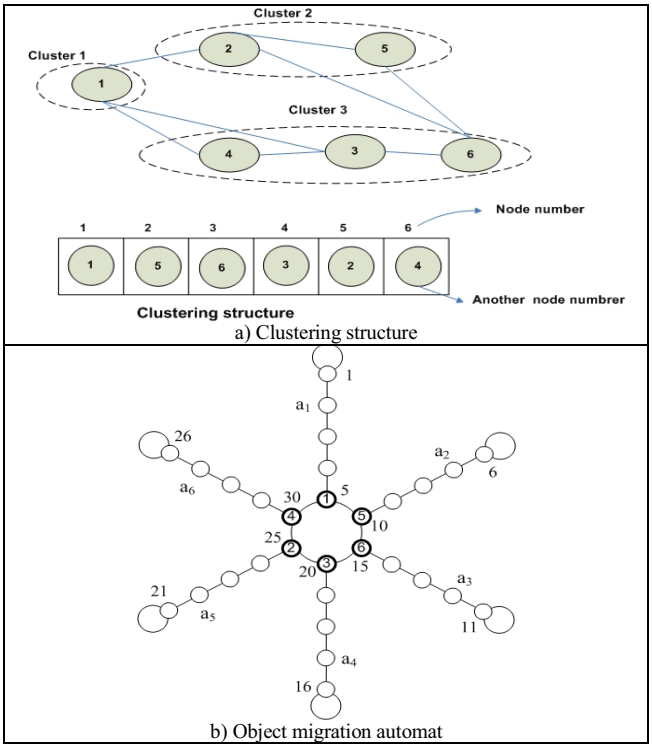

Figure 1. An example of permutation based encoding

In the afore-mentioned encoding, every clustering is created by a permutation from nodes of graph. Since the number of permutation, can be at most equal to n!, Therefore, search space will be reduced in comparison with the space created by coding applied in Bunch genetic algorithm that it is n^n. This reduction of search space leads to the fast convergence of algorithm.

```
DecodeChromosome- Algorithm
    Array C holds a permutation of graph nodes 1 to n
    For node i=1   to  N  do
        if C[i]>=i   Create a new cluster and assign node i to it.
        Else
            Allocate i to the same cluster as node C[i].
End DecodeChromosome.
```

Figure 2. Decoding algorithm of the permutation based encoding

Penalty and reward operator: In the initial solution, an object is chosen randomly, then, it takes penalty or reward. If the difference between cohesion and coupling of chosen node goes beyond threshold (ration of MQ created by

partitioning for the number of clusters of a partitioning), then this node will get reward and moves toward the more internal states of this action. Otherwise, the node gets penalty. Having taken reward or penalty, the state of node in relevant set states change. If a node is located in boundary state of an action, its getting penalty leads to the change of action and creating a new permutation. Penalty and reward operator will be different in terms of kind of learning automata.

Now regarding pervious descriptions, we can show the algorithm applied for solving the problem. Pseudo code of this algorithm is shown in figure 3.

```
Function SoftwareClustering_OMA( MDG): ):partitioned MDG;
n = |V_G|;
randomly create a permutation of MDG nodes and assign them as
objects to the boundary states of corresponding actions;
EvalFitness();
Iteration=1;
Repeat
   For a random node u do //1 ≤ u ≤ n
      If node-fit(u) ≥ MQ/K then reward(u); //k is number of clusters
      Else penalize(u);
   Inc(iteration);
   EvalFitness();
Until(iteration=Max_iterations or all of objects appear in states of
maximumcertainty);
End Function.
```

Figure 3. Object migration automata based algorithm for solving software clustering problem

IV. HYBRID ALGORITHM FOR SOLVING SOFTWARE CLUSTERING THE PROBLEM

Speed for reaching to solution in search process get high if genetic algorithm, learning automata, integeration of concepts of gene, action and depth are combined and algorithm is prevented from being trapped into the local minima [10, 11]. The suggested algorithm in this part is an attempt in this direction. Self-remedy, reproduction and penalty and reward (guidance) are some of the characteristics of hybrid algorithm. For more information, refer simply to the reference [12, 13].

Chromosome and gene: in spite of classic genetic algorithms, in the suggested algorithm, we don't use binary coding for chromosomes. Chromosomes are shown by a learning automata from the kind of object migration so that any gene of chromosomes is associated to one of the automata actions and is located in a specific depth of that action.

Operations: Since every chromosome is represented as a learning automaton in hybrid algorithm, crossover and mutation operators are not similar to classic genetic operators.

A. *Selection operator*

Roulette wheel is used for selecting learning automaton (chromosome) for mutation or crossover.

B. *Crossover operator*

In order to do this operation, we can use any crossover operator which are suitable for working with permutations. New crossover operator is shown in figure 4.

```
Procedure Pcrossover (LA1, LA2)
Begin
   Generate two random numbers r1 and r2 between 1 to n.
   r1 = Random *n; r2 = Random *n;
   r1 = Min(r1, r2); r2 = Max(r1, r2)
   for i = r1 to r2 do
      j = Action of LA1 where
            LA1 .Object(LA1.Action(j)= A2.Object(LA2.Action(i)));
      Swap(LA1.State(LA1.Action(i),LA1.State(LA1.Action(j));
      Swap(LA1.Object(LA1.Action(i)),LA1.Object(LA2.Action(i)));
       j = Action of LA2 where
            LA2 .Object(LA2.Action(j)= LA1.Object(LA1.Action(i)));
       Swap(LA2.State(LA2.Action(i), A2.State(LA2.Action(j));

Swap(LA2.Object(LA1.Action(i)),LA2.Object(LA2.Action(i)));
end for
End Procedure
```

Figure 4. Pseudo code of crossover operator

C. *Mutation operator*

In order to do this operation we can use one of methods which are suitable for working with permutations. A mutaion operator for this encoding is described in figure 5.

```
Procedure Mutation (LA)
Begin
     i = Random *n;
     j = Random *n;
      Swap( LA.Object( LA.Action(i)),LA.Object(LA.Action(j)));
End Procedure
```

Figure 5. Pseudo code of Mutation operator

Now regarding pervious descriptions, we can show the hybrid algorithm applied for solving the problem. Pseudo code of this algorithm is shown in figure 6.

V. EXPERIMENTAL RESULTS

In this section, the results of the new algorithms and the comparison them with other algorithms was shown. In order to examine these algorithms, two groups of module dependency graphs are used. First group includes graphs which are created randomly. Second group includes software graphs which are obtained by code analyzer tools such as CIA [6]. These tools are applied on source codes of software system and produce their MDG graphs. Tests are done on the basis of Basic MQ and Turbo MQ as the objective function. The parameters of the tests are as follows:

(population size:120, number of algorithm iteration:400, crossover rate:0.8 , mutation rate:0.004 $\log_2 n$(n stands for number of nodes of graph), depth of automata: 5).

A. Experimental results on random graphs

In these testes, graphs from 1 to 200 nodes were produced randomly. Because of randomness of algorithms which were used, every test was iterated for 5 times on any algorithm. First, comparison between different algorithms in figures 7 and 8 was done in order to measure of Basic MQ and Turbo MQ respectively. In these figures, HC, Bunch, DAGC, LA and GALA stands for hill-climbing algorithm, Bunch genetic algorithm, DAGC genetic algorithm, learning automata based algorithm and hybrid algorithm respectively. Comparison of the fitness function on the basis of the number of nodes of input graphs we come to the conclusion that hybrid algorithms based Tsetline links are dominant over other algorithms. As you see in the figures 7 and 8, other algorithms are located under curve of hybrid algorithm in most cases. And this shows suitability of hybrid algorithm in comparison with other current algorithms.

The next point is that amounts of curve of hill-climbing algorithm and algorithm based on learning automata is almost the same as hybrid amounts. In spite of this, considerable dominance of hybrid algorithm in comparison with other methods can be observed if we pay attention to figure 9. The results show that average deviation for hybrid algorithm in exchange for random different graphs are lower. Considering random nature of current algorithms, this low amount represents stability of algorithm in different runs. So we can conclude that one of the very positive aspects of this algorithm in comparison with other algorithms is its stability and resistance.

B. Experimental results on software graphs

In this part of tests, the module dependency graph of some software systems was created and then algorithms of software clustering were applied on them. A list of these software systems with descriptions can be shown in Table1.

Applying algorithms and considering the criterion of quality of clustering Basic MQ as the objective function lead to the figures 10, the results show dominance of hybrid algorithm based on Tsetline links in comparison with other algorithms and in most cases, current algorithms are located under hybrid algorithms and this indicates that quality of clustering of this algorithm is high. This shows that hybrid algorithm can be suitable for random graphs and clustering of module dependency graph which were produced from software systems and can be considered as a suitable strategy for solving the problem of software clustering.

TABLE I. DESCRIPTION OF SOFTWARE SYSTEMS IN CONFIGURATION CASE STUDY

SOFTWARE SYSTEM	MODULE IN MDG	EDGES IN MDG	SYSTEM DESCRIPTION
Compiler	*13*	*32*	*A small Compiler*
Boxer	*20*	*29*	*A graph designing system*
Mini-Tunis	*20*	*57*	*A simple operating system*
FSS	*16*	*51*	*A file system service*
Ispell	*24*	*103*	*An open source spell checker*

```
Function SotwareClustering_Hybrid(MDG):partitioned MDG;
  Begin
    n = Size of Population;
    Create the initial population LA1 … LAn;
    EvalFitness();
    Iteration=1;
    Repeat
      NewLA1 = NewLA2 = LA with max Value of MQ;
      for i = 2 to n do
        Select LA1; Select LA2 ;
        if (Random ≤ CrossoverRate) then Crossover ( LA1, LA2 );
        if (Random ≤ MutationRate) then
              Mutation ( LA1 ); Mutation ( LA2 );
        NewLAi+1 = LA1;
        NewLAi+2 = LA2 ;
        i=i+2;
      end for
      for i = 1 to n do
        LAi = NewLAi;
        u = Random *n;
        If (Node_Fit_u(LA_i) ≥ MQ(LA_i)/K) then
              Reward(LAi , u );
            else
              Penalize(LAi , u );
      end for
      EvalFitness();
      Inc(iteration);
    Until(iteration=maxiteration)
End Function
```

Figure 6. Hybrid algorithm for solving the problem of software clustering

VI. CONCLUSIONS

Graphs are powerful tools which are diversely used in software engineering, VLSI design, Databse design, etc. An example of the application of graphs in software engineering is the representation of modules of software system and relations between them in a graph. When the modules of a software system increase, in order to increase the capability of system understanding and maintenance we need to partition the structure of software systems into meaningful systems (clusters). This is called software clustering and that is a NP problem. Different algorithms have been applied to solve this problem that quality of their results are different. In this paper, an algorithm based on object migration learning automata and another algorithm based on combination of object migration learning automata and genetic algorithm were used to the problem. Meanwhile it is proved that simultaneous using of genetic algorithms and learning automata in search process increases the speed of reaching to the solution and also prevents the algorithm from being trapped into local optimal. Apart from high quality of results, one of the positive aspects of this algorithm is its high stability and resistance.

REFERENCES

[1] B. S. Mitchell, A Heuristic Search Approach to Solving the Software Clustering Problem, Ph.D Thesis, Drexel University, Philadelphia, 2002.

[2] D. Doval and S. Mancoridis, "Automatic Clustering of Software Systems using a Genetic Algorithm", In Proc. Of the IEEE Int. Conf. on Software Tools and Engineering Practice(STEP'99), 1999.

[3] S. Mancordis, B. S. Mitchell, Y. Chen and E. R. Gansner, "Bunch: A clustering tool for the recovery and maintenance of software system Structures", in Proc. Of Int. Conf. of Software Maintenance, pp. 50-59, 1999.

[4] B. S. Mitchell, M. Traverso and S. Mancoridis, "An architecture for distributing the computation of software clustering algorithms", In Proc. IEEE/IFIP Conf. on Software Architecture , 2001.

[5] S. Mancordis, B. S. Mitchell, Y. Chen and E. R. Gansner, "Bunch: A clustering tool for the recovery and maintenance of software system Structures", in Proc. Of Int. Conf. of Software Maintenance, pp. 50-59, 1999.

[6] Y. Chen, E. Gansner and E. Koutsofios, "A C++ Data Model Supporting Reachability Analysis and Dead Code Detection", In proc. Of 6th European Software Engineering Conf. and 5th ACM SIGSOFT Symposioum On The Foundations Of Ssoftware Engineering, 1997.

[7] S. Parsa, and O. Bushehrian, "A New Encoding Scheme and a Framwork to investigate Genetic Clustering Algorithms", Journal of Research and Practice in Information Technology, Vol. 37, No. 1, pp. 127-143, 2005.

[8] B. J. Oommen, and D. C .Y. Ma, "Deterministic Learning Automata Solutions to the equipartitioning problem", IEEE Traactions on Computers, Vol. 37 , pp. 2-13, 1998.

[9] K. Narendra and M. A. L. Thathachar, Learning Automata: An Introduction, Prentice Hall, 1989.

[10] H. Beygi and M. R. Meybodi, Solving the Graph Isomorphism Problem Using Learning Automata, In Proc. of CSI int. Conf. pp. 402-415, Tehran, 2000.

[11] M. Rezapour, Solving Graph Isomorphism problem Using Learning Automata, M. Sc. Thesis, Amirkabir University of technology, Tehran, Iran, 2004.

[12] A. Safari Mamaghani, Designing Hybrid Algorithms to Solving Graph Hard Problems, M. Sc. Thesis, Islamic Azad University, Qazvin, 2008.

[13] K. Asghari, A. Safari Mamaghani and M. R. Meybodi,"An Evolutionary Approach for Query Optimization Problem in Database", In Proc. of Int. Joint Conf. on Computers, Information and System Sciences, and Engineering (CISSE2007), University of Bridgeport, England, 2007.

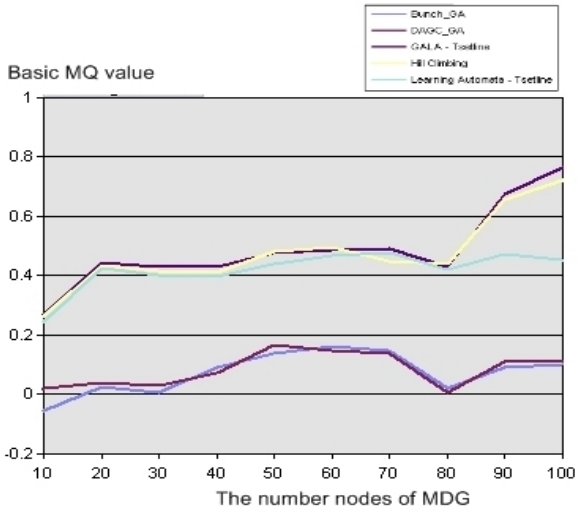

Figure 7. Comparison of average of Basic MQ criteria between different algorithms on random graphs.

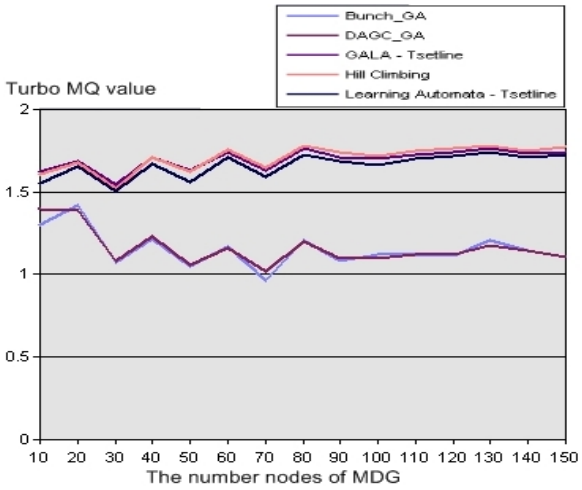

Figure 8. Comparison of average of Turbo MQ criteria between different algorithms on random graphs

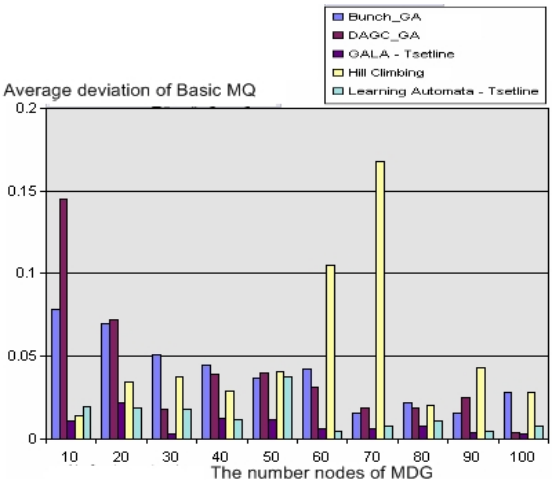

Figure 9. Comparison of stability of algorithms considering Basic MQ criterion as objective function

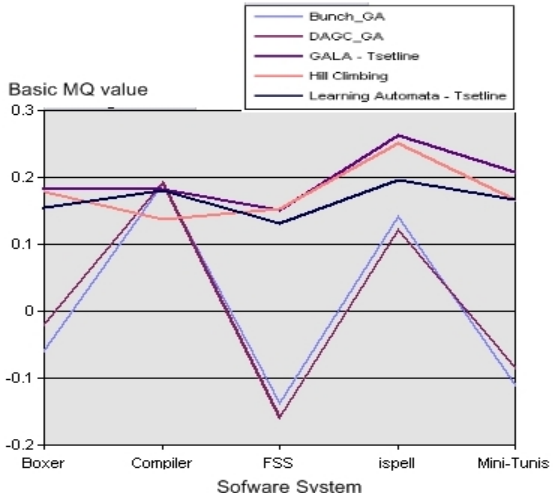

Figure 10. Comparison of average of Basic MQ criteria between different algorithms on software graphs