IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 4, No 3, July 2010
ISSN (Online): 1694-0784
ISSN (Print): 1694-0814

A New Approach for Optimal Clustering of Distributed Program's Call Flow Graph

Yousef Abofathy[1], Bager Zarei[2]

**[1] Department of Computer Engineering, Islamic Azad University, Shabestar Branch
Tabriz, East-Azarbaijan, Iran**

**[2] Department of Computer Engineering, Islamic Azad University, Shabestar Branch
Tabriz, East-Azarbaijan, Iran**

Abstract

Optimal clustering of call flow graph for reaching maximum concurrency in execution of distributable components is one of the NP-Complete problems. Both learning automatas (LAs) and genetic algorithms (GAs) are search tools which are used for solving many NP-Complete problems. In this paper a hybrid algorithm is proposed to optimal clustering of call flow graph and appropriate distributing of programs in network level. The algorithm uses both GAs and LAs simultaneously to search in state space. It has been shown that the speed of reaching to solution increases remarkably using LA and GA simultaneously in search process, and it also prevents algorithm from being trapped in local minimums. Experimental results show the superiority of proposed algorithm over others.

Keywords: *Call Flow Graph, Clustering, Learning Automata, Genetic Algorithm, Concurrency, Distributed Code*.

1. Introduction

Optimization of distributed code with goal of achieving maximum concurrency in execution of distributable components in network level is considered as a new aspect in optimization discussions. Concurrency in execution of distributed code is obtained from remote asynchronous calls. The problem is specifying appropriate calls, with considering amount of yielding concurrency from remote asynchronous calls. In this way, dependency graph between objects are clustered with the base of amount of calls between objects, and each cluster is considered as a component in distributed architecture. Since clustering is a NP-Complete problem, in this paper a hybrid non-deterministic algorithm is proposed for optimal clustering of call flow graph.

The proposed algorithm uses both GAs and LAs simultaneously to search in state space. With the combination of the GA and LA, and incorporation of concepts of gene, chromosome, action and depth, background of the evolution of problem's solution was extracted effectively and is used in the search process. Resisting against the superficial changes of solutions is the most important characteristic of proposed algorithm. In other words, there is a flexible balance between the effectiveness of GA and stability of LA in proposed algorithm. Self-recovery, reproduction, penalty and reward are characteristics of proposed algorithm.

2. Genetic algorithms

GAs act on the basis of evolution in nature, and search for the final solution among a population of potential solutions. In every generation the fittest individuals of that generation are selected and after recombination, produce a new set of children. In this process the fittest individuals will survive more probably in next generations.

At the beginning of algorithm a number of individuals (initial population) are created randomly and the fitness function is evaluated for all of them. If termination condition does not satisfied, parents are selected based on their fitness and are recombined. Then produced children mutate with a fixed probability and their fitness values are calculated. Next new population (generation) is generated by replacing of children with parents, and this process is repeated until the termination condition is satisfied.

3. Learning automata

Learning in LAs is choosing an optimal action from a set of automata's allowable actions. This action is applied on a random environment and the environment gives a random answer to this action from a set of allowable answers. The environment's answer depends statistically on the automata's action. The environment term includes a collection of all outside conditions and their effects on the automata's operation. The interaction between environment and LA is shown in figure 1.

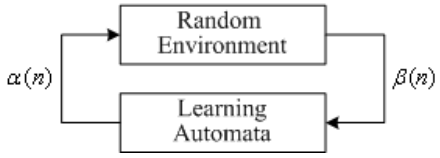


Fig. 1 Interaction between environment and LA

4. Proposed searching algorithm for optimal clustering of call flow graph

In fact, this algorithm determines appropriate clustering in a finite search space. Finally each cluster is considered as a component in distributed architecture.
In section 4-3 an algorithm is represented for determining amount of yielding concurrency from a given clustering. This algorithm is used as a goal function for evaluating of chromosomes in evolutionary process. In fact, in evolutionary process each chromosome is indicator of a clustering and its fitness value specifies amount of yielding concurrency from remote calls in a given clustering.
In the following, the main parameters of proposed algorithm will be explained.

4.1 Gene and chromosome

Unlike classic GAs, in the proposed algorithm binary coding was not used for chromosomes. Each chromosome is shown by a LA of the object migrating type, in a way that each of chromosome's genes is assigned to the one of the automata's actions and it is placed at a certain depth of that action. Therefore in proposed algorithm concepts of, chromosome and automata, gene and action are same.
In this automata $\alpha = \{\alpha_1, \ldots, \alpha_k\}$ is the set of allowable actions for the LA. This automata has k actions (actions number of this automata is equal to the number of distributable components or clusters. Determining of appropriate number of clusters is an optimization problem which has not discussed in this paper). Each action shows a cluster.
$\Phi = \{\Phi_1, \Phi_2, \ldots, \Phi_{kN}\}$ is the set of states and N is the depth of memory for automata. The states set of this automata is partitioned into the k subset $\{\Phi_1, \Phi_2, \ldots, \Phi_N\}$, $\{\Phi_{N+1}, \Phi_{N+2}, \ldots, \Phi_{2N}\}$, …, and $\{\Phi_{(k-1)N+1}, \Phi_{(k-1)N+2}, \ldots, \Phi_{kN}\}$. Call flow graph nodes are classified on the basis of their states. If node n_i from call flow graph is in the states set $\{\Phi_{(j-1)N+1}, \Phi_{(j-1)N+2}, \ldots, \Phi_{jN}\}$, then node n_i will be j^{th} cluster. In the states set of action j, state $\Phi_{(j-1)N+1}$ is called inner (stable) state and state Φ_{jN} is called outer (unstable) state. For example, consider call flow graph of figure 2.

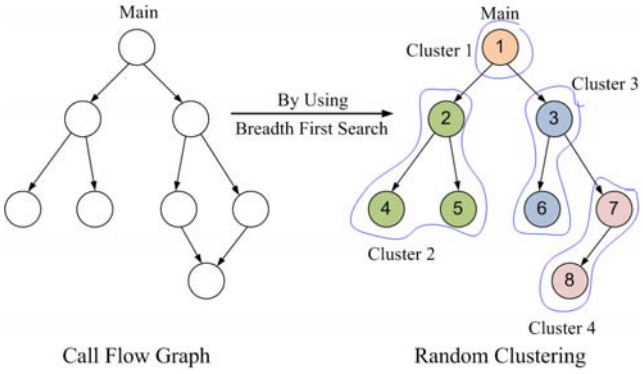
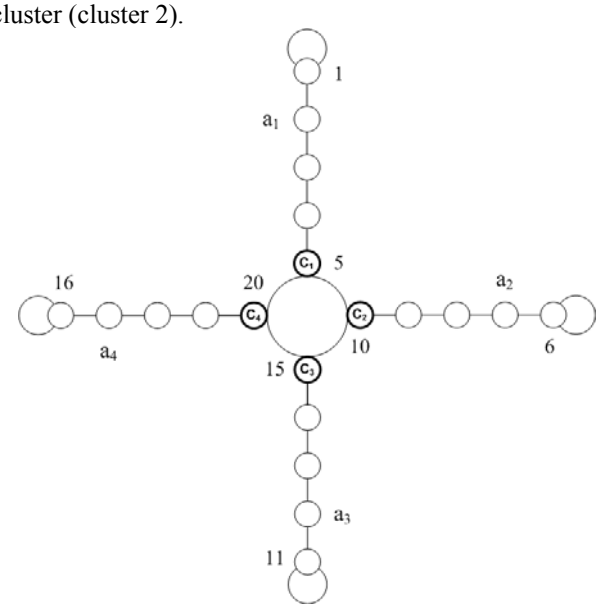


Fig. 2 An instance of call flow graph

Clustering of figure 2 is shown in figure 3 by a LA with similar connections to Tsetline automata. This automata has 4 (equal to the number of clusters) actions a_1, a_2, a_3, and a_4 and its depth is 5. States set {1,6,11,16} are inner states and states set {5,10,15,20} are outer states of the automata. At the beginning each cluster is placed at the outer state of relative action. For instance since in figure 3 $C_2=\{n_2, n_4, n_5\}$, so nodes n2, n4, n5 are placed in same cluster (cluster 2).



$C_1=\{n_1\}$ / $C_2=\{n_2, n_4, n_5\}$ / $C_3=\{n_3, n_6\}$ / $C_4=\{n_7, n_8\}$

Fig. 3 Showing clustering of figure 2 by a LA with similar connections to Tsetline automata

4.2 Initial population

Suppose that the number of population's members is n. Initial population members are generated by creation of n random clustering. As an example, initial population for call flow graph of figure 2 with the assumption n=6 is shown in figure 4. At the beginning each cluster is placed at the outer state of its action.

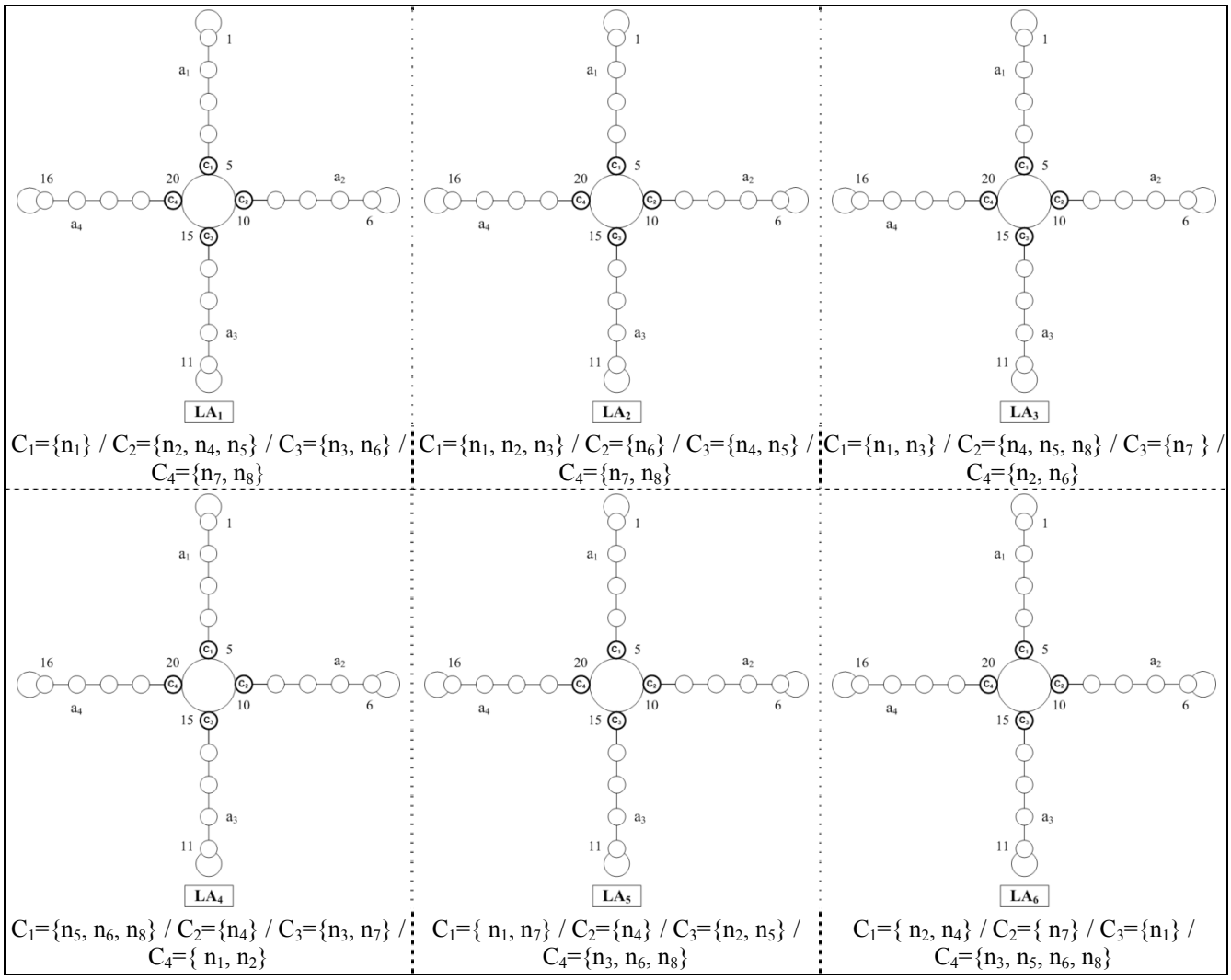


Fig. 4 Initial population for call flow graph of figure 2

4.3 Fitness function

In GAs fitness is indicator of chromosomes survival. In clustering problem fitness of an automata has a direct relation with the amount of yielding concurrency from execution of distributed code. Therefore, for determining fitness the amount of yielding concurrency from conversion of local calls to remote asynchronous calls must be calculated. When a function such as r is called via function m in a way i=a.r(), caller function m can run synchronous with function r until it does not need to the return value of r. For example consider the figure 5. In this figure interval between call of function r to the using point of this call's outcome is denoted by T_d, and execution time of function r is denoted by EET_r. Indisputable in the best case $2T_c+EETr \leq Td$, which T_c is required time for getting or sending of parameters. In general, waiting time of function m for getting return value from function r is calculated from the following criteria.

$T_{wait} = (T_d > 2T_c + EET_r) \; ? \; 0 : (2T_c + EET_r) - T_d$

The problem is calculating T_d and EET_r. Because, for instance, in the time of between call point and using point of this call's outcome or in context of called function may be exist other calls, and since it is unknown these functions will be executed local or remote, the execution time of them are unpredictable. For solving this problem, calculation of execution time must be started from a method in call flow graph in which it has not any call to other methods in call flow graph. Fitness function pseudo code is shown in figure 6.

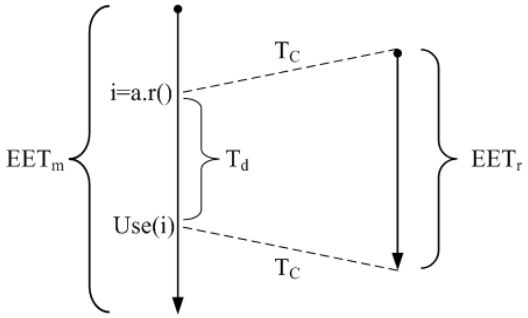


Fig. 5 Calculation of exection time in remote calls

```
Function FitnessEvaluation(CallFlowGraph, Clusters):Speedup
Begin
    Call TopologicalSort(CallFlowGraph);
    for each method m in call flow graph
        if NoCalls_m = 0 then
            EET_m = 0;
            for each not call statement, i, within m
                EET_m = EET_m + ExecutionTime(i);
            for each local call statement, c, within m
                EET_m = EET_m + EET_c;
            for each remote asynchronous call statement, r,
            within m
                EET_m = EET_m + Max(T_d, 2T_c + EET_r);
            for each parent, p, of m within the call flow
            graph
                NoCalls_p = NoCalls_p – 1;
        End if
    End for
End function
```

Fig. 6 Fitness function pseudo code

4.4 Operators

Since in proposed algorithm, every chromosome is shown in the form of LA, crossover and mutation operators are not similar to genetic traditional operators.

a) Selection operator: In order to choose LAs (chromosomes) for recombination or mutation operators, one of these methods can be used: Ranking Selection, Roulette Wheel Selection, and Tournament Selection.

b) Crossover operator: To do this operator a new method is proposed for working with sets namely NewCrossover. In this method two nodes n_i and n_j are selected randomly from call flow graph and their relative actions is swapped in two parents. With doing this operator two new automatas are created which are called children of two parents.

For example, suppose that LA_2 and LA_5 automatas are selected randomly as parents from the previously formed population. By random selection of two nodes n_3 and n_7 from call flow graph and swapping their relative actions in two parents, two new automats are created. This is shown in figure 7.

c) Mutation operator: To do this operator a node n_i is selected randomly from call flow graph and its action is swapped randomly. State of random node n_5 before and after mutation is shown in figure 8.

d) Penalty and reward operator: Since in proposed algorithm, every chromosome is shown in the form of LA, in each automata after examining fitness of a cluster (action) which is selected randomly, that cluster will be rewarded or penalized. State of a cluster in the relative action states set will be changed as the result of rewarding or penalizing it. If a cluster is placed at the outer state of an action, penalizing it leads to action of one of its nodes is changed and so a new clustering will created. The rate of this operator should be low, because this operator is a random search operator, and reduces the effectiveness of algorithm if it is applied with high rate. Reward and penalty operator vary according to the LA types.

For example, in an automata with similar connections to Tsetline automata if cluster C_3 is in the states set {11, 12, 13, 14, 15}, and its execution time without considering remote calls is less than threshold, this cluster is rewarded and it moves to the inner states of its action. If cluster C_3 is in the innermost state (state number 11) and rewarded, it will remain in that state. The movement of such cluster is shown in figure 9.

If execution time of a cluster without considering remote calls is greater than threshold, this cluster is not appropriate and it is penalized. The movement of such cluster for two different cases is as follows:

a) The cluster is at a state other than outer state: Panelizing this cluster reduces its importance and it moves to the outer states. The movement of such cluster is shown in figure 10.

b) The cluster is in outer state: In this case, we find a cluster of automata in which, if one of the cluster nodes is moved to the founded cluster, maximum increase in fitness outcome. In this case, if the founded cluster is not in outer state, first it is moved to outer state of its action and then node is moved to it. The movement of such cluster is shown in figure 11.

Important subject is determining of threshold value. For this consider concurrent execution time in following three different cases:

Best Case (100% Concurrency): $T_C = \frac{T_S}{NoClusters}$

Average Case (50% Concurrency): $T_T = \frac{T_S}{\frac{1}{2}NoClusters}$

Worst Case (0% Concurrency): $T_C = T_S$

In above criteria, T_c is concurrent execution time, T_s is sequent execution time, and NoClusters is number of clusters. In this paper we considered threshold value be concurrent execution time in average case.

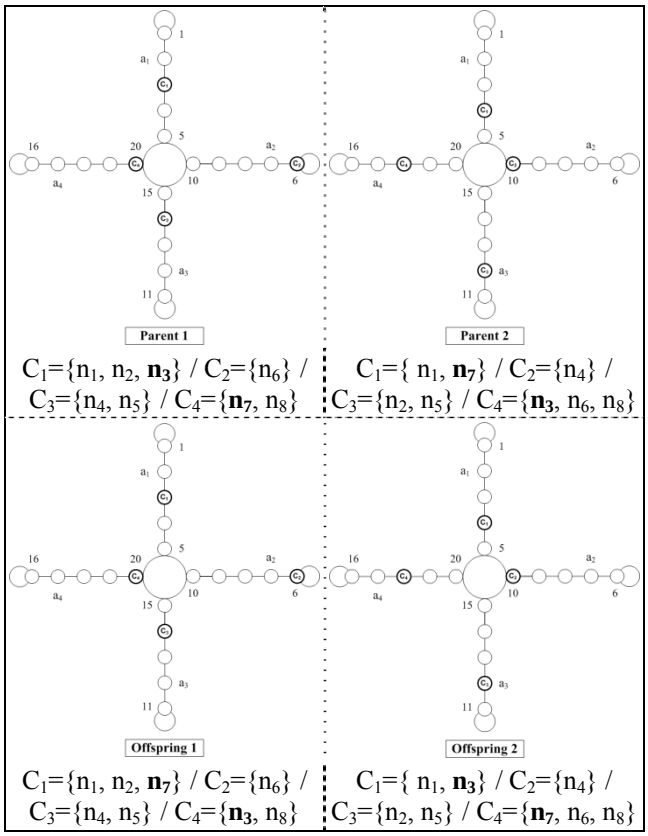


Fig. 7 The manner of doing crossover operator

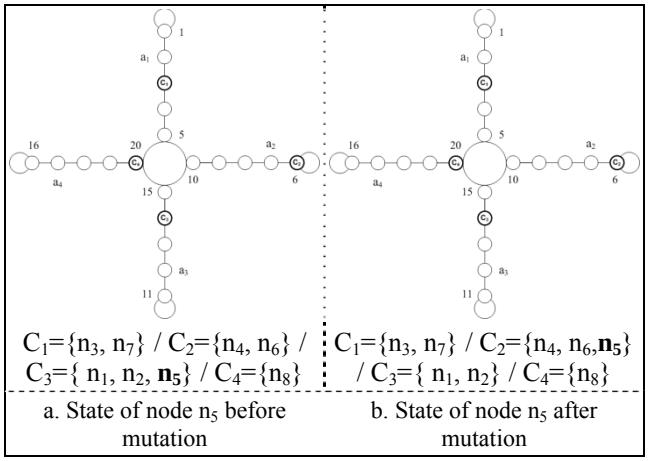


Fig. 8 The manner of doing mutation operator

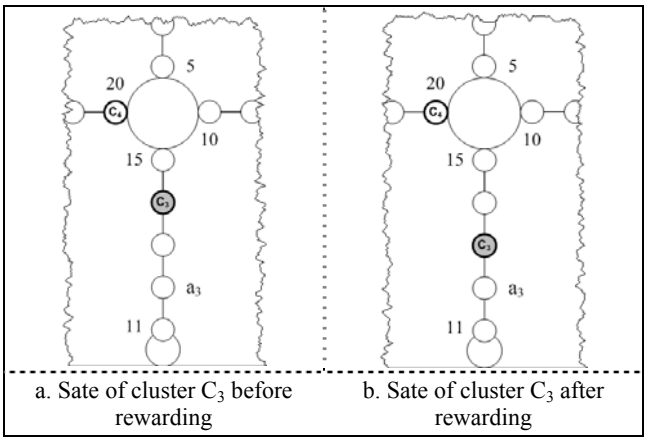


Fig. 9 The manner of rewarding a cluster

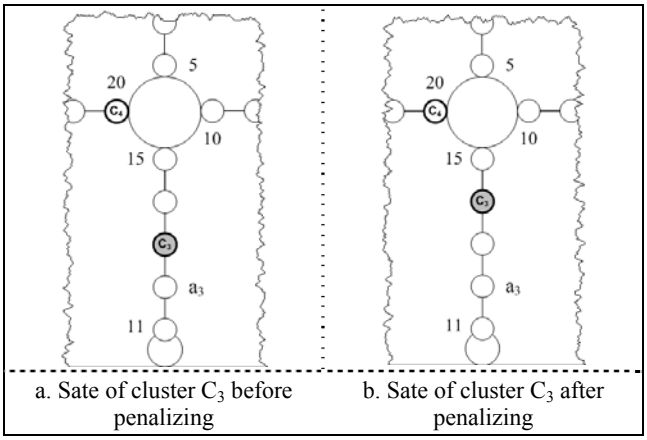


Fig. 10 The manner of penalizing a cluster palced in a state other than outer state

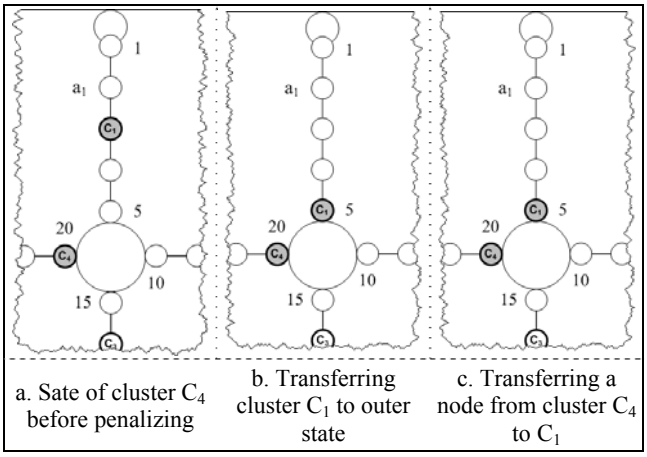


Fig. 11 The manner of penalizing a cluster palced in outer state

Proposed algorithm pseudo code is shown in figure 12.

```
Function CFG_Clustering(CallFlowGraph):Clusters
Begin
    n = Size of Population;
    Create the initial population LA1 … LAn;
    FitnessEvaluation(CallFlowGraph, LAi); //for all LAs
    while(Termination Condition not Satisfied) do
        NewLA1 = NewLA2 = LA with minimum Value of
        Execution Time; //move best two individuals to the new population
        for i = 2 to n do //Create new population
            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 //Replace old population with new population
            LAi = NewLAi;
            Cj = Random * NoClusters;
            if (ExecutionTime(LAi.Cj) < Threshold) then
                Reward(LAi , Cj);
            else
                Penalize(LAi , Cj);
        end for
        FitnessEvaluation(CallFlowGraph, LAi); //for all LAs
    end while
End function
```

Fig. 12 Proposed algorithm pseudo code

5. Proposed method evaluation

In order to evaluating proposed algorithm, distributed code of implementation of TSP was used. This distributed code solves TSP by using dynamic methods and finding optimal spanning tree.

Table and diagram 1 shows execution time of TSP program for three cases sequential, distributed by reference [2] algorithm clustering, and distributed by proposed algorithm clustering for graphs with different node and edges number.

As you observed average execution time of TSP program by proposed algorithm clustering is less than average execution time of sequential and distributed by reference [2] algorithm clustering. This shows that proposed algorithm is efficient than other algorithms, and it can be used for clustering of call flow graph of large application programs.

Table 1: Execution time of TSP program for three cases sequential, distributed by reference [2] algorithm clustering, and distributed by proposed algorithm clustering for graphs with different node and edges number

Number of Graph Nodes	Number of Graph Edges	Sequential Execution Time	Distributed Execution Time with Reference [2] Algorithm Clustering	Distributed Execution Time with Proposed Algorithm Clustering
20	40	0.573	7.357	4.347
40	81	1.383	7.810	5.057
60	122	3.246	8.163	6.163
80	163	11.214	11.109	8.713
100	204	19.773	14.741	10.677
120	245	43.517	30.722	22.222
140	286	85.362	60.871	46.546
160	327	145.721	105.227	73.379
180	368	234.871	168.280	112.511
200	409	360.143	261.412	196.615
220	450	576.655	440.343	316.741
240	491	997.653	774.142	542.524
Average Execution Time (Second)		**206.6759**	**157.5148**	**112.125**

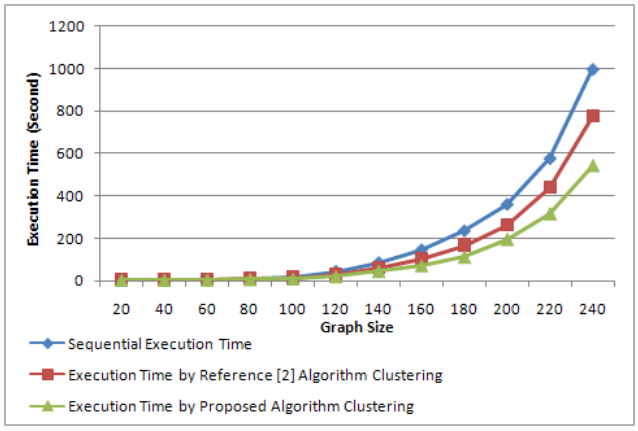


Diagram 1: Execution time of TSP program for three cases sequential, distributed by reference [2] algorithm clustering, and distributed by proposed algorithm clustering for graphs with different node and edges number

6. Conclusion

Problem of finding optimal distribution for reaching maximum concurrency in distributed programs is a NP-Complete problem. So, Deterministic methods are not appropriate for this problem. In this paper an evolutionary

non-deterministic method is proposed for this problem. Proposed method uses both GAs and LAs simultaneously to search in state space. Evaluation results and amount of yielding concurrency from using proposed algorithm, indicator of proposed method efficiency over others.

References

[1] S. Parsa, and O. Bushehrian, "Performance-Driven Object-Oriented Program Remodularization", ISSN: 1751-8806, INSPEC Accession Number: 10118318, Digital Object Identifier: 10.1049/iet-sen: 20070065, Aug, 2008.

[2] S. Parsa, and V. Khalilpoor, "Automatic Distribution of Sequential Code Using JavaSymphony Middleware", 32th International Conference On Current Trends in Theory and Practice of Computer Science, 2006.

[3] Roxana Diaconescu, Lei Wang, Zachary Mouri, and Matt Chu, "A Compiler and Runtime Infrastructure for Automatic Program Distribution", 19th International Parallel and Distributed Processing Symposium (IPDPS 2005), IEEE, 2005.

[4] S. Parsa, O. Bushehrian, "The Design and Implementation of a Tool for Automatic Software Modularization", Journal of Supercomputing, Volume 32, Issue 1, April 2005.

[5] Mohammad M. Fuad, and Michael J. Oudshoorn, "AdJava-Automatic Distribution of Java Applications", 25th Australasian Computer Science Conference (ACSC2002), Monash University, Melbourne, 2002.

[6] S. Mitchell Brian, "A Heuristic Search Approach to Solving the Software Clustering Problem", Thesis, Drexel University, March 2002.

[7] Thomas Fahringer, and Alexandru Jugravu, "JavaSymphony: New Directives to Control and Synchronize Locality, Parallelism, and Load Balancing for Cluster and GRID-Computing", Proceedings of Joint ACM Java Grande ISCOPE 2002 Conference, Seattle, Washington, Nov 2002.

[8] Michiaki Tatsubori, Toshiyuki Sasaki, Shigeru Chiba1, and Kozo Itano, "A Bytecode Translator for Distributed Execution of Legacy Java Software", LNCS 2072, pp. 236-255, 2001.

[9] Markus Dahm, "Doorastha—A Step Towards Distribution Transparency", JIT, 2000.

[10] Michael Philippsen, and Bernhard Haumacher, "Locality Optimization in JavaParty by Means of Static Type Analysis", Concurrency: Practice & Experience, pp. 613-628, July 2000.

[11] Andre Spiegel, "Pangaea: An Automatic Distribution Front-End for Java", 4th IEEE Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS '99), San Juan, Puerto Rico, April 1999.

[12] Saeed Parsa, and Omid Bushehrian, "Genetic Clustering with Constraints", Journal of Research and Practice in Information Technology, 2007.

[13] Leng Mingwei, Tang Haitao, and Chen Xiaoyun, "An Efficient K-means Clustering Algorithm Based on Influence Factors", Eighth ACIS Int. Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, pp. 815-820, July 2007.

[14] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu, "An Efficient K-Means Clustering Algorithm: Analysis and Implementation", IEEE Transaction on Pattern Analysis and Machine Intelligence, Vol. 24, No. 7, July 2002.

[15] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu, "The Analysis of a Simple K-Means Clustering Algorithm", Proc. of the Sixteenth Annual Symposium on Computational Geometry, pp. 162, June 2000.

[16] B. Hendrickson, and R. Leland, "A Multilevel Algorithm for Partitioning Graphs", Proceedings of the 1995 ACM/IEEE Conference on Supercomputing (CDROM), pp. 28, ACM Press, 1995.

[17] V. R. Vemuri, "Genetic Algorithms", Computer Society meeting, Department of Applied Science, University of California, 1997.

[18] E. Cantu-Paz, "A Survey of Parallel Genetic Algorithms", IlliGAL Reprot, No. 97003, May 1997.

[19] D. E. Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning", Reading, MA, Addition-Wesley, 1989.

[20] F. Busetti, "Genetic Algorithm Overview".

[21] K. A. Dejong , and W. M. Spears, "Using Genetic Algorithms to Solve NP-Complete Problems", Proceedings of the Third International Conference on Genetic Algorithms, 1989.

[22] K. S. Narendra, and M. A. L. Thathachar, "Learning Automata: An Introduction", Prentice-hall, Englewood cliffs, 1989.

[23] M. R. Meybodi, and H. Beigy, "Solving Graph Isomorphism Problem by Learning Automata", Thesis, Computer Engineering Faculty, Amirkabir Technology University, Tehran, Iran, 2000.

[24] H. Beigy, and M. R. Meybodi, "Optimization of Topology of Neural Networks Using Learning Automata", Proceedings of 3th Annual International Computer Society of Iran Computer Conference (CSICC-98), Tehran, Iran, pp. 417-428, 1999.

[25] B. J. Oommen, R. S. Valiveti, and J. R. Zgierski, "An Adaptive Learning Solution to the Keyboard Optimization Problem", IEEE Transaction On Systems. Man. And Cybernetics, Vol. 21, No. 6, pp. 1608-1618, 1991.

[26] B. J. Oommen, and D. C. Y. Ma, "Deterministic Learning Automata Solution to the Keyboard Optimization Problem", IEEE Transaction on Computers, Vol. 37, No. 1, pp. 2-3, 1988.

[27] A. A. Hashim, S. Amir, and P. Mars, "Application of Learning Automata to Data Compression", in Adaptive and Learning Systems, K. S. Narendra, Editor, New York, Plenum Press, pp. 229-234, 1986.

[28] M. R. Meybodi, and S. Lakshmivarhan, "A Learning Approach to Priority Assignment in a Two Class M/M/1 Queuing System with Unknown Parameters", Proceedings of Third Yale Workshop on Applications of Adaptive System Theory, Yale University, pp. 106-109, 1983.

[29] Bager Zarei, M. R. Meybodi, and Mortaza Abbaszadeh, "A Hybrid Method for Solving Traveling Salesman Problem", Proceedings of the 6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007), IEEE Computer Society, Melbourne, Australia, pp. 394-399, 11-13 July 2007.