A Learning Automata Based Graph Isomorphism Algorithm

Hamid Beigy
Soft Computing Laboratory
Computer Engineering Departement
Amirkabir University of Technology
Tehran, Iran
beigy@ce.aku.ac.ir

M. R. Meybodi
Soft Computing Laboratory
Computer Engineering Departement
Amirkabir University of Technology
Tehran, Iran
meybodi@ce.aku.ac.ir

Abstract

Learning automata (LA) is a global search method and used for solving several NP-Complete problems, such as object partitioning problem and optimizing the structure of neural networks. In this paper, we introduce an algorithm based on the learning automata for solving graph isomorphism problem. The feasibility of this algorithm is shown through computer simulations. Based on the simulation results, we conjecture that the proposed algorithm has average time complexity of order $\theta(n^3)$, which shows an improvement in order of magnitude.

1 Introduction

Let $G = (V, E)$ be a graph where V is a list of nodes, $E \subset V \times V$ is a set of edges of graph G, and a cost is associated to the edge $(i, j) \in E$. By definition, two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic, if there exist and one to one and onto function $\sigma : V_1 \rightarrow V_2$ such that for every edge $(u, w) \in E_1$ there exists an edge $(\sigma(u), \sigma(w)) \in E_2$. The class of difficulty of the graph isomorphism problem is not known. But, a brute force approach requires effort $O(n!)$ for a graph with n nodes. Therefore, the graph isomorphism problem belongs to class of *NP* problems. The graph isomorphism problem (GIP) has many applications, such as, VLSI circuit verification [1], PC-Board testing [2], equality of two programs [3], pattern recognition [4], and machine vision [5], to mention a few. In many practical applications, a certain amount of uncertainty or deformation may exist in structural description and two graphs are not perfectly isomorphic. In this case, the goal is to find a mapping with minimum error. This mapping is called *optimal mapping* and the problem is referred to as *error correcting graph isomorphism problem* (ECGIP) [5, 6]. The ECGIP belongs to the class of NP-Hard problems. Since the graph isomorphism problem is an intractable problem, its run time increases exponentially as the size of graph increases. There has been many efforts by researchers for finding algorithms with lower time complexity and many approximate algorithms are reported such as solving the GIP by finding the maximum clique [6], tree searches [7], algebraic analysis [6], neural networks [8], and genetic algorithms [6] which produce reasonable results.

Since the error-correcting graph isomorphism problem is *NP-Search* problem of combinatorial optimization, the development of an appropriate search strategy is crucial in reducing time complexity, and guaranteeing the correctness of mapping. Learning automata is a global search method and used for solving many NP-Complete problems, such as object partitioning problem [9] and optimizing the structure of neural networks [10], to mention a few. The object migrating automata (OMA) is a fixed structure learning automata that its actions represent a certain class and is useful for object partitioning problem [9]. In OMA, objects (in this paper, the nodes of graph) are migrated by object migrating automata and this migration results different partitioning.

By random searching in the state space, we could find different mapping. In this paper, we propose an LA based algorithm for solving graph isomorphism problem. To our knowledge, there is no learning automata based solution for this problem. This algorithm uses learning automata as a search method. The proposed algorithms are tested on graphs with different sizes and different noise ratios. Based on the simulation results, we conjecture that the proposed algorithms have average time complexity of order $\theta(n^3)$, which shows an improvement in order of magnitude over other algorithms. The rest of this paper is organized as follows: Section 2 describes the learning automata. The proposed algorithm is given in section 3 and the experimental results are given in section 4. The section 5 concludes the paper.

2 Learning Automata

The automata approach to learning involves determination of an optimal action from a set of allowable actions. It selects an action from its finite set of actions. The selected action serves as the input to the environment which in turn emits a stochastic response $\beta(n)$ at the time n. $\beta(n)$ is an element of $\beta = \{0, 1\}$ and is the feedback response of the environment to the automata. The environment penalize (i.e. $\beta(n) = 1$) the automata with the penalty probability c_i, which is the action dependent. Based on the response $\beta(n)$, the state of the automata is updated. Note that the c_i are unknown initially and it is desired that as a result of the interaction between the automata and the environment arrives at the action which presents it with the minimum penalty response in an expected sense.

An automaton acting in an unknown random environment and improves its performance in some specified manner, is referred to as *learning automata* (LA). Learning automata can be classified into two main families: *fixed structure learning automata* and *variable structure learning automata* [11]. Krylov, Krinsky, and Tsetline automata are examples of fixed structure learning automata and L_{R-I} and L_{R-P} automata are examples of variable structure learning automata. For more information about learning automata, refer to [11].

3 Proposed Algorithm

In this section, we first give different methods for construction of random graphs and then introduce a criterion for measuring the goodness of a mapping between two graphs, which will be used in the proposed algorithm for solving the GIP. This algorithm uses an object migrating automaton. Finally, an automaton that is used by the algorithm is described.

Constructing the Random Graphs Two models are typically used to create random graphs: Model A, and B. Model A uses a given probability to determine the existence of an edge in each entry of the adjacency matrix. Model B starts with a given number of edges and places these edges randomly between nodes. In this paper, we use the Model A for constructing the random graphs where a random number is generated for the value of the cost of a particular edge in a weighted graph.

Error Criteria Let $G = (V_G, W_G)$ be a weighted and directed graph G, where V_G is set of nodes and W_G is weight matrix for graph G. When $|W_{i,j}| > 0$, in graph G there is an edge (i, j) in graph G with weight $W_{i,j}$. Permutations of rows and the corresponding columns in the weight matrix W_G imply reordering of the vertices. Hence, two graphs $G = (V_G, W_G)$ and $H = (V_H, W_H)$ are isomorphic if and only if their weight matrices W_G and W_H differ only by permutations of rows and columns, i.e. W_G and W_H are related with a permutation σ which is given by $W_H = P.W_G.P^T$, where P is a permutation matrix of mapping σ. In each row (column) of matrix P there is only one element with value *one* and remaining elements in this row (column) are *zero*, where the element P_{ji} is mapping from node $v_i \in V_G$ to node $v_j \in V_H$. Based on the value of matrix P, the mapping $\sigma : V_G \to V_H$ is defined by the following equation.

$$\sigma = \{(v_i, v_j) | P_{ji} = 1, v_i \in V_G, v_j \in V_H\}\ i, j = 1, \cdots, n$$

The error of the mapping σ can be defined by following equation.

$$J(\sigma) = ||W_H - P.W_G.P^T|| \tag{1}$$

where $||.||$ is the matrix norm. So, solution of graph isomorphism problem can be formulated as searching a permutation σ that is solution of the equation $J(\sigma) = 0$. For ECGIP, solution of this problem can be formulated as searching a permutation σ that minimizes the function $J(\sigma)$. It is not difficult to see that for a graph with n nodes, the computation of $J(\sigma)$ requires $\theta(n^3)$ operations. To reduce the time complexity of computation of $J(\sigma)$, it can be reformulated as $[P.W_G.P^T]_{i,j} = [W_G]_{\sigma(i),\sigma(j)}$ where $[A]_{i,j}$ denotes the element in the ith row and jth column of matrix A. If the norm $||.||$ in equation (1) is replaced with L_1 norm, a simpler form of error function is obtained. For the sake of simplicity, therefore, the error of mapping node $v_k \in V_G$ to node $\sigma(v_k) \in V_H$ is called *error of mapping for node k* and is defined by the following equation.

$$\begin{aligned} J_k(\sigma) &= \sum_{m=1}^{n} |[W_H]_{k,m} - [W_G]_{\sigma(k),\sigma(m)}| \\ &+ \sum_{m=1}^{n} |[W_H]_{m,k} - [W_G]_{\sigma(m),\sigma(k)}| \end{aligned}$$

Therefore, the error of mapping σ is given by $J(\sigma) = \sum_{k=1}^{n} J_k(\sigma(m))$. It is not difficult to see that for a graph with n nodes, the computation of $J(\sigma)$ requires $\theta(n^2)$ operations.

Proposed Algorithm In what follows, we propose an LA based algorithm for solving the graph isomorphism problem. In this algorithm, as shown in figure 1, we assume that two graphs are undirected weighted graphs which are isomorphic with same size of n. Now, we describe the algorithm for finding the mapping σ. First, a one to one and onto mapping σ is created randomly. In the second step, we compute the error of mapping ($J_u(\sigma)$) for all nodes $u \in V_G$ and sort them in decreasing order. Then a node $u \in V_G$ is selected randomly such that $J_u(\sigma)$ is the Kth largest element in the set $\{J_1(\sigma), J_2(\sigma), \cdots, J_n(\sigma)\}$, where K is a random number in the interval $[1, M]$ and M is a constant. After this, the error of mapping for node $u \in V_G$ is compared with a quantity called *dynamic threshold* and depending on the result of comparison, the node $u \in V_G$ is penalized or rewarded. According to the graph isomorphism learning automata (described later), if $J_u(\sigma)$ is less than the dynamic threshold, then node $u \in V_G$ rewarded and penalized otherwise. The process of rewarding and penalizing is repeated until stopping criteria ($J(\sigma) = 0$) is reached, which at this point the last mapping is a considered as the solution to GIP. The dynamic threshold at instant $k \geq 1$ is defined as $T(k) = \frac{1}{n}\sum_{m=1}^{n} J_m(\sigma)$. In this algorithm, function $Uniform(1, M)$ generates a random number with uniform distribution in the interval $[1, M]$ and function $Select(K)$ chooses node $u \in V_G$ such that $J_u(\sigma)$ is the Kth largest element in $\{J_1(\sigma), J_2(\sigma), \cdots, J_n(\sigma)\}$.

```
procedure GI
    Input: Graphs $G$, and $H$
    Output: Mapping $\sigma$
    begin
        Construct a random mapping $\sigma$
        Set $k \leftarrow 1$
        repeat
            for $u = 1$ to $n$ do
                $J_u(\sigma) \leftarrow 2\sum_{m=1}^{n} |[W_H]_{u,m} - [W_G]_{\sigma(u),\sigma(m)}|$
            end for
            Compute $T(k) \leftarrow \frac{1}{n}\sum_{m=1}^{n} J_m(\sigma)$
            Compute $J(\sigma) \leftarrow \sum_{m=1}^{n} J_m(\sigma)$
            $u \leftarrow Select(Uniform(1, M))$
            If $J_u(\sigma) < T(k)$ Then
                Reward ($u$)
            else
                Penalize($u$)
            end if
            $k \leftarrow k + 1$
        until $J(\sigma) = 0$
    end GI
```

Figure 1: The LA based algorithm for solving GIP

Error Correcting Graph Isomorphism Algorithm The algorithm given in figure 1 can also be used for solving the ECGIP, except for the stopping criteria. We stop the algorithm if for a long period $J(\sigma)$ remains constant. The mapping σ which minimizes function $J(\sigma)$ can be considered as solution to ECGIP. Since the learning automata are a global search method, it seems having such a stopping criteria can guarantee the convergence to the global minima.

Graph Isomorphism Learning Automata In what follows, we introduce one object migrating automaton used by the proposed algorithm for process of rewarding and punishing the nodes. We call this automaton as *graph isomorphism learning automaton*.

Definition 3.1 A graph isomorphism learning automaton (GILA) is defined by tuple $(V, \alpha, \Phi, \beta, \mathcal{F}, \mathcal{G})$, where $V = \{v_1, v_2, \cdots, v_n\}$ is the set of nodes for the graph $G = (V_G, W_G)$. These nodes are migrated on the states of automaton and produce different mapping σ and $\alpha = \{\alpha_1, \alpha_2, \cdots, \alpha_n\}$ is the set of allowable actions for automaton. The number of actions of this automaton equals to number of nodes in the graph. The action α_k of automaton represents node $v_k \in V_H$ for $k = 1, 2, \ldots, n$. If node $v_u \in V_G$ is in the states of action α_m, then node $v_u \in V_G$ is mapped to node $v_m \in V_H$ (or $u = \sigma(m)$). $\Phi = \{\phi_1, \phi_2, \cdots, \phi_{nN}\}$ is the set of states of GILA and N is memory depth of the automaton. The set of states for this automaton is partitioned into n subsets $\{\phi_1, \cdots, \phi_N\}$, $\{\phi_{n+1}, \cdots, \phi_{2N}\}$, $\cdots$, $\{\phi_{(n-1)N+1}, \cdots, \phi_{nN}\}$. In GILA, N states $\{\phi_{(j-1)N+1}, \cdots, \phi_{jN}\}$ are associated to the action α_j for $j = 1, 2, \cdots, n$. The state of node u at instant k is denoted by $\phi_u(k)$. Assume that the states $\phi_{(j-1)N+1}$ and ϕ_{jN} to be the most internal and boundary states of action α_j, respectively. These states are called states of *maximum certainty* and *minimum certainty*, respectively. $\beta = \{0, 1\}$ is the set of inputs for the automaton. $\mathcal{F} : \Phi \times \beta \to \Phi$ is state transition function. This function determines the next state of the automaton on the basis of the current state and current input of automaton. Actually, this function determines how the nodes of graph G are migrated on the states of automata. $\mathcal{G} : \Phi \to \alpha$ is output mapping. If the node $v_u \in V_G$ is in one of states $\{\phi_{(m-1)N+1}, \cdots, \phi_{mN}\}$, then the automaton chooses action α_m.

In what follows, we describe the state transition function for Tsetline Like GILA which is used by the proposed algorithm. The state transition function of this automata can be described as:

1. At instant k, if node $u \in V_G$ is in states of action α_j (i.e. $\phi_u(k) \in \{\phi_{(j-1)N+1}, \cdots, \phi_{jN}\}$) and

automaton receives reward then node u is moved toward the internal states of action α_j. The state of automaton is remained unchanged when node u is in state $\phi_{(j-1)N+1}$.

2. At instant k, if node $u \in V_G$ is in action α_j and automaton receives penalty we have the following two cases.
   (a) When node u is not in boundary state of action α_j, the penalizing operation causes that node u be moved toward the boundary node of this action.
   (b) When node u is in boundary state of action α_j a node $w \in V_G$ $(u \neq w)$ is found and then nodes u and w are swapped. This swapping results in highest decrement in $J(\sigma)$. If selected node w is not in boundary state of its action, first it is moved to boundary state and then swapped with node u.

Since the transition graph of this automaton for the case of reward is like to transition graph for Tsetline automaton [11], we call it *Tsetline like graph isomorphism learning automaton* and denotes it by $Tsetline - GILA(n, N)$, where n is the number of actions and N is the memory depth.

4 Experimental Results

In this section the results of computer simulation on different graphs with different sizes are presented. Model A is used for constructing the undirected weighted graph G. Weight of each edge is a random number in the interval $[0, 99]$ where zero indicates the absence of an edge. We construct graph H from graph G with randomly relabelling the nodes of graph G. In all our simulations, the average number of iterations in every row of tables are obtained by averaging over 10 runs. The repeat–until loop of the algorithm is iterated at most 5000 times. The simulation results are given in table 1. For ECGIP, when graph H is obtained from graph G, weights of its all edges are corrupted with a random number $x \in [-X, X]$ where X is the percentage of noise. We considered three different noise ratios 5%, 10%, and 15% [12]. The simulation results for 10% noise is given in table 2. In all simulations for ECGIP, the algorithms are terminated if $J(\sigma)$ remains unchanged for 20 iterations.

5 Conclusion

In this paper, we proposed an algorithm based on learning automata for solving graph isomorphism problem. To our knowledge, no learning automata based solution for this problem has been reported. This algorithm uses learning automata as a search method. The proposed algorithm has been tested on number of graphs with different sizes and different noise ratios. Based on the simulation results, we conjectured that the proposed algorithm has average time complexity of order $\theta(n^3)$, which shows an improvement in order of magnitude.

References

[1] P. M. Maurer and A. D. Shapira, "A logic-to-logic comparator for vlsi layout verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, no. 8, pp. 897–907, 1988.

[2] K. T. Huang and D. Overhauser, "A novel graph algorithm for circuit recognition," in *Proceedings of IEEE European International Sysmposium on Circuits and Systems*, pp. 1695–1698, 1995.

[3] E. Petrank and R. M. Roth, "Is code equivalence easy to decide," *IEEE Transactions on Information Theory*, vol. 43, no. 5, pp. 1602–1604, 1997.

[4] R. J. Schalkoff, *Pattern Recognition: Statistical, Syntactical, and Neural Approaches.* New York: John Whiley, 1992.

[5] L. Cinque, D. Yasuda, L. G. Shapiro, S. Tanimoto, and B. Allen, "An improved algorithm for relational distance graph matching," *Pattern Recognition*, vol. 29, no. 2, pp. 349–359, 1996.

[6] Y. Wang, K. Fan, and J. Horng, "Genetic based search for error-correcting graph isomorphism," *IEEE Transactions on Systems, Man, and Cybernetics-Part B:Cybernetics*, vol. 27, no. 4, pp. 588–597, 1997.

[7] F. Depiero, M. Trived, and S. Serbin, "Graph matching using direct classification of node attendance," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1031–1048, 1996.

[8] K. Agusa, S. Fujita, S. Yamashita, and T. Ae, "On neural networks for graph isomorphism problem," in *Proceedings of RNNS/IEEE European International Symposium on Neuroinformatics and Neurocomputers*, pp. 1142–1148, 1992.

[9] B. J. Oommen and D. C. Y. Ma, "Deterministic learning automata solutions to the equipartitioning problem," *IEEE Transactions on Commputers*, vol. 37, pp. 2–13, Jan. 1988.

[10] H. Beigy and M. R. Meybodi, “An Algorithm Based on Learning Automata for Determining the Minimum Number of Hidden Units in Three Layers Neural Networks,” *Amirkabir Journal of Science and Technology*, vol. 12, no. 46, pp. 111–136, 2001.

[11] K. S. Narendra and K. S. Thathachar, *Learning Automata: An Introduction.* New York: Printice-Hall, 1989.

[12] H. Beigy and M. R. Meybodi, “Solving the graph isomorphism problem using learning automata,” tech. rep., Computer Engineering Departement, Amirkabir University of Technology, Tehran, Iran, 1998.

Table 1: The simulation results of Tsetline-GILA for graphs with 0% noise

	Average Iterations					Runs Not Converged				
	N					N				
n	1	3	5	7	9	1	3	5	7	9
20	80	131	189	101	111	0	0	0	1	1
30	166	124	107	112	87	0	0	0	0	0
40	70	179	95	146	144	0	0	0	0	0
50	53	79	89	112	129	0	0	0	0	0
60	69	80	113	130	150	0	0	0	0	0
70	89	114	174	168	215	0	1	0	0	0
80	84	114	149	161	199	0	0	0	0	0
90	95	133	155	183	227	0	0	0	0	0
100	98	149	185	216	278	0	0	0	0	0
110	119	161	213	241	283	0	0	0	0	0
120	100	144	193	241	262	0	0	0	0	0

Table 2: The simulation results of Tsetline-GILA for graphs with 10% noise

n	Error	Average Iterations					Runs Not Converged				
		N					N				
		1	3	5	7	9	1	3	5	7	9
20	3834	153	258	220	256	215	0	0	0	0	1
30	8552	86	94	105	130	144	0	0	0	0	0
40	15416	686	912	954	1098	1146	0	0	1	0	0
50	24122	145	218	153	169	197	0	0	0	0	0
60	34278	1768	1600	1052	2592	1469	3	6	3	4	5
70	47570	132	159	182	224	262	0	0	0	0	0
80	62298	132	164	192	217	264	0	0	0	0	0
90	78200	167	230	234	286	298	0	0	0	0	0
100	97388	210	212	267	343	367	1	0	0	0	0
110	117778	192	237	271	323	352	0	0	0	0	0
120	139828	176	220	273	327	347	0	0	0	0	0