Randomized Las Vegas Algorithms For Graph Isomorphism

Hamid Beigy M. R. Meybodi

Computer Engineering Departement
Amirkabir University of Technology
Tehran, Iran
beigy@ce.aku.ac.ir meybodi@ce.aku.ac.ir

Abstract. Graph isomorphism is an important problem in graph theory which has many applications, such as VLSI circuit verification, PC-Board testing, equality of two programs, pattern recognition, and machine vision, for mention a few. This problem belongs to class of NP problems and there is no polynomial time algorithm for solving it. A brute force approach requires effort $O(n!)$ for graphs with n nodes. In many applications, the graphs are noisy and are not perfect isomorphism and goal is to find a mapping with minimum error. This problem is called error correcting graph isomorphism problem and belongs to class of NP-Hard problems which has exponential time complexity. In this paper, six randomized Las Vegas algorithms are proposed for solving the graph isomorphism problems. These algorithms use random search and can be used to solve both graph isomorphism and error correcting graph isomorphism problems. To our knowledge, these algorithms are the first randomized algorithms for solving these problems. The power of these algorithms is shown through the computer simulations. The experimental results show that some of these algorithms have the time complexity $\theta(n^4)$ and others have time complexity $\theta(n^3)$ which show an improvement in order of magnitude over brute force approach.

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 an 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 complexity of graph isomorphism problem is not known. A brute force approach, which is shown in figure 1, requires effort $O(n!)$ for a graph with n nodes. Thus, graph isomorphism problem belongs to class of NP problems.

The graph isomorphism problem (GIP) has many applications, such as VLSI circuit verification [1][2], PC-Board testing [3], equality of two programs [4], pattern recognition [5], and machine vision [6][7], for mention a few. For example, in

```
function GraphIsomorphism (V_G, V_H)
    Input: Graphs G = (V_G, E_G), and H = (V_H, E_H)
    Output: Mapping σ
    begin
        Let k ← |V_H| and n ← |V_G|
        Let σ ← φ
        while k > 0 do
            σ = {(u_n, w_k)} ⋃ GraphIsomrphism(V_G − {u_n}, V_H − {w_k})
            if error of mapping σ is zero Then
                return σ
            end if
            dec (k)
        end while
        return σ
    end GraphIsomorphism
```

Fig. 1. Backtracking algorithm for graph isomorphism problem

machine vision applications, the structural representation of objects and models are given by graphs. To recognize an object, its structural representation will be found and then the recognition problem becomes the graph isomorphism problem, but there may exist noise such that two graphs can not be considered isomorphic to each other. 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) [6][8]. This problem belongs to the class of NP-Hard problems.

To improve the time complexity of GIP, approximation algorithms such as solving the GIP by finding the maximum clique [8], relaxation methods [7], tree searches [9], algebraic analysis [8], neural networks [10], and genetic algorithms [8][11] have been proposed which produce reasonable results. In order to find answer of intractable problems such as GIP and ECGIP, the algorithm needs to search the solution space. In GIP and ECGIP the running time is proportional to the number of mapping. However, most of these mapping are not correct mapping but algorithm needs to consider all of them. The lesser is the number of the mapping considered by the algorithm the lesser will be the time complexity. The randomized algorithms can be used to reduce the time complexity [12].

In this paper, six randomized Las Vegas algorithms are proposed for solving the graph isomorphism problem. These algorithms use random search and can be used to solve both graph isomorphism and error correcting graph isomorphism problems. To our knowledge, these algorithms are the first randomized algorithms for solving these problems. The power of these algorithms is shown through the computer simulations. The experimental results show that some of these algorithms have the time complexity $\theta(n^4)$ and others have time complexity $\theta(n^3)$ which shows an improvement in order of magnitude over brute force approach.

The rest of this paper is organized as follows: Section 2 describes the brief review of randomized algorithms. The proposed algorithms are given in section 3 and the experimental results are given in section 4. The last section concludes this paper.

2 Randomized Algorithms

The randomized algorithms are those algorithms that take random choice in their execution. Randomized algorithms have two advantages of simplicity and performance over deterministic algorithms. This means that randomized algorithms find answer for most of complex problems faster than deterministic algorithms and implementation of these algorithms is very simpler than the deterministic algorithms with same complexity. Randomized algorithms can be divided into following four main groups [13].

Numerical Algorithms Randomness was first used to find approximate solution of numerical problems when there exists no deterministic algorithms for solving them. For example, to estimate the mean length of a queue in a system so complex that is infeasible to get closed-form solutions or even numerical answers by deterministic algorithms. The answers obtained by a numerical randomized algorithm is always approximate, but its expected precision improves as the time available to the algorithm increases.

Monte Carlo Algorithms These algorithms are used for solving the decision problems and always converge to the solution but the answer obtained by Monte Carlo algorithms are not necessarily correct. The probability of correctness of answers can be increased by repeatedly running the algorithm which increases the time complexity of the algorithm. One disadvantage of Monte Carlo algorithms is the verification time of the correctness of answers obtained by these algorithms.

Sherwood Algorithms These algorithms always converge to correct answers. Sherwood algorithms are often used when a known deterministic algorithm to solve the problem of interest runs much faster on the average than the worst case. Incorporating an element of randomness allow a Sherwood algorithm to reduce and sometimes even to eliminate this difference between good and bad instances. The performance of a Sherwood algorithm is not better than of the corresponding deterministic algorithm when we consider the average over all instances of a given size, but the upper bound of run time of these algorithms are predictable.

Las Vegas Algorithms The answer obtained by these algorithms always are correct but it is possible that the algorithm does not converge. The Las Vegas algorithms are faster than the deterministic algorithms and are very suitable when there is no efficient deterministic algorithm. The upper bound of execution time of these algorithms are not predictable but the probability of converging to the correct answer can be increased by consecutive execution of the algorithm. A Las Vegas algorithm is said to be correct if it produces the correct answers for every input with the probability of greater than zero.

We now discuss a number of fundamental techniques of randomizing used by designers of randomized algorithms. These techniques have overlap in the sense that more than one may apply to a given randomized algorithm. These Techniques can be divided in the following five main groups [12].

Input Randomization Regularity of input data causes that the average case time complexity becomes better than the worst case. Input randomization is rearranging or permuting input to rid it of any existing patterns, ensures that for all inputs, the expected running time matches the average running time. This technique can be effective on problems that have algorithms with good average running time but poor worst-case running time due to some unfavorable input patterns.

Random Search Random search is one of most widely used probabilistic techniques. Many problems naturally involve searching a large space for an element having a desired property. If the property in question is easily verified and if the elements processing it are abundant, random search can be very effective.

Control Randomization Consider a problem, such as sorting, for which many algorithms exist. If each of these algorithms has good average performance but poor worst-case performance, it is very risky to use any single one of them. This is especially true if the input probability distribution is not known. A way around this problem is to pick one of algorithms for each input instance.

Random Sampling Sometimes it is possible to ascertain with high probability, certain properties of a set S from a randomly chosen subset of S. In this case, a subset of S is randomly chosen and the properties of S is obtained from this subset. The nearest neighborhood algorithm is an example of this technique [12].

Symmetry Breaking There are certain problems in distributing computing, in particular, problems in which process must reach some sort of agreement, that do not have deterministic solutions. By introducing the randomization in to behavior of the processes themselves, these patterns of identical or symmetric behavior can be broken, thereby leading to agreement.

3 Proposed Algorithms

In this section, we first give the methods for construction of random graphs which is used as inputs and then introduce a criteria for measuring the goodness of a mapping between two graphs. Finally, we present six randomized Las Vegas algorithms for solving the graph isomorphism problem.

3.1 Constructing the Random Graphs

Two models are typically used to create random graphs: Model A, and B. Model A uses a given probability (often 0.5) 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 used 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.

Let $G = (V_G, W_G)$ be the weighted and directed graph G, where V_G is set of nodes of graph and W_G is weight matrix[1] for graph G. When $|W_{i,j}| > 0$ in graph G, there is an edge (i, j) in graph G with weight $W_{i,j}$. Therefore, the undirected graphs has symmetric weight matrix. Permutations of rows and of the corresponding column 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 equation (1).

$$W_H = P.W_G.P^T \tag{1}$$

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 reminder 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.

$$\sigma = \{(v_i, v_j)|P_{ji} = 1, v_i \in V_G, v_j \in V_H\} \ \forall i, j = 1, 2 \cdots, n \tag{2}$$

For example, let $\sigma = \{(A, b), (B, d), (C, c), (D, a)\}$ be a mapping, therefore the permutation matrix P is equals to:

$$P = \begin{bmatrix} 0\,0\,0\,1 \\ 1\,0\,0\,0 \\ 0\,0\,1\,0 \\ 0\,1\,0\,0 \end{bmatrix} \tag{3}$$

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

$$J(\sigma) = ||W_H - P.W_G.P^T|| \tag{4}$$

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

$$[P.W_G.P^T]_{i,j} = [W_G]_{\sigma(i),\sigma(j)}$$

[1] When the elements of matrix W are from set $\{0, 1\}$, the matrix W is called adjacency matrix.

where $[A]_{i,j}$ denotes the element in the ith row and jth column of matrix A. If the norm $||.||$ in equation (4) is replaced with L_1 norm [2], the 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.

$$J_k(\sigma) = \sum_{m=1}^{n} |[W_H]_{k,m} - [W_G]_{\sigma(k),\sigma(m)}| + \sum_{m=1}^{n} |[W_H]_{m,k} - [W_G]_{\sigma(m),\sigma(k)}| \tag{5}$$

For undirected graphs error of mapping for node k is equal to:

$$J_k(\sigma) = 2\sum_{m=1}^{n} |[W_H]_{k,m} - [W_G]_{\sigma(k),\sigma(m)}| \tag{6}$$

Therefore, the error of mapping σ for two graphs is given by the following equation.

$$J(\sigma) = \sum_{k=1}^{n} J_k(\sigma(m)) \tag{7}$$

It is not difficult to see that for a graph with n nodes, the computation of $J(\sigma)$ in equation (7) requires $\theta(n^2)$ operations.

3.2 Randomized Graph Isomorphism Algorithms

In what follows, we propose six Las Vegas algorithms for solving the GIP. The high level description of these algorithms is given in figure 2. The input to the algorithms is two undirected weighted graphs having equal number of nodes and isomorphic to each other.

Now, we describe the proposed algorithms.

Randomized Algorithm 1 The first Las Vegas algorithm for graph isomorphism problem is shown in figure 3 and can be described as follows. In the first step, a random mapping σ is created. This mapping is an one to one and onto mapping. In the second step, we compute the error of mapping for all nodes $u \in V_G$ and sort $J_u(\sigma)$ in decreasing order. Then a node $u \in V_G$ is selected randomly such that $J_u(\sigma)$ is kth largest element in $\{J_1(\sigma), J_2(\sigma), \cdots, J_n(\sigma)\}$, where k is a random number in the interval $[1, N]$ and N is a predefined constant. After this, the $J_u(\sigma)$ is compared with a quantity called *dynamic threshold* and depending on the result of comparison, the mapping σ is reordered. If $J_u(\sigma)$ is greater

[2] The L_1 norm of matrix M is given by $||M|| = \sum_i \sum_j |m_{ij}|$

```
procedure GI
   Input: Graphs G, and H
   Output: Mapping σ
   begin
      Construct a random mapping σ
      repeat
         for u = 1 to n do
            Compute J_u(σ)
         end for
         Select a pair (u ∈ V_G, v ∈ V_H) according to some criteria
         Reorder mapping σ in order to find global minimum of J(σ)
      until J(σ) = 0
  end GI
```

Fig. 2. High level description of randomized algorithm for solving the GIP

than the dynamic threshold, then the node $u \in V_G$ is swapped with node $v \in V_G$ such that the $J_v(\sigma)$ is mth largest element in $\{J_1(\sigma), J_2(\sigma), \cdots, J_n(\sigma)\}$, where m is a random number in the interval $[1, N]$. In other words in the new mapping, two pairs $(u, \sigma(u))$ and $(v, \sigma(v))$ is changed to pairs $(u, \sigma(v))$ and $(v, \sigma(u))$ and other pairs in the mapping σ remain unchanged. This process is repeated until a stopping criteria $(J(\sigma) = 0)$ is reached, which at this point the last mapping is the solution of graph isomorphism problem. The dynamic threshold at instant $k \geq 1$ is defined by equation (8).

$$T(k) = \frac{1}{n} \sum_{m=1}^{n} J_m(\sigma) \tag{8}$$

In the algorithm given in figure 3, the function $Uniform(1, N)$ generates a random number with uniform distribution in the interval $[1, N]$ and the function $Select(k)$ selects the node $u \in V_G$ such that $J_u(\sigma)$ is kth largest element in $\{J_1(\sigma), J_2(\sigma), \cdots, J_n(\sigma)\}$.

In order to increase the performance of the algorithm, when the size of the underlying problem is less than a threshold search will be continued for the solution using a deterministic algorithm. This leads to higher probability of convergence of the randomized algorithms. When the size of the problem $(J(\sigma))$ is less than the threshold instead of random selection of node u, the node with largest error is chosen.

This algorithm is suitable for graphs with small number of nodes and its performance decreases when the graphs become large. Since, this algorithm searches the search space blindly which may lead taking wrong decisions and wrong swapping.

Randomized Algorithm 2 This algorithm which is shown in figure 4 similar to algorithm 1 uses the blind search for finding the mapping σ. In this algorithm

```
procedure GI
   Input:  Graphs G, and H
   Output:  Mapping σ
   begin
      Construct a random mapping σ
      Set k ← 1
      repeat
         for  u = 1 to  n do
            J_u(σ) ← 2 Σ_{m=1}^{n} |[W(H)]_{u,m} − [W(G)]_{σ(u),σ(m)}|
         end for
         Compute T(k) ← (1/n) Σ_{m=1}^{n} J_m(σ)
         Compute J(σ) ← Σ_{m=1}^{n} J_m(σ)
         u ← Select(Uniform(1, N))
         if  J_u(σ) > T(k) Then
            v ← Select(Uniform(1, N))
            Swap(u, v)
         end if
         k ← k + 1
      until  J(σ) = 0
   end  GI
```

Fig. 3. Randomized algorithm 1

instead of random selection of node v, as in algorithm 1, node with largest error is chosen. This may result in larger decrease in the value of $J(\sigma)$.

Randomized Algorithm 3 The algorithm 3 (figure 5) like algorithm 2 chooses node v with largest value of $J_v(\sigma)$, but it swaps node u with node v if this swapping results in decrement in $J(\sigma)$. Such a modification results in lesser number swapping made by the algorithm.

Randomized Algorithm 4 In this algorithm, node u is selected in the same manner as preceding algorithms but node v is selected randomly from the graph H. A new mapping σ^* is created from σ by swapping the nodes u and $\sigma^{-1}(v)$. Depending on the situation one of following actions will be taken.

1. If $J(\sigma^*)$ is less than the dynamic threshold and $\sigma(u) \neq v$ and $J_u(\sigma^*)$ is less than $J_u(\sigma)$, then two nodes u and $\sigma^{-1}(v)$ will be swapped.
2. If $J(\sigma^*)$ is greater than or equal the dynamic threshold and $\sigma(u) = v$ then node u will be swapped with a node w which results in smallest $J(\sigma)$.

This algorithm, as shown in figure 6. This algorithm works well for small graphs.

Randomized Algorithm 5 Unlike algorithm 4, in this algorithm he node v is selected in such a manner $J_v(\sigma)$ is kth largest error, where k is a random number in the interval $[1, N]$. To obtain algorithm 5, we replace statement $v \leftarrow \sigma(Uniform(1, n))$ in algorithm 4 by statement $v \leftarrow \sigma(Select(Uniform(1, n)))$.

```
procedure GI
    Input: Graphs G, and H
    Output: Mapping σ
    begin
        Construct a random mapping σ
        Set k ← 1
        repeat
            for u = 1 to n do
                J_u(σ) ← 2 Σ_{m=1}^{n} |[W(H)]_{u,m} − [W(G)]_{σ(u),σ(m)}|
            end for
            Compute T(k) ← (1/n) Σ_{m=1}^{n} J_m(σ)
            Compute J(σ) ← Σ_{m=1}^{n} J_m(σ)
            u ← Select(Uniform(1, N))
            if J_u(σ) > T(k) Then
                v ← Select(1)
                Swap(u, v)
            end if
            k ← k + 1
        until J(σ) = 0
    end GI
```

Fig. 4. Randomized algorithm 2

```
procedure GI
    Input: Graphs G, and H
    Output: Mapping σ
    begin
        Construct a random mapping σ
        Set k ← 1
        repeat
            for u = 1 to n do
                J_u(σ) ← 2 Σ_{m=1}^{n} |[W(H)]_{u,m} − [W(G)]_{σ(u),σ(m)}|
            end for
            Compute T(k) ← (1/n) Σ_{m=1}^{n} J_m(σ)
            Compute J(σ) ← Σ_{m=1}^{n} J_m(σ)
            u ← Select(Uniform(1, N))
            if J_u(σ) > T(k) Then
                v ← Select(1)
                create mapping σ* by swapping nodes u and v
                if J_u(σ) > J_u(σ*) Then
                    Swap(u, v)
                end if
            k ← k + 1
        until J(σ) = 0
    end GI
```

Fig. 5. Randomized algorithm 3

```
procedure GI
    Input: Graphs G, and H
    Output: Mapping σ
    begin
        Construct a random mapping σ
        Set k ← 1
        repeat
            for u = 1 to n do
                J_u(σ) ← 2 Σ_{m=1}^{n} |[W(H)]_{u,m} − [W(G)]_{σ(u),σ(m)}|
            end for
            Compute T(k) ← (1/n) Σ_{m=1}^{n} J_m(σ)
            Compute J(σ) ← Σ_{m=1}^{n} J_m(σ)
            u ← Select(Uniform(1, N))
            v ← Uniform(1, n)
            create mapping σ* from σ by swapping u and σ^{-1}(v)
            if J_u(σ*) < T(k) Then
                if σ(u) ≠ v and J_u(σ*) < J_u(σ) Then
                    Swap (u, σ^{-1}(v))
                end if
            else
                if σ(u) = v Then
                    MoveNode (u)
                end if
            end if
            k ← k + 1
        until J(σ) = 0
    end GI

procedure MoveNode (u)
    begin
        bestError ← J_u(σ)
        bestNode ← u
        for w = 1 to n do
            Create σ* by swapping nodes u and w in mapping σ
            if J_w(σ*) < bestError Then
                bestError ← J_w(σ*)
                bestNode ← w
            end if
        end for
        Swap (u, bestNode)
    end MoveNode
```

Fig. 6. Randomized algorithm 4

Randomized Algorithm 6 In this algorithm, as shown in figure 7, node u is selected in the same manner as algorithm 3, but node v with which u will be swapped is computed differently. Node v is computed in such a way that minimizes $J(\sigma)$. Such modification will increase the probability choosing a better match for node u.

```
procedure GI
    Input: Graphs G, and H
    Output: Mapping σ
    begin
        Construct a random mapping σ
        Set k ← 1
        repeat
            for u = 1 to n do
                J_u(σ) ← 2 Σ_{m=1}^{n} |[W(H)]_{u,m} − [W(G)]_{σ(u),σ(m)}|
            end for
            Compute T(k) ← (1/n) Σ_{m=1}^{n} J_m(σ)
            Compute J(σ) ← Σ_{m=1}^{n} J_m(σ)
            u ← Select(Uniform(1, N))
            if J_u(σ) > T(k) Then
                MoveNode (u)
            end if
            k ← k + 1
        until J(σ) = 0
    end GI

procedure MoveNode (u)
    begin
        bestError ← J_u(σ)
        bestNode ← u
        for w = 1 to n do
            Create σ* by swapping nodes u and w in mapping σ
            if J_w(σ*) < bestError Then
                bestError ← J_w(σ*)
                bestNode ← w
            end if
        end for
        Swap (u, bestNode)
    end MoveNode
```

Fig. 7. Randomized algorithm 6

3.3 Error Correcting Graph Isomorphism Algorithm

By having appropriate stopping criteria, all preceding algorithms can be used for solving the ECGIP problem. It should be mentioned that since these algorithms do not use local information for finding solution, the probability of trapping algorithms at local minima will be small. Therefore, if for long period of time $J(\sigma)$ remains constant then σ can be considered as solution of ECGIP. For ECGIP only algorithm 6 tested and the result is given later. To obtain algorithm 6 for ECGIP, we replace statement **until** $J(\sigma) = 0$ in algorithm 6 by statement **until** $J(\sigma) = 0$ *OR for long period* $J(\sigma)$ *is constant.*

4 Experimental Results

In this section the results of computer simulation on different graphs with different sizes are given. We used the Model A for constructing the undirected weighted graph G such that the weight of each edge is a random number in the interval $[0, 99]$ where the zero weight means there is no edge. We construct the graph H from graph G with randomly relabeling the n nodes of graph G. In all our simulations, the average number of iterations in every row of each table is obtained by 10 times running the given algorithm on two random graphs G and H. This number is average over the runs that converge in 10000 iterations. The results of simulations for different algorithms are given in table 1. Figure 8 shows the average number of iterations versus size of graph. From figure 8, we claim that the algorithms 1 through 4 have average time complexity of $\theta(n^4)$ and algorithms 5 and 6 have time complexity of $\theta(n^3)$. For ECGIP, when graph H is obtained from graph G, weight of its all edges are corrupted with a random number $x \in [-X, X]$, where X is percent of noise. We considered three different noise ratios 5%, 10%, and 15%. The results of simulation for different algorithms are given in table 2. In all simulations for ECGIP, the algorithms are terminated if for 20 iterations $J(\sigma)$ remains unchanged. The computer simulations show that the algorithm 6 for ECGIP has the same average time complexity as GIP. For more experimentation refer to [14].

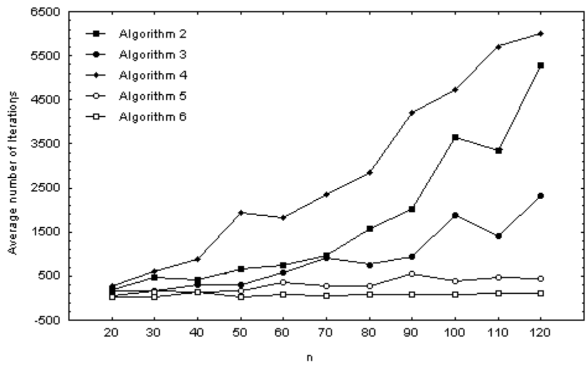

Fig. 8. Average number of iterations

5 Conclusion

In this paper we introduced six randomized algorithms for graph isomorphism problem. The computer experiment show that for graphs with size of n some of these algorithms has time complexity of $\theta(n^4)$ and some others have time complexity of $\theta(n^3)$. Since the backtracking algorithm has time complexity of

Table 1. The simulation results of randomized algorithms

a) Average number of iterations

n	Algorithm 1	2	3	4	5	6
20	424	189	151	287	65	18
30	238	479	173	620	151	32
40	554	419	292	873	135	145
50	593	666	315	1934	171	37
60	1280	759	568	1812	353	69
70	2976	964	923	2355	268	49
80	3533	1581	760	2861	286	74
90	4477	2007	948	4200	544	91
100	*	3656	1881	4731	385	84
110	*	3340	1406	5712	465	109
120	*	5280	2315	5991	436	105

b) Number of runs that does not converge

n	Algorithm 1	2	3	4	5	6
20	0	1	0	0	0	0
30	0	0	0	0	0	0
40	0	0	1	0	0	0
50	0	0	0	0	0	0
60	0	0	0	0	0	0
70	0	0	0	0	0	0
80	0	0	0	0	0	0
90	0	0	0	1	0	0
100	10	0	0	0	0	0
110	10	0	3	0	0	0
120	10	0	0	0	0	0

*The algorithm 1 does not converge for graphs with more than 100 nodes in 10000 iterations.

Table 2. The results of algorithm 6 for graphs with different noise

n	Error			Average Iterations			Runs Not Converge		
	5%	10%	15%	5%	10%	15%	5%	10%	15%
20	1796	3712	5538	116	16	93	0	0	0
30	4516	8938	12856	432	65	160	0	0	0
40	7620	15104	22946	67	23	64	1	0	0
50	12156	24006	35940	138	38	71	0	0	0
60	17232	34464	52124	229	56	119	0	0	0
70	23736	47554	70000	85	122	107	0	0	0
80	30960	61798	91776	115	92	107	0	0	0
90	39660	78298	115770	122	71	89	0	0	0
100	48398	98224	144596	125	122	156	0	1	0
110	59246	117040	175928	112	67	204	0	0	0
120	70376	139146	208132	154	120	131	0	0	0

$O(n!)$, the proposed algorithms result in an improvement in order of magnitude. The computer simulations show that the algorithm 6 for error correcting graph isomorphism problems has the same time complexity as graph isomorphism problem.

References

1. M. S. Abadir and J. Ferguson, "An improved layout verification algorithm," in *Proceedings of IEEE European International Conference on Design and Automation*, pp. 391–395, 1990.
2. 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.
3. 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.
4. 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.
5. R. J. Schalkoff, *Pattern Recognition: Statistical, Syntactical, and Neural Approaches.* New York: John Whiley, 1992.
6. 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.
7. R. C. Wilson and E. R. Hancock, "Structural matching by discrete relaxation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 6, pp. 634–648, 1997.
8. 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.
9. 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.
10. 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.
11. M. Kremar and A. P. Dhawan, "Application of genetic algorithms for graph matching," in *Proceedings of IEEE European International Conferences on Neural Networks (ICNN-94)*, pp. 3872–3876, 1994.
12. R. Gupta, S. A. Smolka, and S. Bhaskar, "On randomization in sequential and distributed algorithms," *ACM Computing Surveys*, vol. 26, no. 1, pp. 7–85, 1994.
13. G. Brassard and P. Bratley, *Algorithmics: Theory and Practice.* New York: Printce-Hall, 1988.
14. H. Beigy and M. R. Meybodi, "Randomized las vegas algorithms for graph isomorphism problem," tech. rep., Computer Engineering Departement, Amirkabir University of Technology, Tehran, Iran, 1998.