Solving Stochastic Shortest Path Problem Using Distributed Learning Automata

M. R. Meybodi H. Beigy

Soft Computing Laboratory
Computer Engineering Department
Amirkabir University of Technology
Tehran, Iran
meybodi@ce.aku.ac.ir beigy@ce.aku.ac.ir

Abstract

Distributed learning automata (DLA) is introduced and its application to shortest path problem in stochastic graph is described. The objective is to use DLA to find a policy that determines a path from a source node to a destination node with minimal expected cost.

Keywords: Distributed learning automata, learning automata, shortest path problem, stochastic graph

1 Introduction

Shortest path problem when the edge cost is deterministic can be solved using one of the many shortest path algorithms, such as Dijkstra, Floyd, and Warshall in polynomial time [1]. However, when the edge lengths are allowed to be random, the problem becomes considerably more difficult. Unlike the deterministic graph, stochastic graph are related to distinct models, interpretations and applications.

Recently, attention has been given to analysis of shortest path in such stochastic networks, and the following questions have been addressed.

1. distribution of the length of the shortest path
2. moments of the length of the shortest path
3. the probability that a given path is the shortest path in the graph

The analysis of some of the problems for general stochastic graphs has been attempted by Pritsker [2] and Frank [3]. Martin [4] has analyzed the problem using the unique arcs concept, and Sigal and et al.[5] have used the uniformly directed cuts in their analysis of shortest paths. Each one of the above four papers represents the required probabilistic quantities as multiple integrals. Numerical evaluation of these integrals quickly gets out of hand, even for small networks. Michandani [6] uses an entirely different approach that avoids the evaluation of multiple integrals, but his approach works only when the arc lengths are discrete random variables. Due to the computational difficulties in the exact computation of the distribution of the length of the shortest path, recently attention has shifted to Monte Carlo simulation of stochastic networks to obtain estimates of the required distributions. Simulation methods for estimating some of the quantities listed above are developed by Sigal and et al. [7] and Adlakha and Fishman [8][9].

In this paper an adaptive procedure based on the distributed learning automata for finding the shortest path in stochastic graph is presented. This problem is particularly useful in routing in transportation or communication networks and in scheduling. For example in transportation application the random attribute denotes a random travel time.

We consider a stochastic directed graph $G = (V, E, X)$ with $V = \{1, 2, \cdots, n\}$ being the set of nodes, $E \subset V \times V$ being the set of directed edges, and X being the statistics of the edge costs. In particular X_{ij} of each edge (i, j) is assumed to be a random variable. We assume that each edge (i, j) has a discrete positive random length X_{ij} taking values $X_{ij}^1 < X_{ij}^2 < \cdots < X_{ij}^{n_{ij}}$ with respective probabilities $q_{ij}^1, q_{ij}^2, \cdots, q_{ij}^{n_{ij}}$. In this paper we are interested in finding the sum of the minimum of these variables over the set of all source-destination paths in the graph.

The rest of the paper is organized as follows: The learning automata and distributed learning automata are introduced in section 2. Section 3 presents the proposed Algorithms. Simulation results and discussion are given in section 4. Section 5 concludes the paper.

2 Learning Automata and Distributed Learning Automata

2.1 Learning Automata

The automata approach to learning involves the determination of an optimal action from a set of allowable actions. An automaton can be regarded as an abstract object that has finite number of actions. It selects an action from its finite set of actions. This action is applied to a random environment. The random environment evaluates the applied action and gives a grade to the selected action of automata. The response from environment (i.e. grade of action) is used by automata to select its next action. By continuing this process, the automata learns to select an action with best grade. The learning algorithm used by automata to determine the selection of next action from the response of environment. 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*.

Variable structure learning automata is represented by triple $< \beta, \alpha, T >$, where β is a set of inputs actions, α is a set of actions, and T is learning algorithm. The learning algorithm is a recurrence relation and is used to modify the state probability vector.

It is evident that the crucial factor affecting the performance of the variable structure learning automata, is learning algorithm for updating the action probabilities. Various learning algorithms have been reported in the literature. Let α_i be the action chosen at time k as a sample realization from probability distribution $p(k)$. The linear reward-inaction algorithm (L_{R-I}) is one of the earliest learning schemes and its recurrence equation for updating action probability vector p is defined as

$$p_j(k+1) = \begin{cases} p_j(k) + a \times [1 - p_j(k)] & \text{if} \quad i = j \\ p_j(k) - a \times p_j(k) & \text{if} \quad i \neq j \end{cases} \tag{1}$$

if $\beta(k)$ is zero and p is unchanged if $\beta(k)$ is one. The parameter $0 < a < 1$ is called *step length* and determines the amount of increases (decreases) of the action probabilities.

In linear reward-ϵpenalty algorithm ($L_{R-\epsilon P}$) scheme the recurrence equation for updating p is defined as

$$p_j(k+1) = \begin{cases} p_j(k) + a \times [1 - p_j(k)] & \text{if} \quad i = j \\ \\ p_j(k) - a \times p_j(k) & \text{if} \quad i \neq j \end{cases} \quad \text{if} \quad \beta(k) = 0 \tag{2}$$

$$p_j(k+1) = \begin{cases} p_j(k) \times (1 - b) & \text{if} \quad i = j \\ \\ \frac{b}{r-1} + p_j(k)(1 - b) & \text{if} \quad i \neq j \end{cases} \quad \text{if} \quad \beta(k) = 1 \tag{3}$$

The parameters $0 < a < 1$ and $0 < b \ll a$ represent *step lengths* and r is the number of actions for learning automata. The a and b determine the amount of increase and decreases of the action probabilities, respectively. If the a equals to b the recurrence equations (2) and (3) is called *linear reward penalty*(L_{R-P}) algorithm.

For more information about the theory and applications of learning automata refer to [10][11][12][13].

2.2 Distributed Learning Automata

Distributed learning automata (DLA) is a network of LA which collectively cooperated to solve a particular problem. The number of actions for a particular LA in DLA is equal to the number of LA's that are connected to this LA. Selection of an action by an LA in the network activate one LA corresponding to this action. For example in figure 1, every automata has two actions. Selection of action α_2 by LA_1 will activate automata LA_3. Activated LA_3 choose one of its action which results in activating one of the LA connected to LA_3 corresponding to the selected action. At any time only one of the automata in the network will be active. Formally, a distributed learning automata can be defined by a graph $DLA = (V, E)$, where the set $V = \{LA_1, LA_2, \cdots, LA_n\}$ is the set of learning automata, where n is the number of LA's in DLA, and $E \subset V \times V$ is the set of edges in the graph. The edge (i, j) shows the action j of the automata LA_i. In other words, LA_j is activated when action j of automata LA_i is selected. The number of actions for a particular automata LA_k (for $k = 1, 2, \cdots, n$) is equal to the out degree of that node.

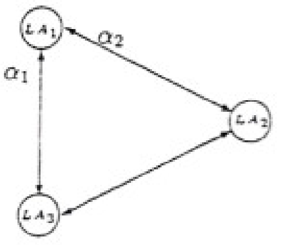


Figure 1:

3 The Proposed Algorithm

First, a network of learning automata which is isomorphic to the input graph is created. In this network each node is a learning automata and each outgoing edge of this node is one of the actions of this learning automata. For example, for graph of figure 2.a the network of learning automata of figure 2.b is created. Selection of one of these actions causes one of LA, LA_2, LA_3 or LA_4 be activated. The stochastic graph plays the role of random environment for the distributed learning automata. Feedback connection of environment and DLA is shown in figure 3. The output of DLA is a sequence of actions that represent a particular path in the stochastic graph.

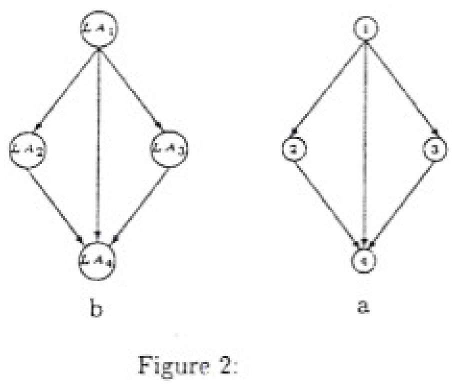


Figure 2:

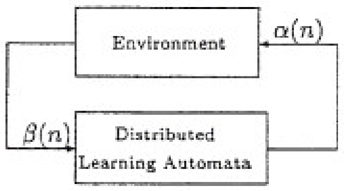


Figure 3: Feedback connection of DLA and random environment

The environment uses the length of this path to produce its response. This response, depending on whether is *favorable* or *unfavorable*, causes the actions along this particular path be *rewarded* or *penalized*.

Now, we give a general description of algorithm for finding the shortest path in stochastic graph. In the first step, the source LA (corresponding to the source node in the input graph) chooses one of its actions (as a sample realization of its action probability vector). This action activate the LA on the other end of the edge corresponding to the chosen action. This automata using its action probability vector will activate another automata. The process of selection of an action and activating an LA is repeated until the destination LA (corresponding to the destination node in the input graph) is reached or for some reason moving along the edges of the graph is not possible or the number of visited nodes exceeds the number of nodes in the graph. For example in figure 2 in which source node is 1 and destination node is 4, at first LA_1 chooses one of its actions. Assume as a result of choosing action by LA_1, LA_3 become active, automata LA_3 activate LA_4 which is the destination automata.

After the destination node is reached, the length of the traversed path is computed and then compared with a quantity called *dynamic threshold* and depending on the result of comparison all the LAs (except the destination LA) along the traversed path update their action probabilities. Updating is done in direction from destination to source. If the length of traversed path is less than or equal to the dynamic threshold then all the automata along that path receive reward and receive penalty if the length of traversed path is greater than dynamic threshold or the destination node is not reached. Reward and penalty to actions are given according to the L_{R-I} learning algorithm.

The process of traveling from source LA to destination LA is repeated until the stopping condition is reached which at this point the path last traveled is the path which has the minimum expected length among all the paths from source to destination. The dynamic threshold at time $k > 1$ is defined as

$$\Delta T = \frac{1}{k-1}\sum_{i=1}^{k-1} l_i \tag{4}$$

where l_i is the length of traversed path at iteration i. The algorithm stops if the product of the probability of choosing the edge of the traversed path, called *path probability*, is greater than a certain threshold.

For the sake of simplicity, we use notations shown in figure 4. The variables *sourceNode* and *destinationNode* denote the source and destination nodes, respectively. The probability vector for LA_j is shown by $p^j = (p_1^j, p_2^j, \cdots, p_{r_j}^j)$. p_m^j denotes the probability of selecting action α_m that is, the edge from node j to node m. For simplicity in the presentation of algorithms, the actions of an automata are shown by the indices of adjacent nodes to the node corresponding to that action. For example, the set of actions of automata LA_2 in figure 4 is represented by $\{\alpha_5, \alpha_6\}$. Variables *pathLen* and *pathCost* represent the length and cost of the generated path which is stored in array *path*, respectively and the variable *path*[*k*] denotes the *k*th node in the generated path. In algorithms of figure 3, the statement *node.getAdjNode*() selects an action from LA_{node} based on the action probability vector p^{node} and statement $node \leftarrow nextNode$ activates the automata $LA_{nextNode}$. The proposed algorithm is given in figure 3.

In the first step of the algorithm, the LA_{source} chooses one of its actions, say *nextNode*. Selection of this action activates the $LA_{nextNode}$. $LA_{nextNode}$ using its action probability vector will activate another automata. The process of selection of an action and activating an LA is repeated until the $LA_{destination}$ is reached or for some reason moving along the edges of the graph is not possible or the number of

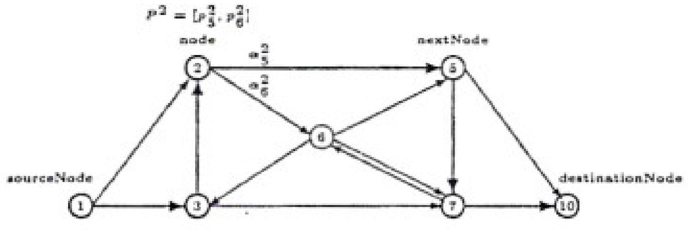


Figure 4:

visited nodes exceeds the number of nodes in the graph. After the destination node is reached, the length of the traversed path is computed and then compared with dynamic threshold and depending on the result of comparison all the LA's (except the $LA_{destination}$) along the traversed path update their action probabilities. Updating is done in the direction of destination to source. If the length of traversed path is less than or equal to the dynamic threshold then all the automata along the path receive reward and otherwise receive penalty according to the L_{R-I} learning scheme. In the rest of this paper, we call this algorithm, algorithm 1.

```
procedure SSPP
    Input: Graph $G = (V, E)$, sourceNode, destinationNode
    Output: path , pathLen, pathCost
    begin
        Construct the DLA $DLA = (V, E)$ from graph $G$
        repeat
            k ← pathCost ← 0
            pathLen ← 1                    // length of the generated path
            node ← path [1]← sourceNode         // Start from sourceNode
            while node ≠ destinationNode and pathLen < $n$ do
                nextNode ← node.getAdjNode ()      // Select an adjacent node prescribed by $LA_{node}$
                pathLen ← pathLen + 1     // increment length of the path
                path [pathLen]← nextNode    // Add the selected edge to the generated path
                pathCost ← pathCost + edgeCost (node, nextNode)     // Sample the cost of an edge (node, nextNode)
                node ← nextNode      // Activate automaton $LA_{nextNode}$
            end while
            k ← k + 1
            if pathCost ≤ prevPathCost and path[pathLen]=destinationNode then
                for m ← pathLen-1 downto 1 do
                    reward action path[m+1] of $LA_{path[m]}$ // Traverse the path from destinationNode to sourceNode
                end for
            end if
            $prevPathCost \leftarrow prevPathCost + \frac{1}{k}[pathCost - prevPathCost]$
        until probability of selected path ≥ threshold
end SSPP
```

Figure 5: Algorithm 1

This algorithm can be improved in several ways some of which are described below.

- In algorithm 1, number of actions for every automata is fixed and does not vary as algorithm proceeds. This may cause the algorithm to traverse paths from source to destination that may contain loops. Existence of loops in the traversed path (if the cost is positive) excludes that path from being the optimal path. If we disallow loops in the paths being traversed, the number of paths sampled inorder to find the optimal path will be reduced leading to lower computation time. This is done by allowing the algorithm to meet every node along a path being traversed at most once. To implement this, when an LA chooses action k from the list of its actions, at the same time all the unactivated LA's will disable action k (but not removed) in their list of actions. When traversing back from destination node to the source node, all the disabled actions will be enabled. Algorithm 1 in which this modification is made is called algorithm 2.

- In algorithms 1 and 2, the learning algorithms residing in different LA's use the same learning rate which remains fixed during the execution of the algorithm. This gives equal importance to all the nodes along the path being traversed, which may not be appropriate. Nodes which are closer to the source node should be given higher degree of participation in the optimal path (higher probability of being part of the optimal path)than those which are far from the source node. Therefore, the amount of the reward given to an action taken at a particular time must increases as we approaching the destination node. To achieve this, we halve the step length as moving from one node to another when we traversing back toward the source node. Algorithm 2 in which this modification is made will be called algorithm 3.

- Updating schemes for step lengths can be different. Another way to update the step length is given by equation (5). Here the step lengths are dependent on the action probabilities. Algorithm 2 in which this modification is made will be called algorithm 4.

$$a_{path[k]} = \frac{a_{path[k-1]}}{p_{path[k]}^{path[k-1]}} \qquad \text{for } k > 1 \tag{5}$$

- It seems reasonable that for computation of the cost of a path to use the mean of all the samples taken from an edge rather than the sample taken at an iteration for that edge. This modification, according to the central limit theorem [14], allows the algorithm to have an estimate of the cost of traversed path which is closer to its mean value. Algorithm 2 in which this modification is made is called algorithm 5.

- Different combination of above mentioned algorithms can be used. We examine the combination of algorithms 3 and 5 and call it algorithm 6.

4 Experiments

To study the feasibility of the proposed algorithms, experiments are conducted on five different stochastic graphs given in figures 15 through 18. Graphs which are shown in figures 17 through 18 are borrowed from [15].

- Graph 1 which is shown in figure 15 is a graph with 6 nodes, 10 arcs, $sourceNode = 1$, and $destinationNode = 6$. Edge cost distribution is given in table 7. For instance, edge (1,2) has length 7.0, 7.3, or 9.4 with probabilities 0.2, 0.5, or 0.3, respectively.

- Graph 2 which is shown in figure 16 is a graph with 4 nodes, 5 arcs, $sourceNode = 1$, and $destinationNode = 4$. Edge cost distribution is given in table 8.

- Graph 3 which is shown in figure 17 is a graph with 10 nodes, 23 arcs, $sourceNode = 1$, and $destinationNode = 10$. Edge cost distribution is given in table 9.

- Graph 4 which is shown in figure 17 is a graph with 10 nodes, 23 arcs, $sourceNode = 1$, and $destinationNode = 10$. Edge cost distribution is given in table 10.

- Graph 5 which is shown in figure 18 is a graph with 15 nodes, 42 arcs, $sourceNode = 1$, and $destinationNode = 15$. Edge cost distribution is given in table 11.

In all the simulations presented in this paper, the updating scheme for vector probability is L_{R-I}, and each algorithm terminates when the probability of traversed path is 0.9 or 300,000 paths are traversed. Each algorithm is tested on five different graphs and the results for each algorithm are summarized in two tables. Every figure in this is the average value over 100 runs. The first column of the first table for each graph includes the average number of iteration required by the algorithm over the runs which converge. The second column shows the percentage of 100 runs converge to the solution after 300,000 iterations to the solution, that is the path with minimum expected cost. The second table for each algorithm, shows the total number of samples taken form all the edges in the graph and also the total number of samples taken from the edges along the shortest path. The results of simulations for different graphs are summarized in table 1 through 6. From the result of the simulations, the following points can be made.

1. Simulation results show that for all graph except graph 3 the number of converged runs increases as the learning parameter a decreases. For instance, in table 1 for graph 1 whereas 100 of the runs converge when $a = 0.001$, this percentage decreases to 98 when $a = 0.151$.

2. Simulation results show that for algorithm 2 the average number of iterations required to reach the termination criteria is lesser comparing the algorithm 1, but percentage of the number of runs converge is higher. This is due to the fact that algorithm 2 does not traverse paths which contain loops, and as a result the number of sampled paths is smaller.

3. Algorithm 3 improves the rate of convergence because as we approaching destination node, the action probability vectors are updated more conservatively. On the other hand, the average number of iterations for this algorithm is higher. The reason for the increase in the average number of iterations is that the nodes closer to the destination node use smaller step length.

4. The updating scheme for step length in algorithm 4 results in degradation in the performance of the algorithm. One reason for such degradation is that there is no dependency between the step lengths and action probabilities.

5. It can be noted that for large value of learning parameter a all algorithms perform the same. The figures 7, 7 and 8 show the average iterations for graphs 1, 2 and 5 with respect to step length parameter a. These figures point out that the average iterations required by the algorithms for finding the optimal path is heavily dependent on the input graph and there is no general ordering according to which the algorithms can be ordered in terms of their performance.

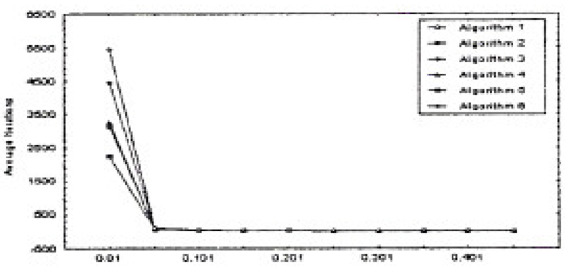


Figure 6: Average number of iterations for graph 1

6. Figure 9 shows the number of converged runs over 100 runs for graph 1 for different algorithms. According to this figure, algorithms 3 and 6 are not very sensitive to step length parameter a in terms of number of runs converged.

7. Probability of optimal path (POP) at each iteration of algorithm is defined as the product of probability of edges along the optimal path. The plots of average probability of optimal path over the converged runs out of 100 runs for all graphs and different algorithms are given in figures 10 through 14. Higher slop of POP for an algorithm indicates that a path selected in a particular

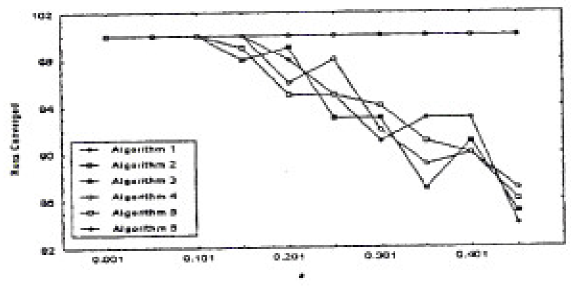


Figure 9: Probability of optimal path for graph 1

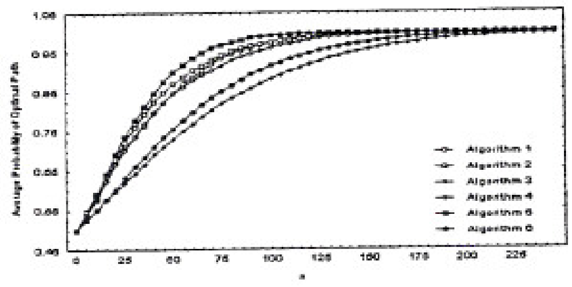


Figure 10: Probability of optimal path for graph 1

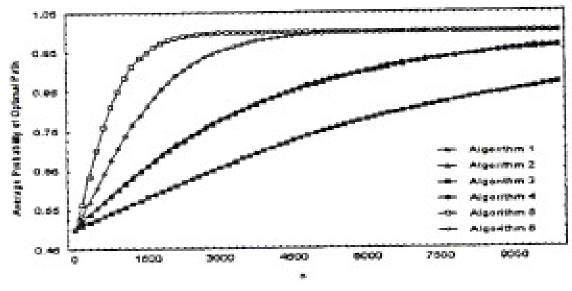


Figure 11: Probability of optimal path for graph 2

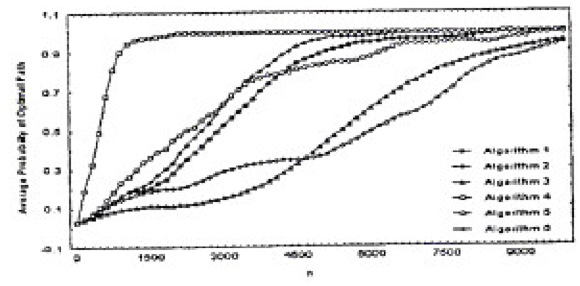


Figure 12: Probability of optimal path for graph 3

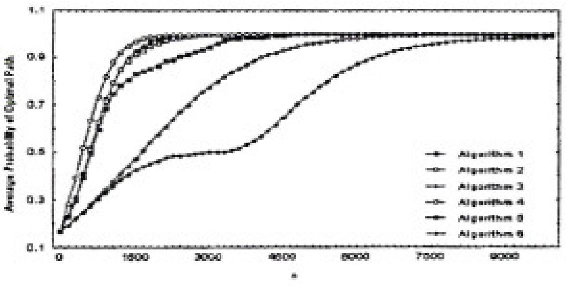


Figure 13: Probability of optimal path for graph 4

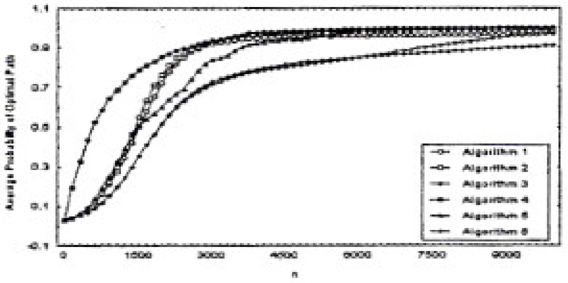


Figure 14: Probability of optimal path for graph 5

Table 1: The simulation results of Algorithm 1
a) Average number of iterations and runs converged

a	Graph 1		Graph 2		Graph 3		Graph 4		Graph 5	
	Average Iterations	Runs Converged	Average Iterations	Runs Converged	Average Iterations	Runs Converged	Average Iterations	Runs Converged	Average Iterations	Runs Converged
0.001	3158.8	100	11943	100	37710	12	10706	100	21027	100
0.051	63	100	211.7	100	379	31	214.7	85	425	66
0.101	33.88	100	91.46	100	159.4	20	93.7	80	189	33
0.151	20.46	100	57.46	93	91.1	17	46.6	57	86.2	26
0.201	17.57	98	37.6	86	59	12	29.8	60	57.4	21
0.251	11.7	95	26.76	90	32.4	16	23.65	55	35.6	14
0.301	9.43	91	15.9	78	27.3	12	18.9	46	17.1	15
0.351	7.45	93	15	73	18.5	13	13.9	49	18.25	16
0.401	7.08	93	13	66	16.3	10	10.5	36	12.7	12
0.451	6.01	84	8.52	67	12.7	3	9.8	43	12.5	4

b) Average samples taken from graph and optimal path

a	Graph 1		Graph 2		Graph 3		Graph 4		Graph 5	
	Total Samples	Shortest Path	Average Samples	Shortest Path	Average Samples	Shortest Path	Average Samples	Shortest Path	Average Samples	Shortest Path
0.001	7376	5004	23890	17738	62525	10692	41256	19109	88574	31038
0.051	153.5	100	427.4	1288	1062	268	882.6	318	1836	563
0.101	85.6	53	186.9	124	397.8	84	384.7	138	769.6	173
0.151	53.6	33	121.3	70	284.9	61	237.5	65	419	81
0.201	47.9	27	77.9	44	200.5	42	159.7	42	289.2	53
0.251	35.7	19	57.4	34	144.8	28	130	32	192	25
0.301	29.6	16	36.4	20	113.5	21	110	24	146.3	22
0.351	24.7	14	32.7	18	94.2	16	82.4	20	113	16
0.401	23.4	14	30.5	14	73.6	11	71.5	15	100	12
0.451	22.1	10	21.5	10	69	19	68.4	15	85.9	9

Table 2: The simulation results of Algorithm 2
a) Average number of iterations and runs converged

a	Graph 1		Graph 2		Graph 3		Graph 4		Graph 5	
	Average Iterations	Runs Converged	Average Iterations	Runs Converged	Average Iterations	Runs Converged	Average Iterations	Runs Converged	Average Iterations	Runs Converged
0.001	3132.8	100	12094	100	36807	18	10054	100	20276	100
0.051	58.5	100	192.3	100	429.4	33	196.9	87	402.9	64
0.101	31.9	100	93.4	100	139.4	39	85.6	69	175.3	34
0.151	20.1	98	53.4	94	95.2	12	50.5	66	84.7	29
0.201	14.5	99	35.6	92	62.2	12	34.3	66	57.9	20
0.251	12.6	93	22.7	81	44.2	13	24.4	63	42.8	18
0.301	8.7	93	18.9	77	27.3	14	16.9	48	26.9	19
0.351	7.4	87	14.5	77	21.2	10	13.8	50	18.9	12
0.401	7.5	91	11.1	77	13.8	8	11.2	47	17.6	9
0.451	6.1	85	9.1	74	15.7	11	11	40	12.9	9

b) Average samples taken from graph and optimal path

a	Graph 1		Graph 2		Graph 3		Graph 4		Graph 5	
	Total Samples	Shortest Path	Average Samples	Shortest Path	Average Samples	Shortest Path	Average Samples	Shortest Path	Average Samples	Shortest Path
0.001	7311	5004	24193	17968	63684	11576	37763	18144	83532	32556
0.051	141.7	100	388	270	1090	297	736	297	1814	494
0.101	80.2	53	190.8	128	409.4	92	376.1	122	765.7	160
0.151	53.8	33	108.7	66	274	54	225.6	70	414.6	90
0.201	36.7	27	75.7	44	176.1	36	169	52	271.8	50
0.251	39.5	19	49.6	26	140.4	26	118	37	205.8	32
0.301	29.1	16	43.1	24	108.3	22	94	25	142	24
0.351	25.8	14	32.9	18	95.3	19	78.8	21	111.4	14
0.401	24.7	14	27.6	14	78.1	12	68.1	18	102	14
0.451	22.8	10	22.5	12	68.4	9	62.4	14	84	12

Table 3: The simulation results of Algorithm 3
a) Average number of iterations and runs converged

a	Graph 1		Graph 2		Graph 3		Graph 4		Graph 5	
	Average Iterations	Runs Converged	Average Iterations	Runs Converged	Average Iterations	Runs Converged	Average Iterations	Runs Converged	Average Iterations	Runs Converged
0.001	5454.5	100	23550	100	66340	100	34940	100	63274	100
0.051	104.5	100	406.4	100	1139	64	718	100	1218.5	91
0.101	52.9	100	206.5	100	512.7	48	339	99	616.4	77
0.151	35.3	100	130.1	99	308.3	58	230.3	95	335.8	74
0.201	26.9	100	87.4	99	224.3	38	169.8	89	185.3	68
0.251	20.5	100	67.7	98	155.8	41	139.4	93	176	66
0.301	18	100	55.5	98	122	40	105.7	86	127.6	63
0.351	15.4	100	42.7	96	106.8	34	85.9	75	87	67
0.401	14	100	35.4	91	84.4	35	70.3	81	75.2	48
0.451	12.3	100	33.1	86	66.2	37	64.5	70	66.3	49

b) Average samples taken from graph and optimal path

a	Graph 1		Graph 2		Graph 3		Graph 4		Graph 5	
	Total Samples	Shortest Path	Average Samples	Shortest Path	Average Samples	Shortest Path	Average Samples	Shortest Path	Average Samples	Shortest Path
0.001	12721	8648	47105	35029	21404	68190	117805	63255	220013	125001
0.051	248.4	168	816.8	581.4	3136.2	904.6	2416.5	1245.5	4094	1870
0.101	128.7	86.2	416.9	288.2	1392.2	346.2	1181.2	572	2012	810.9
0.151	87	58.7	266.2	173.5	904.4	258.4	777.2	367.8	1148.9	446.9
0.201	67.9	45	179.7	117.4	612.9	142.2	615.5	255.6	676.3	227.6
0.251	52.4	35.5	139.2	89.1	479	115	477.5	216.6	609	196.2
0.301	47.8	30.5	115.1	7.4	347.6	86	400.5	153.4	471.7	141.2
0.351	41.1	27.2	97.3	59.1	296.2	66.8	331.1	123.7	365.2	110.3
0.401	37.6	24.8	73.6	44.3	252.6	57.7	251.7	106.7	351.5	82.5
0.451	34.1	21.8	70.3	41.7	215.6	51	231	85.6	291.7	66

Table 4: The simulation results of Algorithm 4
a) Average number of iterations and runs converged

a	Graph 1		Graph 2		Graph 3		Graph 4		Graph 5	
	Average Iterations	Runs Converged	Average Iterations	Runs Converged	Average Iterations	Runs Converged	Average Iterations	Runs Converged	Average Iterations	Runs Converged
0.001	3272	100	11896	100	7301	100	9867	100	16241	100
0.051	64	100	207.9	100	174.4	40	174.9	90	220.3	42
0.101	30.6	100	101.8	98	72.3	34	74.8	81	97.9	27
0.151	19.7	100	58.7	98	46.5	22	44.9	69	60.1	28
0.201	17	96	37	92	31.7	14	30.5	53	36.4	19
0.251	11.8	98	25.3	82	20.7	17	16.9	57	26.9	20
0.301	9.4	92	18.7	80	21	9	14.8	56	17.8	6
0.351	8.4	89	13.5	81	14	14	11.6	43	14.4	10
0.401	6.8	90	13.1	76	12	9	10.2	50	12.8	8
0.451	5.6	87	7.3	71	9.9	7	7.9	38	9.1	8

b) Average samples taken from graph and optimal path

a	Graph 1		Graph 2		Graph 3		Graph 4		Graph 5	
	Total Samples	Shortest Path	Average Samples	Shortest Path	Average Samples	Shortest Path	Average Samples	Shortest Path	Average Samples	Shortest Path
0.001	7607	5269	23797	17695	24046	11693	31357	17188	53474	31949
0.051	155	103.8	419.8	289.3	952.7	256.3	631	258.3	1152.5	264.8
0.101	76.4	50.9	205.9	125.2	322.3	83	295	111.7	401.9	78.3
0.151	51.9	33.5	121.2	75.7	240.6	49.6	175	61.3	253.5	56.3
0.201	46.8	28.8	77.3	47.5	131.9	25.8	143.9	40.1	159.5	32.6
0.251	33.7	21.3	54.9	30	92.5	21.7	86.2	27	121.5	24
0.301	29.8	16.9	40.7	23.2	92.1	16	74.9	24.8	102.8	14.1
0.351	26.6	15.3	30.3	16.9	63.7	12.9	61	17.2	80.6	10.1
0.401	23	13.4	31.1	17	60.9	9.8	53.9	18.1	73.7	9.7
0.451	20.3	11.3	19.9	10.6	48.4	8.6	43.6	11.8	61.6	8.2

Table 5: The simulation results of Algorithm 5
a) Average number of iterations and runs converged

a	Graph 1		Graph 2		Graph 3		Graph 4		Graph 5	
	Average Iterations	Runs Converged	Average Iterations	Runs Converged	Average Iterations	Runs Converged	Average Iterations	Runs Converged	Average Iterations	Runs Converged
0.001	2235	100	2230	100	55614	9	19770	98	26302	100
0.051	50	100	57.9	100	482.8	18	189.3	65	478	43
0.101	25.6	100	33.2	99	165.2	18	73.4	58	139.3	26
0.151	17.9	99	24.4	96	72.4	14	23.6	58	81.9	15
0.201	14	95	23.9	96	44.9	16	31.8	53	59.2	20
0.251	10.8	95	17.2	89	32.4	11	22.4	60	35.3	15
0.301	9.7	94	13.1	91	22.9	9	17.9	47	23.6	21
0.351	6.8	91	14.4	90	18.8	10	14.7	45	20.6	11
0.401	6.5	90	12.1	87	14.9	7	13.4	45	17.3	12
0.451	5.4	86	9.4	82	10.9	17	10.2	34	12.9	8

b) Average samples taken from graph and optimal path

a	Graph 1		Graph 2		Graph 3		Graph 4		Graph 5	
	Total Samples	Shortest Path	Average Samples	Shortest Path	Average Samples	Shortest Path	Average Samples	Shortest Path	Average Samples	Shortest Path
0.001	5219	3522	4464	3232	101566	20390	68612	35540	103965	48325
0.051	119	81	118	82.5	1186	284.9	840	278.7	1824	543.7
0.101	60	42.7	68	43.5	459	125.2	352	108.2	651	166.6
0.151	43	30.5	50	32.3	234	53.2	203	62.4	344	64.2
0.201	35	24.3	50	30.9	166	37.9	135	43.1	213	47.5
0.251	27	19.4	36	21.3	114	24.1	104	34.9	153	26.5
0.301	26	17.8	28	17.2	83	17.4	81	26.2	113	20.8
0.351	18	13.3	30	18.2	71	14.2	71	21.7	84	14.3
0.401	17	12.8	26	16	58	11.5	52	18.7	70	12.5
0.451	16	11	20	11.8	44	11.7	47	15.4	56	11.2

Table 4: The simulation results of Algorithm 4
a) Average number of iterations and runs converged

a	Graph 1		Graph 2		Graph 3		Graph 4		Graph 5	
	Average Iterations	Runs Converged	Average Iterations	Runs Converged	Average Iterations	Runs Converged	Average Iterations	Runs Converged	Average Iterations	Runs Converged
0.001	3272	100	11896	100	7301	100	9867	100	16241	100
0.051	64	100	207.9	100	174.4	40	174.9	90	220.3	42
0.101	30.6	100	101.8	98	72.3	34	74.8	81	97.9	27
0.151	19.7	100	58.7	98	46.5	22	44.9	69	60.1	28
0.201	17	96	37	92	31.7	14	30.5	53	36.4	19
0.251	11.8	98	25.3	82	20.7	17	16.9	57	26.9	20
0.301	9.4	92	18.7	80	21	9	14.8	56	17.8	6
0.351	8.4	89	13.5	81	14	14	11.6	43	14.4	10
0.401	6.8	90	13.1	76	12	9	10.2	50	12.8	8
0.451	5.6	87	7.3	71	9.9	7	7.9	38	9.1	8

b) Average samples taken from graph and optimal path

a	Graph 1		Graph 2		Graph 3		Graph 4		Graph 5	
	Total Samples	Shortest Path	Average Samples	Shortest Path	Average Samples	Shortest Path	Average Samples	Shortest Path	Average Samples	Shortest Path
0.001	7607	5269	23797	17595	24046	11693	31357	17188	53474	31949
0.051	155	103.8	419.8	289.3	952.7	256.3	631	258.3	1152.5	264.8
0.101	76.4	50.9	205.9	125.2	322.3	83	295	111.7	401.9	78.3
0.151	51.9	33.5	121.2	75.7	240.6	49.6	175	61.3	253.5	56.3
0.201	46.8	28.8	77.3	47.5	131.9	25.8	143.9	40.1	159.5	32.6
0.251	33.7	21.3	54.9	30	92.5	21.7	86.2	27	121.5	24
0.301	29.8	16.9	40.7	23.2	92.1	16	74.9	24.8	102.8	14.1
0.351	26.6	15.3	30.3	16.9	63.7	12.9	61	17.2	80.6	10.1
0.401	23	13.4	31.1	17	60.9	9.8	53.9	18.1	73.7	9.7
0.451	20.3	11.3	19.9	10.6	48.4	8.6	43.6	11.8	61.6	8.2

Table 5: The simulation results of Algorithm 5
a) Average number of iterations and runs converged

a	Graph 1		Graph 2		Graph 3		Graph 4		Graph 5	
	Average Iterations	Runs Converged	Average Iterations	Runs Converged	Average Iterations	Runs Converged	Average Iterations	Runs Converged	Average Iterations	Runs Converged
0.001	2235	100	2230	100	55614	9	19770	98	26302	100
0.051	50	100	57.9	100	482.8	18	189.3	65	478	43
0.101	25.6	100	33.2	99	165.2	18	73.4	58	139.3	26
0.151	17.9	99	24.4	96	72.4	14	23.6	38	81.9	15
0.201	14	95	23.9	96	44.9	16	31.8	53	59.2	20
0.251	10.8	95	17.2	89	32.4	11	22.4	60	35.3	15
0.301	9.7	94	13.1	91	22.9	9	17.9	47	23.6	21
0.351	6.8	91	14.4	90	18.6	10	14.7	45	20.6	11
0.401	6.5	90	12.1	87	14.9	7	13.4	45	17.3	12
0.451	5.4	86	9.4	82	10.9	17	10.2	34	12.9	8

b) Average samples taken from graph and optimal path

a	Graph 1		Graph 2		Graph 3		Graph 4		Graph 5	
	Total Samples	Shortest Path	Average Samples	Shortest Path	Average Samples	Shortest Path	Average Samples	Shortest Path	Average Samples	Shortest Path
0.001	5219	3522	4464	3232	101566	20390	68612	35540	103965	48325
0.051	119	81	118	82.5	1186	284.9	840	278.7	1824	543.7
0.101	60	42.7	68	43.5	459	125.2	352	108.2	651	166.6
0.151	43	30.5	50	32.3	234	53.2	203	62.4	344	64.2
0.201	35	24.3	50	30.9	166	37.9	135	43.1	213	47.5
0.251	27	19.4	36	21.3	114	24.1	104	34.9	153	25.5
0.301	26	17.8	28	17.2	83	17.4	81	26.2	113	20.8
0.351	18	13.3	30	18.2	71	14.2	71	21.7	84	14.3
0.401	17	12.8	26	16	58	11.5	52	18.7	70	12.5
0.451	16	11	20	11.8	44	11.7	47	15.4	56	11.2

Table 6: The simulation results of Algorithm 6
a) Average number of iterations and runs converged

a	Graph 1		Graph 2		Graph 3		Graph 4		Graph 5	
	Average Iterations	Runs Converged	Average Iterations	Runs Converged	Average Iterations	Runs Converged	Average Iterations	Runs Converged	Average Iterations	Runs Converged
0.001	4447	100	4437	100	18686	51	50217	100	77186	100
0.051	90.9	100	103	100	1294.5	39	1009	91	1576	93
0.101	46.2	100	55.1	100	587	31	435.1	73	568.9	81
0.151	30.6	100	43.1	100	266.2	31	295.9	67	306.7	67
0.201	24.1	100	35.3	99	186.4	28	166.4	60	209.4	68
0.251	19.5	100	30.6	98	145.9	28	125.3	62	159.4	67
0.301	18.9	100	27.6	98	133.4	23	103.2	68	110.3	68
0.351	14.9	100	24.8	98	81.3	28	78.6	62	84.9	57
0.401	13	100	21.9	93	81.4	31	59.7	56	73.9	53
0.451	12	100	22.3	92	67.7	20	54.9	52	65.5	49

b) Average samples taken from graph and optimal path

a	Graph 1		Graph 2		Graph 3		Graph 4		Graph 5	
	Total Samples	Shortest Path	Average Samples	Shortest Path	Average Samples	Shortest Path	Average Samples	Shortest Path	Average Samples	Shortest Path
0.001	10392	7006	8878	6454	325030	65152	163184	77186	265575	163689
0.051	217.8	145.3	210	147.3	2391.8	769	3256.7	1392.9	5230	2571
0.101	113.1	75.3	114.2	78.3	1561.6	304.9	1518.1	574.8	1945	829
0.151	76.3	51.3	90.1	60.5	868.7	179.6	991.8	366.9	1099.6	370.6
0.201	61.3	40.7	74.6	47.5	617.2	112.8	564.5	211.5	790.8	256.6
0.251	50.3	33.7	66	40.4	455	93	451.3	173.4	586.6	193.3
0.301	44.9	28.6	58.9	35.7	370.4	69.1	373	144.7	476.8	153.7
0.351	40.2	26.8	53.3	33.6	280.9	61.4	289.7	103.7	390	112.9
0.401	35.8	22.8	47.3	29.2	253.8	54.2	238.2	81.8	351.7	90.2
0.451	33.3	21.4	47.2	30.6	214.8	38.9	224.3	77.1	307.8	71.2

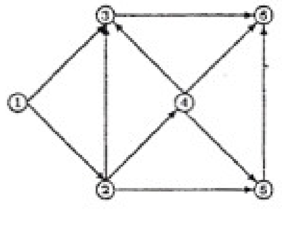


Figure 15: Graph 1

Table 7: Weight distribution of graph 1 (figure 15)

Edge	Lengths			Probabilities		
(1,3)	1	10		0.8	0.2	
(1,2)	4	10	100	0.2	0.4	0.4
(2,3)	10	50	100	0.1	0.8	0.1
(2,4)	20	100		0.1	0.9	
(2,5)	10	90		0.1	0.9	
(3,6)	1	100		0.9	0.1	
(4,3)	10	100		0.2	0.8	
(4,5)	50	50		0.5	0.5	
(4,6)	100	100		0.2	0.8	
(5,6)	1	10	100	0.2	0.2	0.6

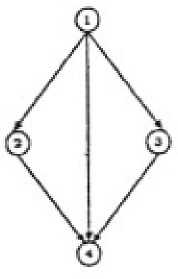


Figure 16: Graph 2

Table 8: Weight distribution of graph 2(figure 16)

Edge	Lengths				Probabilities			
(1,2)	6	14	15		0.3	0.45	0.25	
(1,3)	2	9	10		0.1	0.7	0.2	
(2,4)	4	11	18	19	0.25	0.1	0.3	0.35
(2,4)	6	13	15		0.15	0.3	0.55	

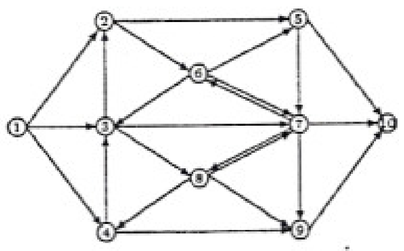


Figure 17: Graph 3

Table 9: Weight distribution of graph 3 (figure 17)

Edge	Lengths					Probabilities				
(1,2)	7.0	7.3	9.4			0.2	0.5	0.3		
(1,3)	2.5	3.5	8.2			0.5	0.4	0.1		
(1,4)	4.2	4.8	6.1			0.2	0.3	0.5		
(2,5)	2.6	3.1	5.5	8.8	9.0	0.1	0.2	0.4	0.2	0.1
(2,6)	5.8	7.0	9.5			0.3	0.3	0.4		
(3,2)	1.5	7.3				0.4	0.6			
(3,7)	6.5	7.4	7.5			0.4	0.5	0.1		
(3,8)	5.9	7.2	9.8			0.6	0.3	0.1		
(4,3)	2.1	3.2	8.5	9.8		0.3	0.2	0.3	0.2	
(4,9)	8.9	9.6				0.7	0.3			
(5,7)	3.2	4.8	6.7			0.2	0.2	0.6		
(5,10)	6.3	6.9				0.5	0.5			
(6,3)	6.6	8.5	9.8			0.8	0.1	0.1		
(6,5)	0.6	1.5	3.9	5.8		0.1	0.4	0.3	0.2	
(6,7)	0.2	4.8				0.4	0.6			
(7,6)	6.1	6.3	8.5			0.2	0.3	0.5		
(7,8)	1.6	1.8	4.0	5.2		0.2	0.3	0.3	0.2	
(7,10)	1.6	3.4	7.1			0.1	0.5	0.4		
(8,4)	9.0	9.6				0.5	0.5			
(8,7)	2.1	4.6	8.5			0.3	0.4	0.3		
(8,9)	1.7	4.9	5.3	6.5		0.1	0.4	0.4	0.1	
(7,9)	0.3	3.0	5.0			0.1	0.4	0.5		
(9,10)	0.6	1.2	5.4	6.6		0.1	0.1	0.3	0.5	

Table 10: Weight distribution of graph 4(figure 17)

Edge	Lengths				Probabilities			
(1,2)	3	5.3	7.4	9.4	0.2	0.2	0.3	0.2
(1,3)	3.5	6.2	7.9	8.5	0.3	0.3	0.2	0.2
(1,4)	4.2	6.1	6.9	8.9	0.2	0.3	0.2	0.3
(2,5)	2.6	4.1	5.5	9.0	0.2	0.2	0.4	0.2
(2,6)	5.8	7.0	8.5	9.6	0.3	0.3	0.2	0.2
(3,2)	1.5	2.3	3.6	4.5	0.2	0.2	0.3	0.3
(3,7)	6.5	7.2	8.3	9.4	0.5	0.2	0.2	0.1
(3,8)	5.9	7.8	8.6	9.9	0.4	0.3	0.1	0.2
(4,3)	2.1	3.2	4.5	6.8	0.2	0.2	0.3	0.3
(4,9)	1.1	2.2	3.5	4.3	0.2	0.3	0.4	0.1
(5,7)	3.2	4.8	6.7	8.2	0.2	0.2	0.3	0.3
(5,10)	6.3	7.8	8.4	9.1	0.2	0.2	0.4	0.2
(6,3)	6.8	7.7	8.5	9.6	0.4	0.1	0.1	0.4
(6,5)	0.6	1.5	3.9	5.8	0.2	0.2	0.3	0.3
(6,7)	2.1	4.8	6.6	7.5	0.2	0.4	0.2	0.2
(7,6)	4.1	6.3	8.5	9.7	0.2	0.3	0.4	0.1
(7,8)	1.6	2.8	5.2	6.0	0.2	0.3	0.3	0.2
(7,10)	1.6	3.4	8.2	9.3	0.2	0.3	0.3	0.2
(8,4)	7.0	8.0	8.8	9.4	0.2	0.2	0.2	0.4
(8,7)	2.1	4.6	8.5	9.6	0.4	0.2	0.2	0.2
(8,9)	1.7	4.9	6.5	7.8	0.2	0.2	0.2	0.4
(7,9)	3.5	4.0	5.0	7.7	0.1	0.2	0.4	0.3
(9,10)	4.6	6.4	7.6	8.9	0.4	0.1	0.2	0.3

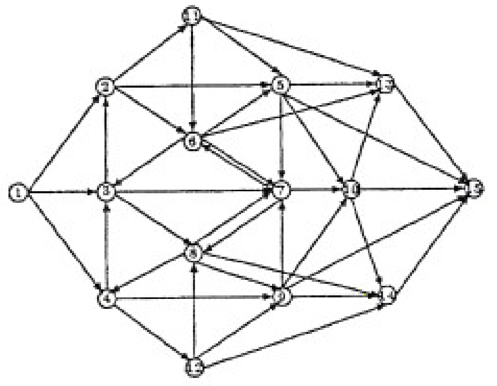
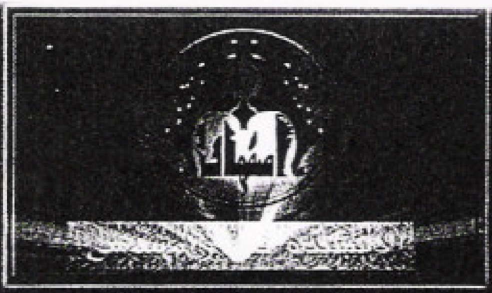


Figure 18: Graph 5

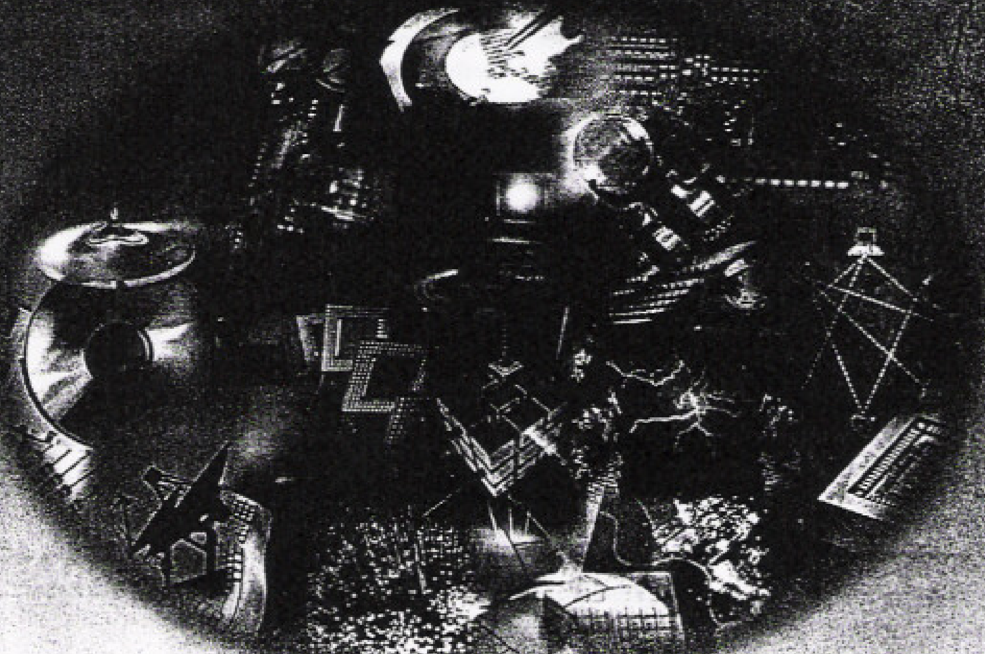
Table 11: Weight distribution of graph 5 (figure 18)

Edge	Lengths				Probabilities			
(1,2)	16	25	36		0.6	0.3	0.1	
(1,3)	21	24	25	39	0.5	0.2	0.2	0.1
(1,4)	11	13	26		0.4	0.4	0.2	
(2,11)	24	28	31		0.5	0.3	0.2	
(2,5)	11	30			0.7	0.3		
(2,6)	13	37	39		0.6	0.2	0.2	
(3,2)	11	20	24		0.6	0.3	0.1	
(3,7)	23	30	34		0.4	0.3	0.3	
(3,8)	14	23	34		0.5	0.4	0.1	
(4,3)	22	30			0.7	0.3		
(4,9)	35	40			0.6	0.4		
(4,12)	16	25	37		0.5	0.4	0.1	
(5,13)	28	35	37	40	0.4	0.3	0.2	0.1
(5,15)	25	32			0.7	0.3		
(5,10)	27	33	40		0.4	0.3	0.3	
(5,7)	15	17	19	26	0.3	0.3	0.3	0.1
(6,5)	18	25	29		0.5	0.3	0.2	
(6,13)	21	23			0.5	0.5		
(6,7)	11	31	37		0.5	0.5	0.1	
(6,3)	18	24			0.7	0.3		
(7,10)	19	23	37		0.6	0.2	0.2	
(7,8)	12	15	22	24	0.3	0.3	0.3	0.2
(7,6)	12	23	31		0.5	0.3	0.2	
(8,7)	14	34	39		0.6	0.2	0.2	
(8,14)	14	15	27	32	0.3	0.3	0.2	0.2
(8,9)	13	31	32		0.8	0.1	0.1	
(8,4)	13	23	34		0.4	0.3	0.3	
(9,7)	10	17	20		0.6	0.3	0.1	
(9,10)	16	18	36	39	0.3	0.3	0.2	0.2
(9,15)	12	13	25	32	0.4	0.3	0.2	0.1
(9,14)	19	24	29		0.4	0.3	0.3	
(10,13)	14	20	25	32	0.3	0.3	0.2	0.2
(10,15)	15	19	25		0.4	0.3	0.3	
(10,14)	23	34			0.9	0.1		
(11,13)	13	31	25		0.6	0.3	0.1	
(11,5)	18	19	20	23	0.3	0.3	0.3	0.1
(11,6)	10	19	39		0.5	0.4	0.1	
(12,8)	15	36	39		0.5	0.3	0.2	
(12,9)	16	22			0.7	0.3		
(12,14)	10	13	18	34	0.3	0.3	0.3	0.1
(13,15)	12	31			0.9	0.1		
(14,15)	14	19	32		0.5	0.3	0.2	



Proceedings
6th Annual CSI Computer Conference

Computer Engineering Department
The University Of Isfahan



20-22 February 2001