ALGORITHMS FOR SOLVING THE MAXIMUM CLIQUE PROBLEM

Chartchai Leenawong* and Rungaree Santawakup

Department of Mathematics and Computer Science
Faculty of Science
King Mongkut's Institute of Technology Ladkrabang
Bangkok 10520, THAILAND

ABSTRACT

Given a graph, in a well-known combinatorial optimization problem, the search for a maximum clique is investigated here. A clique of a graph is a set of vertices, any two of which are adjacent. The maximum clique problem asks for a clique having a largest vertex (node) set. This research modifies a specific previous algorithm that uses branch and bound as well as pruning strategy by reordering the vertices according to the degrees of vertices. Then the new algorithm and some previous algorithms are implemented on a computer to compare the results of these algorithms on a certain type of graphs, namely, random graphs. In addition, for approximation proposes, an algorithm for solving the minimum vertex color problem is also implemented because of the fact that the solution to this problem is an upper bound to the solution of the maximum clique problem.

KEYWORDS: Maximum Clique, Combinatorial Optimization, Computer Algorithm.

1. INTRODUCTION

The maximum clique problem is one of the classical problems in combinatorial optimization and graph theory. Its applications can be found in various areas when the problem can be transformed into a graph problem and the objective is to find the largest set of vertices, each two of which has a direct edge. This section is to state the problem by starting with basic graph theory concepts.

Let $G = (V,E)$ denote an undirected graph, where V is the set of vertices (nodes) and E is the set of edges. Two vertices are said to be adjacent if they are connected by an edge. A clique of a graph is a set of vertices, any two of which are adjacent. An independent set of a graph is a subset S of V, such that every two nodes in S are not joined by an edge of E. The maximum independent set problem consists of finding the largest cardinality of an independent set. A vertex cover of a graph is a subset C of V, such that every edge of E has an endpoint in C. The minimum vertex cover problem consists of finding the smallest cardinality of a vertex cover.

* Corresponding author. Tel/Fax: 02-326-4341 Ext. 313/284.
E-mail: klchartc@kmitl.ac.th

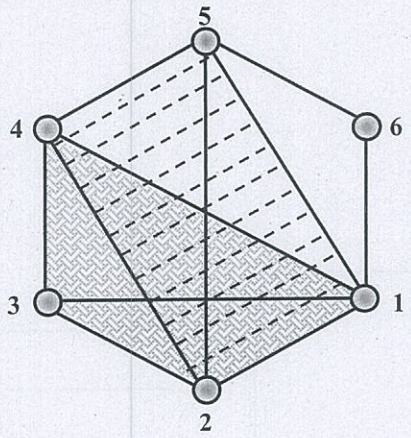

Figure 1.1: An Example of Two Maximum Cliques of Size 4 in a 5-Vertex Graph.

The maximum clique problem searches for a clique having a largest node set. And example of a maximum clique is shown in Figure 1.1. This problem is computationally equivalent to some other important graph problems, for example, the maximum independent set problem and the minimum vertex cover problem. Since these are NP-hard problems [1], no polynomial time algorithms are expected to be found. Therefore almost all types of algorithms have been used to solve it. Nevertheless, as these problems have several important practical applications, it is of great interest to try to develop fast, exact algorithms for small instances. Another direction of research, which has recently been fairly popular, is that of using heuristic methods to find as large cliques as possible, without proving optimality. At the survey of Pardalos and Xue [2] provides an extensive bibliography on the maximum clique problem.

2. PREVIOUS ALGORITHMS

In this section, two algorithms proposed by Carraghan and Pardalos [3] in 1990 and Östergård [4] in 2002 are presented. The idea behind each algorithm is given now.

Algorithm 1
Given an arbitrary graph, the algorithm basically starts at a vertex to see how large a maximal clique size can be identified if that vertex is included in the clique. Repeat the process for other vertices until the last vertex is considered. In the mean time, after each iteration, update the maximum clique size found so far.

Details and steps of this algorithm are as follows.

Step 0 : Set $max = 0$, $i = 1$, $size = 0$.
Step 1 : Given a graph $G(V, E)$, randomly assign numbers 1 to n to the vertices.
Step 2 : If ($i > n$) or the possible clique size of the remaining graph is not greater than the current maximum clique, stop, *max* is the maximum clique size. Otherwise, go to step 3.
Step 3 : Remove vertex i from V, set $size = size + 1$.
Step 4 : Set U equal to the intersection of neighboring vertices of the just-removed vertex in Graph V and the remaining vertices.
Step 5 : If the possible clique size of the remaining graph is not greater than the current maximum clique. Set $i = i + 1$, $size = 0$ and go back to step 2. Otherwise, go to step 6.
Step 6 : If $U = \emptyset$, *size* is a maximal clique size and set $max = size$ only if *max* is less than *size*. Also set $i = i + 1$, $size = 0$ and go back to step 2. Otherwise, remove a smallest vertex in U, set $size = size + 1$ and go back to step 4.

Algorithm 2

This algorithm starts at a subgraph that has one vertex to see how large a clique can be. Then insert another vertex and consider again the maximal clique size in the induced subgraphs with those vertices. Repeat the process until the last vertex is inserted. Furthermore, to make it faster, the variable *found* is used to see if the maximum clique size is found.

Details and steps of the algorithm are as follows.

Step 0 : Set $max = 0$, $i = n$, $size = 1$, *found* = false, $c[i] = 0$, $W = \varnothing$.

Step 1 : Given a graph $G(V,E)$, randomly assign numbers 1 to n to the vertices.

Step 2 : If ($i < 1$), stop, *max* is the maximum clique size. Otherwise, go to step 3.

Step 3 : Insert vertex i (from V) into W.

Step 4 : Set U equal to the intersection of neighboring vertices of the just added vertex in Graph W and the vertices in Graph W.

Step 5 : If the possible clique size of the remaining graph (depending on U) is not greater than the current maximum clique, set $c[i] = max$, $i = i - 1$, $size = 1$ and go back to step 2. Otherwise, go to step 6.

Step 6 : If $U = \varnothing$, *size* is a maximal clique size and set $max = size$ only if *max* is less than *size*. Also set $c[i] = max$, $i = i - 1$, size = 0 and go back to step 2. Otherwise, go to step 7.

Step 7 : If the possible clique size of the remaining graph (depending on $c[i]$) is ot greater than the current maximum clique or *found* = true then set $i = i-1$, $size = 1$ and go back to step 2. Otherwise, remove a smallest vertex in U, set $size = size + 1$ and go back to step 4.

3. PROPOSED ALGORITHMS AND EXPERIMENTAL DESIGN

In this section, two proposed algorithms are presented. It is hoped that the proposed algorithms will outperform the previous two algorithms, especially on a certain type of graphs, namely, random graphs. The definition of random graphs is given later in this section. A way of generating random graphs by the computer is then followed.

3.1 The Proposed Algorithms

In this section, two proposed algorithms (and henceforth called Algorithm 3 and Algorithm 4) for solving the maximum clique problem are presented. Again, the general concepts along with the steps of the algorithms are presented here.

Algorithm 3

For good performance of the algorithm, a proper heuristic for reordering the vertices has to be chosen. One can think of several ways of doing this, and these orderings may have different effects for different types of graphs.

We will now consider an improved algorithm by reordering the vertices in descending order of their degree (the number of *incident* edges) using Selection Sort. Additional notation to be used is defined here. Let $S_i = \{v_1, v_2, \ldots, v_i\}$ where $\deg_G(v_1) \geq \deg_G(v_2) \geq \ldots \geq \deg_G(v_i) \geq \ldots \geq \deg_G(v_n)$. This improved algorithm searches for the maximum clique by first considering S_1, then S_2, S_3, and so on (or W in each iteration of the following algorithm).

Details and steps of the algorithm are as follows.

Step 0 : Set $max = 0$, $i = 1$, $size = 1$, *found* = false, $c[i] = 0$, $W = \varnothing$,

Step 1 : Given a graph $G(V,E)$, sort the vertices in descending order of their degrees and assign numbers 1 to n to the sorted vertices.

Step 2 : If ($i > n$) or the possible clique size of the remaining graph (depending on U, $c[i]$) is not greater than the current maximum clique, stop, *max* is the maximum clique size. Otherwise, go to step 3.

Step 3 : Insert vertex i (from V) into W (or S_i).

Step 4 : Set U equal to the intersection of neighboring vertices of the just added vertex in Graph W and the vertices in Graph W.

Step 5 : If the possible clique size of the remaining graph (depending on U) is not greater than the current maximum clique, set $c[i] = max$, $i = i - 1$, $size = 1$ and go back to step 2. Otherwise, go to step 6.

Step 6 : If $U = \emptyset$, $size$ is a maximal clique size and set $max = size$ only if max less than $size$. Also set $c[i] = max$, $i = i - 1$, size = 0 and go back to step 2. Otherwise, go to step 7.

Step 7 : If the possible clique size of the remaining graph (depending on $c[i]$) is not greater than the current maximum clique or *found* = true then set $i = i-1$, $size = 1$ and go back to step 2.

Otherwise, remove a smallest vertex in U , set $size = size + 1$ and go back to step 4.

Algorithm 4

This algorithm uses minimum vertex coloring problem to approximate the maximum clique size. That is doable because the maximum clique size is less than or equal to the minimum number of vertex colors [5], or $\omega(G) \leq \chi(G)$.

A vertex coloring is an assignment of labels or colors to each vertex of a graph such that no edge connects two identically colored. The most common type of vertex coloring seeks to minimize the number of colors for a given graph and this number is an upper bound of the maximum clique size.

First, initialize parameters i and color c to be assigned to v_i. Then assign the minimum possible color c to v_i. Repeat the process until the last vertex is assigned.

Details and steps of the algorithm are as follows.

Step 0 : [This step initializes the parameter i and color c , used for naming the current vertex v_i and to be assigned to v_i.] $i = 1$, $c = 1$.

Step 1 : Given a graph $G(V,E)$, assign numbers 1 to n to the vertices.

Step 2 : [The minimum possible color c is assigned to v_i.]

2.1 Sort the colors adjacent with v_i in non-decreasing order and call the resulting list L_i.

2.2 If c does not appear on L_i , then assign color c to v_i and go to Step 4. Otherwise, continue.

Step 3 : [The color c is incremented.]

$c = c + 1$ and return to Step 2.2.

Step 4 : [The parameter i is incremented.]

If $i < n$, then $i = i + 1$, and return to Step 1.

Otherwise, stop.

3.2 Experimental Design

Random graphs are graphs randomly generated by the computer. Input parameters are the number of vertices and the edge density. The edge density is defined as the proportion of number of edges in the graph being considered and the number of edges in the complete graph on the same set of vertices. For example, for a graph of 10 vertices, if our graph has 27 edges, the edge density will be $27/\binom{10}{2} = 0.6$ or 60%. For each instance, a random graph is constructed by the following steps.

Step 1 : Define an edge density.

Step 2 : Select a pair of vertices.

Step 3 : Random a number between 0 – 1 uniformly. If the random number is not greater than edge density then insert an edge.

Step 4 : Repeat Step 2 and Step 3 for every other pair of vertices.

Eventually, after many instances have been undertaken, the average edge density of the graphs actually constructed will be very close to the predetermined edge density.

4. COMPUTER EXPERIMENTAL RESULTS

In this section, computer experimental results of the maximum clique problem when solved by the four algorithms in section 3 are presented. The two quantities of interest here are the average CPU time used to obtain an optimal solution and the average maximum clique number of the optimal solution.

To see the performance of each algorithm from section 3 in solving the maximum clique problem, algorithms 1 to 4 are applied to the same sets of random graphs having various numbers of vertices and different values of edge densities. For each value of number of vertices and edge density, 500 randomly generated problems are created using C^{++} programming on a 1.8 GHz Pentium M laptop computer with 768 MB RAM.

Vertices	Edge Density	Average CPU Time (Seconds)			
		Algorithm 1	Algorithm 2	Algorithm 3	Algorithm 4
5	0.1	0.000030	0.000009	0.000001	0.000001
	0.2	0.000048	0.000017	0.000007	0.000001
	0.5	0.000077	0.000049	0.000017	0.000004
	0.8	0.000082	0.000059	0.000025	0.000005
	0.9	0.000119	0.000073	0.000051	0.000009
10	0.1	0.000173	0.000100	0.000073	0.000013
	0.2	0.000203	0.000197	0.000159	0.000075
	0.5	0.000252	0.000202	0.000197	0.000102
	0.8	0.000300	0.000288	0.000210	0.000139
	0.9	0.000800	0.000499	0.000307	0.000201
25	0.1	0.000806	0.000531	0.000441	0.000294
	0.2	0.001000	0.000747	0.000500	0.000417
	0.5	0.004600	0.001906	0.000981	0.000500
	0.8	0.081500	0.009482	0.003220	0.000529
	0.9	0.707400	0.083146	0.010180	0.000779
50	0.1	0.003200	0.001900	0.001239	0.000802
	0.2	0.006700	0.002300	0.001700	0.000927
	0.5	0.084000	0.004530	0.002420	0.001040
	0.8	22.205072	0.036634	0.004370	0.002700
	0.9	558.863600	0.551210	0.013278	0.007500
100	0.1	0.030200	0.002400	0.002000	0.003540
	0.2	0.067800	0.059400	0.005152	0.020200
	0.5	3.210400	0.160240	0.011900	0.033700
	0.8	949.599100	6.027900	1.964358	0.040700
	0.9	3156.749600	417.359448	18.785400	0.052100
150	0.1	0.081600	0.004200	0.003600	0.007840
	0.2	0.591600	0.125080	0.024342	0.060400
	0.5	34.278208	0.156900	0.038596	0.097500
	0.8	4907.210400	58.003200	34.491617	0.157500
	0.9	11452.98070	8993.601200	1829.406020	0.178000
200	0.1	0.202800	0.009900	0.007828	0.016982
	0.2	4.009500	0.513200	0.072464	0.131300
	0.5	468.734900	1.349800	0.158844	0.141600
	0.8	51782.39820	1252.552400	827.210100	0.497100

Table 4.1 Average CPU Time to a Maximum Clique When Solved by Algorithms 1 to 4 in Random Graphs with Different Numbers of Vertices and Different Values of Edge Density

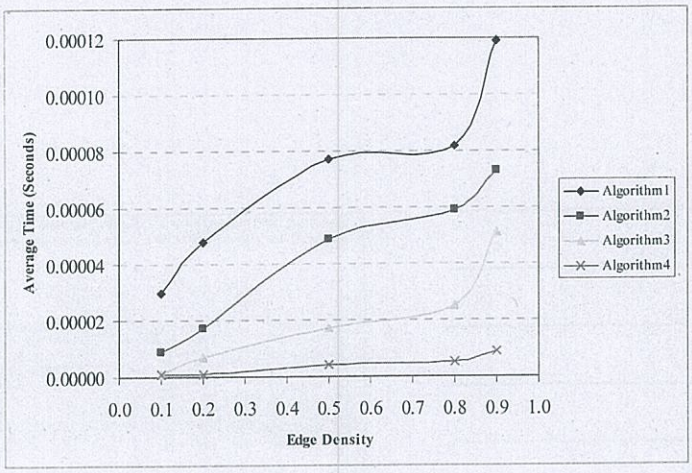

Figure 4.1: Average CPU Time to a Maximum Clique When Solved by Algorithms 1 to 4 as a Function of Edge Density in Random Graphs with 5 Vertices.

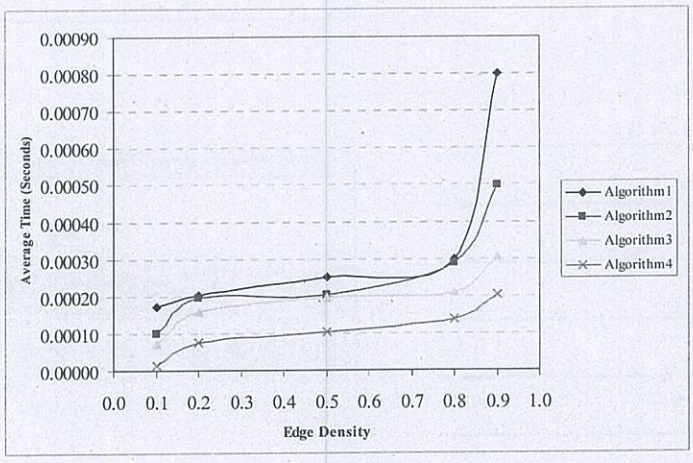

Figure 4.2: Average CPU Time to a Maximum Clique When Solved by Algorithms 1 to 4 as a Function of Edge Density in Random Graphs with 10 Vertices.

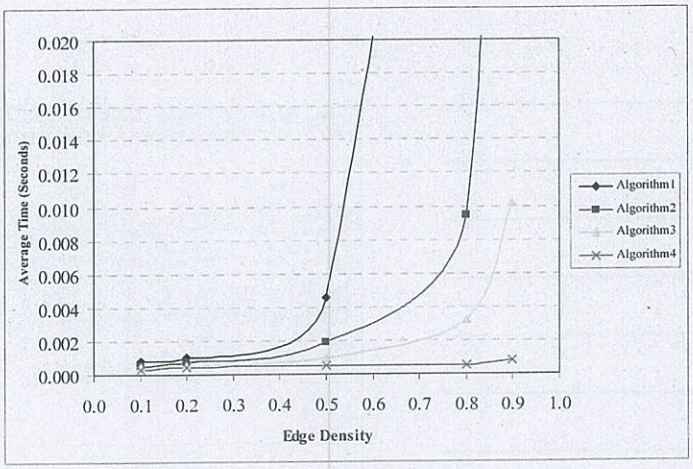

Figure 4.3: Average CPU Time to a Maximum Clique When Solved by Algorithms 1 to 4 as a Function of Edge Density in Random Graphs with 25 Vertices.

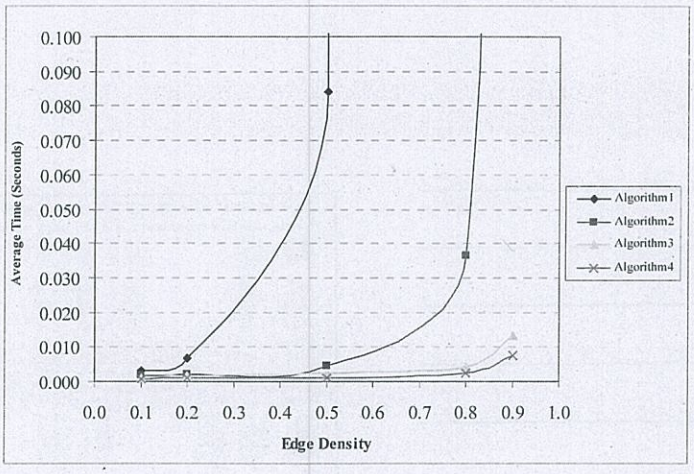

Figure 4.4: Average CPU Time to a Maximum Clique When Solved by Algorithms 1 to 4 as a Function of Edge Density in Random Graphs with 50 Vertices.

4.1 Average CPU Time to a Maximum Clique

Computer experiments for solving the maximum clique problem by algorithms 1 to 4 have been conducted. Table 4.1 shows the results on the average CPU time to an optimal solution in random graphs for different numbers of vertices----5, 10, 25, 50, 100, and 200---- and different values of edge density ---0.1, 0.2, 0.5, 0.8, and 0.9 (0.9 is skipped for the case of 200 vertices due to its complexity). To understand every aspect of the results better, various graphs are plotted in Figures 4.1 to 4.4

The results in Table 4.1 and Figures 4.1 through 4.4 show that for small numbers of vertices and any values of edge density, algorithms 1 to 4 can solve the problem in reasonable and comparable time even though algorithms 3 and 4 seem to perform slightly better (see Figures 4.1 and 4.2). Notice that the values of the *y*-axis in all the figures are not the same. As for larger numbers of vertices, when the values of edge density remain small (such as 0.1 and 0.2 in Figures 4.3 and 4.4), the four algorithms still perform relatively well. However, when the values of edge density increase, the average time to a maximum clique enormously improves in algorithms 3 and 4. The growing rate of the average CPU time with respect to the increasing values of edge density is highest for algorithm 1 followed by algorithm 2. Meanwhile, the CPU time increases much slower for algorithms 3 and 4 even with larger numbers of vertices.

4.2 Average Maximum Clique Size of a Maximum Clique

The results of these algorithms on the average maximum clique size are presented. Table 4.2 shows the results on the average maximum clique size in random graphs for different numbers of vertices---5, 10, 25, 50, 100 and 200---and different values of edge density---0.1, 0.2, 0.5, 0.8 and 0.9 (0.9 is skipped for the case of 200 vertices due to its complexity). To understand every aspect of the results better, various graphs are plotted in Figures 4.5 to 4.8

The results in Table 4.2 and Figures 4.5 through 4.8 show that for small numbers of vertices and any values of edge density the average maximum clique size obtained from the four algorithms are indifferent (see Figures 4.5 through 4.6). As for larger numbers of vertices, when the values of edge density = 0.1 (see Figures 4.7 and 4.8), the four algorithms still perform relatively well. However, when the values of edge density increase, algorithm 1, 2, and 3 still give comparable results while the curve of algorithm 4 increase faster than the curves of the other three algorithms (see Figures 4.7 and 4.8).

5. CONCLUSIONS

Among all the algorithms presented in this paper for solving the maximum clique problem, the results show that algorithm 3 absolutely outperforms algorithms 1 and 2 for random graphs according to the average CPU time to a local maximum clique. Note that the maximum clique sizes obtained from algorithms 1, 2 and 3 are insignificantly different.

In terms of the average CPU time to a local maximum clique, algorithm 4 is the best. However, the average maximum clique size is really far from those of the remaining three algorithms. This is because algorithm 4 is meant for the minimum vertex coloring problem, not the maximum clique problem. Note that the number of minimum colors obtained from algorithm 4 may be used in another kind of algorithms for finding the maximum clique, more precisely, when such algorithms start at a highest-degree vertex and work downward. A vertex having the degree equal to the number of minimum colors can be used as a starting vertex instead. This will definitely improve the efficiency of such algorithms.

Vertices	Edge Density	Average Maximum Clique Size			
		Algorithm	Algorithm 2	Algorithm	Algorithm 4
5	0.1	1.709	1.640	1.632	1.702
	0.2	1.970	1.950	1.974	1.970
	0.5	2.710	2.678	2.680	2.790
	0.8	3.690	3.674	3.726	3.820
	0.9	4.240	4.248	4.200	5.180
10	0.1	2.100	2.084	2.120	2.109
	0.2	2.530	2.572	2.538	2.550
	0.5	3.960	3.966	3.856	4.030
	0.8	6.080	6.052	6.066	6.100
	0.9	7.480	7.394	7.450	7.490
25	0.1	2.890	2.872	2.862	2.970
	0.2	3.510	3.470	3.484	3.620
	0.5	6.250	5.808	5.970	6.710
	0.8	10.630	10.546	10.894	11.670
	0.9	14.180	14.140	14.254	14.841
50	0.1	3.210	3.164	3.204	3.640
	0.2	4.180	4.552	4.196	6.310
	0.5	7.550	7.810	7.994	11.250
	0.8	15.860	15.140	14.960	20.000
	0.9	22.390	21.818	21.488	24.470
100	0.1	3.960	3.958	3.954	5.040
	0.2	5.030	5.034	5.042	9.730
	0.5	9.340	9.288	9.284	19.840
	0.8	20.170	20.180	20.174	34.780
	0.9	31.129	31.113	31.825	43.940
150	0.1	4.060	4.019	4.189	8.190
	0.2	5.950	5.351	5.632	12.680
	0.5	10.180	10.245	10.832	26.920
	0.8	23.460	23.779	23.006	48.190
	0.9	36.887	36.530	37.328	62.010
200	0.1	4.220	4.193	4.229	9.811
	0.2	6.728	6.523	6.420	17.080
	0.5	11.090	11.695	11.832	33.662
	0.8	29.420	29.920	28.100	61.209

Table 4.2 Average Maximum Clique Size Obtained from Algorithms 1 to 4 in Random Graphs with Different Numbers of Vertices and Different Values of Edge Density

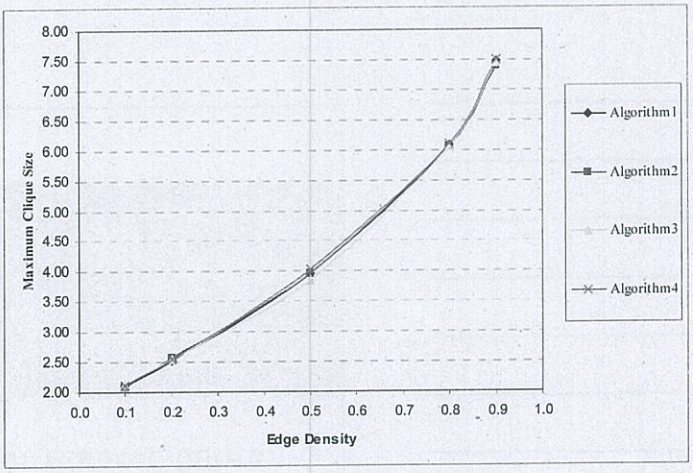

Figure 4.5: Average Maximum Clique Size Obtained from Algorithms 1 to 4 As a Function of Edge Density in Random Graphs with 10 Vertices.

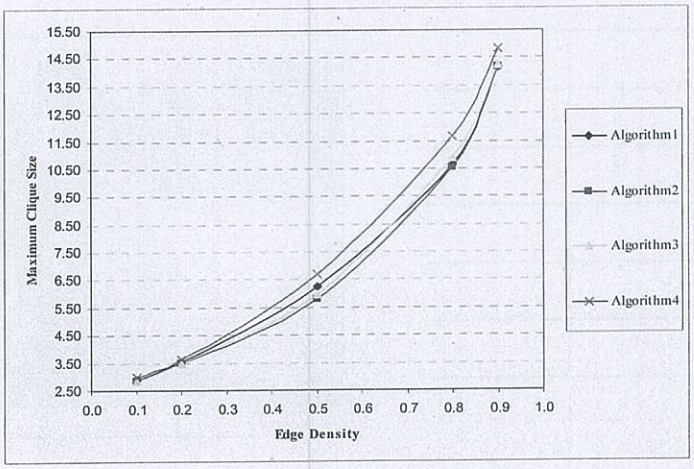

Figure 4.6: Average Maximum Clique Size Obtained from Algorithms 1 to 4 As a Function of Edge Density in Random Graphs with 25 Vertices.

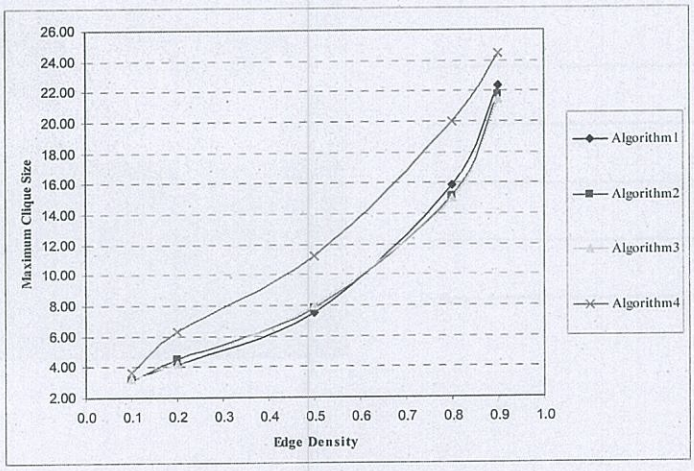

Figure 4.7: Average Maximum Clique Size Obtained from Algorithms 1 to 4 As a Function of Edge Density in Random Graphs with 50 Vertices.

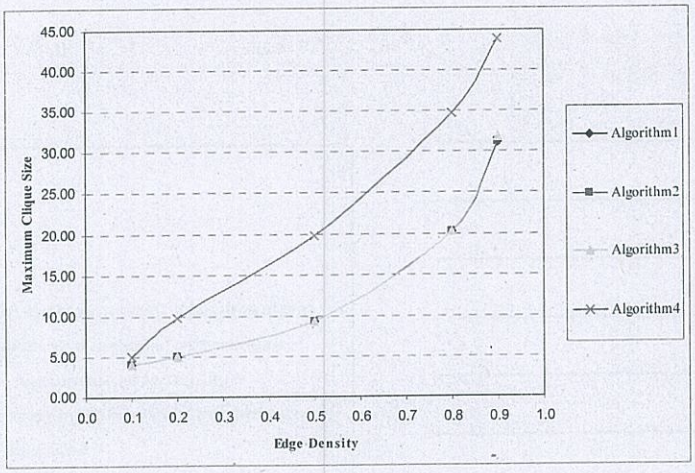

Figure 4.8: Average Maximum Clique Size Obtained from Algorithms 1 to 4 As a Function of Edge Density in Random Graphs with 100 Vertices.

6. REFERENCES

[1] Garey, M.R. and Johnson, D.S. **1979** *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, W. H. Freeman.

[2] Pardalos, P.M. and Xue, J. **1994** The Maximum Clique Problem, *Journal of Global Optimization*, *4*, 301-328.

[3] Carraghan, R. and Pardalos, P.M. **1990** An Exact Algorithm for the Maximum Clique problem, *Operation Research Letters*, *9*, 375-382.

[4] Östergård, P.R.J. **2002** A Fast Algorithm for the Maximum Clique Problem, *Discrete Applied Mathematics*, *120*, 195-205.

[5] Gould, R. **1988** *Graph Theory*. California, Benjamin/Cummings Publishing.