Yugoslav Journal of Operations Research
2 (1992), Number 1, 3–11

A COMPARISON OF ALGORITHMS FOR THE MAXIMUM CLIQUE PROBLEM

Pierre HANSEN

GERAD and Ecole des Hautes Etudes Commerciales
5255 Avenue Decelles, Montreal H3T 1V6, Canada

Nenad MLADENOVIĆ

Laboratory for Operations Research
Faculty of Organizational Sciences, University of Belgrade
Jove Ilića 154, 11000 Belgrade, Yugoslavia

Abstract. Five recent practically efficient methods for solving the maximum clique problem are briefly described and compared on randomly generated graphs.

Key words and phrases: Graph, clique, algorithm.

1. INTRODUCTION

Let $G = (V, E)$ denote a graph with vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and edge set $E = \{e_1, e_2, \ldots, e_n\}$ (see Berge [4] for basic definitions). A set $C \subseteq V$ of vertices is a **clique** if any two of them are adjacent, i.e., if they are the end points of an edge of E. A set $S \subseteq V$ of vertices is a **stable** (or independent) set if any two of them are not adjacent. A set $T \subseteq V$ of vertices is a transversal (or vertex cover) if any edge of E contains at least one vertex of T. Clearly, a clique of G is a stable set of the **complementary graph** $\overline{G} = (V, \overline{E})$ of G, in which a pair of vertices is joined by an edge if and only if it is not so in G. Moreover, any minimal transversal T of G is the vertex complement of a **maximal stable set** S of G (to which no vertex can be added without losing stability) i.e., a vertex belongs to T if and only if it does not belong to S.

Research of the first author supported by NSERC (Natural Sciences and Engineering Research Council of Canada) grant to HEC, grant GPO 105574 and FCAR (Fonds pour la Formation de Chercheurs et l'Aide à la Recherche) grant 92EQ1048.

Research of the second author done at GERAD during a sabbatical leave from Faculty of Organizational Sciences, supported in part by CETAI, GERAD and NSERC grant to HEC.

A clique C (or a stable set S) is **maximum** if it has the largest possible number of vertices. The **clique number** $\omega(G)$ of G (respectively, the **stability number** $\alpha(G)$ of G) is equal to the cardinality of a maximum clique (respectively a maximum stable set). So finding a maximum clique is tantamount to finding a maximum stable set or a minimum transversal in the complementary graph. A (vertex) **coloring** of a graph G is an assignment of colors to the vertices of G such that no two adjacent vertices get the same color. The cardinality of a coloring in the minimum number of colors is called the **chromatic number** $\gamma(G)$ of G. For every graph $\omega(G) \leq \gamma(G)$ holds.

Finding maximum cliques or stable sets, or minimum transversals are classical problems in applied graph theory. They have many applications in various fields, e.g., experimental design; information retrieval systems; signal transmission analysis; computer vision; sociological structures; economy (see [2] and [8] for references). Before discussing solution techniques let us mention two more recent applications of the maximum stable set and minimum transversal problems.

The *Forest planning* problem, studied by Barahona, Weintraub and Epstein [3], consists in defining cutting patterns for a forest that respect the habitat of wildlife. This implies large zones should not be cut at the same time. Scenic beauty may also be a reason for such dispersion constraints. The forest is first partitioned into units. The decision variables specify whether cutting is performed in a unit or not in a given time period. If it takes place, it is required that no timber cutting be carried out in neighboring units in the same period. A graph G that has one vertex per unit and per period is then constructed. This graph has edges between any two vertices that represent the same period in neighboring units as well as edges between any two vertices in the same unit. A managing alternative corresponds to a set of vertices, such that there is no edge between any two of them, i.e., to a stable set of G. Finding the largest number of units which may be cut in the periods considered is a maximum stable set problem.

Non-convex quadratic programs with non-convex quadratic constraints arise in many engineering contexts. They may be reformulated as *bilinear programs* with bilinear constraints which are easier to solve. In the latter, variables can be partitioned into two sets such that any nonlinear term contains exactly one variable from each set. Consequently, when any one set of variables is fixed, a linear program is obtained. This allows solution through a *generalized Benders decomposition* approach (e.g. Geoffrion [9], Floudas and Visweswaran [6]). Finding the reformulation which has the smallest number of *complicating variables*, i.e., of variables in the smallest of the two sets, can be done using the *co-occurrence* graph of the quadratic program. This graph has vertices associated with the variables of the quadratic program and edges joining vertices associated with variables appearing jointly in one term of the objective function or constraints. In practice it is usually sparse, and hence has a dense complementary graph. Hansen and Jaumard [14] show that a minimum set of complicating variables corresponds to a minimal transversal of the co-occurrence graph (or to a maximum clique of the complementary graph of this co-occurrence graph). Therefore, to find a *narrow reduction* of

a general quadratic program to a bilinear program, i.e., a reformulation with minimum number of complicating variables, one must solve a maximum clique problem in a dense graph.

The maximum clique problem is NP-hard and numerous heuristic or exact algorithms have been proposed to solve it. The exact algorithms are usually enumerative in nature. Recent, practically efficient exact algorithms have been proposed by Balas and Yu [2], Friden, Hertz and de Werra [7], Carraghan and Pardalos [5] and Babel and Tinhofer [1]. These algorithms all allow to solve large problems, with 400 vertices or more when the graph G is sparse. Finding a solution for dense graphs (i.e., when the density d, or number of edges divided by the number of vertex pairs, is 80% or more) appears to be much more difficult. For $d = 90\%$ only problems with up to 100 vertices have been reported to have been solved exactly. Recently, Hansen and Mladenović [15] have proposed two algorithms for the maximum clique problem in dense graphs (a modification of the Carraghan-Pardalos algorithm (*MCP*) and the so called *"Dense CLique" DCL* algorithm).

In this paper, five of those algorithms are briefly described (Section 2) and compared (Section 3): Carraghan-Pardalos [5] (*C-P*); Balas and Yu [2] (*B-Y*); Friden, Hertz and de Werra [7] (*TABARIS*); Modification of Carraghan-Pardalos [15] (MCP) and Dense clique [15] (DCL). The Babel-Tinhofer [1] algorithm has not been included in the comparison as their code was not available to us and their algorithm is quite close to *TABARIS* [7].

2. PRINCIPLES OF THE ALGORITHMS

All algorithms considered here are based on the *Branch and Bound* (or Backtrack or Recursive programming) technique, where an **exhaustive search tree** is constructed. The nodes of this tree are associated with subproblems induced by current sets V_i of visited and U_i of unvisited vertices. The initial problem corresponds to the root. The subproblems obtained by branching are associated with the nodes of the tree. The shape of the tree depends on the vertex ordering. If no further branching is possible, i.e., if a leaf in the tree is obtained, a solution is found and backtracking takes place, after storing this solution if it is better then the incumbent one. Any leaf with greatest distance to the root corresponds to a maximum clique. Lower and upper bounds on the size of the maximum clique for the current subproblem may be computed and used to curtail the enumeration. Moreover, local feasibility and optimality conditions are exploited: vertices not adjacent to an included one are excluded and vertices all of whose neighbors are included are included too.

Enumerative algorithm (C-P). Carraghan and Pardalos [5] present an enumerative, depth-first search algorithm in which a few simple tests are implemented very efficiently. This algorithm performs a lexicographic search, i.e., vertices are first ranked (in order of non-decreasing degrees if the density is greater or equal to 0.4 and in an arbitrary order otherwise) and always considered in that order. Initially, indices of all variables are included in a list, at depth 1. At this level, step

i, for $i = 1, 2, \ldots, n$, consists in including the i^{th} vertex in the list into the current clique, increasing the level by 1 and copying those indices in the previous list which correspond to vertices adjacent to the chosen one. This procedure is iterated to greater and greater depths until the condition

$$\textit{depth} + \textit{current list size} - \textit{step} \leq \textit{size of current best clique}$$

is satisfied or the current list becomes empty (in which case the current best clique size is updated). Then backtracking takes place, i.e., the level is reduced by 1 and the step increased by 1 (if not yet at the end of the current list, in which case backtracking occurs again).

Coloring triangulated graphs (B-Y). As the maximum clique problem is NP-hard there is no polynomial algorithm to solve it for arbitrary graphs. Nevertheless there are polynomial algorithms for special classes of graphs. Perfect graphs are one of them (recall G is perfect if $\omega(H) = \gamma(H)$ for all induced subgraphs H of G) and a special class of perfect graphs is the class of triangulated graphs (G is **triangulated** if every cycle of G of length at least 4 has a cord). The main ingredients in the Balas and Yu [2] algorithm are: (i) finding a maximal triangulated induced subgraph (in $O(n+m)$ time) and its maximum clique (let us denote the cardinality of this maximum clique by k); (ii) coloring the so-obtained triangulated graph, i.e., finding a maximal k-chromatic induced subgraph (again in $O(m+n)$ time). In other words, the strategy of this algorithm consists in finding maximal subgraphs (subproblems, i.e., nodes in the search tree) for which the current lower bound is also an upper bound, and which can thus be eliminated. Since no upper bound on the value of each subproblem is available, the search strategy is depth-first.

Using Tabu Search (TABARIS). Glover [10, 11, 12], has developed a metaheuristic for solving difficult combinatorial optimization problems known as *Tabu Search*. It combines local search procedures with a number of anti-cycling rules which prevent being trapped in local optima. Moreover, exploitation of short, medium and long-term memory allow as to guide, intensify or diversify the search. It has been successfully applied to the Maximum satisfiability [13], Graph coloring [17], Timetabling [16], Neural networks [18] among others problems. TABARIS (Tabu And Branch and bound Applied Repeatedly for Independent Sets, Friden, Hertz and de Werra [7]) is an implicit enumeration algorithm (fairly similar to the Balas and Yu algorithm [2]), which uses at some steps the Tabu Search technique for getting bounds on the stability number $\alpha(H)$. This is done by: (i) finding a large stable set in the set of candidate nodes for inclusion in the stable set (set of unvisited nodes); (ii) covering as many nodes of that set as possible with cliques. Vertices are ordered in advance according to non-increasing degrees.

Finding Simplicial vertices (MCP), (DCL). A vertex v of an arbitrary graph G is called **simplicial** if all vertices adjacent to v are pairwise adjacent, i.e., if the subgraph induced by its neighbors is a clique. In the Balas and Yu [2] algorithm, simplicial vertices are used in finding a prefect ordering of the vertices of the subgraph of G (an **ordering** $v_1, \ldots, v_n$ of the vertices of G is called **perfect** if

for $k = 1, \ldots, n$, v_k is simplicial in $G(\{v_k, \ldots, v_n\}))$, i.e., in generating a maximal triangulated induced subgraph. Indeed, $G = (V, E)$ is triangulated if and only if V admits a perfect ordering. In [15] it has been proved that any vertex which is simplicial in the complementary graph $\overline{G} = (V, \overline{E})$ of a given graph G belongs to at least one maximum clique of G. This property is exploited in two algorithms. The first one (MCP) is a variation of the algorithm of Carraghan and Pardalos [5], checking only for leaves in the complementary graph. The second one (DCL) detects systematically simplicial vertices within small cliques. If there is a simplicial vertex in the current subgraph, then there is only one branch from that node in the search tree; thus, backtracking at that subproblem is not necessary. If there are no more simplicial vertices, then a vertex with maximum degree in the subgraph induced by the current set of unvisited vertices is chosen for branching.

3. COMPUTATIONAL RESULTS

Computational experiments have been run on a *Sun Sparc* station with random graphs (generated according to the so called *uniform random graph scheme*, see e.g. Gendreau, Salvail and Soriano [8]). The number of vertices n is chosen as well as the probability of presence of an edge (d). Each pair of nodes is then examined and becomes an edge with probability d (comparing d with a number drawn randomly from a uniform distribution on $(0, 1)$. The same pseudo-random generator as in [5] is used.

In the tables below, CPU times for the five following methods are compared: Carraghan-Pardalos [15] (C-P); Balas and Yu [2] (B-Y); Friden, Hertz and de Werra [7] (TBS); Modification of Carraghan-Pardalos [15] (MCP) and Dense clique [15] (DCL). For each problem (specified by a given density d and given size n) we generate 10 graphs and consider the average CPU times on these graphs. Methods C-P, MCP and DCL are coded in Fortran 77, while B-Y and TBS are coded in Pascal. We tested CPU times of Fortran and Pascal compiler on simple programs with same instructions and found that on the Sun Sparc station the object programs obtained with Fortran compiler are between 2.5 and 3 times faster then those obtained with Pascal compiler.

Table 1 contains results for small graphs with $n = 80, 90$ and 100. We vary the density d between 0.1 and 0.9 as is usual in the literature. C_{opt}, given in the last column is the average number of vertices in a maximum clique; its nonlinear growth with respect to the density d is worth noting. It explains in part the very rapid increase in CPU time for all codes except DCL when d becomes large. It can be observed that for graphs with density from 10 to 70% C-P and MCP are better than others, while for graphs with density equal to 80% and higher the DCL code is the best. However, if a correction is made for the difference in compilers, then the TBS and B-Y codes are the best for densities of 60% to 80%.

Table 2 presents more detailed results for small dense graphs, i.e., graphs with $n = 80, 90, 100$ and $d \geq 80\%$. It can be seen that for very dense graphs there are examples ($d = 0.96$) where DCL is more than 4.000 times faster on average than

Table 1. Results for small graphs and all densities

n	$dens$	C-P	MCP	DCL	B-Y	TBS	C_{opt}
80	0.10	0.01	0.01	1.36	0.10	0.07	3.8
80	0.20	0.03	0.02	1.54	0.14	0.09	4.8
80	0.30	0.05	0.04	1.81	0.22	0.38	6.0
80	0.40	0.13	0.12	2.20	0.34	0.52	7.0
80	0.50	0.25	0.23	3.09	0.64	1.00	8.7
80	0.60	0.64	0.58	4.67	1.43	2.22	10.4
80	0.70	2.28	2.11	6.57	4.16	3.23	13.7
80	0.80	13.10	11.08	7.82	16.66	9.45	18.3
80	0.90	169.52	87.49	1.96	45.94	34.66	27.5
90	0.10	0.02	0.01	1.98	0.12	0.08	3.9
90	0.20	0.03	0.03	2.35	0.20	0.13	5.0
90	0.30	0.07	0.07	2.84	0.27	0.56	6.2
90	0.40	0.19	0.18	3.79	0.50	0.60	7.3
90	0.50	0.39	0.37	5.93	1.05	1.51	9.0
90	0.60	1.15	1.07	9.31	2.49	2.58	10.9
90	0.70	5.11	4.77	16.40	8.60	6.67	14.1
90	0.80	41.00	33.86	24.47	42.80	29.21	18.6
90	0.90	715.96	374.44	6.55	283.85	160.85	29.1
100	0.10	0.02	0.02	2.80	0.15	0.10	4.0
100	0.20	0.04	0.04	3.37	0.24	0.14	5.1
100	0.30	0.10	0.09	4.30	0.35	0.91	6.1
100	0.40	0.26	0.25	6.17	0.76	0.82	7.6
100	0.50	0.58	0.55	10.15	1.64	2.95	9.2
100	0.60	2.00	1.86	16.53	4.75	3.59	11.7
100	0.70	10.36	9.76	33.16	16.89	11.56	14.7
100	0.80	105.98	84.00	56.31	100.21	84.24	20.0
100	0.90	3721.50	1827.17	31.49	1037.31	657.95	30.6

the best among other methods (*TABARIS*). There is no result in *Table 2*, if CPU time exceed 3,000 seconds.

Table 3 gives results for large graphs. The limit time in experimentations was 1,000 seconds, except for $d = 0.6$, where it was 3,000 seconds. It appears that MCP is the best code for large sparse graphs, but not significantly better than the C-P code. For $n = 300$ and $d = 0.6$, *TABARIS* has the best CPU time. As only small to medium densities are considered here (problems with $n \geq 300$ and $d > 60\%$ appear out of search for exact solution at this time), the DCL code is not competitive.

Table 2. Results for small dense graphs

n	*dens*	*C-P*	*MCP*	*DCL*	*B-Y*	*TBS*	C_{opt}
80	0.80	13.10	11.08	7.82	16.66	9.77	18.3
80	0.82	22.16	17.87	8.06	21.54	15.18	19.5
80	0.84	36.23	26.85	6.93	33.88	17.58	21.0
80	0.86	55.16	39.14	5.73	32.92	23.51	22.9
80	0.88	100.31	59.93	3.50	50.46	26.87	25.0
80	0.90	169.52	87.49	1.96	45.94	34.08	27.5
80	0.92	325.62	139.05	0.74	75.91	40.20	30.4
80	0.94	647.39	187.71	0.17	72.25	53.30	35.2
80	0.96	796.90	102.73	0.01	61.32	37.32	41.5
80	0.98	1040.85	17.59	0.01	2.17	23.78	51.7
80	1.00	0.04	0.04	0.01	0.08	0.02	80.0
90	0.80	41.00	33.86	24.47	42.80	28.94	18.6
90	0.82	65.23	52.12	23.67	54.12	39.01	20.5
90	0.84	117.78	85.70	21.79	77.23	63.33	22.0
90	0.86	196.04	136.40	18.14	121.95	59.10	23.8
90	0.88	343.15	210.90	11.76	175.64	122.98	26.3
90	0.90	715.96	374.44	6.55	283.85	126.43	29.1
90	0.92	1809.40	679.54	2.53	402.11	182.21	32.4
90	0.94	2668.07	788.94	0.48	447.96	140.46	37.8
90	0.96	—	660.23	0.03	262.29	138.40	44.9
90	0.98	917.77	127.89	0.01	10.36	57.22	56.7
90	1.00	0.04	0.04	0.01	0.10	0.02	90.0
100	0.80	105.98	84.00	56.31	100.21	59.15	20.0
100	0.82	187.63	143.27	62.64	140.14	82.10	21.7
100	0.84	372.41	265.41	60.93	254.37	154.57	23.4
100	0.86	700.69	460.88	58.54	413.84	265.57	25.4
100	0.88	1430.42	882.35	52.12	692.80	401.36	27.6
100	0.90	—	1827.17	31.49	1037.31	657.95	30.6
100	0.92	—	2861.44	10.80	1476.06	841.51	34.9
100	0.94	—	—	1.66	1170.43	563.47	40.5
100	0.96	—	—	0.05	1148.52	342.12	48.4
100	0.98	—	523.41	0.02	158.74	82.68	61.3
100	1.00	0.06	0.05	0.02	0.12	0.02	100.0

In conclusion. the *C-P* (or *MCP*) code appears to be the best for finding maximum cliques in large and sparse graphs. while the *DCL* code appears to be the best for such purpose in dense graphs.

Acknowledgements. The authors thank Alain Hertz for sending them copies of the codes for the TABARIS and Balas-Yu algorithms written at the Department of Applied Mathematics. Ecole Polytechnique Fédérale de Lausanne. They also thank Brigitte Jaumard and Olivier Play for their advice on computing matters.

Note. A Fortran 77 code for the algorithm "Dense Clique" is available from the authors.

Table 3. Results for large sparse graphs

n	*dens*	*C-P*	*MCP*	*DCL*	*B-Y*	*TBS*	C_{opt}
200	0.10	0.10	0.10	27.83	0.69	1.24	4.1
200	0.20	0.27	0.27	42.55	1.34	2.23	6.0
200	0.30	0.93	0.93	79.90	3.46	6.28	7.1
200	0.40	3.26	3.21	187.16	10.13	14.40	8.9
200	0.50	16.10	15.78	501.61	42.65	32.88	10.9
200	0.60	121.70	117.89	—	215.10	134.72	13.9
300	0.10	0.26	0.26	110.62	1.74	3.12	5.0
300	0.20	0.98	0.96	209.29	5.15	15.83	6.0
300	0.30	4.37	4.33	577.03	17.14	21.45	8.0
300	0.40	20.55	20.33	—	70.94	57.59	9.6
300	0.50	155.68	153.96	—	412.87	239.31	12.1
300	0.60	2302.07	2178.30	—	—	1767.25	15.1
400	0.10	0.56	0.56	303.62	3.29	6.24	5.0
400	0.20	2.52	2.50	725.26	14.76	36.44	6.4
400	0.30	13.81	13.79	—	53.91	56.67	8.2
400	0.40	83.40	83.15	—	313.87	226.66	10.1
400	0.50	950.94	942.94	—	—	—	12.9
500	0.10	1.05	1.05	683.36	7.14	11.18	5.0
500	0.20	5.34	5.29	—	31.10	54.50	7.0
500	0.30	35.66	35.43	—	149.33	124.83	8.5
500	0.40	253.18	250.94	—	1022.28	649.88	10.5
1000	0.10	7.67	7.59	—	50.19	66.61	6.0
1000	0.20	71.26	70.75	—	577.54	—	7.0
1000	0.30	846.62	842.37	—	—	—	9.5

REFERENCES

[1] L. Babel and G. Tinhofer, *A Branch and bound algorithm for the maximum clique problem*, ZOR — Methods and Models of Operations Research **34** (1990), 207–217.

[2] E. Balas and C.S. Yu, *Finding a maximum clique in an arbitrary graph*, SIAM Journal on Computing **15/4** (1986), 1054–1068.

[3] F. Barahona, A. Weintraub, and R. Epstein, *Habitat dispersion in forest planning and the stable set problem*, Operations Research **40**, Supp. No. 1 (1992), S14–S21.

[4] C. Berge, *Graphes*, Gauthier-Villars, Paris, 1983.

[5] R. Carraghan and P.M. Pardalos, *An exact algorithm for the maximum clique problem*, Operations Research Letters 9 (1990), 375–382.

[6] C.A. Floudas and V. Visweswaran, *A global optimization algorithm (GOP) for certain classes of non-convex NLPs — I. Theory*, Computers and Chemical Engineering 14 (12) (1990), 1297–1417.

[7] C. Friden, A. Hertz, and D. de Werra, *TABARIS: An exact algorithm based on tabu search for finding a maximum independent set in a graph*, Computers and Operations Research 17 (1990), 437–445.

[8] M. Gendreau, L. Salvail, and P. Soriano, *Solving the maximum clique problem using a tabu search approach*, Centre de Recherche sur les Transports, Montreal-Publication No. 675, 1991.

[9] A.M. Geoffrion, *Generalized Benders decomposition*, Journal of Optimization Theory and Applications 10 (1972), 237–260.

[10] F. Glover, *Future paths for integer programming and links to artificial intelligence*, Computer and O.R. 13 (1986), 533–549.

[11] F. Glover, *Tabu search - Part I*, ORSA Journal on Computing 1 (1989), 190–206.

[12] F. Glover, *Tabu search — Part II*, ORSA Journal on Computing 2 (1990), 4–32.

[13] P. Hansen and B. Jaumard, *Algorithms for the maximum satisfiability problem*, Computing 44 (1990), 279–303.

[14] P. Hansen and R. Jaumard, *Reduction of indefinite quadratic programs to bilinear programs*, Journal of Global Optimization 2 (1992), 41–60.

[15] P. Hansen and N. Mladenović *Two algorithms for maximum cliques in dense graphs*, GERAD research report G-9218, 1992.

[16] A. Hertz, *Tabu search for large scale timetabling problems*, European Journal of Operational Research 54 (1991), 39–47.

[17] A. Hertz and D. de Werra, *Using tabu search techniques for graph coloring*, Computing 29 (1987), 345–351.

[18] D. de Werra and A. Hertz, *Tabu search techniques: A tutorial and application to neural networks*, OR Spectrum 11 (1989), 131–141.