ELSEVIER

Operations Research Letters 21 (1997) 211–217

operations research letters

An algorithm for finding a maximum clique in a graph

David R. Wood [*]

School of Computer Science and Software Engineering, Monash University, Wellington Road, Clayton VIC 3168, Australia

Received 1 August 1995; revised 1 August 1997

Abstract

This paper introduces a branch-and-bound algorithm for the maximum clique problem which applies existing clique finding and vertex coloring heuristics to determine lower and upper bounds for the size of a maximum clique. Computational results on a variety of graphs indicate the proposed procedure in most instances outperforms leading algorithms. © 1997 Elsevier Science B.V. All rights reserved.

Keywords: Graph algorithm; Maximum clique problem; Branch-and-bound algorithm; Vertex coloring; Fractional coloring

1. Introduction

A *clique* of an undirected graph $G = (V,E)$ is a maximal set of pairwise adjacent vertices. A set of pairwise nonadjacent vertices is called an *independent set*. In this paper we address the following problem:

Maximum Clique Problem: For a given undirected graph G find a maximum clique of G (whose cardinality we denote by $\omega(G)$).

Clearly, the maximum clique problem is equivalent to that of finding a maximum independent set in the complementary graph. Applications for this problem exist in signal processing, computer vision and experimental design for example (see Ref. [7]). However, not only is the exact problem NP-hard (see Ref. [12]), but approximating the maximum clique problem within a factor of $|V|^{\varepsilon}$ for some $\varepsilon > 0$ is NP-hard [1].

Early algorithms included the branch and bound algorithm of Bron and Kerbosch [10] to generate all the cliques of a graph and the recursive algorithm of Tarjan and Trojanowski [18] to determine a maximum independent set of an n-vertex graph in $O(2^{n/3})$ time. Recent approaches to the maximum clique problem have included the branch-and-bound algorithms of Carraghan and Pardalos [11], Pardalos and Rodgers [15], Balas and Yu [7], Balas and Xue [5, 6], Babel and Tinhofer [3], and Babel [2]. In their survey paper, Pardalos and Xue [16] identify the following key issues in a branch-and-bound algorithm for the maximum clique problem.

1. How to find a good lower bound, i.e. a clique of large size?
2. How to find a good upper bound on the size of a maximum clique?
3. How to branch, i.e. break a problem into smaller subproblems?

In the following section of this paper we address the first two of these questions. In Section 3 we present our

[*] E-mail: davidw@cs.monash.edu.au.

branch-and-bound algorithm, and in Section 4 we discuss computational results of our algorithm in comparison with leading algorithms for the maximum clique problem. We denote the set of vertices adjacent to $v \in V$ by $N_G(v)$ and the subgraph of G induced by $S \subseteq V$ by $G(S)$.

2. Heuristics for the maximum clique problem

The algorithm of Ref. [7] concentrates on the determination of lower bounds using an algorithm to find a maximum clique of a maximal triangulated induced subgraph at selected search tree nodes. This method is extended to the maximum weight clique problem in Ref. [5]. The algorithm presented in this paper and the algorithm of Ref. [6] determine a lower bound at the root node of the search tree, using the algorithm of Balas [4] to find a maximum clique of an edge-maximal triangulated subgraph. To provide lower bounds at non-root search tree nodes we use the following simple heuristic to determine a clique Q of a graph $G = (V, E)$.

CLIQUE: Set $S := V$ and $Q := \emptyset$. While $S \neq \emptyset$, choose a vertex $v \in S$ with maximum degree in G, and set $Q := Q \cup \{v\}$ and $S := S \cap N_G(v)$.

We now turn our attention to the determination of upper bounds. The algorithms of Refs. [11, 15] use the size of a given subgraph as an upper bound for the size of a clique in that subgraph. Vertex colorings provide much tighter upper bounds. A *vertex coloring* (or *k-coloring*) of a graph $G = (V, E)$ is a partition of V into independent sets $(C_1, C_2, \ldots, C_k)$ called *color classes*. Each C_i contains those vertices assigned the ith color. A k-coloring of G must color each vertex of a clique differently, so k is an upper bound for $\omega(G)$. Determining the minimum k such that an arbitrary graph G is k-colorable is NP-hard [12].

The following vertex coloring heuristic, described in Biggs [8] as the *greedy* or *sequential* method, assigns the first color to every vertex available; repeats for the second color, and so on, until all the vertices are colored.

COLOR: To determine a color class C_k, set $C_k := \emptyset$ and initialize S to be the set of uncolored vertices. While $S \neq \emptyset$, assign color k to a vertex $v \in S$ with maximum degree in G, and set $C_k := C_k \cup \{v\}$ and $S := (S \setminus \{v\}) \setminus N_G(v)$.

In Refs. [2, 3, 6] upper bounds are determined using the following vertex coloring heuristic of Brelaz [9]:

DSATUR: While uncolored vertices remain, choose an uncolored vertex v adjacent to the maximum number of different colors (called the *saturation degree* of v), breaking ties by higher degree. Color v with the minimum color not already assigned to an adjacent vertex.

This method colors the connected components of G in turn, and within each connected component the initial vertices chosen form a clique. So DSATUR provides both a lower and upper bound for $\omega(G)$. Comparisons of COLOR and DSATUR in Refs. [6, 19] show that for all but a few of the tested graphs DSATUR requires (up to 27.5%) less colors than COLOR, although DSATUR uses considerably more CPU time. For very sparse and very dense graphs, DSATUR is an order of magnitude more expensive than COLOR [6].

A *fractional coloring* of a graph $G = (V, E)$ is a set C of (possibly intersecting) weighted color classes (i.e. independent sets), such that for each vertex $v \in V$ the sum of the weights of the color classes containing v is at least 1.

Since a color class can contain at most one vertex of a clique, in a fractional coloring the sum of the weights of those color classes intersecting a clique Q is at least $|Q|$. Therefore, the total weight of a fractional coloring of a graph G is an upper bound for $\omega(G)$. The upper bound from a minimum weight fractional coloring is in general tighter than that provided by a minimum vertex coloring [6], however, Grötschel, Lovász and Schrijver show that determining such a fractional coloring is NP-hard [13].

In Ref. [6] the following heuristic for the fractional coloring problem provides upper bounds for the maximum clique problem. After i iterations of FCP each vertex is colored exactly i times, and each color class is assigned weight $1/i$, so $t_i = |C|/i$ is an upper bound for $\omega(G)$. Initially $C := \emptyset$, $i := 1$ and $t_0 := \infty$.

FCP (at iteration *i*): For each vertex v, include v in the first color class $C_j \in C$, if one exists, such that $C_j \cup \{v\}$ is an independent set. Let U be the set of vertices not included in a color class. Find a vertex coloring $(C_1, C_2, \dots, C_k)$ of $G(U)$ (using COLOR or DSATUR), and set $C := C \cup \{C_1, C_2, \dots, C_k\}$ and $t_i := |C|/i$. If $t_i < t_{i-1}$ then set $i := i+1$ and repeat, otherwise return the upper bound $\lfloor t_{i-1} \rfloor$.

To prove a complexity result for FCP the authors amend the stopping rule so that the number of color classes $|C|$ does not exceed the number of vertices $|V|$. Our implementation also includes this feature. Note that for many graphs a tighter upper bound can be calculated by reiterating the algorithm after either stopping condition is satisfied. By $\mathrm{FCP_C}$ and $\mathrm{FCP_D}$ we refer to FCP with COLOR and DSATUR determining vertex colorings respectively. The comparison of these heuristics in [6, 19] show that the improvements in upper bound by $\mathrm{FCP_C}$ over COLOR range from 0 to 21 colors, and for $\mathrm{FCP_D}$ over DSATUR the improvements range from 0 to 7 colors.

3. The maximum clique algorithm

We now present our branch and bound algorithm MC for the maximum clique problem, which uses the FCP heuristic to determine upper bounds, and, like the algorithms of [11, 15], activates exactly one new search tree node at each branching stage. Given a graph $G = (V, E)$ algorithm MC maintains the following conditions:

- If h is the depth of the search tree the set $\{v_1, v_2, \dots, v_{h-1}\}$ consists of pairwise adjacent vertices.
- M is the largest clique found by the algorithm; $h - 1 \leqslant |M| \leqslant \omega(G)$.
- For $1 \leqslant i \leqslant h$, the vertex set $S_i \subseteq \bigcap_{j=1}^{i-1} N_G(v_j)$ consists of candidates for enlarging $\{v_1, v_2, \dots, v_{i-1}\}$ into a clique.
- For $1 \leqslant i \leqslant h$, $(C_1^i, C_2^i, \dots, C_{k_i}^i)$ is a vertex coloring of $G(S_i)$. Both k_i and k_i' (determined by FCP) are upper bounds for $\omega(G(S_i))$, with $k_i' \leqslant k_i$.
- An active node of the search tree corresponds to the subproblem of finding a maximum clique larger than M of the subgraph:

$$G_i = G(\{v_1, v_2, \dots, v_{i-1}\} \cup S_i), \quad \text{for } 1 \leqslant i \leqslant h.$$

Clearly, $\omega(G_i) \leqslant i - 1 + k_i' \leqslant i - 1 + k_i$.

Algorithm MC

```
Step 0 (Initialisation): Find a maximum clique $M$ of
    an edge-maximal triangulated subgraph of $G$.
    Set $h := 1$, $S_h := V$ and go to Step 2.
Step 1 (Calculate Lower Bound): Find a clique $Q$ of
    $G(S_h)$.
    If $h - 1 + |Q| > |M|$ then set $M := \{v_1, v_2, \dots, v_{h-1}\} \cup Q$.
    Go to step 2.
Step 2 (Calculate Upper Bound): Find a vertex coloring $(C_1^h, C_2^h, \dots, C_{k_h}^h)$ of $G(S_h)$.
    If $h - 1 + k_h \leqslant |M|$ then go to Step 4.
    Apply FCP to $G(S_h)$ to obtain a further upper
    bound $k_h' \geqslant \omega(G(S_h))$.
    If $h - 1 + k_h' \leqslant |M|$ then go to Step 4.
    Go to Step 3.
Step 3 (Branching): Choose a vertex $v_h \in C_{k_h}^h$ with
    maximum degree in $G$.
    Set $S_{h+1} := S_h \cap N_G(v_h)$, $S_h := S_h \backslash \{v_h\}$,
    $C_{k_h}^h := C_{k_h}^h \backslash \{v_h\}$.
    If $C_{k_h}^h = \emptyset$ then decrement $k_h$ and if $k_h < k_h'$
    then set $k_h' := k_h$.
    Increment $h$ and go to Step 1.
Step 4 (Backtracking): If $h = 1$ then stop: $M$ is a
    maximum clique of $G$.
    Decrement $h$ and if $h - 1 + k_h' \leqslant |M|$ then go
    to Step 4.
    Go to Step 3.
```

In line 2 of Step 3, the problem of finding a maximum clique of G_h is divided into two sub-problems. If v_h is a vertex of $G(S_h)$ then a clique Q of G_h will be contained in either:

$$G_{h+1} = G(\{v_1, v_2, \dots, v_h\} \cup (S_h \cap N_G(v_h))) \quad \text{if } v_h \in Q$$

or

$$G_h = G(\{v_1, v_2, \dots, v_{h-1}\} \cup (S_h \backslash \{v_h\})) \quad \text{if } v_h \notin Q.$$

We choose v_h from the final color class $C_{k_h}^h$ as the latter color classes generated by COLOR and by DSATUR tend to be smaller than the initial ones. Therefore the upper bound k_h is reduced more quickly than if an arbitrary vertex in S_h was chosen. Note that since $|M| \geqslant h - 1$ and $h - 1 + k_h > |M|$ whenever the algorithm goes to Step 3, we have $k_h \geqslant 1$ at this stage, and therefore the color class $C_{k_h}^h$ must exist.

Table 1
Maximum clique finding algorithms – uniform random graphs

n	d	LB	$\lvert M\rvert$	CPU time (s)					Search tree nodes				
				MC_C	MC_D	$MC1_C$	$MC1_D$	BXB	MC_C	MC_D	$MC1_C$	$MC1_D$	BXB
100	10	3.7	3.9	0.222	0.428	0.160	0.280	0.432	24.3	18.7	24.8	18.7	23.1
100	20	4.7	5.1	0.363	0.755	0.277	0.523	0.752	38.1	33.9	40.6	34.7	39.2
100	30	5.6	6.3	0.959	1.590	0.422	0.883	1.390	53.4	45.6	79.2	52.9	50.3
100	40	6.9	7.6	1.325	3.148	0.613	1.508	3.020	109.8	82.4	165.6	102.8	89.1
100	50	8.1	9.1	2.515	6.894	1.478	3.780	6.458	254.1	198.7	344.5	234.9	201.9
100	60	10.4	11.6	5.497	14.18	1.932	6.860	14.87	468.4	328.7	707.5	405.8	365.4
100	70	12.8	14.8	14.31	36.85	3.445	18.08	38.38	1048	672.7	1705	893.4	698.1
100	80	18.0	20.0	35.43	92.84	6.525	46.46	88.62	1786	1253	2961	1696	1160
100	90	28.0	30.7	73.84	150.1	12.12	71.30	134.1	2126	1109	4043	1523	974.3
200	10	4.0	4.3	1.013	2.498	0.962	1.715	2.705	92.3	83.5	98.2	83.7	91.2
200	20	5.1	5.9	2.708	5.810	1.548	4.217	5.965	140.3	120.7	202.2	137.6	126.9
200	30	6.1	7.3	7.030	17.71	3.187	9.095	18.56	519.9	396.2	699.5	476.8	386.0
200	40	7.6	9.0	16.04	47.64	5.510	26.04	49.85	1539	1162	2011	1279	1317
200	50	10.0	11.1	57.49	161.5	12.68	81.31	168.1	4295	2810	6846	3622	2889
200	60	12.1	14.0	249.9	755.6	45.66	380.4	820.4	17 461	11 704	26 857	14 712	13 109
200	70	15.3	18.1	1993	5830	341.9	2945	5829	102 122	64 430	173 810	88 354	63 972

Theorem 1. *Given an undirected graph $G = (V, E)$ algorithm MC finds a maximum clique M of G.*

Proof: This result follows immediately from the observation that algorithm MC maintains the abovementioned conditions throughout the algorithm. □

To evaluate the effectiveness of the FCP heuristic as an upper bounding device for the maximum clique problem, we have also developed an algorithm MC1, which skips lines 3 and 4 of step 2, thus not using FCP to calculate a further upper bound. MC_C ($MC1_C$) uses CLIQUE to determine a clique in Step 1, and FCP_C (COLOR) to determine upper bounds in Step 2. MC_D ($MC1_D$) uses FCP_D (DSATUR) for these purposes.

4. Computational results

See [19] for a complete description of the implementation of our algorithms in GAP [17] on a Sun Sparcstation 10. We now compare the performance of algorithms MC_C, MC_D, $MC1_C$ and $MC1_D$ with existing algorithms for the maximum clique problem. By BXB we refer to a combination of the algorithms of [2, 6], the most efficient known algorithms for the maximum clique problem. BXB uses FCP_D to calculate lower and upper bounds at each search tree node, and uses branching rule II of [2], their best performing branching rule. The branching rules of Refs. [2, 6] (which is stated for weighted graphs) both generally activate more than one new search tree node.

Table 1 shows the average size of the lower bound determined at the root node (LB), the average size of a maximum clique ($\lvert M\rvert$), the average CPU time taken by each of the algorithms, and the average number of search tree nodes generated by each algorithm, for 10 uniform random graphs of size $n = \lvert V\rvert$ and edge probability (or *density*) $d = 2\lvert E\rvert/n(n-1)$. In Table 2 we compare the algorithms for a selection of the DIMACS benchmark graphs which were developed as part of the 1993 DIMACS Challenge (see Johnson and Trick Ref. [14]). They include non-uniform random graphs with relatively large clique sizes, and graphs which have arisen in coding theory, the Steiner Triple Problem, tilings of hypercubes, vertex cover problems and fault diagnosis. Table 2 shows the size n and density d of the graph, the CPU time taken by each algorithm, and the number of search tree nodes generated by each

Table 2
Maximum clique finding algorithms – DIMACS benchmark graphs

DIMACS Graph	n	d	$\|M\|$	CPU time (s)					Search tree nodes					
				MC_C	MC_D	$MC1_C$	$MC1_D$	BXB	MC_C	MC_D	$MC1_C$	$MC1_D$	BXB	BX
brock200_1	200	75	21	4911	15 186	805.2	7951	16 320	218 853	149 153	379 810	211 013	163 348	113 244
brock200_2	200	50	12	26.72	149.7	3.833	74.22	158.4	1790	3018	2594	3593	3018	2965
brock200_3	200	61	15	230.1	573.6	38.50	281.0	815.9	15 354	7818	24 113	10 113	12 717	8155
brock200_4	200	66	17	568.2	1926	92.95	931.5	1530	31 751	25 105	52 332	33 693	19 316	25 705
c-fat200-1	200	8	12	0.283	2.200	0.017	0.150	2.133	8	1	8	4	1	1
c-fat200-2	200	16	24	0.317	0.183	0.017	0.183	0.167	7	1	7	1	1	1
c-fat200-5	200	43	58	0.683	3.467	0.133	2.217	3.284	27	27	27	27	27	29
c-fat500-1	500	4	14	0.534	0.616	0.017	0.617	2.217	13	1	13	1	1	1
c-fat500-2	500	7	26	1.417	0.700	0.083	0.700	0.750	23	1	23	1	1	1
c-fat500-5	500	19	64	1.450	0.984	0.166	0.950	0.983	23	1	23	1	1	1
c-fat500-10	500	37	126	0.017	1.400	0.033	1.400	1.450	1	1	1	1	1	1
hamming6-2	64	90	32	0.017	0.050	0.001	0.067	0.066	1	1	1	1	1	1
hamming6-4	64	35	4	0.133	0.850	0.067	0.300	0.800	81	29	81	58	29	48
hamming8-2	256	97	128	0.017	0.733	0.001	0.750	0.717	1	1	1	1	1	1
hamming8-4	256	64	16	344.2	155.7	79.15	137.6	156.5	28 593	357	36 441	2045	357	373
hamming10-2	1024	99	512	0.050	10.57	0.066	10.47	12.28	1	1	1	1	1	1
johnson8-2-4	28	56	4	0.050	0.050	0.017	0.083	0.033	20	1	23	26	1	1
johnson8-4-4	70	77	14	0.533	0.300	0.183	0.534	0.300	115	1	115	19	1	1
johnson16-2-4	120	76	8	770.8	0.417	195.8	2046	0.384	190 084	1	256 099	355 522	1	1
keller4	171	65	11	113.1	256.5	18.45	137.5	256.7	6543	3700	12 829	5195	3700	4164
MANN_a9	45	93	16	0.617	1.033	0.100	0.384	1.017	46	19	60	20	19	23
MANN_a27	378	99	126	23 286	26 524	704.3	9753	25 549	39 087	8704	47 264	9874	8714	14 145
p_hat300-1	300	24	8	8.800	38.93	1.467	20.12	37.53	1032	819	1310	928	819	832
p_hat300-2	300	49	25	75.05	225.6	10.05	129.2	225.5	1888	1304	2801	1579	1304	1613
p_hat500-1	500	25	9	76.48	384.8	13.72	231.4	389.5	7454	6179	9772	6724	6179	6105
p_hat500-2	500	50	36	2695	9790	267.1	5796	6320	35 657	27 182	59 393	34 787	17 019	31 746
p_hat700-1	700	25	11	272.8	1915	40.32	1060	1408	17 629	19 337	25 805	23 150	15 310	14 040
p_hat1000-1	1000	24	10	1883	13 060	283.2	6974	13 150	122 182	90 607	179 082	111 897	91 159	93 004
san200_0.7_1	200	70	30	6.617	36.37	0.917	18.85	95.73	53	231	206	348	645	635
san200_0.7_2	200	70	18	3.700	20.80	0.466	10.65	36.53	110	154	195	182	363	852
san200_0.9_1	200	90	70	73.75	45.72	11.48	24.92	255.4	715	121	2069	201	631	10
san200_0.9_2	200	90	60	5988	612.6	1052	348.0	2036	71 114	1553	211 889	2365	5655	1825
san400_0.5_1	400	50	13	51.03	81.73	11.22	64.83	247.7	1223	378	3465	523	1689	1194
san400_0.7_1	400	70	40	1681	2455	198.7	1430	10 263	15 903	5604	38 989	8507	30 707	20 913
san400_0.7_2	400	70	30	36 486	39 100	6228	24 285	66 579	690 806	139 092	1 591 030	231 593	295 314	75 773
san1000	1000	50	15	2281	32 630	653.9	40 814	9277	43 623	44 408	106 823	78 698	12 996	21 897
sanr200_0.7	200	70	18	1711	4608	338.2	2372	4076	87 012	51 610	150 861	71 799	44 278	40 496
sanr400_0.5	400	50	13	2352	9094	350.9	4955	8617	155 285	115 210	233 381	136 636	114 208	112 932

Table 3
Maximum clique finding algorithms – uniform random graphs with $n = 100$ and $d = 90\%$

$\lvert M\rvert$	CPU time (s)					Search tree nodes				
	MC_C	MC_D	$MC1_C$	$MC1_D$	BXB	MC_C	MC_D	$MC1_C$	$MC1_D$	BXB
29	160.0	263.1	26.34	122.5	285.8	4957	2014	9854	2721	2216
30	66.09	158.2	10.38	74.80	134.4	1885	1183	3259	1620	966
31	53.27	94.34	9.740	45.92	79.52	1392	643.5	2815	938.5	522
32	50.80	138.0	7.809	66.97	92.19	1323	1005	2465	1372	623
33	12.03	36.32	2.300	17.90	22.80	256	217	391	307	123

algorithm. Column BX refers to the number of search tree nodes for the algorithm of Ref. [6] as stated in their paper. To accurately compare algorithms we use the values presented in Ref. [6] for the lower bound at the root node for each of the tested algorithms.

In most cases those algorithms MC_D, BXB and BX which use the upper bound heuristic FCP_D, generate the least number of search tree nodes. MC_D on average generates less search tree nodes than BXB for 12 of the 16 sets of random graphs. For 12 of the DIMACS benchmark graphs the lower bound and upper bound calculated at the root node by these algorithms are equal, and therefore only one search tree node is generated. Of the other 26 DIMACS benchmark graphs, MC_D uses the least search tree nodes of these algorithms 15 times, BXB 10 times, and BX 8 times.

Those algorithms which use the vertex coloring heuristic COLOR, while generating the most search tree nodes, are generally the fastest. In particular, for the random graphs, $MC1_C$ is the fastest of the tested algorithms, using on average only 18.41% of the CPU time used by BXB. $MC1_C$ is again the fastest for all but four of the DIMACS benchmark graphs (and for two of these the difference is only a few microseconds). We have also implemented a variant $MC2_C$ of $MC1_C$ which only finds a lower bound at the root node of the search tree. For the random graphs (DIMACS benchmark graphs) this algorithm uses 0.65% (0.20%) more search tree nodes than $MC1_C$, yet is on average 4.34% (12.04%) faster than $MC1_C$. This indicates that the determination of lower bounds at non-root nodes is not time-efficient.

We have observed that for graphs with fixed size and density the difficulty of the maximum clique problem is generally inversely proportional to the size of a maximum clique in the graph. This is apparent for the *san* graphs with equal n and d. Similar results occur with the random graphs. For example, the 10 uniform random graphs (used in Table 1) with $n = 100$ and $d = 90\%$ have a maximum clique of size 29(2), 30(3), 31(2), 32(2) or 33(1). For each maximum clique size, Table 3 shows the average CPU time taken, and the average number of search tree nodes generated by each algorithm.

Acknowledgements

The author wishes to thank Graham Farr, Cheryl Praeger, Leonard Soicher and an anonymous referee for their helpful discussion and advice. Thanks also to Christopher Pickett for the use of his computer.

References

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, M. Szegedy, Proof verification and intractability of approximation problems, Proc. 33rd IEEE Symp. on Foundations of Computer Science, 1992, pp. 13–22.

[2] L. Babel, Finding maximum cliques in arbitrary and in special graphs, computing 46 (1991) 321–341.

[3] L. Babel, G. Tinhofer, A branch and bound algorithm for the maximum clique problem, Meth. Oper. Res. 34 (1990) 207–217.

[4] E. Balas, A fast algorithm for finding an edge-maximal subgraph with a TR-formative coloring, Discrete Appl. Math. 15 (1986) 123–134.

[5] E. Balas, J. Xue, Minimum weighted coloring of triangulated graphs, with application to maximum weight vertex packing and clique finding in arbitrary graphs, SIAM J. Comput. 20(2) (1991) 209–221.

[6] E. Balas, J. Xue, Weighted and unweighted maximum clique algorithms with upper bounds from fractional coloring, Algorithmica 15 (1996) 397–412.

[7] E. Balas, C.S. Yu, Finding a maximum clique in an arbitrary graph, SIAM J. Comput. 15(4) (1986) 1054–1068.

[8] N. Biggs, Some Heuristics for Graph Coloring, in: R. Nelson, R.J. Wilson, (Eds.), Graph Colourings, Longman, New York, 1990, pp. 87–96.

[9] D. Brelaz, New methods to color the vertices of a graph, Comm. ACM 22 (1979) 251–256.

[10] C. Bron, J. Kerbosch, Finding all cliques of an undirected graph, Comm. ACM 16(9) (1973) 575–577.

[11] R. Garraghan, P.M. Pardalos, An exact algorithm for the maximum clique problem, Oper. Res. Lett. 9 (1990) 375–382.

[12] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.

[13] M. Grötschel, L. Lovász, A. Schrijver, Polynomial algorithms for perfect graphs, Ann. Discrete Math. 21 (1984) 325–356.

[14] D.S. Johnson, M.A. Trick (Eds.), Cliques, Coloring, and Satisfiability: Second DIMACS Challenge. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, Providence, 1996.

[15] P.M. Pardalos, G.P. Rodgers, A branch and bound algorithm for the maximum clique problem, Comput. Oper. Res. 19(5) (1992) 363–375.

[16] P.M. Pardalos, J. Xue, The maximum clique problem. J. Global Optim. 4 (1994) 301–328.

[17] M. Schönert et al., GAP-Groups, Algorithms and Programming, Lehrstuhl D für Mathematik, Rheinisch Westfälische Technische Hochschule, Aachen, Germany, 1995.

[18] R.E. Tarjan, A.E. Trojanowski, Finding a maximum independent set, SIAM J. Comput. 6(3) (1977) 537–546.

[19] D.R. Wood, An algorithm for finding a maximum clique in a graph, Technical Report 96/260, Department of Computer Science, Monash University, Australia, 1996, available at ftp.cs.monash.edu.au.