SIAM J. COMPUT.
Vol. 2, No. 4, December 1973

AN ALGORITHM FOR DETERMINING THE CHROMATIC NUMBER OF A GRAPH*

D. G. CORNEIL AND B. GRAHAM†

Abstract. A heuristic algorithm for the determination of the chromatic number of a finite graph is presented. This algorithm is based on Zykov's theorem for chromatic polynomials, and extensive empirical tests show that it is the best algorithm available. Christofides' algorithm for the determination of chromatic number is described and is used in the comparison tests.

Key words. chromatic number, coloring algorithm, Zykov's theorem

1. Introduction. This paper presents an algorithm for the determination of the chromatic number of a graph. Empirical evidence indicates that this algorithm is the best available.

The chromatic number of a graph G, denoted $\chi(G)$, is the minimum number of colors required to color the graph such that no two adjacent vertices have the same color. Christofides [1] and Welsh and Powell [10] mention several practical applications of this property of a graph. For example, it can be used to solve an examination scheduling problem; in this application each examination is represented by a node of a graph and an edge joining two vertices indicates that because of a candidate's conflict, the corresponding examinations may not be held at the same time. The chromatic number of this graph is the minimum number of periods required to hold all examinations. In addition, the chromatic number problem belongs to the Cook–Karp class [5] which also includes the subgraph isomorphism, the Hamiltonian circuit and the maximal clique problems. Problems in the Cook–Karp class are polynomially equivalent, i.e., either each of them possesses a polynomially bounded algorithm or none of them does.

The best known algorithms for the determination of chromatic number are Christofides' algorithm [1] and an algorithm based on a corollary to Zykov's theorem for chromatic polynomials [13].[1] As mentioned by Hedetniemi [4], no extensive work has been done to compare these two methods.

Our algorithm is a modification and extension of the Zykov algorithm. We also present the results of extensive empirical tests [3] which were conducted to compare the storage and time requirements of our algorithm with that of Christofides.

2. Our algorithm.

2.1. The Zykov algorithm. As mentioned in the Introduction, our algorithm is a modification and extension of the Zykov algorithm. Before describing Zykov's method, we introduce the concept of reduction of a graph.

* Received by the editors March 13, 1973. This research was supported by the National Research Council of Canada.

† Department of Computer Science, University of Toronto, Toronto, Canada.

[1] This algorithm will be referred to as Zykov's algorithm.

DEFINITION. Let $G(V, E)$ be a graph with nonadjacent vertices x and y. The *reduced graphs* of G are G'_{xy} and G''_{xy}, where

(i) $G'_{xy} = G'_{xy}(V', E')$; $V' = V$; $E' = E + (x, y)$ (i.e., G'_{xy} is obtained from G by adding the edge (x, y)).

(ii) $G''_{xy} = G''_{xy}(V'', E'')$; $V'' = V - y$;

$$(i,j) \in E'' \quad \text{iff} \quad (i,j) \in E \quad \text{for} \quad i,j \neq x,$$

$$(i,x) \in E'' \quad \text{iff} \quad (i,x) \in E \quad \text{or} \quad (i,y) \in E,$$

(i.e., G''_{xy} is obtained from G by coalescing (identifying) the vertices x and y).

For example, Fig. 1 shows a graph G together with a pair of reduced graphs.

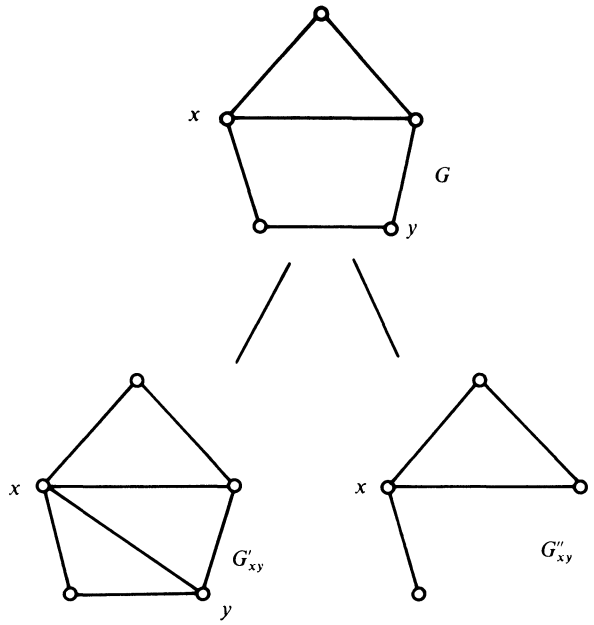

FIG. 1

Obviously any graph which is not complete can be reduced; a complete graph is irreducible, and the chromatic number of the irreducible graph K_α is α. If either G'_{xy} or G''_{xy} is reducible, the process can be continued until all graphs obtained are irreducible; at this stage G is said to be *completely reduced*, and

$$R(G) = \{G_1, G_2, \cdots, G_s\} \tag{1}$$

is the set of irreducible graphs thus obtained. Also let

$$R'(G) = \{G'_1, G'_2, \cdots, G'_{s'}\} \tag{2}$$

denote a set of graphs produced at some intermediate stage of the reduction process. For example,

$$R'(G) = \{G'_{xy}, G''_{xy}\}$$

after just one reduction step.

Theorem 1 below is the corollary to Zykov's theorem [13] on which the Zykov algorithm is based.

THEOREM 1. *If x and y are nonadjacent vertices in G, then* $\chi(G) = \min[\chi(G'_{xy}), \chi(G''_{xy})]$.

Proof. For a proof of this theorem, see [3].

Theorem 1 can be applied recursively, so when G is completely reduced to $R(G)$ (cf. (1)), we have

$$\chi(G) = \min[|G_1|, |G_2|, \cdots, |G_s|]. \tag{3}$$

Similarly when G is partially reduced to $R'(G)$ (cf. (2)), we have

$$\chi(G) = \min[\chi(G'_1), \chi(G'_2), \cdots, \chi(G'_{s'})]. \tag{4}$$

Figure 2 shows an application of the Zykov algorithm. We note from the figure that the algorithm has produced a binary tree (the Zykov tree) where each node of the tree is a graph.

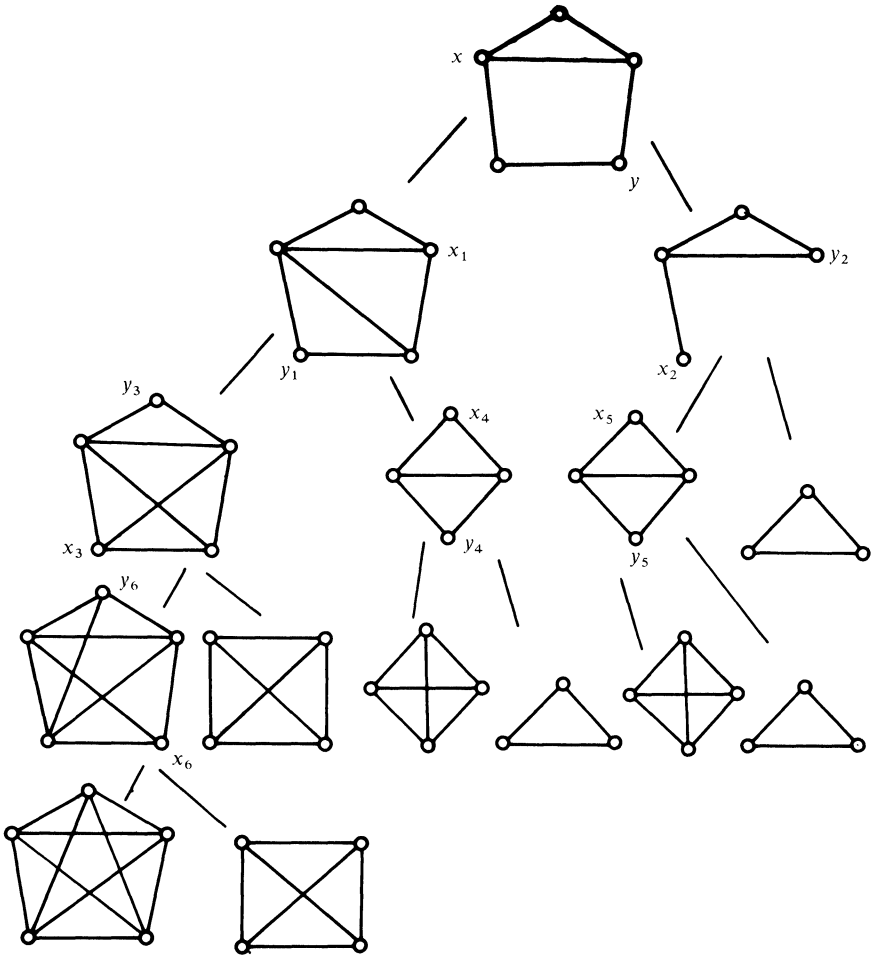

FIG. 2. $\chi(G) = \min[5, 4, 4, 4, 3, 4, 3, 3] = 3$

2.2. The improved algorithm. We will now show how the number of reduction steps required by the Zykov algorithm may be reduced by using a branch and bound approach [7]. In our case the branching is the reduction of G to G'_{xy} and G''_{xy}. We now show how the branching is bounded. Consider Fig. 3, an intermediate stage in the development of Fig. 2. At this stage we know from (4) that

$$\chi(G) = \min [\chi(A), \chi(B), \chi(C)].$$

Graph C is complete (K_3), so $\chi(C) = 3$ and 3 is thus an upper bound for $\chi(G)$; note that each of A and B contains a 3-clique, so that 3 is a lower bound for $\chi(A)$ and $\chi(B)$. Consequently, branching from A and B is unnecessary; we conclude that $\chi(G) = 3$ without finding $\chi(A)$ or $\chi(B)$. In general whenever α has been established as an upper bound for $\chi(G)$, then a graph encountered in the reduction of G which is known to contain an α-clique need not be reduced further. This pruning of the Zykov tree is the essence of bounding and is a major step in the development of our algorithm.

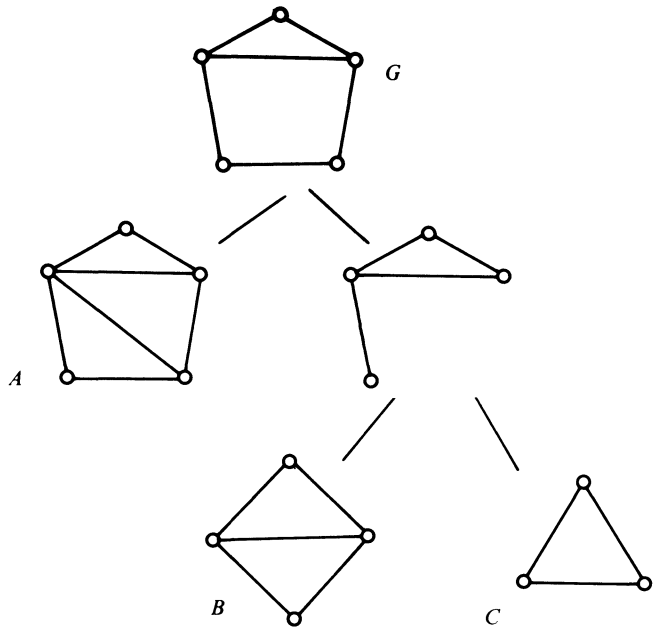

FIG. 3

The basic bounding strategy, once α (an upper bound for $\chi(G)$) has been obtained, is now obvious. It would, however, be poor practice to determine whether a reducible graph H contains an α-clique since the clique finding problem is in the Cook–Karp class mentioned in the Introduction. Instead the strategy is to find an α-cluster[2] in H, where an α-cluster is a set of α vertices which has a high density of edges; let H_α denote the subgraph of H determined by this α-cluster. Next H is reduced to H' and H'' such that H' is formed by adding an edge to H_α.

[2] The present version of the algorithm uses an $O(n^3)$ threshold-matrix method for finding clusters. This method is fully described in [3].

This process is continued with H' until the α-cluster becomes an α-clique, whereupon this branch is terminated. Graphs formed by coalescence (e.g. H'') are treated similarly.

If α is the exact value of $\chi(G)$, then the above procedure will always terminate successfully by building α-cliques. However, if $\alpha > \chi(G)$, then at some stage a graph introduced by coalescence will have only $\alpha - 1$ vertices, implying that $\chi(G) \leqq \alpha - 1$. Whenever this occurs, the value $\alpha - 1$ is substituted for α and the execution of the algorithm continues uninterrupted (i.e., a decrease of α does not require a repetition of any of the previous steps). It terminates when every branch of the Zykov tree introduced by reduction has been forced to contain an α-clique. Upon termination, the value of $\chi(G)$ will be exactly α.

We now present a concise recursive definition of the algorithm. Its most important feature is the procedure Reduce (which might be more aptly labeled "Reduce-if-necessary") which has two parameters: H, a graph, and N, the order of H. Note that the algorithm essentially performs a preorder traversal [6, p. 316] of a pruned Zykov tree such as the one in Fig. 3.

2.3. Recursive definition of our algorithm.

```
Main Program: (α is a global variable)
      Find Initial α, an upper bound for χ(G);[3]
      Reduce (G, n);
      Stop (now χ(G) = α).
Procedure Reduce (H, N);
      if α > N then α := N;
      β := α; (β is a local variable)
      if H is complete then GO TO EXIT;
      Find H_α, a cluster on α-vertices;
A:    if H_α is an α-clique then GO TO EXIT;
      choose nonadjacent vertices x and y in H_α;
      Form H'_xy and H''_xy by reduction of H;
      H := H'_xy;
      Reduce (H''_xy, N − 1);
      If β = α then GO TO A
                else REDUCE (H, N);
      (the else is necessary in case α was changed during the previous step)
EXIT: END;
```

3. Christofides' algorithm. The Christofides' algorithm [1] depends strongly on the concept of a *Maximal Internally Stable Set* (or MISS) which is defined as follows:

A MISS of a graph is a set of vertices, no two of which are adjacent, and which is maximal with respect to this property. Thus a MISS in G is a clique in $\bar{G}$, the complement of G.

As shown in [8], for some graphs, the number of MISS grows exponentially with n. For example, a graph consisting of t disjoint triangles will have 3^t MISS.

[3] There are several methods for finding an initial upper bound for $\chi(G)$, for example $\alpha := n$. However in [3] and [11] methods for finding a good upper bound for $\chi(G)$ are described.

Having found all the MISS of the graph, Theorem 2 is used to find its chromatic number.

THEOREM 2 (Christofides [1]). *If a graph is r-chromatic, then it can be colored with r colors, coloring first with one color a* MISS *of G, say* M_i, *next coloring with another color a* MISS *of* $G - M_i$ *and so on until all the vertices are colored.*

Unfortunately, Theorem 2 does not determine which MISS should be colored at each step, so every MISS must be considered. A modification introduced by Furtado et al [2], while not eliminating the exponential nature of the algorithm, reduces the number of MISS investigated. Whenever a new color is introduced, they concentrate on coloring only one of the uncolored vertices, and consequently they consider only the MISS which contain this vertex. For obvious reasons, the vertex which appears in fewest MISS is selected. They also suggest that it is necessary to use the MISS-finding algorithm only once, since the MISS of G can be used to find the MISS of $G - M_i$.

4. Tests and results. Both our algorithm and Christofides' algorithm were programmed in ALGOL-W [12] and run on an IBM 370 model 165 computer. The most efficient available version of Christofides' algorithm [2] was used for the tests. In order to improve the speed of this algorithm, recursive programming and dynamic storage techniques were not employed.

4.1. Storage comparison. It has been shown [3] that any program based on Christofides' method may require an amount of storage which grows exponentially with n. By use of dynamic storage allocation the maximum storage required by our algorithm is bounded by $O(n^3)$.

4.2. Timing comparison. An extensive empirical comparison of the time requirements of the two algorithms was made [3]. For the test, the following three families of graphs were used.

(i) *Pseudo-random graphs* $G(n, \rho)$.

A pseudo-random graph $G(n, \rho)$ has n vertices and $(n/2)(n-1)\cdot\rho$ edges. The location of the edges within the graph is determined by some pseudo-random mechanism.

(ii) *Modified Moon–Moser graphs* $G(n)$.

Modified Moon–Moser graphs are a family of graphs with $n = 0 \pmod 3$ vertices. Each graph is formed by constructing $n/3$ disjoint triangles and then adding $n/3 - 1$ additional edges to form a chain of triangles.

(iii) *Starred polygons* $G(n, s)$ [9].

A starred polygon $G(n, s)$ is defined as follows:

$$V = \{v_1, v_2, \cdots, v_n\},$$

$$E = \{(v_i, v_k), k = (i + j) \text{ modulo } n, 1 \leqq i \leqq n, 1 \leqq j \leqq s\}.$$

Approximately two hundred test graphs were used. Our algorithm paid the overheads of recursion and dynamic storage allocation; these techniques were not employed in the tested version of Christofides' algorithm.

4.3. Results. Tables 1–3 are representative of the results obtained from the three families of graphs tested. From these tests we conclude:

TABLE 1

Observed tine and storage requirements on pseudo-random graphs

		Times[4]		Core used[5]	
ρ	n	C–G	Christofides	C–G	Christofides
4	10	3.0	3.3	22.0	29.2
	15	11.7	32.7	57.6	94.2
	20	66.3	608.7	268.8	653.0
	25	354.2	6324.2	720.0	4197.5
	30	972.3	—	830.8	—
	35	12785.0	—	1620.0	—
.8	10	1.3	2.7	26.4	23.4
	15	6.0	11.0	38.4	41.8
	20	16.7	27.3	75.6	61.6
	25	177.7	361.3	291.2	308.0
	30	845.3	4909.7	570.4	3323.2
	35	2006.0	—	950.4	—
	40	7825.0	—	1284.7	—

[4] 1 time unit = .01 sec.
[5] 1 unit of core = 1 word = 32 bits.

(i) For all graphs tested, the execution time of our algorithm was either significantly less than that of Christofides or at worst approximately equal to it.

(ii) With one exception, the asymptotic behavior (i.e., the slope of the log (time) vs. n plots) of our algorithm was superior to that of Christofides' algorithm. The exception was for pseudo-random graphs with a high density ($\rho = 0.9$) of edges; in this case the plots had approximately equal slopes.

(iii) For low values of n, the storage requirements of the algorithms are approximately equal, but for larger graphs, Christofides' method requires more storage. This reflects the result that the storage requirement of our algorithm is bounded by $O(n^3)$, while that of Christofides can grow exponentially.

TABLE 2

Observed time and storage requirements on modified Moon–Moser graphs

	Time		Storage	
n	C–G	Christofides	C–G	Christofides
3	0.0	0.0	8	12
6	1.7	1.7	14	25
9	3.3	8.3	20	60
12	5.0	56.7	26	166
15	8.3	705.0	32	533
18	13.3	8781.7	38	2014
21	18.3	—	44	—
—	—	—	—	—
60	278.3	—	122	—
63	320.0	—	128	—

TABLE 3
Observed time and storage requirements on starred polygons with s = 1 (i.e., cycles)

	Time		Core Used	
n	C-G	Christofides	C-G	Christofides
3	0.0	0.0	8	12
6	1.7	1.7	14	19
9	5.0	5.0	80	42
12	5.0	20.0	26	97
15	16.7	138.3	224	281
18	11.7	1235.0	38	778
21	48.3	11593.3	440	2862
24	23.3	—	50	—
27	101.7	—	728	—
—	—	—	—	—
54	185.0	—	110	—
60	243.0	—	122	—
63	1320.0	—	3968	—

REFERENCES

[1] N. CHRISTOFIDES, *An algorithm for the chromatic number of a graph*, Comput. J., 14 (1971), pp. 38–39.

[2] A. FURTADO, S. ROSCHKE, C. DOS SANTOS AND J. BAUER, *Finding the optimal colourings of a graph*, unpublished report, Departmento de Informatica Pontificia Universidade Catolica do Rio de Janeiro, Brazil.

[3] B. GRAHAM, *An algorithm to determine the chromatic number of a graph*, M.Sc. thesis, Tech. Rep., no. 47, Dept. of Computer Sci., Univ. of Toronto, Toronto, Canada, 1972.

[4] S. T. HEDETNIEMI, Rev. no. 22063, Comput. Rev., 12 (1971), pp. 446–447.

[5] R. M. KARP, *Reducibility among combinatorial problems*, Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–104.

[6] D. E. KNUTH, *The Art of Computer Programming*, vol. 1, Addison-Wesley, Reading, Mass., 1968.

[7] E. L. LAWLER AND D. E. WOOD, *Branch and bound methods: A survey*, Operation Res., 14 (1966), pp. 699–719.

[8] J. W. MOON AND L. MOSER, *On cliques in graphs*, Israel J. Math., 3 (1965), pp. 23–28.

[9] J. TURNER, *Point symmetric graphs with a prime number of points*, J. Combinatorial Theory, 3 (1967), pp. 136–145.

[10] D. J. A. WELSH AND M. B. POWELL, *An upper bound for the chromatic number of a graph and its application to large scale time-tabling problems*, Comput. J., 10 (1967), pp. 85–86.

[11] M. R. WILLIAMS, *The colouring of very large graphs*, Combinatorial Structures and their Applications, R. Guy et al., eds., Gordon and Breach, New York, 1970, pp. 477–478.

[12] N. WIRTH AND C. A. R. HOARE, *A contribution to the development of ALGOL*, Comm. ACM, 9 (1966), pp. 413–431.

[13] A. A. ZYKOV, *On some properties of linear complexes*, Mat. Sb., 24 (1949), pp. 163–188; English transl., Amer. Math. Soc. Translation no. 79, 1952.