ALGORITHMS FOR COMPUTING THE CHROMATIC POLYNOMIAL

D. R. Shier
College of William and Mary
Williamsburg, VA

N. Chandrasekharan
Clemson University
Clemson, SC

Abstract

The chromatic polynomial captures a good deal of combinatorial information about a graph, describing its acyclic orientations, its all terminal reliability, its spanning trees, as well as its colorings. Several methods for computing the chromatic polynomial of a graph G construct a computation tree for G whose leaves are "simple" base graphs for which the chromatic polynomial is readily found. Previously studied methods involved base graphs which are complete graphs, completely disconnected graphs, forests, and trees. In this paper we consider chordal graphs as base graphs. Algorithms for computing the chromatic polynomial based on these concepts are developed, and computational results are presented.

1. INTRODUCTION

In this section we review properties of the chromatic polynomial and various representations for this polynomial. Throughout, $G = (V, E)$ will be an undirected graph with vertex set $V = I_n = \{1, 2, \dots, n\}$ and edge set E, with $n = |V|$ and $m = |E|$. A *proper coloring* of G is simply a function ϕ from V into a set of colors such that $\phi(v) \neq \phi(w)$ whenever $(v,w) \in E$. For a given integer $\lambda \geq 0$, the *chromatic polynomial* $P(G;\lambda)$ is the number of distinct proper colorings of G using at most λ colors. For example, $P(\bar{K}_n;\lambda)$ is simply the total number of functions from I_n to I_λ, whereas $P(K_n;\lambda)$ counts the number of injective functions from I_n to I_λ. Thus,

$$P(\bar{K}_n;\lambda) = \lambda^n \, , \tag{1.1}$$

$$P(K_n;\lambda) = \lambda^{(n)} = \lambda(\lambda-1) \cdots (\lambda-n+1) \, . \tag{1.2}$$

If m_k denotes the number of ways to properly color G using exactly k colors, then it is clear that

$$P(G;\lambda) = \sum_{k=0}^{n} m_k \binom{\lambda}{k} = \sum_{k=0}^{n} \frac{m_k}{k!} \lambda^{(k)} \, , \tag{1.3}$$

showing that $P(G;\lambda)$ is indeed a polynomial. For example, when $G = \bar{K}_n$ then m_k counts the number of surjective functions from I_n to I_k. Using (1.1), (1.3), and the fact that the Stirling numbers of the second kind $S_{n,k}$ satisfy $\lambda^n = \Sigma\, S_{n,k}\, \lambda^{(k)}$, we obtain the well-known result that the number of surjective functions from I_n to I_k is $m_k = k!\, S_{n,k}$.

Not only does $P(G;\lambda)$ have combinatorial significance as a generalized way of counting functions, but its coefficients have a number of interesting interpretations. Since the coefficients of $P(G;\lambda)$, when expressed as a polynomial in powers of λ, are known to alternate in sign [9], we can write the chromatic polynomial as

$$P(G;\lambda) = \sum_{i=1}^{n} (-1)^{n-i} a_i \lambda^i , \qquad (1.4)$$

where all $a_i \geq 0$. It is easy to show [9] that $a_n = 1$, $a_{n-1} = m$ (the number of edges), and that the smallest index k for which $a_k > 0$ is the number of connected components of G. Recently, it has been demonstrated that a_1 is the "all terminal domination" of G, a graph invariant important in network reliability [1]. Whitney [18] showed that the coefficient a_i also has the interpretation as the number of spanning forests of G with i components and having "external activity" 0. Another interesting aspect of these coefficients is that Σa_i counts the number of acyclic orientations of G [15].

An alternative representation of the chromatic polynomial in "factorial form" is given by (1.3) or

$$P(G;\lambda) = \sum_{i=1}^{n} b_i \lambda^{(i)} , \qquad (1.5)$$

where the $b_i \geq 0$. It is known that $b_n = 1$, $b_{n-1} = n(n-1)/2 - m$, and that the smallest index k for which $b_k > 0$ is the chromatic number of G. The coefficient b_i has an interpretation as the number of partitions of V into i disjoint subsets, each of which forms an independent set [9]. Equivalently, b_i is the number of clique covers of $\overline{G}$ using i cliques, whence Σb_i counts the number of clique covers of $\overline{G}$.

A third representation of $P(G;\lambda)$ is the Tutte polynomial form [17], given by

$$P(G;\lambda) = \sum_{i=1}^{n-1} (-1)^{n-i+1} c_i \lambda(\lambda-1)^i , \qquad (1.6)$$

with $c_i \geq 0$. As discussed in [3], $c_{n-1} = 1$, $c_{n-2} = m-n+1$ (the cycle rank of G), and the smallest index k for which $c_k > 0$ is the number of blocks in G. Tutte has shown that c_i is the number of spanning trees with external activity 0 and internal activity i. Notice also that $\Sigma c_i = a_1$ is the all terminal domination of G.

2. COMPUTING THE CHROMATIC POLYNOMIAL

Since computing the chromatic polynomial for an arbitrary graph is known to be NP-hard, effective algorithms for determining $P(G;\lambda)$ are unlikely to be found. However, it is possible to express the chromatic polynomial of G in terms of the chromatic polynomials of related graphs. Namely, if G+e, G-e, and G/e represent the graphs obtained from G by adding, deleting, and contracting edge e, then the following relations can be easily established [9]:

$$P(G;\lambda) = P(G+e;\lambda) + P(G/e;\lambda), \quad (2.1)$$

$$P(G;\lambda) = P(G-e;\lambda) - P(G/e;\lambda). \quad (2.2)$$

Either equation (2.1) or (2.2) can be applied repeatedly as a means of obtaining the chromatic polynomial. For example, the evaluation of $P(G;\lambda)$ using (2.1) can be viewed as a certain *binary computation tree* with G at the root, G+e as its left child, and G/e as its right child. These two children in turn have left and right children, with the process continued until all resulting graphs are complete. Thus the leaves of this computation tree are complete graphs, whose chromatic polynomials are easily found via (1.2). Another interpretation of the coefficient b_i in (1.5) is then the number of leaf graphs on i vertices in this computation tree, clearly an invariant of the graph G. Alternatively, if the graph is relatively sparse it might be advisable to delete edges using (2.2), resulting in another binary computation tree, with each graph H having children H-e and H/e. If carried on to the very end, each leaf of this tree is a completely disconnected graph, whose chromatic polynomial is known from (1.1). Again, the number of leaves of "size" i (i vertices) in this computation tree is just the coefficient a_i in (1.4). On the other hand, it is not necessary to carry out the reduction in its entirety; rather one can terminate the process when each leaf is a forest [6] or a tree [7, 10]. The latter case is computationally more advantageous [3], and the number of leaves having size i+1 in this computation tree is simply the coefficient c_i in (1.6), since the chromatic polynomial of any tree on i+1 vertices is known to be $\lambda(\lambda-1)^i$.

As suggested in [3], the total number of leaves in a given computation tree for G is a reasonable measure for the complexity of the associated method of determining $P(G;\lambda)$. From the previous section, the number of leaves in these three computation trees are (a) the number of acyclic orientations of G (leaves $\leftrightarrow$ disconnected graphs), (b) the number of clique covers of $\overline{G}$ (leaves $\leftrightarrow$ complete graphs), and (c) the all terminal domination of G (leaves $\leftrightarrow$ trees). For the example graph shown in Figure 1, the associated representations of $P(G;\lambda)$ are:

(a) $P(G;\lambda) = \lambda^5 - 6\lambda^4 + 14\lambda^3 - 15\lambda^2 + 6\lambda$,

(b) $P(G;\lambda) = \lambda^{(5)} + 4\lambda^{(4)} + 3\lambda^{(3)}$,

(c) $P(G;\lambda) = \lambda(\lambda-1)^4 - 2\lambda(\lambda-1)^3 + 2\lambda(\lambda-1)^2 - \lambda(\lambda-1)$.

Thus the respective computation trees have 42, 8, and 6 leaves, and here the last method is clearly preferred for computational purposes.

Our objective here is to study a reduction method based on (2.2) in which the leaves are chordal graphs. Since chordal graphs span a range of sparsity from trees to complete graphs, they may more easily adapt to the sparsity or density of a given graph and thus might require fewer leaves (a computational advantage). Moreover, the chromatic polynomial of any chordal graph can be efficiently calculated, as discussed in the next section.

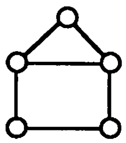
Figure 1

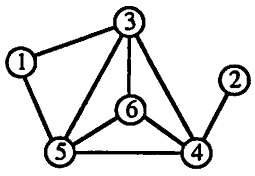

Figure 2

3. CHROMATIC POLYNOMIALS AND CHORDAL GRAPHS

A *chordal* (triangulated, rigid circuit) graph is one in which there are no induced cycles of length > 3. The class of chordal graphs includes trees, k-trees [8, 13], interval graphs [5], indifference graphs [11], and complete graphs, among others. Moreover, they find application in a number of areas including scheduling, location, evolutionary trees, acyclic relational databases, and sparse matrices. The last application area is especially noteworthy since chordal graphs correspond precisely to the zero-nonzero patterns of symmetric matrices that have a perfect elimination ordering: namely, an ordering so that no fill-in is introduced during Gaussian elimination [12].

More precisely an *ordering* of G is a bijection α: $I_n \rightarrow V$. Relative to this ordering, the *monotone adjacency set* of vertex v is defined as $MADJ(v) = \{w \in V: (v,w) \in E$ and $\alpha^{-1}(w) > \alpha^{-1}(v)\}$. A *perfect elimination ordering* is an ordering for which MADJ(v) is complete for all $v \in V$. It is well known that G is chordal if and only if it possesses a perfect elimination ordering [4]. Efficient algorithms for recognizing chordal graphs and generating perfect elimination orderings have been studied by [5, 14, 16].

Suppose that α is a perfect elimination ordering for G, and let $d_i = |MADJ(\alpha(i))|$. Then it can be established that the chromatic polynomial for G is simply

$$P(G;\lambda) = \prod_{i=1}^{n} (\lambda - d_i) . \qquad (3.1)$$

This result is proved by induction using the fact that the vertices adjacent to $\alpha(1)$ induce a complete subgraph in G. Moreover, (3.1) shows that the multiset $\{d_i\}$ is an invariant of any chordal graph G, independent of the particular ordering α, since the integers d_i are the roots of the chromatic polynomial. It is also worth pointing out that the all terminal domination of a chordal graph can be expressed as $\Pi\{d_i: i < n\}$. In the example of Figure 2, the vertices have been numbered so that 1, 2, ..., 6 forms a perfect elimination ordering, and $\{d_i\} = \{2, 1, 3, 2, 1, 0\}$. Thus $P(G;\lambda)$ is readily computed as $\lambda(\lambda-1)^2(\lambda-2)^2(\lambda-3)$.

If we now apply the reduction formula (2.2) to the graph of Figure 1, we obtain the binary computation tree in Figure 3, with only 2 leaves. It is then straightforward to compute $P(G;\lambda) = \lambda(\lambda-1)^3(\lambda-2) - \lambda(\lambda-1)(\lambda-2)^2$. In general, the use of chordal graphs as leaves of the computation tree may afford a reduction in the number of leaves, and hence the computational effort in determining the chromatic polynomial. The next section discusses two implementations of this method and presents empirical comparisons of the new method with existing techniques.

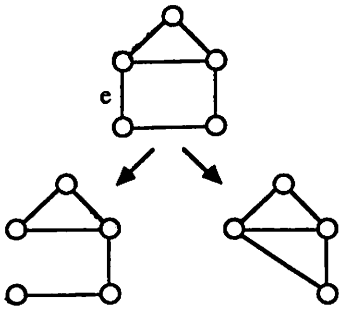

Figure 3

4. COMPUTATIONAL CONSIDERATIONS AND RESULTS

The major task in implementing the "chordal reduction" method discussed in the previous section involves determining whether a given graph H (a node in the computation tree) is chordal or if not what edge e should be deleted for use in (2.2). Efficient methods for determining if a graph is chordal [5] typically work "backward" and then "forward." That is, they first generate an ordering in reverse $\alpha(n), \alpha(n-1), \ldots, \alpha(1)$, after which a pass is made through the vertices in the natural order $\alpha(1), \alpha(2), \ldots, \alpha(n)$ to check whether this is indeed a perfect elimination ordering. In our case, however, we wish to determine the chordality of G as soon as possible (without separate backward and forward passes). The following algorithm accomplishes the task in a single pass. Here, card(v) records the cardinality of MADJ(v), relative to the vertices in S (those already ordered by α), and min(v) indicates a vertex $w \in$ MADJ(v) having minimum $\alpha^{-1}(w)$.

<u>Algorithm CARDINALITY</u>. Given a connected graph $G = (V, E)$ with $n = |V|$, this procedure either yields a perfect elimination ordering α (if G is chordal), or an edge e to be deleted (if G is not chordal).

```
1. Select $v_0 \in V$;  card(v) $\leftarrow 0$ for all $v \neq v_0$;  min($v_0$) $\leftarrow v_0$;
   $S \leftarrow \{v_0\}$;  $k \leftarrow n$;  $\alpha(k) \leftarrow v_0$.
2. m $\leftarrow$ min($v_0$);
   for all $u \in S$, $u \neq m$ with $(u,v_0) \in E$:
       if $(u,m) \notin E$ then **return** $e = (u,v_0)$.
```

```
3. for all $u \in V\text{-}S$ with $(u,v_0) \in E$:
       $card(u) \leftarrow card(u)+1$
       $min(u) \leftarrow v_0$.
4. if $k = 1$ then **return** $\{\alpha(1), \alpha(2), \ldots, \alpha(n)\}$, else $k \leftarrow k - 1$.
5. Select $v_0 \in V\text{-}S$ such that $card(v_0) = \max\{card(x): x \in V\text{-}S\}$;
   $\alpha(k) \leftarrow v_0$;  $S \leftarrow S \cup \{v_0\}$;  go to Step 2.
```

Step 5 selects the next vertex to be ordered based on having the maximum cardinality monotone adjacency set; as shown in [5, 14, 16] this ensures that the resulting order is a perfect elimination ordering when G is chordal. Step 3 updates the arrays *card* and *min* after vertex v_0 is added to the set S. Step 2 checks whether the ordering generated so far qualifies as a perfect elimination ordering relative to vertex v_0. It is important to note that we do not need to check here whether MADJ(v_0) is complete; rather it suffices to check whether each $u \in$ MADJ(v_0) other than $m = \min(v_0)$ is in fact adjacent to m. Using this fact, the algorithm can be implemented to have a worst-case complexity of $O(n\Delta)$, where Δ is the maximum degree of any vertex in G.

In the algorithm above, if the graph G is chordal no edges will be produced for deletion, and we obtain a leaf of the binary computation tree. When G is not chordal, the indicated edge will be deleted, giving a smaller graph G′. This graph G′ becomes the left child of G and is then itself processed by the algorithm, and so forth, until the resulting (leaf) graph is chordal. There is no assurance that this final chordal graph is in fact an edge maximal chordal subgraph of G. The following algorithm, adapted from [2], does however ensure that this leftmost descendant is always a maximal chordal subgraph. Rather than keeping track of the cardinality of MADJ(v) at every step, this algorithm explicitly maintains the current clique MADJ(v) itself, represented by the set cliq(v).

Algorithm CLIQUE. Given a connected graph $G = (V, E)$ with $n = |V|$, this procedure either yields a perfect elimination ordering α (if G is chordal), or an edge e to be deleted (if G is not chordal).

```
1. Select $v_0 \in V$;  $cliq(v) \leftarrow \phi$ for all $v \in V$;  $S \leftarrow \{v_0\}$;
   $k \leftarrow n$;  $\alpha(k) \leftarrow v_0$.
2. for all $u \in V\text{-}S$ with $(u,v_0) \in E$:
       if $cliq(u) \subseteq cliq(v_0)$ then $cliq(u) \leftarrow cliq(u) \cup \{v_0\}$
       else **return** $e = (u,v_0)$.
3. Select $v_0 \in V\text{-}S$ such that $|cliq(v_0)| = \max\{|cliq(x)|: x \in V\text{-}S\}$;
   $k \leftarrow k - 1$;  $\alpha(k) \leftarrow v_0$;  $S \leftarrow S \cup \{v_0\}$.
4. if $k = 1$ then **return** $\{\alpha(1), \alpha(2), \ldots, \alpha(n)\}$, else go to Step 2.
```

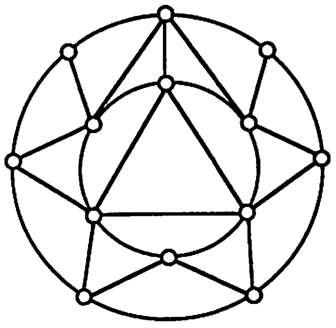

Figure 4

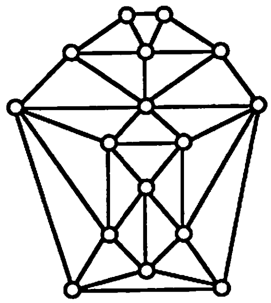

Figure 5

It should be noted that in contrast to algorithm CARDINALITY, which detects edges (u,v_0) to be deleted among vertices $u \in S$, the algorithm above finds (if possible) an edge (u,v_0) with $u \in V-S$. Using appropriate data structures, algorithm CLIQUE can be implemented to run in $O(m\Delta)$ time. Thus, this second algorithm is not as efficient (in the worst case) as the first one, but it does guarantee that the leftmost descendant leaf of any graph H in the computation tree is a maximal chordal subgraph of H. In turn, this might reduce the total number of leaves generated in the computation tree.

We now report computational results obtained by using both algorithms (to produce a binary tree with chordal leaves) as well as an existing algorithm TREE that uses trees as the leaves of its binary tree [3, 7, 10]. All procedures were coded in Pascal and run on the NAS AS/XL mainframe at Clemson University. Table 1 shows the number of leaves generated when the three algorithms were run on *circulant* graphs on n vertices. In these circulant graphs, vertices $v_0, v_1, \ldots, v_{n-1}$ are numbered along a circle and edges are of the form (v_i,v_{i+1}), (v_i,v_{i+2}) with subscripts taken mod n. The corresponding CPU times to obtain the chromatic polynomial are also shown in Table 1 (in seconds). These results show that algorithms CARDINALITY and CLIQUE produce considerably smaller binary trees (fewer leaves) than TREE. Moreover, both algorithms reduce the overall computation time by a factor of 8-16, when compared with TREE. Algorithm CARDINALITY is seen to run slightly faster than algorithm CLIQUE on these graphs.

Figure 4 shows another test graph with 13 vertices and 29 edges. For this example, TREE produced 77,544 leaves whereas CARDINALITY generated 1444 and CLIQUE generated 629. The corresponding CPU times were 22.49, 1.536, and 0.687 seconds. The superiority of the chordal-based algorithms is again apparent, with CLIQUE being twice as fast as CARDINALITY. Figure 5 shows another example, having 16 vertices and 37 edges. TREE was unable to handle this problem in the allocated time and space, but both of the chordal algorithms were successful. CARDINALITY generated 6904 leaves in

8.589 seconds, whereas CLIQUE generated 16,395 leaves in 17.45 seconds. In this example CARDINALITY was twice as fast as CLIQUE.

Table 1. Comparison of Algorithms for Circulant Graphs on n Vertices

			number of leaves			
n	7	8	9	10	11	12
TREE	174	426	1028	2388	5466	12,286
CARDINALITY	7	16	27	58	103	196
CLIQUE	7	18	33	69	130	255

			CPU time (secs.)			
n	7	8	9	10	11	12
TREE	.041	.104	.261	.633	1.499	3.487
CARDINALITY	.005	.013	.025	.055	.101	.198
CLIQUE	.005	.013	.027	.057	.110	.222

We have also randomly generated a number of connected graphs on n vertices and m edges for further study. Two sample graphs were generated for each size (n, m) and computational results are presented in Table 2. It is seen that either in terms of numbers of leaves or actual CPU time, the new algorithms completely dominate the TREE algorithm. Indeed, for some of the larger graphs, TREE was unable to complete its execution in the allocated time and space (indicated by the entry ***).

In summary, both algorithms presented here provide substantial improvements in empirical performance over existing reduction techniques for computing $P(G;\lambda)$. Further investigation is necessary to delineate those circumstances in which CARDINALITY is preferred to CLIQUE, and vice versa.

5. ACKNOWLEDGMENT

This research was supported by the US Air Force Office of Scientific Research (AFSC) under Grant AFOSR-84-0154.

Table 2. Comparison of Algorithms on Randomly Generated Graphs

		CPU time (secs.)			number of leaves		
n	m	TREE	CARD	CLIQUE	TREE	CARD	CLIQUE
5	6	0.001	0.000	0.000	4	1	1
		0.002	0.000	0.000	6	2	2
	8	0.003	0.000	0.000	12	1	1
		0.003	0.000	0.000	14	2	2
7	8	0.002	0.001	0.001	6	2	2
		0.001	0.001	0.001	6	2	2
	12	0.018	0.003	0.006	76	4	10
		0.018	0.004	0.003	74	7	5
	17	0.075	0.002	0.002	312	2	2
		0.080	0.001	0.003	336	2	4
9	13	0.014	0.003	0.003	44	3	4
		0.026	0.008	0.018	102	10	32
	20	0.315	0.005	0.013	1200	4	12
		0.345	0.007	0.006	1356	6	5
	25	1.644	0.018	0.025	6618	15	20
		1.655	0.011	0.030	6714	8	25
	30	4.124	0.008	0.009	16800	5	5
		4.247	0.013	0.007	17280	8	4
11	15	0.044	0.024	0.028	144	26	38
		0.027	0.009	0.021	84	10	32
	20	0.768	0.137	0.115	2756	157	138
		0.567	0.047	0.073	2000	45	84
	30	13.195	0.028	0.102	50088	17	74
		13.179	0.058	0.330	50148	38	299
	40	130.000	0.078	0.198	517200	39	107
		140.000	0.101	0.532	526800	52	340
13	15	0.019	0.020	0.023	48	17	28
		0.008	0.007	0.004	20	7	4
	25	2.835	0.095	0.236	8844	75	237
		4.331	0.167	0.131	14092	137	110
	35	170.000	1.252	1.143	583488	907	849
		170.000	1.046	2.293	600156	1239	2351
	45	***	0.806	1.668	***	445	937
		***	4.107	2.544	***	2511	1396
15	20	0.225	0.159	0.164	580	140	183
		0.159	0.086	0.083	416	78	82
	25	2.032	0.125	0.110	5316	87	88
		3.503	0.427	0.104	10428	390	84
	35	400.000	3.556	10.364	1312752	2529	9263
		320.000	2.612	0.981	1057740	1949	645
	45	***	42.977	54.285	***	30013	38911
		***	13.926	35.707	***	8496	24887
	50	***	38.782	79.456	***	23311	50247
		***	31.536	59.880	***	18832	39141
17	25	1.996	0.126	1.027	4800	82	1125
		1.879	0.610	0.234	4928	509	201
	35	610.000	24.040	37.763	1747242	22129	34341
		***	20.142	68.603	***	15149	70342

6. REFERENCES

[1] Boesch, F., Satyanarayana, A., and C. L. Suffel, "On Some Alternate Characterizations of a Graph Invariant Called Domination," Technical Report, Computer Science Dept., Stevens Institute of Technology, 1986.

[2] Dearing, P. M., Shier, D. R., and D. D. Warner, "Maximal Chordal Subgraphs," *Discrete Appl. Math.*, 1988, to appear.

[3] Frank, S., and D. R. Shier, "The Chromatic Polynomial Revisited," *Congressus Numerantium* 55 (1986) 57-68.

[4] Fulkerson, D. R., and O. A. Gross, "Incidence Matrices and Interval Graphs," *Pacific J. Math.* **15** (1965) 835-855.

[5] Golumbic, M. C., *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.

[6] James, K. R., and W. Riha, "Algorithm 24: Algorithm for Deriving the Chromatic Polynomial of a Graph," *Computing* **14** (1975) 195-203.

[7] Nijenhuis, A., and H. S. Wilf, *Combinatorial Algorithms*, Academic Press, New York, 1978.

[8] Pippert, R., and L. Beineke, "Characterizations of 2-Dimensional Trees," in *The Many Facets of Graph Theory*, Springer-Verlag, 1968, 263-270.

[9] Read, R.C., "An Introduction to Chromatic Polynomials," *J. Combinatorial Theory* 4 (1968) 52-71.

[10] Read, R. C., "An Improved Method for Computing the Chromatic Polynomials of Sparse Graphs," Research Report CORR 87-20, Faculty of Mathematics, Univ. of Waterloo, 1987.

[11] Roberts, F. S., *Graph Theory and its Applications to Problems of Society*, SIAM, 1978.

[12] Rose, D., "Triangulated Graphs and the Elimination Process," *J. Math. Anal. Appl.* 32 (1970) 597-609.

[13] Rose, D., "On Simple Characterizations of k-Trees," *Discrete Math.* 7 (1974) 317-322.

[14] Shier, D. R., "Some Aspects of Perfect Elimination Orderings in Chordal Graphs," *Discrete Appl. Math.* 7 (1984) 325-331.

[15] Stanley, R., "Acyclic Orientations of Graphs," *Discrete Math.* 5 (1973) 171-178.

[16] Tarjan, R E., and M. Yannakakis, "Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs, " *SIAM J. Comput.* **13** (1984) 566-579.

[17] Tutte, W. T., "A Contribution to the Theory of Chromatic Polynomials," *Canad. J. Math.* **6** (1954) 80-91.

[18] Whitney, H., "A Logical Expansion in Mathematics," *Bull. Amer. Math. Soc.* **38** (1932) 572-579.