TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

A simple linear time algorithm for triangulating three-colored graphs

Citation for published version (APA):
Bodlaender, H. L., & Kloks, A. J. J. (1991). *A simple linear time algorithm for triangulating three-colored graphs*. (Universiteit Utrecht. UU-CS, Department of Computer Science; Vol. 9113). Utrecht University.

Document status and date:
Published: 01/01/1991

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 25. Mar. 2023

A simple linear time algorithm for triangulating three-colored graphs

H. Bodlaender, T. Kloks

RUU-CS-91-13
May 1991

Utrecht University

Department of Computer Science

Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

A simple linear time algorithm for triangulating three-colored graphs

H. Bodlaender, T. Kloks

Technical Report RUU-CS-91-13
May 1991

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

ISSN: 0924-3275

A simple linear time algorithm for triangulating three-colored graphs*

Hans Bodlaender and Ton Kloks †

Abstract

In this paper we consider the problem of determining whether a given colored graph can be triangulated, such that no edges between vertices of the same color are added. This problem originated from the Perfect Phylogeny problem from molecular biology, and is strongly related with the problem of recognizing partial k-trees. In this paper we give a simple linear time algorithm that solves the problem when there are three colors. We do this by first giving a complete structural characterization of the class of partial 2-trees. We also give an algorithm that solves the problem for partial 2-trees.

1 Introduction

Consider a graph of which the vertices are colored such that no two adjacent vertices have the same color. In this paper we consider the problem to determine whether we can triangulate the graph (i.e. add edges, such that the resulting graph does not have an induced cycle of length at least 4), such that in the triangulated graph no two adjacent vertices have the same color.

The problem of triangulating a colored graph such that no two adjacent vertices have the same color is polynomially equivalent to the *Perfect Phylogeny problem*, see e.g., [7]. This problem, which is concerned with the inference of evolutionary history from DNA sequences, is of major importance to molecular biologists. Very recently, this problem has been shown to be NP-complete [2]. In [11] it was shown that the problem is solvable in $O(n^{k+1})$ time for k-colored graphs. Another special case was solved in [7]. In this paper we consider the problem, for the case that there are at most three colors. In [8] this problem was solved in $O(n\alpha(n))$ time. Very recently,

*This research was supported in part by the Foundation for Computer Science (S.I.O.N.) of the Netherlands Organization for Scientific Research (N.W.O.), and in part by the ESPRIT II Basic Research Actions Program of the EC under contract no. 3075 (project ALCOM).

†Department of Computer Science, Utrecht University, P.O.Box 80.089, 3508 TB Utrecht, The Netherlands.

Kannan and Warnow improved on their algorithm, and found a variant, that uses linear time. In this paper we give another, much simpler, linear time algorithm.

This paper is organized as follows. In section 2 we give a number of important definitions and properties of triangulated graphs. In section 3 we give a complete characterization of partial 2-trees. In section 4, precise conditions are given when a 3-colored graph is c-triangulatable, and when a colored partial 2-tree is c-triangulatable. In section 5 we give a linear time algorithm, that solves the problem of triangulating a three-colored graph, using the characterization of section 4. In section 6 we discuss a simpler variant of the algorithm, that only decides whether a c-triangulation exists, but does not yield the triangulation itself, and give an algorithm for triangulating t-colored partial 2-trees.

2 Definitions and basic properties

In this section we discuss some basic properties of triangulated graphs. Triangulated graphs are also known as *chordal graphs.* For further information we refer to [6]. The subgraph of graph $G = (V, E)$, induced by a set of vertices $W \subseteq V$, is denoted by $G[W] = (W, \{(v, w) \in E \mid v, w \in W\})$.

Definition 2.1
A graph is *triangulated* (or chordal) if it has no induced cycle of length strictly greater than three.

Definition 2.2
A vertex x of a graph G is *simplicial* if its adjacency set induces a complete subgraph (i.e. $G[\{v \in V \mid (v, x) \in E\}]$ is a complete graph.)

Definition 2.3
Let G be a graph and let $\sigma = [v_1, \ldots, v_n]$ be an ordering of the vertices. σ is called a *perfect elimination scheme* for G if for all i, v_i is simplicial in $G[\{v_i, \ldots, v_n\}]$.

Fulkerson and Gross ([5]) gave the following characterization of triangulated graphs.

Theorem 2.1 *A graph G is triangulated if and only if it has a perfect elimination scheme. Furthermore, if a graph is triangulated then* any *simplicial vertex can start a perfect elimination scheme.*

This theorem gives us an easy algorithm for the recognition of triangulated graphs, namely repeatedly locate a simplicial vertex and remove it from the graph. The graph is triangulated if and only if this process ends with the empty graph. If the graph is not a clique then there are at least *two* nonadjacent simplicial vertices.

Definition 2.4
A subset S of vertices is called a vertex separator for nonadjacent vertices a and b, if a and b are in different connected components of $G[V - S]$.

The following characterization was found by Dirac ([4]).

Theorem 2.2 *A graph is triangulated if and only if every minimal vertex separator induces a complete subgraph.*

There is yet another characterization, which says that a graph is triangulated if and only if it is the intersection graph of a family of subtrees of a tree. Examples of triangulated graphs are interval graphs and k-trees. Triangulated graphs are *perfect* (i.e. for every induced subgraph the chromatic number is equal to the size of a maximum clique, or equivalently, for every induced subgraph the size of a clique cover is equal to the size of a maximum stable set). Triangulated graphs can be recognized in linear time. There exist linear time algorithms for many NP-complete problems when restricted to triangulated graphs, for example coloring, clique, stable set and clique-cover. (See [6].)

Definition 2.5
A triangulation of a graph G is a graph H with the same number of vertices such that G is a subgraph of H and such that H is triangulated. We say that G is *triangulated into* H.

Clearly, every graph can be triangulated (into a clique). Triangulating a graph such that the number of added edges is minimum is called the minimum fill-in problem, and triangulating such that the maximum clique is minimum is called the treewidth problem (the treewidth of a graph is one less than the minimum size of a maximum clique in any triangulation). Both these problem (treewidth and minimum fill-in) are NP-complete.

In the next section we give a definition of a special type of triangulated graphs, called k-trees, and we characterize the biconnected partial 2-trees.

3 Characterization of biconnected partial 2-trees

Definition 3.1
A *k-tree* is a graph for which there exists a perfect elimination scheme $\sigma = [v_1, \ldots, v_n]$ such that for all $i \leq n - k$, v_i is adjacent to a clique with k vertices in the subgraph $G[\{v_i, \ldots, v_n\}]$.

From this definition it follows that every maximal clique in a k-tree has size $k+1$, and that every minimal vertex separator has size k.

Definition 3.2
A *partial k-tree* is a subgraph of a k-tree, or equivalently, a partial k-tree is a graph that can be triangulated into a k-tree.

It is an easy exercise to show that every triangulated graph with a maximum clique of size at most $k+1$ is a partial k-tree. It turns out that many interesting classes of graphs are contained in a class of partial k-trees for some k (see e.g. [1]). A large number of NP-hard problems become solvable in polynomial or even linear time when restricted to the class of partial k-trees for some constant k. Partial k-trees are recognizable in $O(n \log^2 n)$ time, see [3].

In this section we address the problem of characterizing partial 2-trees. A graph is a partial 2-tree if and only if all its biconnected components are partial 2-trees, so when characterizing partial 2-trees we can restrict ourselves to biconnected partial 2-trees.

Lemma 3.1 *Let G be a biconnected partial 2-tree. Let $S = \{x, y\}$ be a separator such that $G[V - S]$ has at least three connected components. Then in any 2-tree embedding (x, y) is an edge.*

Proof:
Let x and y be vertices such that $G[V - \{x, y\}]$ has at least three connected components. Since the graph is biconnected, both x and y have at least one neighbor in every connected component and so there are paths between x and y with internal vertices in every connected component. Assume that G can be triangulated into a 2-tree H such that (x, y) is not an edge in H. In any 2-tree every minimal vertex separator is an edge. Every minimal vertex separator for x and y contains at least one vertex of every connected component of $G[V - \{x, y\}]$ (otherwise there would be a path between x and y). So the minimal separator contains at least three vertices, which is a contradiction. □

Definition 3.3
Let the *cell-completion* of G be the graph $\bar{G}$, obtained from G by adding an edge between all pairs $\{x, y\}$ for which $G[V - \{x, y\}]$ has at least three connected components. A *cell* of G is a set of vertices which form a chordless cycle in the cell-completion.

Note that by lemma 3.1 the cell-completion $\bar{G}$ is a subgraph of any triangulation of G.

Definition 3.4
A *tree of cycles* is a graph which can be obtained by gluing together chordless cycles edge by edge. At each stage a new cycle is glued to exactly one edge of the part of the tree of cycles that has already been constructed.

For example, a 2-tree is a tree of triangles (i.e. a tree of cycles in which every cycle is a triangle). The following characterization appears in [9], where it is used to enumerate all biconnected partial 2-trees. By triangulating a chordless cycle with m vertices, we mean adding $m-3$ edges (i.e. the minimum number) such that the new graph is triangulated.

Theorem 3.2 *A biconnected graph G is a partial 2-tree if and only if its cell-completion $\bar{G}$ is a tree of cycles. By triangulating each chordless cycle of $\bar{G}$ in all possible ways we obtain all possible triangulations of G into a 2-tree.*

Proof:
The only if part can be seen as follows. We must show that $\bar{G}$ is a tree of cycles. Consider a triangulation of $\bar{G}$ into a 2-tree H. Consider the edges in H that are not in $\bar{G}$. Such an edge of H must be incident with at least two triangles since $\bar{G}$ is biconnected. On the other hand, if there are three triangles incident with an edge of H, the edge is also present in $\bar{G}$. Since H is a tree of triangles, and the only edges that are not in $\bar{G}$ are edges that are incident with two triangles, it follows that $\bar{G}$ must be a tree of cycles.

Now let $\bar{G}$ be a tree of cycles. Then by triangulating each cycle into a 2-tree, the resulting graph is a tree of triangles, and hence a 2-tree. □

In the next section we focus on c-triangulating 3-colored graphs.

4 c-Triangulating 3-colored graphs

Recall that the problem that we consider in this paper is the following. Given a graph G with a vertex coloring $c : V \rightarrow C$, where C is a set of t colors, such that each vertex is colored with one of t colors. Triangulate G (if possible) without introducing edges between vertices of the same color. If such a triangulation exists we call it a *c-triangulation*.

Note that a graph can be c-triangulated if and only if all the biconnected components can be c-triangulated, and a c-triangulation of a graph G can be obtained by c-triangulating all biconnected components. Hence, in the remainder we assume that G is a biconnected graph. We can also observe that the maximum clique size in a c-triangulation can be at most the number of different colors, since otherwise there would be an edge between vertices of the same color. The following lemma states an even stronger result.

Lemma 4.1 *Let G be a $(k+1)$-colored graph. Then G can be c-triangulated if and only if G can be c-triangulated into a k-tree.*

Proof:
Consider a c-triangulation of G, say H. Clearly, every maximal clique in H has size at most $k+1$ and every minimal vertex separator has size at most k. If the graph G has at most $k+1$ vertices, then the lemma is obviously true. Otherwise, suppose we have a minimal vertex separator S in H of size less than k. Let the vertices of $H[V-S]$ be partitioned into two non-empty sets A and B such that there is no edge between A and B. By induction we may assume that there is a c-triangulation of $H[B \cup S]$ into a k-tree. We add these edges to the graph H. Let H' be the (c-triangulated) graph with vertices $A \cup S \cup B$ such that $H'[B \cup S]$ is a k-tree. Since the subgraph induced by $A \cup S$ is triangulated, and every triangulated graph which is not a clique has at least two nonadjacent simplicial vertices, it follows that A contains a simplicial vertex. This argument can be repeated and we may conclude that there must be a perfect elimination scheme for H' of the form $\sigma = [v_1, \ldots, v_a, \ldots, v_n]$, such that $v_1, \ldots, v_a$ are exactly the vertices of A. Now S is contained in a clique with $k+1$ vertices in $B \cup S$, say C. C contains exactly one vertex x with the same color as v_a. So we can make v_a adjacent to all vertices of $C \setminus \{x\}$. Repeating the process, adding vertices $v_a, v_{a-1}, \ldots, v_1$ in this way we obtain a triangulation of H' into a k-tree. □

In particular, we have that a $(k+1)$-colored graph can be c-triangulated, only if it is a partial k-tree. Consider a biconnected three-colored graph $G = (V, E)$. To be c-triangulatable, G must be a biconnected partial 2-tree. The following results give a precise characterization of c-triangulatable partial 2-trees.

Theorem 4.2 *Let G be a biconnected t-colored partial 2-tree, and let $\bar{G}$ be the cell completion of G. G can be c-triangulated, if and only if:*

1. *No two adjacent vertices of $\bar{G}$ have the same color, and*
2. *Every cell has at least three vertices with different colors.*

Proof:
From theorem 3.2 it follows, that it is sufficient to show that a cycle can be c-triangulated, if and only if it contains three vertices of different colors. This fact was proved in [8]. For reasons of completeness we also present a proof here.

If a cycle has only two colors, then it is easy to see that it can not be c-triangulated, since cycles of length 3 cannot be made. Suppose a cycle S has three colors. If S is a triangle there is nothing to proof. Otherwise we can add a chord to S such that the two new cycles made by this chord each have three colors. To find such a chord, we consider two cases. In case that there is a color that appears

only once, then take any chord containing this color at one of its end-vertices. If every color appears at least twice in S, then there is a vertex v such that the two neighbors have different colors. Then take the chord connecting the two neighbors of v. Recursively apply the argument to the shorter cycles formed by the chord. □

A slightly different characterization is obtained in the following theorem.

Theorem 4.3 *Let G be a biconnected t-colored partial 2-tree, and let $\bar{G}$ be the cell completion of G. G can be c-triangulated, if and only if:*

1. *No two adjacent vertices of $\bar{G}$ have the same color, and*
2. *Every cycle in $\bar{G}$ contains at least three vertices with different colors.*

Proof:
Use theorem 4.2. Clearly, if each cycle in $\bar{G}$ contains at least three different colors, then each cell in the cell completion does so. This shows the 'if'-part. The 'only if'-part follows from the fact, that if $\bar{G}$ contains a cycle with only two colors, then $\bar{G}$ contains a subgraph that cannot be triangulated, hence G cannot be triangulated. □

It follows that a t-colored partial 2-tree can be c-triangulated, if and only if $\bar{G}$ does not contain an edge between vertices with the same color, and for every pair of colors, the subgraph of $\bar{G}$ induced by the vertices with th colors is cycle-free, i.e., is a partial 1-tree. This partly generalizes: If a t-colored graph G can be c-triangulated, then for every subset of $s \leq t$ colors, the subgraph of G induced by vertices with a color in this set is a partial $(s-1)$-tree. This necessary condition is unfortunately not sufficient, not even for $t = 3$. The following result follows directly from lemma 4.1 and theorem 4.3.

Corollary 4.4 *Let G be a biconnected 3-colored graph, and let $\bar{G}$ be the cell completion of G. G can be c-triangulated, if and only if:*

1. *G is a partial 2-tree.*
2. *No two adjacent vertices of $\bar{G}$ have the same color, and*
3. *Every cycle in $\bar{G}$ contains at least three vertices with different colors.*

5 Algorithm for 3-colored graphs

In this section we describe an algorithm to c-triangulate a 3-colored graph, if possible. In section 6, we give an easier variant, that only tests whether it is possible to c-triangulate the graph, without actually yielding a c-triangulation. We also give a variant for t-colored partial 2-trees, for $t \geq 3$.

Suppose G is a biconnected 3-colored graph. Our algorithm to c-triangulate G, if possible, has the following structure:

1. Make *any* triangulation of G into a 2-tree H.
2. Given H we then can make the cell-completion of G, $\bar{G}$. Check if no two adjacent vertices in $\bar{G}$ have the same color.
3. Make a list of all cells.
4. c-Triangulate each cell (if it has three different colors).

Notice that when step 1, 2 or 4 fails, the graph can not be c-triangulated, and otherwise the algorithm outputs a correct c-triangulation. Correctness of this method follows from theorem 4.2.

We now describe each step of this algorithm in more detail. As each step ' linear time implementation, we get the following result.

Theorem 5.1 *There exists a linear time algorithm, that given a 3-colored gra tests whether G can be c-triangulated, and if so, outputs a c-triangulation of G.*

5.1 Triangulating G into a 2-tree H

In this subsection we consider the problem to find a 2-tree H, that contains a given graph G as a subgraph, or output that such a graph does not exist. It is well known that this problem can be solved in linear time, see e.g. [10]. For reasons of completeness we describe the algorithm also here.

If G is a biconnected partial 2-tree, we can make a triangulation into a 2-tree H by successively choosing a vertex of degree two, making the neighbors of this vertex adjacent and removing the vertex from the graph (see for example [12].)

We can implement this as follows. Assume that we have for each vertex in the graph G a (linked) list of neighbors. Assume that with each edge (x, y) in the adjacency list of x, there is a pointer to the edge (y, x) in the adjacency list of y. We also keep a list of vertices of degree 2. Initialize $H := G$ (i.e. make a copy of the adjacency lists).

Choose a vertex of the list of vertices of degree 2, say x. Let y and z be the neighbors of x. We add z to the adjacency list of y and y to the adjacency list of z. It is possible, that we create a duplicate edge (y, z) in this way, i.e. in that case y appears twice in the adjacency list of z, and vice versa.

We now test, whether y, and z have degree 2 (and hence must be put on the list of vertices of degree 2). Look at the adjacency list of y. Scan this list until either we have encountered *three different* neighbors of y, or until the list becomes empty. When we encounter a duplicate edge while scanning the list, say (y, x'), we remove the second copy of it from the list of y, and remove its counterpart from the list of x'. If y has only two different neighbors, we put y in the list of vertices of degree two. We do the same for z.

Iterate the above until there are no vertices of degree 2 left. When the graph now has more than two vertices, then G was no partial 2-tree. Otherwise, H is a 2-tree, containing G as a subgraph. Correctness of the algorithm follows from [12]. The order in which the vertices have been removed, is a perfect elimination scheme for the 2-tree H.

To see that this algorithm runs in linear time, we notice the following. When scanning the adjacency lists, every step we either encounter a duplicate edge (which is then removed) or we find a new neighbor. Notice that the total number of duplicate edges is at most n (the number of vertices of G), since every time a vertex is removed at most one duplicate edge is created.

We remark here that the description given in [12] is insufficient to show linear time of the algorithm. In particular, in [12] the test '*given vertices x, y, test whether $(x, y) \in E$*' is assumed to take constant time. However, when the edges are given as adjacency lists, this test can take more time (when using a standard model of computation).

5.2 Making the cell-completion $\bar{G}$

Suppose that we now have the 2-tree embedding H with a perfect elimination scheme $\sigma = [v_1, \ldots, v_n]$ for H. Note that $\bar{G}$ is a subgraph of H, so to find $\bar{G}$, we only have to test for each edge in H, whether it belongs to $\bar{G}$ or not. We can use the following lemma to make the cell-completion.

Lemma 5.2 *For any edge $e = (x, y)$ in $\bar{G}$, the number of common neighbors of x and y in H is at least three if and only if the number of components in $G[V - \{x, y\}]$ is at least three.*

(Compare with definition 3.3.) The algorithms is as follows. First make a copy of G in $\bar{G}$. If vertices x and y have at least three common neighbors in H, add the edge (x, y) to $\bar{G}$. Testing this property can be done as follows.

From the adjacency list of a vertex v_i in H, remove the vertices v_j for all $j < i$. Each adjacency list now has at most two elements, since σ is a perfect elimination scheme of a 2-tree. Number the edges of H in any order $1, \ldots, 2n - 3$, and let a pointer point from the edges in the adjacency lists to its number and vice versa. Initialize an array $cn(1 \ldots 2n - 3)$ to zero. Start with the triangle $\{v_{n-2}, v_{n-1}, v_n\}$. For each edge of this triangle, look up the number and increase the value in cn by 1.

Consider the other vertices one by one, in the reversed order of the perfect elimination scheme, i.e. in the order $v_{n-3}, v_{n-4}, \ldots, v_1$. For each vertex v_i ($i < n-2$) we do the following. Suppose it is adjacent to vertices v_j and v_k (with $j > i$ and $k > i$). For the edges in this triangle increase the value in cn by one. It is straightforward to see (by induction) that for each edge in the induced subgraph $H[\{v_i, v_{i+1}, \ldots, v_n\}]$ the number of common neighbors is given by the value in cn. (We use here that each triangle v_i, v_j, v_k is considered exactly once, namely when considering the lowest numbered vertex in the triangle.) If a final value in cn is at least three, look up the edge in the adjacency list of H, and add this edge to $\bar{G}$ if it was not already present. Clearly, this procedure uses linear time. Correctness follows from lemma 5.2.

5.3 Making a list of the cells

Notice that the number of cells in $\bar{G}$ is at most $n-2$ (with equality if and only if $\bar{G}$ is already a 2-tree). For each cell we initialize an empty list. During the algorithm we keep a pointer from each edge in H we have encountered to the number of the last cell it is contained in and to its position in this cell. Again let $\sigma = [v_1, \ldots, v_n]$ be a perfect elimination scheme for H. Remember that for each vertex v_i we have removed all vertices v_j with $j < i$ from its adjacency list.

Put the vertices v_n, v_{n-1}, v_{n-2} in the first cell and for each edge in this triangle we make a pointer to the number of this cell and to its position in this cell. Consider the other vertices one by one, in the reversed order of the perfect elimination scheme. Suppose v_i has neighbors v_j and v_k in H with $j > i$ and $k > i$. Let $j < k$. Look in the adjacency list of v_j in $\bar{G}$ (which has at most two elements) if the vertices v_j and v_k are adjacent. If v_j and v_k are adjacent in $\bar{G}$ we make a new cell containing the three vertices v_i, v_j and v_k and for each edge of this triangle we update the number of the cell it is contained in. If v_j and v_k are not adjacent in $\bar{G}$, then they can be contained in at most one cell, since otherwise v_j and v_k would have at least three different neighbors in H and hence would be adjacent in $\bar{G}$. We have a pointer to this cell and to the position of v_j and v_k in this cell. We add the vertex v_i to this cell in the place between vertices v_j and v_k and we update the cell number for the edges (v_i, v_j) and (v_i, v_k). It is straightforward to see by induction that at each step each cell contains the vertices of the cell of $\bar{G}$, restricted to the vertices $v_i, v_{i+1}, \ldots, v_n$. So in this way we obtain a list of all cells in $\bar{G}$ in linear time.

5.4 Triangulating each cell

We have made a list of vertices for each cell such that consecutive vertices in this list are adjacent in $\bar{G}$. For each color we make a list of vertices in the cell that are of this color, and for every vertex we make a pointer to its position in the list. If there is a color with an empty list, the cell can not be triangulated. Suppose every color occurs in the cell. If there is only one vertex of a given color, we make this vertex

References

[1] H.L. Bodlaender, Classes of graphs with bounded tree-width, Tech. Rep. RUU-CS-86-22, Department of Computer Science, Utrecht University, Utrecht, 1986.

[2] H.L. Bodlaender, M.R. Fellows and T. Warnow, paper in preparation, 1991.

[3] H.L. Bodlaender and T. Kloks, Better algorithms for the pathwidth and treewidth of graphs, *To appear in: proceedings ICALP'91.*

[4] G. Dirac, On rigid circuit graphs, *Abh. Math. Sem. Univ. Hamburg* **25**, 71 – 76 (1961).

[5] D. Fulkerson and O. Gross, Incidence matrices and interval graphs, *Pacific J. Math.* **15**, 835 – 855 (1965).

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

[7] S. Kannan and T. Warnow, Inferring evolutionary history from DNA sequences, *in: Proceedings of the 31th Annual Symposium on Foundations of Computer Science*, pp. 362 – 371, 1990.

[8] S. Kannan and T. Warnow, Triangulating three-colored graphs, *in: Proceedings of the 1st Ann. ACM-SIAM Symposium on Discrete Algorithms*, pp. 337 – 343, 1990.

[9] T. Kloks, Enumeration of biconnected partial 2-trees, to appear.

[10] J. Matoušek and R. Thomas, Algorithms finding tree-decompositions of graphs, *Journal of Algorithms* **12**, 1 – 22 (1991).

[11] C.F. McMorris, T. Warnow, and T. Wimer, Triangulating colored graphs, submitted to Inform. Proc. Letters.

[12] J.A. Wald and C.J. Colbourn, Steiner trees, partial 2-trees and minimum IFI networks, *Networks* **13** (1983), 159 – 167.