Mathematics of Operations Research

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

On a Class of Totally Unimodular Matrices

Mihalis Yannakakis,

To cite this article:
Mihalis Yannakakis, (1985) On a Class of Totally Unimodular Matrices. Mathematics of Operations Research 10(2):280-304.
http://dx.doi.org/10.1287/moor.10.2.280

MATHEMATICS OF OPERATIONS RESEARCH
Vol. 10, No. 2, May 1985
Printed in U.S.A.

ON A CLASS OF TOTALLY UNIMODULAR MATRICES*

MIHALIS YANNAKAKIS

AT&T Bell Laboratories

We examine the class of totally unimodular matrices that contain no odd cycles, which we call restricted totally unimodular (RTUM). We show that a matrix is RTUM if and only if it can be decomposed in a very simple way into the incidence matrices (or their transposes) of bipartite graphs or directed graphs, and give a linear time algorithm to perform this task. Based on this decomposition, we show that the 0, 1 Integer Programming Problem with an RTUM matrix of constraints has the same time complexity as the b-matching and the max flow problems.

1. Introduction. A totally unimodular (TUM) matrix is a matrix every square submatrix of which has determinant $-1, 0, 1$. The importance of these matrices lies in the fact that they are exactly those constraints matrices of Integer Programs for which one can relax the integrality restriction without altering the optimal solution; more formally, a matrix A is TUM if and only if the extreme points of the polyhedron $\{\mathbf{x} \mid A\mathbf{x} \leqslant \mathbf{b}, \mathbf{x} \geqslant 0\}$ are integral for all integral vectors $\mathbf{b}$ [HK]. Important examples of totally unimodular matrices are the incidence matrices of directed graphs (constraints matrices of network flow problems) and bipartite (undirected) graphs (constraints matrices of matching problems in bipartite graphs).

Since the discovery of this property of TUM matrices, several characterizations have been found (see [B1], [Tam] for references); these characterizations, though elegant, are hard to use to test whether a matrix is TUM. Only recently, Seymour discovered a nice characterization for TUM matrices [S] which renders itself to a polynomial algorithm. This characterization says roughly that a matrix is TUM if and only if it can be decomposed in a certain way into simple kinds of TUM matrices that are related to networks.

In [C] Commoner gave a very simple sufficient condition for total unimodularity. (According to [Tam], this condition had been found by Edmonds in 1961.) In this paper we study the matrices that satisfy this condition, which we call *restricted totally unimodular* (RTUM). From a $[-1, 0, 1]$ matrix A (i.e. matrix with entries in $\{-1, 0, 1\}$) we can construct a bipartite directed graph $D(A)$ by associating a node with every row and every column of A, and including an edge between a row node r_i and a column node c_j if the entry a_{ij} in that row and column is nonzero. The edge is directed from r_i to c_j or from c_j to r_i according as $a_{ij} = -1$ or $a_{ij} = 1$. Let C be an undirected cycle of $D(A)$ (i.e., a cycle in the underlying undirected graph). We traverse C in one direction and assign a *sign* $+1$ or -1 to every edge of C according as the edge has the same or the opposite direction. The *sign* of C, $\sigma(C)$, is the product of the signs of its edges; the sign does not depend on which one of the two directions we traverse C in, since $D(A)$ is bipartite and therefore c has even length. The cycle C is *even* or *odd* according as $\sigma(C) = 1$ or -1. For, example, if A is a [0, 1] matrix, it is easy to see (see §2) that if the length of C is $2k$, then C is even or odd according as k is even or odd. Commoner's

*Received January 5, 1981; revised December 12, 1983.
AMS 1980 subject classification. Primary: 90C35. Secondary: 05B20
OR/MS Index 1978 subject classification. Primary: 481 networks/graphs.
Secondary: 634 Programming/integer/theory.
Key words. Totally unimodular matrices incidence matrices, integer programming.

0364-765X/85/1002/0280$01.25

sufficient condition for total unimodularity states that if all cycles of $D(A)$ are even, then A is TUM. We say that a matrix A is *restricted totally unimodular* (RTUM) iff all cycles of $D(A)$ are even. The condition that all cycles of $D(A)$ be even is sufficient but not necessary for the matrix A to be totally unimodular; a necessary condition is that all *chordless* cycles of $D(A)$ be even. In the case of [0, 1] matrices, the latter condition is necessary and sufficient for a larger class of matrices, the *balanced matrices* [B1], [B2], [FHO]: these are the matrices A for which the linear program $\min\{\mathbf{1x} | A\mathbf{x} \geq \mathbf{b}, \mathbf{x} \geq 0\}$ has an integral solution for all integral vectors $\mathbf{b}$.

We show that the graphs of RTUM matrices are exactly those that can be decomposed into graphs of incidence matrices (and their transposes) of directed graphs and bipartite graphs, by successively deleting pairs of edges and replacing them by pairs of small appropriate paths. Thus, Seymour's decomposition theorem takes a particularly simple form in the case of RTUM matrices. Furthermore, one can obtain such a decomposition in linear time. Using then this decomposition we prove that the 0, 1 Integer Program with an RTUM constraints matrix can be solved with the same time complexity as the b-matching problem in bipartite graphs and the max flow problem in networks. (Actually, the max flow problem is not 0, 1; however its dual, the min cut, is.) Of course, this problem can be solved using the recent algorithms of Khachian for the general Linear Programming Problem [K], or of Bland and Edmonds, or of Maurras, Truemper and Akgul [MTA] for totally unimodular matrices. However, exploiting the special structure of RTUM matrices gives a much better complexity bound, and the advantage of being able to use directly improvements on the solution of the b-matching and the max flow problem. (See e.g. [GN], [ST] for a list of successive recent improvements on the complexity of the max flow problem.)

In §2 we review basic definitions from Graph-theory and the theory of TUM matrices. §§3, 4 and 5 are concerned with [0, 1] RTUM matrices and their graphs (RTUM graphs). In §3 we prove some basic combinatorial properties of these graphs. In §4 we describe a linear time algorithm that recognizes RTUM graphs and does the decomposition of them as we mentioned above. In §5 we show how to solve the Binary (0, 1) Program with a [0, 1] RTUM matrix. In §6 we show that $[-1, 0, 1]$ RTUM matrices are not essentially different from the [0, 1] ones, and in §7 we conclude with some final remarks.

2. Definitions and preliminaries. A *graph* (*digraph*) $G = (N, E)$ consists of a set N of *nodes* and a set E of *edges* which are unordered (ordered) pairs of distinct nodes. A *multigraph* can have multiple edges, i.e. E is a multiset rather than a set. A graph (or multigraph) is *bipartite* if its set N of nodes can be partitioned into two subsets P and Q so that no edge has both nodes in P or Q. If (u, v) is an edge of G we say that u and v are *adjacent* and that (u, v) is *incident* to u and v. The *degree* $d(v)$ of a node v is the number of edges incident to v. An *independent set of nodes* is a set of nodes no two of which are adjacent. A *node cover* is a set of nodes S such that every edge is incident to a node in S. If $F \subset E$ is a set of edges of G, the graph $G_F = (N, F)$ is a *subgraph* of G. Let $\mathbf{b} = [b_1, \ldots, b_{|N|}]$ be a vector of integers of length $|N|$. A *b-matching* of G is a set F of edges of G such that the degree of each node v_i in G_F is at most b_i. If $V \subset N$ is a set of nodes, the graph G_V *induced* by V is (V, E_V) where $E_V = \{(u, v) | u, v \in V$ and $(u, v) \in E\}$.

A *path* p between nodes u and v of G is a sequence $u, (u, w_1), w_1, (w_1, w_2), w_2, \ldots, w_k, (w_k, v), v$ of distinct nodes and edges of G; the *length* of p (denoted $|p|$) is its number of edges $(k + 1)$. A graph is *connected* if there is a path between any two nodes of it; otherwise it is disconnected. The maximal connected induced subgraphs of a graph are its *connected components*. A set of nodes V (or edges) *separates the connected graph G if* G_{N-V} is disconnected. If $\{u\}$ separates a connected graph G, we

say that u is a *cutpoint* of G. A connected graph with no cutpoints is called *biconnected*. The maximal biconnected induced subgraphs of a graph G are the *blocks* of G. A block is *trivial* if it consists of a single edge. A *cycle* is a sequence of nodes and edges (like a path), where all the nodes and edges are distinct except the first and the last nodes of the sequence which are the same. A *tree* is a connected graph with no cycles. With every connected graph G (or every connected component of G, if G is disconnected) we can associate a tree $BC(G)$ called the *block-cutpoint tree* of G with one node for every block and every cutpoint of G and an edge between a cutpoint v and a block B if v is in B; it can be shown that $BC(G)$ is actually a tree and that two blocks have a node v in common iff v is a cutpoint (see e.g. [H]). The *underlying graph* of a digraph D is obtained from D by regarding the edges of D as unordered pairs. The terms we have defined above when used for a digraph refer to the underlying graph.

A *rooted tree* T is a connected digraph with one node, the *root*, having no edges coming in, and such that each other node has exactly one edge coming in. If (u,v) is an edge of T, u is the *father* of v and v is a *son* of u. The reflexive transitive closure of the father (resp. son) relationship is the *ancestor* (resp. *descendant*) relationship. The underlying graph of a rooted tree is a tree; and conversely, if T is a tree, we can designate any node v of T as its root and direct all edges of T away from v to form a rooted tree. A *spanning tree* of a connected graph G is a tree, which is a subgraph of G containing all its nodes. If G is disconnected, a set of spanning trees for the connected components of G is a *spanning forest*.

A systematic way of searching graphs is *depth-first-search* (DFS) [T], [HT1], [AHU]. DFS produces from a graph G a rooted spanning tree T of G (spanning forest if G is disconnected), according to the order in which nodes are visited. The edges of G are divided into *tree-edges* (edges in T) and *back-edges* (edges not in T); back-edges connect a node u with a proper ancestor v (i.e. ancestor $v \neq u$) of u. For every back-edge e there is a unique cycle containing e and tree edges; it is called the *fundamental cycle* that corresponds to e. The nodes of G receive numbers from 1 to $|N|$ in the order in which they are visited (i.e. a node gets a lower number than its descendants). These numbers can be used to identify the cutpoints and blocks of G (see e.g. [AHU] for an exposition) and construct the block-cutpoint tree $BC(G)$ (rooted at a block of the root of the DFS tree T).

As we mentioned in §1, we can construct from a $[-1,0,1]$ matrix A a digraph $D(A)$ with one node for every row and column of A and an edge (c_j, r_i) if $a_{ij} = 1$, (r_i, c_j) if $a_{ij} = -1$. Nodes that correspond to rows (resp. columns) of A have *type* r (resp. c) or are $r-$ $(c-)$ nodes. A digraph that is constructed in this way from a $[-1,0,1]$ matrix is called a *matrix digraph*. Thus, a matrix digraph is a bipartite digraph with the nodes in the one set of the bipartition having type r and the nodes in the other set having type c. Clearly, there is a 1-1 correspondence between $[-1,0,1]$ matrices and matrix digraphs. The sign of an undirected cycle C of a matrix digraph $D(A)$ is defined as in the Introduction. It is easy to see that an equivalent expression for the sign $\sigma(C)$ is $\sigma(C) = (-1)^n \prod\{a_{ij} \mid (r_i, c_j)) \in C\}$, where the length of the cycle is $|C| = 2n$. A matrix A (and its digraph $D(A)$) is called restricted totally unimodular (RTUM) iff all cycles of $D(A)$ are even. If the matrix A has 0,1 entries only, and C is a cycle of $D(A)$ of length $2n$, then the sign of C is $(-1)^n$; thus, $\sigma(C) = 1$ iff the length of C is a multiple of 4. Let $G(A)$ be the underlying graph of $D(A)$. If A has 0,1 entries, we can ignore the directions on the edges of $D(A)$ and work instead with $G(A)$; A is RTUM iff all cycles of $G(A)$ have length a multiple of 4. We call $G(A)$ a *matrix graph*. That is, a matrix graph is an undirected bipartite graph with the nodes in the one side of the bipartition having type r and the nodes in the other set having type c. There is a 1-1 correspondence between [0, 1] matrices and matrix graphs.

All forbidden subgraphs of RTUM digraphs are cycles. This implies that a matrix

digraph $D(A)$ (and its associated matrix A) is RTUM iff all blocks of $D(A)$ are RTUM. A trivial block (i.e. a single edge) does not contain any cycles, and is therefore RTUM. Thus, to determine if A is RTUM it suffices to look at each nontrivial block of $D(A)$ separately. We shall show that a nontrivial biconnected RTUM digraph can be decomposed in a simple way into RTUM digraphs with all r-nodes or all c-nodes having degree 2. An RTUM matrix A with two nonzero entries per row or per column is called a *basic* RTUM matrix; its associated digraph (or graph if the matrix has 0, 1 entries) is called a basic RTUM digraph (or graph). A *chord* of a cycle is an edge connecting two nonconsecutive nodes of it. A cycle is *chordless* if it does not have any chords. Commoner showed that all cycles of digraphs of restricted totally unimodular (RTUM) matrices are chordless, and conversely, if all cycles of $D(A)$ are chordless then A is TUM iff it is RTUM [C]. Clearly, if a matrix A has two nonzero entries per row or per column, then all cycles of $D(A)$ are chordless; therefore, such a matrix is TUM iff it is basic RTUM.

The constraint matrices of network-flow problems and bipartite matching, paradigms of totally unimodular matrices, are basic RTUM. The *incidence matrix* A of a multigraph G has one row for every node v_i and one column for every edge e_j of G; the ijth entry of A, a_{ij} is 1 iff v_i is a node of e_j. The incidence matrix of a multidigraph G has also one row for each node v_i and one column for each edge e_j of G: if $e_j = (v_i, v_k)$ then $a_{ij} = -1$, $a_{kj} = 1$ and $a_{lj} = 0$ for $l \neq i, k$. Clearly, the incidence matrix of a multigraph or multidigraph has two nonzero entries per column. The incidence matrix of a (multi) digraph is totally unimodular (and basic RTUM); the incidence matrix of a (multi) graph is totally unimodular (and basic RTUM) iff the graph is bipartite. In general, there is a simple necessary and sufficient condition for the total unimodularity of a matrix A with two nonzero entries in each column: A is TUM iff the set of rows can be partitioned into two sets I_1, I_2 so that for every column with two 1's or two -1's one nonzero entry is in I_1 and the other in I_2 and for every column with an 1 and a -1 both nonzero entries are in the same set (I_1 or I_2)—see e.g. [B1] or [L]. A similar condition holds for matrices with two nonzero entries per row, since a matrix A is TUM iff its transpose A^T is TUM.

If u, v are two nodes of $D(A)$ of the same type (r or c) and p a (undirected) path between them of length $2n$ (p must have even length), the sign of p is $\sigma(p) = (-1)^n \prod\{a_{ij} \mid (r_i, c_j) \in p\}$; the path p has even or odd *parity* according as $\sigma(p) = 1$ or -1. Thus, for example, if A is the incidence matrix of a bipartite graph H and r_i, r_j the two nodes of $D(A)$ that correspond to nodes v_i and v_j of H, all paths between r_i and r_j have even (resp. odd) parity iff v_i and v_j are in the same (resp. different) set in the bipartition of H, i.e. all $v_i - v_j$ paths in H have even (resp. odd) length.[1]

3. Properties of RTUM graphs. In this and the next two sections we shall concentrate on [0, 1] RTUM matrices A, and the associated graphs $G(A)$.

At first let us see what the basic RTUM graphs are. Suppose that in a matrix graph G all c-nodes have degree 2. Let H be the multigraph whose nodes are the r-nodes of G; for every c-node u of G connected to the r-nodes x and y, H has an edge (x, y). Thus, G can be obtained from H by inserting a node in the middle of every edge. It follows easily from the definitions that G is the graph of the incidence matrix of H. Therefore, G is TUM (and basic RTUM) iff H is bipartite. If the matrix graph G has all r-nodes of degree 2, we can similarly construct a multigraph H such that G is the

[1] The notions of "parity", "even", "odd" are usually associated with mod 2. Here they apply to cycles and paths that have even length. In the case of 0, 1 matrices their definition here can be thought of as half of the length (of the cycle or path) mod 2. If we associate with a 0, 1 matrix a hypergraph as in [B1] instead of a matrix graph, then these notions of parity, etc. correspond to the lengths of paths in the hypergraph (i.e., number of hyperedges).

graph of the transpose of the incidence matrix H. (Note that the graph $G(A^T)$ of the transpose A^T of a matrix A is $G(A)$ with r and c-nodes switched.) Therefore, we have

LEMMA 1 [C]. *Let $G(A)$ be a matrix graph, all of whose c-nodes (resp. r-nodes) have degree 2. $G(A)$ is TUM (and basic RTUM) if and only if A is the incidence matrix (resp. the transpose of the incidence matrix) of some bipartite multigraph.*

LEMMA 2. *Let G be an RTUM graph, u any c-node, v any r-node of it. There are no 3 disjoint paths between u and v in G.*

PROOF. Suppose that there are 3 disjoint paths p_1, p_2, p_3 between u and v. Any two of these paths form a cycle. Let C_1 be the cycle formed by p_2 and p_3, C_2 by p_1 and p_3, and C_3 by p_1 and p_2. Let l_1, l_2, l_3 be the lengths of the three paths. Since u and v are of opposite type, l_1, l_2, l_3 are odd. We have $|C_1| = l_2 + l_3$, $|C_2| = l_1 + l_3$, $|C_3| = l_1 + l_2$ $\Rightarrow |C_1| + |C_2| + |C_3| = 2(l_1 + l_2 + l_3) = 2 \cdot \text{odd}$. Therefore, at least one of $|C_1|, |C_2|, |C_3|$ is not divided by 4. ∎

COROLLARY 1 [C]. *All cycles of an RTUM graph are chordless.*

If all cycles of a biconnected graph are chordless, then G is minimal biconnected, i.e. removal of any edge will destroy the biconnectivity of G. Thus, a biconnected RTUM graph $G(A)$ is minimally biconnected and consequently has at most $2(m + n) - 4$ edges (see e.g. [H, Example 3.14]), where m, n are the numbers of rows and columns of A. That is, RTUM matrices are sparse.

LEMMA 3. *Let G be an RTUM graph, C any cycle of it, and let G' be the graph obtained by deleting the edges of the cycle. No (connected) component of G' contains two nodes of C of opposite type.*

PROOF. Let K be a component of G' that contains some nodes of C of opposite type. Let p be a shortest path connecting a c- with an r-node of C in K, say c-node u to r-node v. From our choice of p, it does not contain any other nodes from C. But then there are 3 disjoint paths between u and v (p and the two paths of the cycle) contradicting the fact that G is RTUM. ∎

Let us call the components of G' in the previous lemma the *segments* of G with

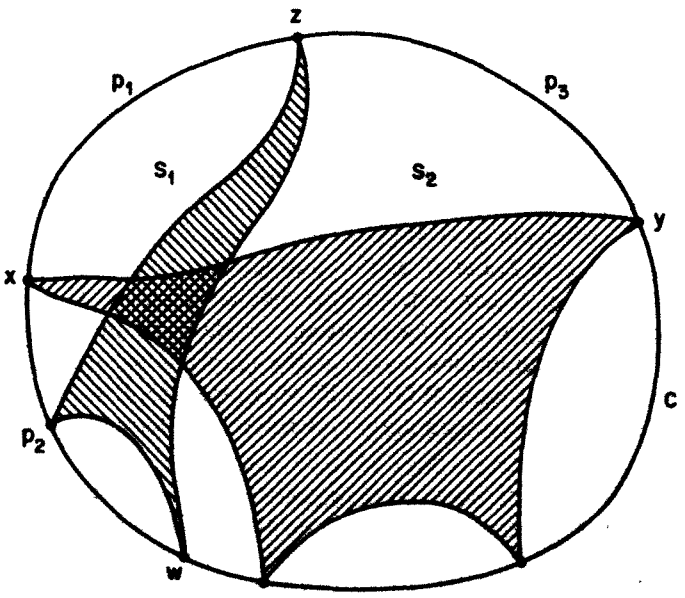

FIGURE 1

respect to the cycle. A segment is *trivial* if it contains only one node. We say that a segment S is of type r (resp. c) if it contains an r- (resp. c-) node of C. We say that a segment S_1 *crosses* another segment S_2 if there are two nodes x, y of S_2 on the cycle such that S_1 contains nodes from both $x - y$ paths on the cycle (see Figure 1). Clearly, if S_1 crosses S_2, then also S_2 crosses S_1, and we say that the two segments *cross* each other.

LEMMA 4. *Let G be an RTUM graph, C any cycle of it. No two segments of G with respect to C of opposite type cross each other.*

PROOF. Suppose that segments S_1 and S_2 cross each other (see Figure 1). Since S_1 and S_2 are of opposite type, the same is true for nodes x and z. Let p_2' be a path from z to w in S_1, and p_3' a path from x to y in S_2. G contains three disjoint paths from x to $z: p_1, p_2$ with p_2', and p_3' with p_3 (see Figure 1). ∎

In Figure 2 we show what a typical picture of the segments of an RTUM graph G with respect to a cycle C will look like. In the figure we have circled the r-nodes of the cycle.

Suppose now that G is a biconnected RTUM graph which is not basic. Then G has some c-node, say u, and some r-node, say v, with degree at least 3. Since G is biconnected, there is a cycle C that passes through u and v. Let $S(u), S(v)$ be the segments with respect to C that contain respectively u and v. Since u and v have opposite type, these two segments are distinct; $S(u)$ is of type c and $S(v)$ of type r. Since u and v have degree ≥ 3, $S(u)$ and $S(v)$ have more than 1 node; in particular they have more nodes (besides u and v respectively) on the cycle, because otherwise u or v would be a cutpoint. Thus, G has nontrivial segments of both types with respect to C.

Let H be a graph with the segments of G with respect to C as nodes and an edge between any two segments that cross each other. From Lemma 4 every connected component of H contains segments of the same type. Let K be a component of H with

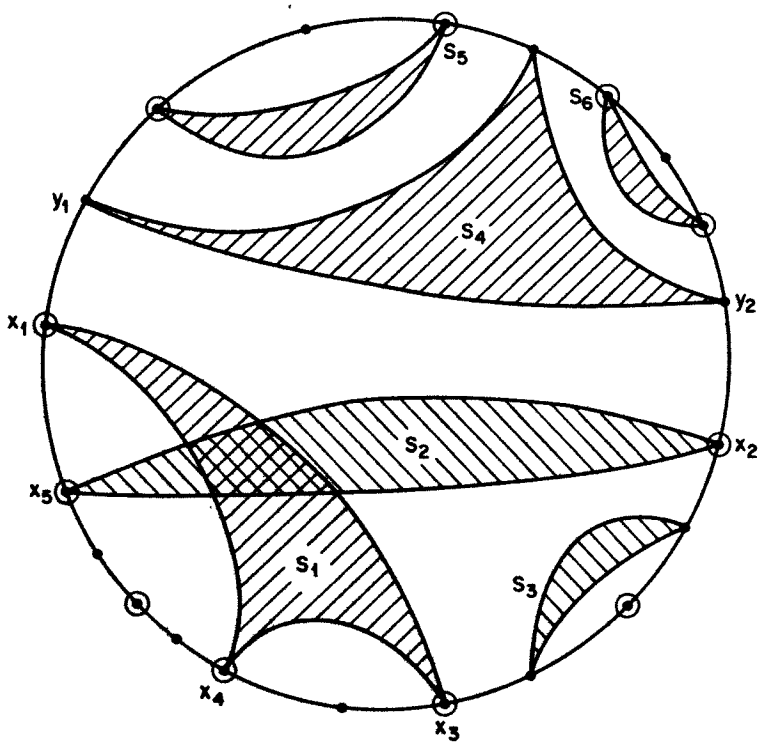

FIGURE 2

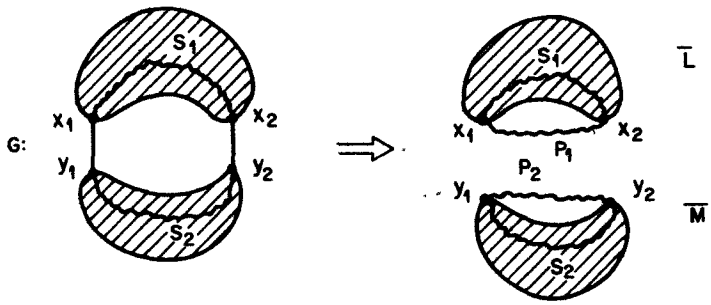

FIGURE 3

segments of type r and let $x_1, \ldots, x_k$ be the nodes of the segments of K that lie on C. All of these nodes must be of type r. They divide the cycle C into k paths. If a segment S not in K had nodes on C on two or more of these paths, then S would cross some segment of K and, consequently, S would belong to K. Therefore, for every other component K' of H, all segments of K' have nodes on C that lie in exactly one of these paths (see Figure 2 where $K = \{S_1, S_2\}$). Let K' be a component of H with segments of type c and suppose without loss of generality that all nodes common to the segments of K' and to C lie on the path $x_1 - x_2$ of C. Let y_1, y_2 be the nodes adjacent to x_1 and x_2 in this path; y_1 and y_2 are of type c. Since no segment not in K crosses a segment of K, the edges (x_1, y_1) and (x_2, y_2) separate the graph G into two components L and M with $x_1, x_2 \in L$ and $y_1, y_2 \in M$. Let k_1 be the length of the $y_1 - y_2$ path in $M \cap C$ modulo 4, and k_2 the length of the $x_1 - x_2$ path in $L \cap C$ modulo 4. Since $|C|$ is a multiple of 4 we have $k_1 + k_2 = 2 \bmod 4$. Let $\overline{L}$ be L with a virtual path of new nodes of length $k_1 + 2$ added between x_1 and x_2, and let $\overline{M}$ be M with a virtual path of new nodes of length $k_2 + 2$ added between y_1 and y_2. Clearly, if G is RTUM then so are $\overline{L}$ and $\overline{M}$ and $(k_1 + 2) + (k_2 + 2) = 2 \bmod 4$. Let us call the operation that takes G into $\overline{L}$ and $\overline{M}$ the *decomposition* G at the edges (x_1, y_1) and (x_2, y_2). Note that every node has the same degree after the decomposition as before, and that $\overline{L}$ has fewer c-nodes than G with degree ≥ 3 and $\overline{M}$ fewer r-nodes than G with degree ≥ 3. Therefore, if we continue decomposing the graph in this way, we will eventually arrive at basic RTUM graphs. In the next section we will see how this can be done systematically in linear time.

Summarizing here our previous discussion we have:

THEOREM 1. *A biconnected graph G is RTUM if and only if it is basic RTUM or can be decomposed at two edges (x_1, y_1) and (x_2, y_2) into two RTUM graphs $\overline{L}, \overline{M}$ with the virtual $x_1 - x_2$ and $y_1 - y_2$ paths in $\overline{L}$ and $\overline{M}$ having opposite parity.*

PROOF. We showed above the (only if) part. For the (if) part, note that a cycle C in G either lies entirely in $\overline{L}$ or $\overline{M}$, or consists of a path s_1 from x_1 to x_2 in $\overline{L}$, a path s_2 from y_2 to y_1 and the two edges (x_1, y_1) and (x_2, y_2). In the first two cases C is even since $\overline{L}$ and $\overline{M}$ are RTUM. In the last case, let p_1 and p_2 be respectively the virtual $x_1 - x_2$ and $y_1 - y_2$ paths in $\overline{L}$ and $\overline{M}$. Since $\overline{L}$ and $\overline{M}$ are RTUM we have: $|s_1| + |p_1| = 0 \bmod 4$ and $|s_2| + |p_2| = 0 \bmod 4$. Therefore, $|C| = |s_1| + |s_2| + 2 = |p_1| + |p_2| + 2, \bmod 4 \Rightarrow |C| = 0 \bmod 4$, since p_1 and p_2 have opposite parity. ∎

4. Recognition of RTUM graphs. Let $G = (N, E)$ be a matrix graph. At first we do a depth-first search of G. In this DFS we do the following.

(1) Identify the blocks and cutpoints of G.

(2) Check that all fundamental cycles are even.

(3) For each node v compute

$Lc(v) = \min\{w \mid$ there exists a back edge (x,w) from a descendant x of v to the c-node $w\}$.

$Lr(v) = \min\{w \mid$ there exists a back edge (x,w) from a descendant x of v to the r-node $w\}$.

$LOW(v) = \min\{Lc(v), Lr(v), v\}$

(4) (i) Check that every node v has at most one back edge (v,x) emanating from it.

(ii) Compute the *first node* $A1(v)$ *adjacent to* v as follows:

$$A1(v) = \begin{cases} x & \text{if } v \text{ has a back edge } (v,x), \\ \text{the son } w \text{ of } v \text{ with the least } LOW(w), & \text{if } v \text{ has no back edges.} \end{cases}$$

(iii) Check that for every son w of v other than $A1(v)$, $Lt'(w) > v$ where t' is the opposite type to $t(v)$, and $LOW(w) > A1(v)$ if $(v, A1(v))$ is a back edge.

We can perform (1), (3) and (4) as in the usual DFS. For (2) we assign levels to the nodes mod 4; every time we encounter a back edge (v,x) we check that $\text{level}(v) - \text{level}(x) = 3 \bmod 4$.

If in (4)(i), a node v with two back edges $(v,x_1),(v,x_2)$ is discovered, with $x_1 < x_2$ say, then the fundamental cycle corresponding to (v,x_1) has a chord (v,x_2) and therefore G is not RTUM by Corollary 1; the cycle that consists of the path in the tree from x_1 to x_2 and the two back edges $(v_1x_1),(v_1x_2)$ must be odd if the fundamental cycles are even.

Suppose that v has a back edge (v,x). If $y = LOW(w) \leqslant A1(v) = x$ for some son w of v (see Figure 4b), then the fundamental cycle corresponding to the back edge (z, y) that gave $LOW(w)$ has a chord (v,x). If $y = Lt'(w) \leqslant v$ for some son w of v in (4)(iii), then since y is of opposite type than v, $y < v$ and y is a proper ancestor of v. If y is an ancestor of x then $LOW(w) \leqslant x$ and as above G is not RTUM. If $x < y$, then (see Figure 4a) there are 3 disjoint paths between v and y and therefore by Lemma 2, G is not RTUM; the cycle that consists of the paths on the tree between x and y, v and z, and the two back edges $(v,x)(z, y)$ must be odd if the fundamental cycles are even.

If v does not have a back edge (and thus $A1(v)$ is a son of v) and $Lt'(w) = y \leqslant v$ for some other son w of v, we have $x = LOW(A1(v)) \leqslant y$ (see Figure 4c) and therefore G has again 3 disjoint paths between v and y.

Next, we do a second DFS of G as a directed graph, with the direction of the edges as in the first DFS, and $A1(v)$ as the first node in the adjacency list of v. This is similar to the method used in [HT2] for dividing a graph into triconnected components. In fact, the task we have here has some similarities: we want to find separation pairs to decompose the graphs into basic parts. Only the basic parts here are different (basic RTUM graphs) and therefore we look for special kinds of separation pairs. Also, we expect the graph to have a particular structure, and this makes the separation much simpler. In the second DFS we are going to decompose every nontrivial block into

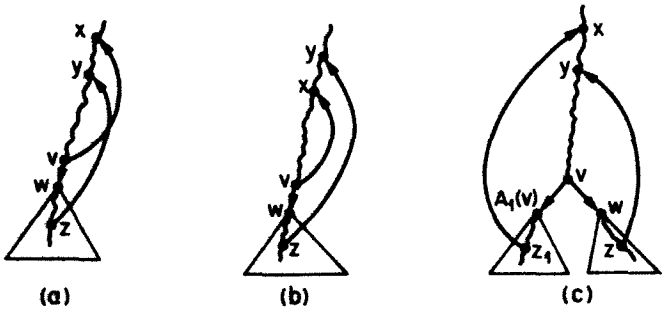

FIGURE 4

basic RTUM graphs. The output of the algorithm will be a rooted tree for each nontrivial block of G. If B is a block of G, the nodes of the tree $T(B)$ of B correspond to basic RTUM graphs, which we call the *basic components* of B. Every node v of B belongs to exactly one such basic RTUM graph H and the degree of v in B is the same as the degree in H. Every basic component H of B has a type r or c. All nodes of H of a different type have degree 2 in H (and B). The least node v of B (in the DFS) belongs to the component H_1 that corresponds to the root of $T(B)$ and $t(H_1) = t(v)$. The types alternate along the paths of the tree, i.e. the father of a component H in $T(B)$ has different type than H. For every edge $e = (i, j)$ of the tree $T(B)$ incident to the node i that corresponds to the basic component H_i, H_i has two nodes of B, $x_{i1}(e), x_{i2}(e)$ of type $t(H_i)$ and a virtual path of length 2 or 4 of new nodes between $x_{i1}(e)$ and $x_{i2}(e)$. The block B is obtained from its basic components by deleting the new paths and adding edges $(x_{i1}(e), x_{j1}(e)), (x_{i2}(e), x_{j2}(e))$ for every edge $e = (i, j)$ of $T(B)$. That is, $T(B)$ describes a decomposition of B into basic RTUM graphs.

In the second DFS we use the following notation. $A(v)$ is the list of nodes adjacent to v, with $A1(v)$ the first node in the list; $b(v)$ is the index of the basic component of B to which v belongs; k is the index of the current component; $F(v)$ is the father of v in the DFS tree; $f(i)$ is the father of i in the tree $T(B)$; t_i is the type of H_i; $l(i)$ is the least numbered node that has been assigned to component H_i; j is the number of components created thus far. A component is created by its father which has opposite type.

Initially $j = 1$, $k = 1$, $t(H_1)$ is $t(v_1)$ where v_1 is the lowest numbered node of the block B (the root of the DFS for this block) $b(v_1) = 1$ and $b(v) = 0$ for $v \neq v_1$; that is, v_1 is assigned to H_1 and all other nodes are unassigned. The main call for the block B is SEARCH(v_1) where SEARCH(v) is as shown below. In the algorithm the degree $d(v)$ of a node v is counted with respect to the block B; i.e., only edges of B count in the degree. The father $F(v_1)$ of the root v_1 of the block is undefined; when line 13 is reached in SEARCH(v_1) the condition is treated as evaluating to false, and the procedure finishes. This reflects the fact that the basic component of the root has no father in the decomposition tree $T(B)$ of the block. Also $f(1)$, the father of the first component is undefined; if on line 3 the current component k is 1, the condition $w < l(f(k))$ is treated as evaluating to false.

What we mean by "appropriate length" in lines 15 and 16 is 2 or 4 according as the difference mod 4 of levels between the two nodes is 2 or 0. As it will turn out later, the test on line 12 is redundant (it will never happen that $b(v) \neq k$) but we have included it here for simplicity.

EXAMPLE. In Figure 5a we show the DFS tree of a biconnected matrix graph B. The nodes are numbered according to their DFS ordering. In the figure we have circled the r-nodes and left c-nodes uncircled. The heavy edges in the figure are the edges $(v, A1(v))$ that connect each node to its first adjacent node. It can be easily verified that the graph passes the test of the first DFS.

The second DFS is initialized by opening a component H_1 of type r which contains the root node 1. From node 1 the search goes to the first adjacent node 2, then to 3, to 4 and the back edge (4, 1) is found. Since node 4 has degree 3 and type c, different from the type of the current component, a new component H_2 of type c is created at lines 7, 8 of SEARCH(4), and node 4 is assigned to it. The one edge (4, 1) that connects H_2 to its father H_1 in the decomposition tree is recorded at line 8.

Now the search moves to node 5, from there to its first adjacent node 6, then to 7, 8, 9, and the back edge (9, 2) is encountered. Node 2 is assigned to the current component H_2, $l(2)$—the highest node in the DFS tree assigned to H_2—becomes node 2. On line 7 of SEARCH(9), a new component is not created although node 9 has type r, different from the type of the current component, because 9 has degree 2. On line 9, node 9 is assigned to H_2 and the search backs up to node 8, which gets also assigned to

```
procedure SEARCH(v)
begin
1.    w ← A1(v);
2.    if (v, w) is a back edge then
3.          if b(w) ≠ 0, k or w < l(f(k)) then reject
4.          else begin b(w) ← k;
            l(k) ← min(l(k), w) end;
      else   begin
5.          SEARCH(w);
6.          if b(v) ≠ 0, k then reject;
          end;
7.    if t(v) ≠ t_k and d(v) > 3 then
            begin  [create new component]
8.          j ← j + 1; f(j) ← k;
            x_{k1}((k, j)) ← w; x_{j1}((k, j)) ← v;
            k ← j; t_k = t(v); l(k) ← v;
          end;
9.    b(v) ← k; l(k) ← min(l(k), v);
10.   for each node u in A(v) other than A1(v) do
          begin
11.         SEARCH(u);
12.         if b(v) ≠ k then reject;
          end;
13.   if l(k) > v and t(F(v)) ≠ t_k , and d(F(v)) > 3 or F(v) = v_1 (the root) then
          begin  [close current component]
14.       x_{k2}((f(k), k)) ← v; x_{f(k)2}((f(k), k)) ← F(v);
15.       add in H_k a virtual path of new nodes between
          x_{k1}((f(k), k)) and x_{k2}((f(k), k)) of appropriate
          length;
16.       add in H_{f(k)} a virtual path of new nodes between
          x_{f(k)1}((f(k), k)) and x_{f(k)2}((f(k), k)) of appropriate
          length;
17.       k ← f(k);
          end;
end
```

H_2. The father of 8 has degree 3 and type $r \neq t_2$ but $l(2) = 2 < 8$ and therefore the current component will not be closed on line 13 of SEARCH(8). When the search backs up to node 7, a new component H_3 of type r is created on line 7, node 7 is assigned to it, and the one edge (7, 8) connecting H_3 to H_2 is recorded. Now, the search proceeds to node 10 and finds the back edge (10, 5). Node 5 is assigned to H_3. As we back up through nodes 10, 7, 6, the nodes 10 and 6 of degree 2 are assigned to H_3 too. When SEARCH(5) reaches line 13, it verifies that the conditions of the if-statement

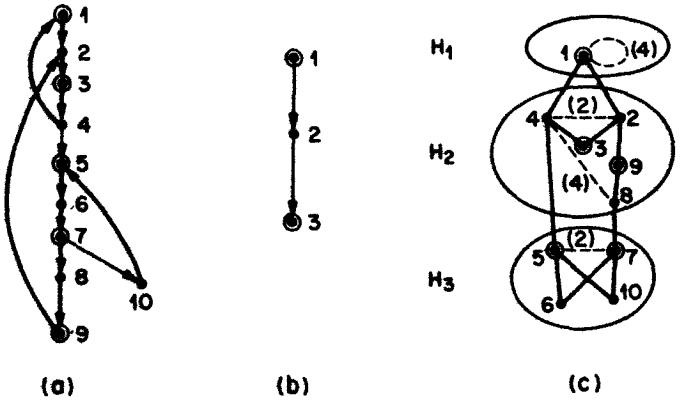

FIGURE 5

are satisfied; therefore, it closes H_3 and records the second edge (5, 4) connecting it to its father H_2 in the decomposition tree.

Subsequently, the search backs up to node 4, to node 3 which gets assigned to H_2, and to node 2. There, on line 13, H_2 is closed because the father of node 2 is the root of the DFS tree and has different type than t_2.

In Figure 5b we show the decomposition tree. In Figure 5c the basic components are shown together with the edges connecting different components. The virtual paths are shown with broken lines with the labels in parentheses indicating the length. ∎

Let us explain informally how the algorithm operates. The principle underlying the algorithm is that every node of degree at least 3 has to be assigned to a basic component of the same type. Such a node is assigned to a component as soon as a third edge incident to it is traversed, and is never reassigned. (We will prove these properties formally later on.) This third edge is either a back edge into the node, in which case the assignment takes place at line 4 of the search of the tail of the back edge, or it is a tree edge, in which case the assignment occurs at line 9 of the node. In the second case (of a tree edge), the type of the node is compared with the type of the current component; if they disagree, a new component is created. Thus, every node of degree ≥ 3 which is assigned to a component at line 9 is assigned to a component of the same type. In the case when an assignment takes place after a back edge (line 4) we do not check the types. We will see later however that it is not necessary; the fact that we always move from a node to its first adjacent node, together with property (4) of the first DFS, ensure in this case the compatibility of types.

The reader might have noticed in the example that the second DFS consisted essentially in exploring a sequence of paths: 1-2-3-4-1, 4-5-6-7-8-9-2, and 7-10-5. The first path started from the root, moved always to the first adjacent node until it found a back edge. The other paths were formed in a similar way except that the first move in the path was not to the first adjacent node. From property (4) of the first DFS, the first node adjacent to a node v leads always to the back edge emanating from a descendant of v and reaching as high a node in the DFS tree as possible (with the least number). That is, either v has a back edge, in which case all back edges coming out of the subtree rooted at v go to proper descendants of $A1(v)$, or v has no back edges and $A1(v)$ is the son of v with the least LOW number.

Thus, in general, a path is formed where the search starts from a node s, goes to a son s' and proceeds down the tree choosing always the first son (the one with the least LOW number) until it finds a node v with a back edge (v, w). The first time, $s = v_1$ (the root of B) and it is easy to see [HT2] that $w = s$. All other times w is an ancestor of s, since B is biconnected; $w = LOW(s')$ where s' is the son of s in this path (see Figure 6). While traversing the path from s' down to v and the back edge to w, the current component remains the same. Thus, w is assigned to the same component as s. If we let C be the fundamental cycle corresponding to the back edge coming into $LOW(A1(s))$, we see that w and s belong to the same segment with respect to C. From §3 we know that if B is RTUM, w and s must be of the same type. If we do the decomposition (of the segments w.r. to C) into two parts that we described before Theorem 1 of the last section, w and s will be in the same part. The algorithm can be viewed as a systematic succession of decompositions as in §3, with respect to the fundamental cycles of the graph.

A new component is created only when it is absolutely necessary; i.e., when a node of degree ≥ 3 cannot be accommodated. The only exception to this is the first component which is created to match the type of the root. This will be convenient later when several blocks have to be combined in solving an Integer Program. A consequence of this fact, however, is that the root has to be treated in line 13 as a degree 3 node because we require it to belong to a component of the same type even if it has

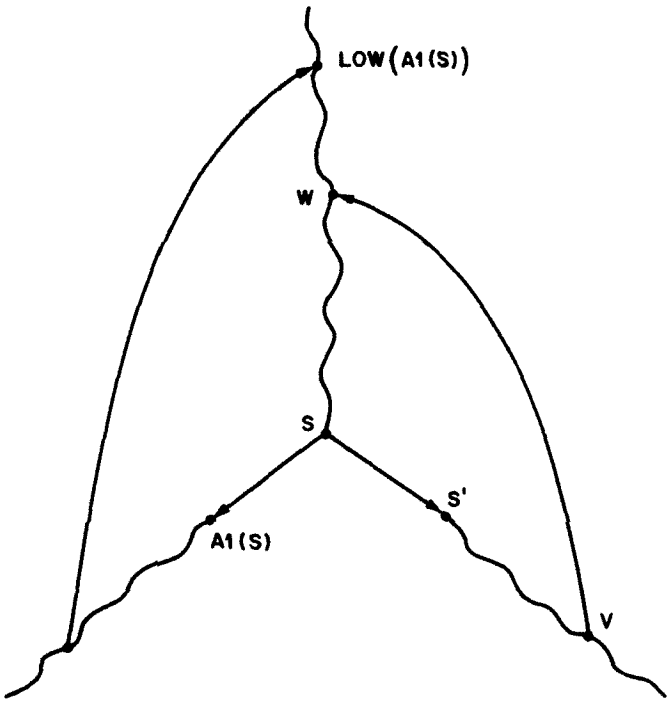

FIGURE 6

degree 2. This may result in the creation of an unnecessary component as in our example. There, the first two components H_1 and H_2 could be combined into one component of type c.

Regarding now the closing of components, we postpone it until it is necessary. The condition $l(k) \geq v$ (line 13) indicates that all nodes assigned to the current component have been already completely searched; thus, we could close the component. However, if the father of v has degree 2, or has degree 3 but is of the same type as the current component, then it can be assigned to the current component too. In the latter case (when $F(v)$ has degree 3 and type t_k), if we closed H_k then we would have to open a new component at $F(v)$ since the father of H_k has different type.

We will proceed now to prove the correctness of the algorithm.

LEMMA 5. *Throughout the execution of the algorithm,* (1) *if a degree* ≥ 3 *node is assigned to a component, the component has the same type as the node, and* (2) *no node changes every component.*

PROOF. The claim is clearly true initially. Consider an assignment, and suppose the claim holds before this assignment. We shall show that it continues to hold.

Suppose at first that the assignment occurs on line 4 after a back edge (v, w). Because of the test on line 3, w was previously either assigned to the same component or not assigned yet. Thus, part (2) is true. Look at the path that ended with the back edge (v, w) (see Figure 6). Let s, s' be the first two nodes on this path. The current component did not change while traversing the path. Node s' is not the first adjacent node to s. From property (4)(iii) of the first DFS, the type of w is the same with the type of s. Before SEARCH(s), called SEARCH(s') either s was not assigned yet to a component, in which case it was assigned to H_k (line 9), or it was already assigned to H_k (lines 6, 12). Thus, when the back edge (v, w) is encountered, s belongs to H_k. Therefore, $t(w) = t(s) = t_k$.

Suppose now that the assignment occurs on line 9. If $(v, A1(v))$ is a back edge then v

was not assigned prior to this moment. If $d(v) \geq 3$, the current component after the execution of lines 7-8 agrees in type with v. Thus, the claim holds. Assume that $(v, A1(v))$ is a tree edge. On line 6 we verify that v is either unassigned or belongs to the (then) current component. In the first case the lemma holds as before. In the second case, v must agree in type with H_k, and therefore no new component will be created in lines 7–8; that is, the assignment on line 9 will have no effect. ∎

LEMMA 6. *Suppose that the algorithm does not reject. Then, for every edge* (v,u) *of* B *one of the following holds*:

(1) $b(u) = b(v)$,

(2) $b(v) = i$, $b(u) = j$, $i = f(j)$, $v = x_{i1}((i,j))$ *and* $u = x_{j1}((i,j))$, *or* $v = x_{i2}((i,j))$ *and* $u = x_{j2}((i,j))$,

(3) *Same as* (2) *with the roles of* u *and* v *reversed.*

PROOF.

Case 1. (v,u) is a tree edge and $u \neq A1(v)$. Let k be the index of the current component when SEARCH(u) finishes. On line 11 we check that $b(v) = k$. If at the end of SEARCH(u) the component was not changed (lines 13–17) then $b(u) = b(v) = k$. If the component was changed then $k = b(v) = i = f(b(u)) = f(j)$. From line 14, $v = x_{i2}((i,j))$ and $u = x_{j2}((i,j))$.

Case 2. (v,u) is a tree edge and $u = A1(v)$. After SEARCH(u) finishes (line 5) we check that $b(v) = 0$ or $b(v) = k$. If $d(v) = 2$, then $k = b(u)$ from line 13, and v gets assigned also to H_k. Suppose, therefore, that $d(v) \geq 3$, and let $j = b(u)$, $i = b(v)$. If $j = k$ (we don't close the component in line 13 of SEARCH(u)), either v was assigned earlier to H_k or is assigned at this point and hence $b(u) = b(v)$, or $t(v) \neq t_k$ and a new component H_i is created by v (line 7), son of H_j. Assume, therefore, that $j \neq k$; i.e. $k = f(j)$. From line 13 of SEARCH(u), $t(v) \neq t_j \Rightarrow t(v) = t_k$. Thus, in line 7 of SEARCH(v), v will not create a new component and therefore $i = k = f(j)$, and as in Case 1, $v = x_{i2}((i,j))$ and $u = x_{j2}((i,j))$.

Case 3. (v,u) is a back edge. If $d(v) = 2$ then $b(v) = k = b(u)$ from lines 4 and 9. If $d(v) \geq 3$, then, since $t_k = t(u) \neq t(v)$, on line 7 we create a new component H_j son of the component H_k to which u is assigned, assign v to it, and set $x_{k1}((k,j)) = u$, $x_{j1}((k,j)) = v$ on line 8. ∎

THEOREM 2. *If the algorithm does not reject then* B *is RTUM.*

PROOF. Since all nodes of degree ≥ 3 are assigned to components of the same type, and because of Lemma 6, when the algorithm terminates, B is decomposed into components each of which has all r-nodes or all c-nodes of degree 2. Thus, each component is the graph of the incidence matrix (or its transpose) of some multigraph. It is easy to see that $G(A)$ (or $G(A^T)$) is RTUM, where A is the incidence matrix of a graph H if and only if all fundamental cycles of $G(A)$ are even; just note that the levels of the r- or c-nodes in the depth-first search spanning tree of $G(A)$ give a bipartition of H. Since in the first DFS we check all fundamental cycles it follows then that all components are basic RTUM. It remains to show that the virtual paths added to the two components at a decomposition have opposite parities. The result then will follow from Theorem 1 using a straightforward induction. Thus, let us look at an edge $e = (i,j)$ or $T(B)$ with i the father of j. The edges $e_1 = (x_{i1}(e), x_{j1}(e))$, $e_2 = (x_{i2}(e), x_{j2}(e))$ of B separate B into two components. It is easy to see that there are two possibilities for the relative position of e_1 and e_2 in the depth-first-search tree: either (1) both e_1 and e_2 are tree-edges and the one lies above the other (see Figure 7a) or (2) e_1 is a back edge and e_2 lies in the fundamental cycle corresponding to e_1 (see Figure 7b).

Component H_j is created at node x_{j1}; the first case happens when $A1(x_{j1}) = x_{i1}$ is a son of x_{j1} (i.e. x_{j1} does not have any back edge), and the second case happens when x_{j1}

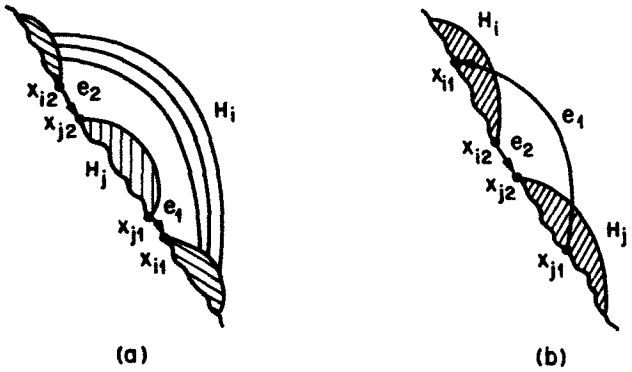

FIGURE 7

has a back edge (to x_{i1}) and degree ≥ 3. The virtual paths added to H_i and H_j between x_{i1}, x_{i2} and x_{j1}, x_{j2} respectively have opposite parity: obviously in Case 1, and because the fundamental cycle corresponding to e_1 is even in Case 2. ∎

The proof of the converse of Theorem 2 is a bit more complicated. In every case of rejection we are going to exhibit three disjoint paths connecting nodes of opposite type. From Lemma 2 it will follow then that the graph is not RTUM if the algorithm rejects.

Let's start with some simple facts; they can be proved by an easy induction.

Fact 1. At every instance, the open components (those that are not finished yet) are the components on the path in the decomposition tree $T(B)$ from the current component to the root one.

Fact 2. Let H_k be the current component when the SEARCH(v) is initiated. All descendants of v in the DFS tree are assigned to a descendant component of H_k.

Fact 3. At any instance, $l(i)$ gives the lowest numbered node assigned to H_i.

Fact 4. At any instance, if k is the index of the current component while the search is at node v, $l(k)$ is an ancestor of v.

Fact 5. At any instance, $l(f(i))$ is a proper ancestor of $l(i)$.

We are going to prove only Fact 5. Suppose that at some point, while the search is at node v, Fact 5 ceases to hold. This means that the current component is i, and $l(i)$ is updated. From Fact 4, $l(i)$ is not updated on line 9. Therefore $l(i)$ is updated on line 3 by some back edge (v, w). Right prior to this moment $l(f(i))$ is a proper ancestor of $l(i)$ which (by Fact 4) is an ancestor of v. Since w is also an ancestor of v, either $l(f(i))$ is a proper ancestor of w (the new $l(i)$), or else w is an ancestor of $l(f(i))$ in which case $w \leq l(f(i))$ and the algorithm will reject.

LEMMA 7. *If the algorithm rejects because it finds at line* 3 *a back edge* (v, w) *with* $w \leq l(f(k))$ *then there are* 3 *disjoint paths between two nodes of opposite type.*

PROOF. Consider the path that ended with the back edge (v, w). Let s, s' be the first two nodes of this path; $s' \neq A1(s)$ (see Figure 8). Node s was assigned to the current component H_k which did not change while traversing the path. From Facts 4 and 5, $l(f(k))$ is a proper ancestor of s. Since $w \leq l(f(k))$, and $t(w) \neq t_{f(k)}$, w is a proper ancestor of $l(f(k))$.

Let i be the index of the current component when SEARCH(s) was initiated. The search proceeded at that time to the first node adjacent to s and continued until it found a back edge leading to $LOW(s)$, which was assigned then to H_i. Since $LOW(s) < s$, and the search has not backed up from s yet, component H_i is still open. From Fact 1, i is an ancestor of k in $T(B)$. Since $LOW(s) \leq w < l(f(k))$, and because of Fact 5, i is a proper ancestor of $f(k)$. Therefore, when SEARCH(s) was initiated, $H_{f(k)}$ was not created yet. From that time on the search has been restricted to

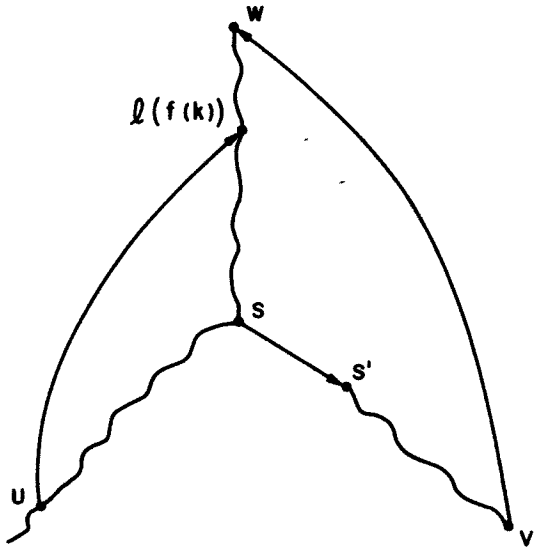

FIGURE 8

descendants of s. Consequently, $l(f(k))$ must have been assigned to $H_{f(k)}$ through a back edge from a descendant u of s. The type of $l(f(k))$ is the same with $t_{f(k)} \neq t_k = t(s)$, and there are three disjoint paths from s to $l(f(k))$ (see Figure 8) ∎

LEMMA 8. *If the algorithm rejects because it finds (line 3) a back edge (v,w) with w having been assigned to a component H'_k other than the current one H_k, then there are three disjoint paths between two nodes of opposite type.*

PROOF. Since $b(w) = k'$, the component $H_{k'}$ is still open and therefore k' is an ancestor of k (Fact 1), and moreover a proper one because $k' \neq k$. Let i be the son of k' in the path from k' to k. We have $t_i \neq t'_k = t_k$, and $l(i) \leq l(f(k)) < w$. Let w' be the son of w in the path from w to v in the DFS tree. Since $b(w) = k'$ and since component H_i is still open when (v,w) is encountered, the search has not backed up to w from the creation of H_i to this moment; i.e. H_i was created at a descendant of w'. Since $l(i) < w$, $l(i)$ must be an ancestor of w assigned to H_i through some back edge from a descendant y of w'. Since $t(w) = t'_k \neq t_i = t(l(i))$, w' must be the first son of w from the first DFS. Therefore w must have been assigned to H'_k through a back edge from a descendant x of w'. Let z be the lowest common ancestor of x and y; z is a descendant of w'. Let z' be the son of z on the path from z to y. Since x was visited before y, $z' \neq A1(z)$ and therefore from the first DFS, $t(z) = t(l(i)) = t_i \neq t(w)$. Thus, there are 3 disjoint paths between two nodes (z and w) of opposite type (see Figure 9) and the algorithm correctly rejects. ∎

LEMMA 9. *If the algorithm rejects on line 6, then there are three disjoint paths between two nodes of opposite type.*

PROOF. Suppose that the algorithm rejects on line 6 of SEARCH(v) because $b(v) = k' \neq k$. Then (v,w) is a tree edge (where $w = A1(v)$), v was assigned to $H_{k'}$ through some back edge (x,v) from a descendant x of w, and therefore $d(v) \geq 3$. Again k' must be a proper ancestor of k in $T(B)$; let i be the son of k' on the path from k' to k and $j = b(w)$ ($j = k$ or a son of k). If $j = i$, then since $k' \neq k$ we must have $k = j$; from line 13, since H_k was not closed at the end of SEARCH(w) and since $t(v) = t_{k'} \neq t_k$, $d(v) \geq 3$, it must be the case that $l(i) < v$. If $j \neq i$ (i.e. j is a proper descendant of i), from Facts 4 and 5, $l(i)$ must be a proper ancestor of w and consequently of v (since the algorithm has not rejected up to this point). In any case,

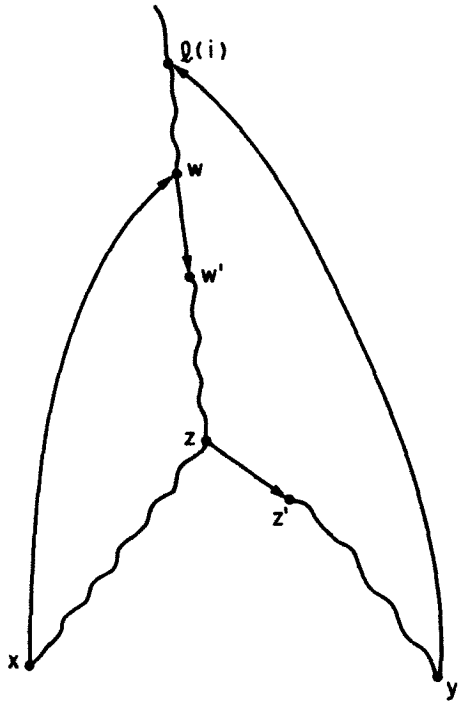

FIGURE 9

$l(i)$ is a proper ancestor of v, and therefore was assigned to H_i through a back edge from a descendant y of w. The proof from here on is similar to that of the previous lemma. ∎

LEMMA 10. *The test on line* 12 *is redundant.*

PROOF. Suppose that $b(v) = k'$, and that after finishing SEARCH(w) where $w \neq A1(v)$, the index of the current component is $k \neq k'$. As before, k' must be a proper ancestor of k, and if i is the son of k' in the path from k' to k, $l(i)$ must be a proper ancestor of v which was assigned to H_i through a back edge from a descendant y of w. But then, since $t(l(i)) = t_i \neq t(v)$, w should be $A1(v)$ from the first DFS. ∎

THEOREM 3. *If the algorithm rejects then the graph is not RTUM.*

PROOF. From Lemmas 7–10. ∎

5. Binary programming with [0, 1] RTUM matrices. We shall show here how the decomposition of the previous sections can be used to reduce the binary programming problem with a [0, 1] RTUM matrix to that with a basic [0, 1] RTUM matrix.

Let A be an RTUM matrix and consider the program:

$$\text{maximize} \quad \mathbf{wx} \qquad \text{subject to} \quad A\mathbf{x} \leqslant \mathbf{b}, \quad \mathbf{x} = 0, 1. \tag{P1}$$

Let G be the graph of A and suppose that v_1, v_2 are two c-nodes that separate G into G_1 and G_2. Let $\alpha_0, \alpha_1, \alpha_2, \alpha_{12}$ be the values of the optimal solutions of the program restricted to the rows and columns of $G_1, G_1 - v_1, G_1 - v_2, G_1 - \{v_1, v_2\}$ respectively, and let us denote in general by $\alpha(H)$ the optimal value for a graph H. (Thus, for example, $\alpha(G_1) = \alpha_0$, and we want to compute $\alpha(G)$.)

From our previous discussions (see e.g. the proof of Theorem 1) it follows that all $v_1 - v_2$ paths in G have the same parity. We are going to show that this parity is related to the sign of $\alpha_0 + \alpha_{12} - \alpha_1 - \alpha_2$.

LEMMA 11. *If* $\alpha_0 + \alpha_{12} - \alpha_1 - \alpha_2 > 0$ *(resp.* < 0*) then all* $v_1 - v_2$ *paths have even (resp. odd) parity.*

PROOF. To prove Lemma 11 we will show at first that it suffices to consider programs with all the right-hand sides (**b**) being 1. We will do this by reducing (P1) to (P1)′ with all the right-hand sides being 1. Suppose that the ith row of A has $b_i > 1$, and let S_i be the set of columns with an 1 in the ith row. Replace the ith row by $2|S_i| - b_i$ new rows with right-hand side 1, and add $|S_i|$ $(|S_i| - b_i)$ new columns and variables with very large weight M. In Figure 10 we show the transformation in terms of the graph of the matrix, when $|S_i| = 5$, $b_i = 3$.

If G' is the new graph (with the new right-hand sides), it is easy to see that $\alpha(G) = \alpha(G') - M\sum(|S_i| - b_i)$: This transformation is similar to the standard reduction of the b-matching problem to the maximum matching problem (see e.g. [B1]). Every path from a c_j to a c_k through the new part of the graph has odd parity. Therefore, the new graph (and matrix) is RTUM if the original was, and if all paths between two c-nodes had the same parity in the old graph, they continue to have the same parity in the new graph. Note now that any separation pair $\{v_1, v_2\}$ of c-nodes in G is also a separation pair in the new graph. Thus, it suffices to prove Lemma 11 when all right-hand sides in (P1) are 1.

A solution to (P1) is described by the set S of variables that have value 1. The *weight* of S is $w(S) = \sum_{x_i \in S} w_i x_i$. Thus, if S is an optimal solution $w(S) = \alpha(G)$. Let S_0, S_1, S_2, S_{12} be optimal solutions to (P1) for $G_1, G_1 - v_1, G_1 - v_2, G_1 - \{v_1, v_2\}$ respectively (with all the right-hand sides 1).

Case 1. (1) $\alpha_0 + \alpha_{12} > \alpha_1 + \alpha_2$. Clearly, $\alpha_0 \geqslant \alpha_1 \geqslant \alpha_{12}$ and $\alpha_0 \geqslant \alpha_2 \geqslant \alpha_{12}$. From (1) we must have $\alpha_0 > \alpha_1$ and $\alpha_0 > \alpha_2$, and therefore $v_1, v_2 \in S_0$. Let S be the symmetric difference of S_0 and S_{12}; i.e. $S = S_0 \oplus S_{12} = (S_0 - S_{12}) \cup (S_{12} - S_0)$, and let R be the set of r-nodes adjacent to nodes in S. Let H be the subgraph of G_1 induced by S and R. Since all right-hand sides are 1, every r-node is adjacent to at most one node of S_0 and one node of S_{12} in H. Since $v_1, v_2 \in S_0$ and $v_1, v_2 \notin S_{12}$, we have $v_1, v_2 \in H$. Suppose that v_1 and v_2 belong to different connected components of H, and let L_0, L_{12} be the set of nodes of S_0, S_{12} in the component to which v_1 belongs. Let $S_0' = S_0 \cup L_{12} - L_0$, and $S_{12}' = S_{12} \cup L_0 - L_{12}$. Clearly S_0' and S_{12}' are solutions of (P1) for $G_1 - v_1$ and $G_1 - v_2$ respectively. We have $w(S_0') = w(S_0) + w(L_{12}) - w(L_0) \leqslant \alpha_1$, $w(S_{12}') = w(S_{12}) + w(L_0) - w(L_{12}) \leqslant \alpha_2$. Therefore $w(S_0') + w(S_{12}') = w(S_0) + w(S_{12})$ $= \alpha_0 + \alpha_{12} \leqslant \alpha_1 + \alpha_2$, contradicting (1). Therefore v_1 and v_2 must belong to the same component K of H, and $v_1, v_2 \in K \cap S_0$. Since every r-node of K is adjacent to at most one node in each of $K \cap S_0$, $K \cap S_{12}$, every path in K connecting two nodes in $K \cap S_0$ must be even. Thus, there is an even path from v_1 to v_2 in G_1 and consequently all $v_1 - v_2$ paths are even.

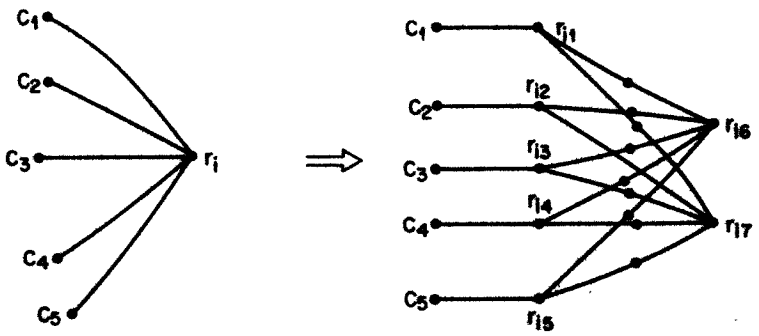

FIGURE 10

Case 2. (2) $\alpha_0 + \alpha_{12} < \alpha_1 + \alpha_2$. From (2) we must have $\alpha_1 > \alpha_{12}$ and $\alpha_2 > \alpha_{12}$, and therefore $v_1 \in S_2$, $v_2 \in S_1$. Let $S = S_1 \oplus S_2$. Then $v_1, v_2 \in S$. Let H be again the subgraph of G_1 induced by S and the set R of r-nodes adjacent to them. Suppose that v_1 and v_2 are in different connected components of H. Let L_1 (resp. L_2) be the set of nodes in S_1 (resp. S_2) that are in the same component of H as v_2. Let $S_1' = S_1 \cup L_2 - L_1$, $S_2' = S_2 \cup L_1 - L_2$. Clearly, $v_1, v_2 \notin S_1'$, and $v_1, v_2 \in S_2'$; therefore $w(S_1') \leqslant \alpha_{12}$, $w(S_2') \leqslant \alpha_0$. We have $w(S_1') + w(S_2') = w(S_1) + w(S_2) = \alpha_1 + \alpha_2 \leqslant \alpha_0 + \alpha_{12}$, contradicting (2). Therefore v_1, v_2 are in the same component of H, and (similarly to Case 1) are connected by an odd path. ∎

We can use Lemma 11 to solve recursively (P1). If v_1, v_2 are two c-nodes of G that separate G into G_1 and G_2, we can find $\alpha_0, \alpha_1, \alpha_2, \alpha_{12}$ and then construct from G_2 an RTUM graph G' with a few more new nodes and a modification of the weights to reflect the solution to G_1. We will give the details in the next two lemmas.

LEMMA 12. *Suppose that all $v_1 - v_2$ paths have even parity. Let G' be as in Figure* 11 *where u_1, u_2 are new r-nodes with right-hand side* 1, *v_3 is a new c-node with weight $w(v_3) = \alpha_{12} + \alpha_0 - \alpha_1 - \alpha_2$, and nodes v_1 and v_2 have new weights $w'(v_1) = \alpha_0 - \alpha_1$, $w'(v_2) = \alpha_0 - \alpha_2$. Then $\alpha(G) = \alpha(G') + \alpha_1 + \alpha_2 - \alpha_0$.*

PROOF. From Lemma 11 we have $\alpha_{12} + \alpha_0 - \alpha_1 - \alpha_2 \geqslant 0$; that is, the weight of v_3 is nonnegative.

(1) $\alpha(G) \geqslant \alpha(G') + \alpha_1 + \alpha_2 - \alpha_0$. Let S' be an optimal solution to G', $S_2 = S' \cap [G_2 - \{v_1, v_2\}]$. Since the weights of v_1, v_2, v_3 are nonnegative, we can assume without loss of generality that at least one of these three nodes is in S'.

(a) $v_3 \in S'$. Then $v_1, v_2 \notin S'$ and $w(S') = w(S_2) + w'(v_3) = w(S_2) + \alpha_{12} + \alpha_0 - \alpha_1 - \alpha_2$. Let S_1 be an optimal solution to $G_1 - \{v_1, v_2\}$. Then $S_1 \cup S_2$ is a solution to G with $w(S_1 \cup S_2) = w(S_1) + w(S_2) = \alpha_{12} + w(S_2) = \alpha(G') + \alpha_1 + \alpha_2 - \alpha_0$.

(b) $v_1 \in S'$, $v_2 \notin S'$ (the case $v_1 \notin S'$, $v_2 \in S'$ is analogous). Then $v_3 \notin S'$. Let S_1 be an optimal solution to $G_1 - \{v_2\}$ and $S = S_1 \cup (S' - v_1)$. S is a solution to G with $w(S) = w(S_1) + w(S') - w'(v_1) = \alpha_2 + \alpha(G') - \alpha_0 + \alpha_1$.

(c) $v_1 \in S'$, $v_2 \in S'$. Let S_1 be an optimal solution to G_1 and $S = S_1 \cup (S' - \{v_1, v_2\})$. S is a solution to G with

$$w(S) = w(S_1) + w(S') - w'(v_1) - w'(v_2)$$
$$= \alpha_0 + \alpha(G') - [2\alpha_0 - \alpha_1 - \alpha_2] = \alpha(G') + \alpha_1 + \alpha_2 - \alpha_0.$$

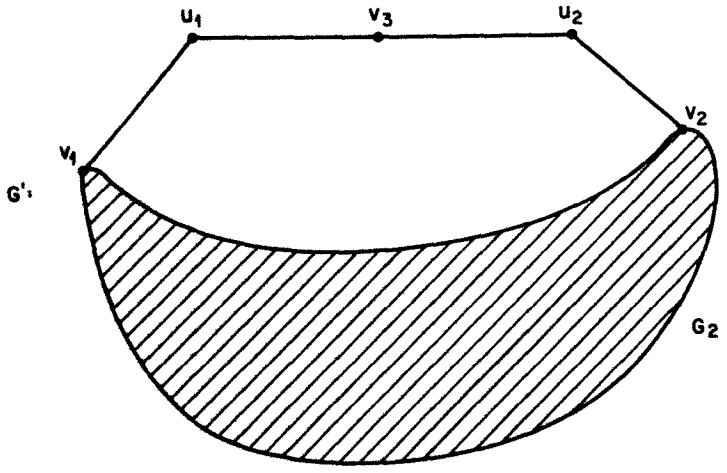

FIGURE 11

(2) $\alpha(G) \leqslant \alpha(G') + \alpha_1 + \alpha_2 - \alpha_0$. Let S be an optimal solution to G, $S_1 = S \cap G_1$, $S_2 = S \cap G_2 - \{v_1, v_2\}$. Let S' be the following solution to G':

$$S' = \begin{cases} S \cap G_1, & \text{if } v_1 \text{ or } v_2 \in S, \\ [S \cap G_2] \cup \{v_3\}, & \text{if } v_1, v_2 \notin S. \end{cases}$$

(a) $v_1, v_2 \notin S$.

$$w(S) = w(S_1) + w(S_2) \leqslant \alpha_{12} + w(S') - w(v_3) \leqslant \alpha(G') + \alpha_1 + \alpha_2 - \alpha_0.$$

(b) $v_1 \in S$, $v_2 \notin S$ (the case $v_1 \notin S$, $v_2 \in S$ is symmetric).

$$w(S) = w(S_1) + w(S_2) \leqslant \alpha_0 + w(S') - w'(v_1) \leqslant \alpha_2 + \alpha(G') - (\alpha_0 - \alpha_1).$$

(c) $v_1, v_2 \in S$.

$$w(S) = w(S_1) + w(S_2) \leqslant \alpha_0 + w(S') - w'(v_1) \leqslant \alpha_2 + \alpha(G') - (\alpha_0 - \alpha_1).$$

$$w(S) = w(S_1) + w(S_2) \leqslant \alpha_0 + w(S') - w'(v_1) - w'(v_2) \leqslant \alpha(G') + \alpha_1 + \alpha_2 - \alpha_0. \quad \blacksquare$$

LEMMA 13. *Suppose that all $v_1 - v_2$ paths have odd parity. Let G' be as in Figure 12 where u_1, u_2, u_3 are new r-nodes with right-hand side 1, v_3, v_4 are new c-nodes with weights $w(v_3) = w(v_4) = \alpha_1 + \alpha_2 - \alpha_0 - \alpha_{12}$, and the nodes v_1 and v_2 have new weights $w'(v_1) = \alpha_2 - \alpha_{12}$, $w'(v_2) = \alpha_1 - \alpha_{12}$. Then, $\alpha(G) = \alpha(G') + 2\alpha_{12} + \alpha_0 - \alpha_1 - \alpha_2$.*

PROOF. From Lemma 11 we have $\alpha_1 + \alpha_2 - \alpha_0 - \alpha_{12} \geqslant 0$; that is, v_3 and v_4 have nonnegative weight.

(1) $\alpha(G) \geqslant \alpha(G') + 2\alpha_{12} + \alpha_0 - \alpha_1 - \alpha_2$. Let S' be an optimal solution to G', and $S_2 = S' \cap [G_2 - \{v_1, v_2\}]$.

(a) $v_1, v_2 \notin S'$. Since v_3 and v_4 have nonnegative weights, we can assume without loss of generality that one of them (but not both) is in S'. So assume that $v_3 \in S'$ (v_4 has the same weight). Let S_1 be an optimal solution to $G_1 - \{v_1, v_2\}$. Then $S = S_1 \cup S_2$ is a solution to G with weight

$$w(S) = w(S_1) + w(S_2) = \alpha_{12} + w(S') - w(v_3) = \alpha_{12} + \alpha(G') - \alpha_1 - \alpha_2 + \alpha_0 + \alpha_{12}.$$

(b) $v_1 \in S'$, $v_2 \notin S'$ (the case $v_1 \notin S'$, $v_2 \in S'$ is analogous). Then $v_3 \notin S'$ and we can assume without loss of generality that $v_4 \in S'$ (since it has nonnegative weight). Let S_1 be an optimal solution to $G_1 - v_2$. Then $S = S_1 \cap S_2$ is a solution to G with

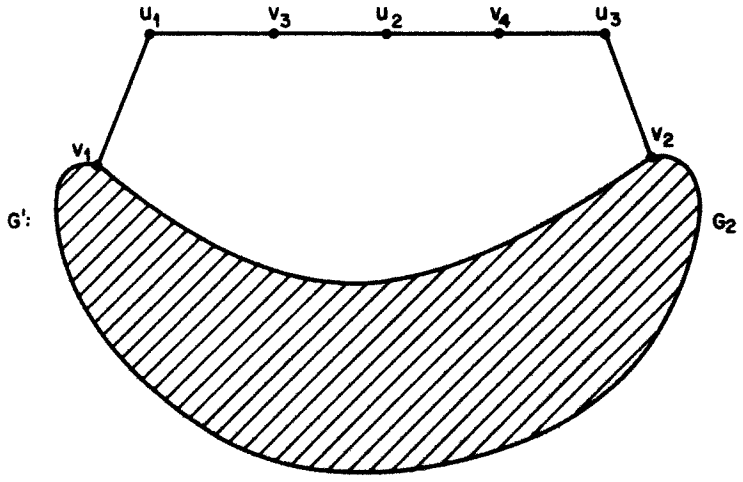

FIGURE 12

weight

$$w(S) = w(S_1) + w(S_2) = \alpha_2 + w(S') - w(v_3) - w'(v_1)$$
$$= \alpha_2 + \alpha(G') - \alpha_1 - \alpha_2 + \alpha_0 + \alpha_{12} - \alpha_2 + \alpha_{12}.$$

(c) $v_1, v_2 \in S'$. Then $v_3, v_4 \notin S'$. Let S_1 be an optimal solution to G_1. Then $S = S_1 \cup S_2$ is a solution to G with weight

$$w(S) = w(S_1 + w(S_2) = \alpha_0 + w(S') - w'(v_1) - w'(v_2)$$
$$= \alpha_0 + \alpha(G') - \alpha_2 + \alpha_{12} - \alpha_1 + \alpha_{12}.$$

(2) $\alpha(G) \leqslant \alpha(G') + 2\alpha_{12} + \alpha_0 - \alpha_1 - \alpha_2$. Let S be an optimal solution to G, $S_1 = S \cap G_1$, $S_2 = S \cap G_2 - \{v_1, v_2\}$. Let S' be the following solution to G':

$$S' = \begin{cases} S_2 \cup \{v_1, v_2\} & \text{if } v_1, v_2 \in S, \\ S_2 \cup \{v_1, v_4\} & \text{if } v_1 \in S, \quad v_2 \notin S, \\ S_2 \cup \{v_2, v_3\} & \text{if } v_1 \notin S, \quad v_2 \in S, \\ S_2 \cup \{v_3\} & \text{if } v_1, v_2 \notin S. \end{cases}$$

(a) $v_1, v_2 \notin S$.

$$w(S) = w(S_1) + w(S_2) \leqslant \alpha_{12} + w(S') - w(v_3) \leqslant \alpha_{12} + \alpha(G') - \alpha_1 - \alpha_2 + \alpha_0 + \alpha_{12}.$$

(b) $v_1 \in S$, $v_2 \notin S$ (the case $v_1 \notin S$, $v_2 \in S$ is symmetric).

$$w(S) = w(S_1) + w(S_2) \leqslant \alpha_2 + w(S') - w'(v_1) - w(v_4)$$
$$\leqslant \alpha_2 + \alpha(G') - \alpha_2 + \alpha_{12} - \alpha_1 - \alpha_2 + \alpha_0 + \alpha_{12}.$$

(c) $v_1, v_2 \in S$.

$$w(S) = w(S_1) + w(S_2) = \alpha_0 + w(S') - w'(v_1) - w'(v_2)$$
$$\leqslant \alpha_0 + \alpha(G') - \alpha_2 + \alpha_{12} - \alpha_1 + \alpha_{12}. \quad \blacksquare$$

The new graph G' in each case is RTUM. We can use Lemmas 12, 13 to solve (P1) for a biconnected graph B as follows. After computing the tree $T(B)$ that describes the decomposition of B into basic components we proceed in $T(B)$ bottom-up. Let H_i be a basic component, and let j be the father of i in $T(B)$. If H_i is a type c component, let G_i be the subgraph of G induced by the nodes of G in the components that are descendants of H_i (including H_i). If H_i is a type r component we include also the c-nodes $x_{j1}((j,i)), x_{j2}((j,i))$ into G_i. We proceed from the leaves of $T(B)$ to the root, computing at each node i, $\alpha(G_i), \alpha(G_i - v_1), \alpha(G_i - v_2), \alpha(G_i - \{v_1, v_2\})$ where $v_{1,2} = x_{i1,2}((j,i))$ or $v_{1,2} = x_{j1,2}((j,i))$ according as H_i is of type c or type r, and then recording the solutions into the virtual path of H_j corresponding to the edge (j,i) of $T(B)$ as described in Lemmas 12, 13. (Only here a virtual path of odd parity has length 6 instead of the length 2 in the previous section.)

At each stage in this bottom-up procedure we have to solve four instances of (P1) on a graph that is almost basic RTUM. More specifically, let i be a node of the decomposition tree $T(B)$ and H_i the corresponding basic component. When we reach node i of the tree, the nodes of the graph that belong to components that are descendants of i have been reflected on the weights of the c-nodes of the virtual paths of H_i. At node i of $T(B)$ we solve 4 problems on a graph $\overline{H}_i$ which is obtained from H_i as follows. If H_i is of type c, $\overline{H}_i$ does not have the virtual path of H_i corresponding to the edge (j,i) of $T(B)$ (where j is the father of i); $v_1 = x_{i1}((j,i))$ and $v_2 = x_{i2}((j,i))$. In this case $\overline{H}_i$ is basic RTUM: all its r-nodes have degree 2. If H_i is of type r, $\overline{H}_i$ does

not contain again the virtual path of H_i corresponding to (j,i), and in addition it contains the c-nodes $v_1 = x_{j1}((j,i))$, $v_2 = x_{j2}((j,i))$ and two edges connecting them respectively to $x_{i1}((j,i))$ and $x_{i2}((j,i))$. In this case $\overline{H}_i$ is not quite basic RTUM because the c-nodes v_1 and v_2 have degree 1 (instead of 2). This can be easily taken care of. Add a path of new nodes between v_1 and v_2 of length 4 or 6 depending on the parity of the $v_1 - v_2$ paths, where the new r-nodes have right-hand side 1 and the new c-nodes have weight 0. With this path $\overline{H}_i$ becomes basic RTUM. Clearly, the new nodes have no effect on the optimal solution: a solution for the old graph is also a solution for the new graph with the same weight and vice-versa. To compute the optimal solution for $\overline{H}_i$ with one or both of v_1, v_2 deleted, just set the corresponding weights to a negative number.

Suppose now that G is not biconnected. To simplify our subsequent discussion we will first get rid of trivial blocks (blocks that consist of single edge). For every trivial block (u,v) of G we add a path of length 3 of new nodes between u and v, where the new r-nodes have right-hand side 1 and the new c-nodes have weight 0. Clearly, the new nodes have no effect on the optimal solution. Thus, we can assume without loss of generality that all blocks are nontrivial and have been decomposed into their basic components.

Let BC be the block-cutpoint tree of G. We proceed bottom-up in BC. Let v be a cutpoint, B_0 the father of v, and $B_1, \ldots, B_k$ the sons of v. Let $H_0, H_1, \ldots, H_k$ be the basic components of $B_0, B_1, \ldots, B_k$ respectively to which v belongs. From our decomposition, $t(H_i) = t(v)$ for $i \geq 1$, because v is the root of the DFS tree for B_i. If H_0 is also of type $t(v)$, then we solve (P1) simultaneously on the union of the H_i's (which is also basic of type $t(v)$). Suppose that $t(H_0) \neq t(v)$. If v is of type c, then let H be the union of the H_i's for $i \geq 1$; H is basic of type c. Compute $\alpha(H)$ and $\alpha(H - v)$, and give new weight to v in H_0: $w'(v) = \alpha(H) - \alpha(H - v)$. It is easy to see that $\alpha(H_0 \cup H) = \alpha'(H_0) + \alpha(H - v)$, where $\alpha'(H_0)$ is the optimal value for H_0 with the new weight for v. If v is of type r, then H_0 is of type c; that is, all its r-nodes have degree 2. Let v_1, v_2 be the two c-nodes adjacent to v in H_0; v_1, v_2 separate the graph into two parts and the paths connecting them have odd parity. So we can use Lemma 13. That is, we let H be the union of the H_i's for $i \geq 1$ together with the nodes v_1, v_2. We solve 4 instances of (P1) on H (with none, one, or both of v_1, v_2 deleted), and add a path of length 6 to H_0 with the weights reflecting the solutions.

Thus, we can solve (P1) on the graph G by solving at most 4 such problems on every basic component of each block of G. The number of new nodes added to the various basic components of G is clearly at most linear in the number of nodes of G. If H is a basic component of type r, then the problem (P1) is a b-matching problem on a bipartite multigraph. If H is a basic component of type c (every row has 2 nonzero entries) then we can assume without loss of generality that all right-hand sides are 1, and (P1) is the problem of finding a maximum weight independent set of nodes in a bipartite graph. Let us call a function $f(n,m)$ *subadditive* if $f(n_1,m_1) + f(n_2,m_2) \leq f(n_1 + n_2, m_1 + m_2)$. From our previous discussion we have:

THEOREM 4. *Suppose that the b-matching problem and the maximum weight independent set problem can be solved in $f(n,m)$ and $g(n,m)$ time respectively on a bipartite multigraph with n nodes and m edges, where f and g are subadditive functions. Then* (P1) *can be solved for an $n \times m$ RTUM matrix A in time $O(f(n,m) + g(m,n))$.*

The b-matching problem can be solved in $O(m^2 \log m)$ time by the Hungarian method (see e.g. [L]). The maximum weight independent set problem on a bipartite graph can be reduced to a maximum flow problem as follows. If G is a bipartite graph with (S,T) a bipartition of its nodes, add two nodes s and t, a directed edge from s to each node v_i in S with capacity w_i, a directed edge from each node v_j in T to t with

capacity w_j, and direct all edges of G from S to T and assign to them a very large capacity M. (Note that we can assume without loss of generality that all weights are positive.) A minimum $s-t$ cut in this network corresponds to a minimum weight node cover of G (see e.g. [L]). Therefore, if f is a value of the maximum $s-t$ flow and $\alpha(G)$ the maximum weight of an independent set of G, we have from the max-flow min-cut theorem: $\alpha(G) = (\sum_{v_i \in G} w_i) - f$. The maximum flow problem can be solved in $O(nm \cdot \log n)$ time [ST]. Thus, from Theorem 4 we have

COROLLARY 2. *For an $n \times m$ RTUM matrix A, the program* (P1) *can be solved in $O(m^2 \log m + mn \log m)$ time.*

In the rest of this section we will show that similar results hold if in (P1) we have both kinds of inequalities ($\leqslant$ and $\geqslant$) and equalities. That is, let (P2) be the following program:

$$\text{maximize} \quad \mathbf{wx} \qquad \text{subject to} \quad \begin{cases} A_1\mathbf{x} \leqslant \mathbf{b}_1, \\ A_2\mathbf{x} = \mathbf{b}_2, \\ A_3\mathbf{x} \geqslant \mathbf{b}_3, \\ \mathbf{x} = 0, 1, \end{cases} \tag{P2}$$

where the constraint matrix

$$A = \begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix}$$

is RTUM.

We can reduce (P2) to (P1) as follows. At first we replace every $\geqslant$ inequality by a $\leqslant$ inequality and some equalities. If $\mathbf{a}_i\mathbf{x} \geqslant b_i$ is a $\geqslant$ inequality of (P2) we introduce k new variables x'_{ij}, one for each column j that has an 1 in the ith row of A_3, add the equalities $x_j + x'_{ij} = 1$, and replace $\mathbf{a}_i x \geqslant b_i$ by $\sum_{a_{ij}=1} x'_{ij} \leqslant k - b_i$. The new variables x'_{ij} are given weight 0. In Figure 13 we show graphically this transformation. From the figure it is obvious that a path $c_j - c_l$, where $a_{ij} = a_{il} = 1$, through the new part of the graph has odd parity (as before the transformation), and therefore the new matrix A' (and the new graph G') is also RTUM. Clearly, there is a 1-1 weight-preserving correspondence between solutions to the new and old program.

Next, we can change all equalities to $\leqslant$ by adding to the weight of each variable y the quantity lM where l is the number of equalities to which y participates and M is a very large number, say $M > 2\sum|w_i|$. Let (P2)′ be the new program, α and α' the optimal values of (P2) and (P2)′ respectively. Let β be the sum of the right-hand sides of all the equalities. It is easy to see that (P2) is feasible iff $\alpha' \geqslant M\beta - (M/2)$, and that an optimal solution for (P2) with weight α is also an optimal solution for (P2)′ with weight $\alpha' = \alpha + M\beta$ and vice-versa.

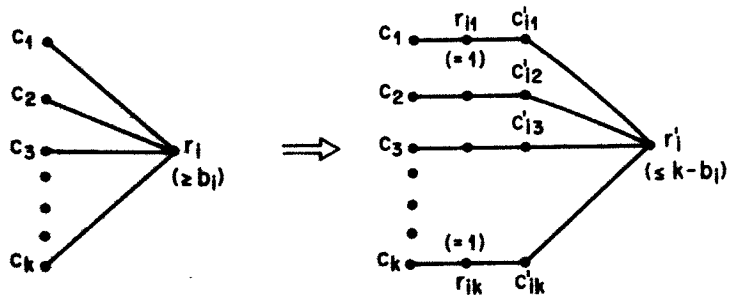

FIGURE 13

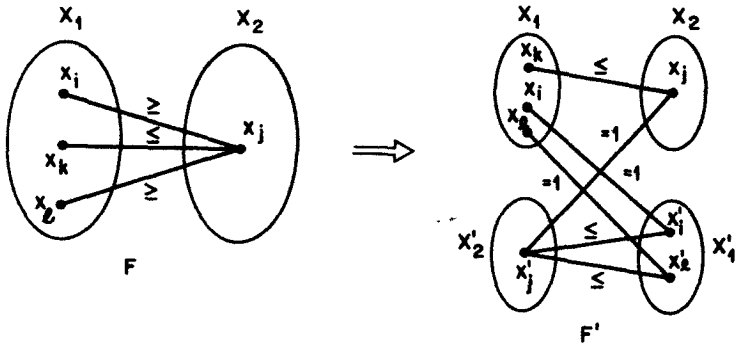

FIGURE 14

The transformation from (P2) to (P2)′ increases the size of A by a linear amount at most. However, it is better not to apply the transformation to the whole graph but only to its basic components; the fact that there is a correspondence between optimal solutions of (P2) and (P2)′, and that the transformation does not affect the parity of paths implies that Lemmas 11–13 hold also for (P2). Therefore, we can combine the optimal solutions of (P2) on the basic components of $G(A)$ as before to obtain the optimal solution for $G(A)$ itself. If H is a basic component of type c (variables correspond to the nodes of a bipartite multigraph F) then in the previous transformation we don't have to add a new variable x'_{ij} for every $\geqslant$ constraint (edge of F) r_i that x_j participates: it suffices to add only one complementary variable x'_j to play the role of all x'_{ij}—see Figure 14. (We could have done this with the whole graph to begin with; however, it would not be as trivial to see that it would not affect the RTUM property of the graph.) The new bipartite multigraph F' has at most $2n$ nodes and $m + n$ edges, if the original F had n nodes and m edges. If H is a basic component of type r (variables correspond to the edges of a bipartite multigraph F) then for every $\geqslant$ constraint (node v of F) we add a new node v' to F, and k edges between v and v', where k is the degree of v in F. If b_v is the right-hand side of the $\geqslant$ constraint that corresponds to node v, we let $k - b_v$ of the k new edges have weight 0 and the others a very large negative weight. We change the $(\geqslant b_v)$ constraint at v into an $(= k)$ constraint and assign a vacuous constraint (e.g. $\leqslant k$) to v'. Clearly the new bipartite multigraph F' has at most $2n$ nodes and $2m$ edges, if the original graph F has n nodes and m edges. Thus, from Theorem 4 we have

THEOREM 5. *Suppose that the b-matching problem and the maximum weight independent set problem can be solved in $f(n,m)$ and $g(n,m)$ time respectively on a bipartite multigraph with n nodes and m edges, where f and g are subadditive functions. Then* (P2) *can be solved for an $n \times m$ RTUM matrix A in time $O(f(n,m) + g(m,n))$.*

6. [−1, 0, 1] RTUM matrices. In this section we will show that [−1, 0, 1] RTUM matrices are not essentially different from the [0, 1] ones. Let D be the digraph of a

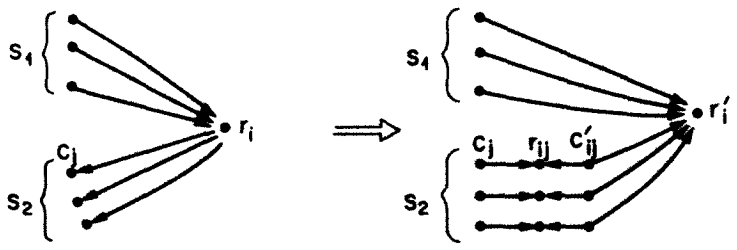

FIGURE 15

$[-1,0,1]$ matrix A. For each r-node of D with some outgoing edges apply the transformation of Figure 15.

Let D' be the new digraph. In D' all edges incident to r-nodes are coming in, and therefore D' is the digraph of a $[0,1]$ matrix A'.

LEMMA 14. *A is RTUM if and only if A' is.*

PROOF. It suffices to show that for every pair of nodes u, v in $S_1 \cup S_2$ the parity of a $u - v$ path p' in the new part of the digraph is the same as the parity of the path $p: u - r_i - v$ in the old digraph.

(a) $u, v \in S_1$. Then both p and p' have odd parity.

(b) $u, v \in S_2$. Then $\sigma(p) = -1$, and $\sigma(p') = (-1)^3 = -1$.

(c) $u \in S_1$, $v \in S_2$. Then $\sigma(p) = 1$, and $\sigma(p') = (-1)^2 = 1$. ∎

Thus, the algorithm of §4 can be applied to A' to test if the $[-1,0,1]$ matrix A is RTUM in $O(n+m)$ time. In fact, we don't have to go through the transformation to test A: The transformation shows that Lemma 2 and therefore also Lemmas 3, 4 and Theorem 1 hold also for RTUM digraphs. Consequently, an RTUM digraph can be decomposed into *basic RTUM digraphs* (digraphs of RTUM matrices with 2 nonzero entries in each row or in each column) exactly as an RTUM graph. A basic digraph is RTUM iff all its fundamental cycles are even: This can be easily shown directly from the condition of §2, or using the transformation of Figure 15 together with the fact that this holds for basic graphs. Therefore we can apply the algorithm of §4 directly to D; the only thing that needs modification is the assignment of levels to the nodes and the test of the fundamental cycles, to reflect the direction of the edges of D.

Regarding now the 0, 1 Programming Problem (P2) with a $[-1,0,1]$ RTUM matrix A, consider again the transformation of Figure 15, where for each $c_j \in S_2$ the weight of the variable x'_{ij} that corresponds to the new node c'_{ij} is 0, the constraint that corresponds to r_{ij} is: $x_j + x'_{ij} = 1$, and if the constraint of r_i was $\leqslant k$ (resp. $=$, $\geqslant k$) the new constraint that corresponds to r'_i is $\leqslant k + |S_2|$ (resp. $=$, $\geqslant k + |S_2|$). If (P2)′ is the new program, it is easy to see that there is a 1-1 correspondence between the solutions of (P2) and (P2)′ that preserves the weights. Thus, the techniques of the previous section carry over directly to $[-1,0,1]$ RTUM matrices as well. Note again that it is not necessary to perform the transformation on the whole digraph but on its basic components. Then, again, we just have to introduce one x'_j to play the role of all x'_{ij}, if the basic component is of type r (the variables correspond to the nodes of a graph), and if the basic component is of type c (variables correspond to edges) we have to introduce at most one new node and an appropriate number of edges for each constraint. Therefore, Theorem 5 holds also for $[-1,0,1]$ RTUM matrices.

7. Conclusions. We showed in this paper how RTUM graphs can be recognized and decomposed into basic parts efficiently. Bipartite graphs are those with cycles having length of a multiple of 2; their recognition is very easy. RTUM graphs are those with cycles having length a multiple of 4; their recognition is not as easy, but as we showed can be efficiently done. A generalization of the problem in this direction (posed by Garey and Tarjan) is: given a k, how hard is it to determine whether all cycles of a graph have length a multiple of k—or more generally, find the greatest common divisor of the lengths of the cycles of a graph. Another related extension concerns the lengths of the *chordless* cycles of a graph. The analogues of bipartite graphs in this case are related to *perfect graphs*, i.e. graphs all of whose induced subgraphs have the chromatic number equal to the maximum clique number [B1]. The (still unproved) Strong Perfect Graph Conjecture says that a graph is perfect if neither it nor its complement contains a chordless cycle of odd length > 5. The analogues of RTUM graphs (i.e. all chordless cycles having length a multiple of 4) are exactly the

graphs of balanced matrices [B2]. (Although not immediately obvious, it is easy to see that these graphs have to be bipartite and therefore can be associated with a [0, 1] matrix.) It would be interesting to see if an analogous decomposition theorem can be proved for balanced matrices, and how it could be used to solve their Integer Programs.

Regarding the general Integer Programs of RTUM matrices, one could apply a similar technique as for the 0, 1 case. The basic problem in this case is a min-cost circulation problem, which can be solved by the out-of-kilter method with successive approximations in $O(m^2p)$ time, where m is the number of edges of the network and p the logarithm of the largest capacity ([EK], [L]). In order to combine the various solutions, one would have to do a parametric analysis of the individual basic problems (i.e. see how the optimal value is affected by changing e.g. the cost of the flow in an edge or the capacity), which for the 0, 1 case was trivial. It can be shown that the parametric analysis will give a piecewise linear convex cost function for the subproblem. The out-of-kilter method works also for such cost functions [L]; in fact, any method should, since one can easily change an edge with a piecewise linear convex cost function into k edges (where k is the number of different slopes of the cost function) with a linear cost.

References

[AHU] Aho, A. V., Hopcroft, J. and Ullman, J. D. (1974). *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass.

[B1] Berge, C. (1973). *Graphs and Hypergraphs*. North-Holland, Amsterdam.

[B2] ——— (1972). Balanced Matrices. *Math. Programming* **2** 19–31.

[C] Commoner, F. G. (1973). A Sufficient Condition for a Matrix to be Totally Unimodular. *Networks* **3** 351–365.

[EK] Edmonds, J. and Karp, R. M. (1972). Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *J. ACM* **19** 248–264.

[FHO] Fulkerson, D. R., Hoffman, A. J. and Oppenheim, R. (1974). On Balanced Matrices. *Math. Programming Study* **1** 120–132.

[GN] Galil, Z. and Naamad, A. (1979). Network Flow and Generalized Path Compression. *Proc. 11th Annual ACM Sympos. on Theory of Computing*, 13–26.

[H] Harary, F. (1969). Graph Theory. Addison-Wesley, Reading, Mass.

[HK] Hoffman, A. J. and Kruskal, J. B., Integral Boundary Points of Convex Polyhedra. H. W. Kuhn and A. W. Tucker eds., (1956). *Linear Inequalities and Related Systems*. Ann. Math. Study 38. Princeton University Press, Princeton, N.J., 223–246.

[HT1] Hopcroft, J. and Tarjan, R. E. (1973). Efficient Algorithms for Graph Manipulation. *Comm. ACM* **16** 372–378.

[HT2] ——— and ———. (1973). Dividing a Graph into Triconnected Components. *SIAM J. Computing* **2** 135–158.

[K] Khachian, L. G. (1979). Polynomial Algorithm for Linear Programming. *Dokl. Akad. Nauk USSR* **244** 1093–1096.

[L] Lawler, E. L. (1976). *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, New York.

[MTA] Maurras, J. F., Truemper, K. and Akgul, M. (1981). Polynomial Algorithms for a Class of Linear Programs. *Math. Programming* **21** 121–136.

[S] Seymour, P. D. Decomposition of Regular Matroids. *J. Combin. Theory Ser.* B **28** 305–359.

[ST] Sleator, D. D. and Tarjan, R. E. (1981). A Data Structure for Dynamic Trees. *Proc. 13th ACM Sympos. on Theory of Computing* 114–122.

[Tam] Tamir, A. (1976). On Totally Unimodular Matrices. *Networks* **6** 373–382.

[T] Tarjan, R. E. (1972). Depth-First Search and Linear Graph Algorithms. *SIAM J. Computing* 1 146–159.

AT&T BELL LABORATORIES, MURRAY HILL, NEW JERSEY 07974