Algorithmica (1989) 4: 471–501

Algorithmica

Algorithms for Multicommodity Flows in Planar Graphs

Hitoshi Suzuki,[1] Takao Nishizeki,[1] and Nobuji Saito[1]

Abstract. This paper gives efficient algorithms for the multicommodity flow problem for two classes C_{12} and C_{01} of planar undirected graphs. Every graph in C_{12} has two face boundaries B_1 and B_2 such that each of the source-sink pairs lies on B_1 or B_2. On the other hand, every graph in C_{01} has a face boundary B_1 such that some of the source-sink pairs lie on B_1 and all the other pairs share a common sink lying on B_1. The algorithms run in $O(kn + nT(n))$ time if a graph has n vertices and k source-sink pairs and $T(n)$ is the time required for finding the single-source shortest paths in a planar graph of n vertices.

Key Words. Algorithm, Cut condition, Multicommodity flow, Network, Planar graph, Shortest path.

1. Introduction. The network flow problem and its variants have been extensively studied. It is well known that the Max Flow–Min Cut theorem holds for single- and two-commodity flows. There are efficient algorithms for finding a maximum single-commodity flow [14]. It is also known that the maximum two-commodity flows in an undirected graph can be found by using an algorithm for a single-commodity flow [13]. The situation is different with regard to flows of more than two commodities. No simple polynomial-time algorithm is known for the multicommodity flow problem on general graphs. Very recently Tardos reported a strongly polynomial algorithm to solve combinatorial linear programs including the multicommodity flow problem [19]. However, it employs a polynomial linear programming algorithm, does not have a polynomial time bound of lower order, nor is easy to implement. Therefore simple efficient algorithms are useful in practice even if they are valid for restricted classes of graphs [2], [5], [7], [8], [12].

In the multicommodity flow problem, we would like to (1) test the feasibility, that is, decide whether a given graph G has multicommodity flows, each from a source to a sink and of a specified demand, and (2) then actually find them if G does have them. The problem can be applied to many practical problems such as traffic control, design of communication networks, and routing of VLSI [9], [16].

This paper deals with the multicommodity flow problem for two classes C_{12} and C_{01} of planar *undirected* graphs. Every planar graph in the first class C_{12} has two specified face boundaries B_1 and B_2 such that each of the source-sink pairs lies on B_1 or B_2. On the other hand, every planar graph in the second class

[1] Department of Electrical Communications, Faculty of Engineering, Tohoku University, Sendai 980, Japan.

Received January 30, 1987; revised November 9, 1987. Communicated by Tatsuo Ohtsuki.

C_{01} has one specified face boundary B_1 such that some of the source-sink pairs lie on B_1 and all the other pairs share a common sink located on B_1. Figure 1 depicts two planar graphs belonging to C_{12} or C_{01}. The Max Flow-Min Cut theorem is known to hold for graphs in C_{12} or C_{01} [10]. We show that the multicommodity flow problem for an *undirected* graph G in C_{12} (resp. C_{01}) can be reduced to the shortest path problem for an *undirected* (resp. a *directed*) graph

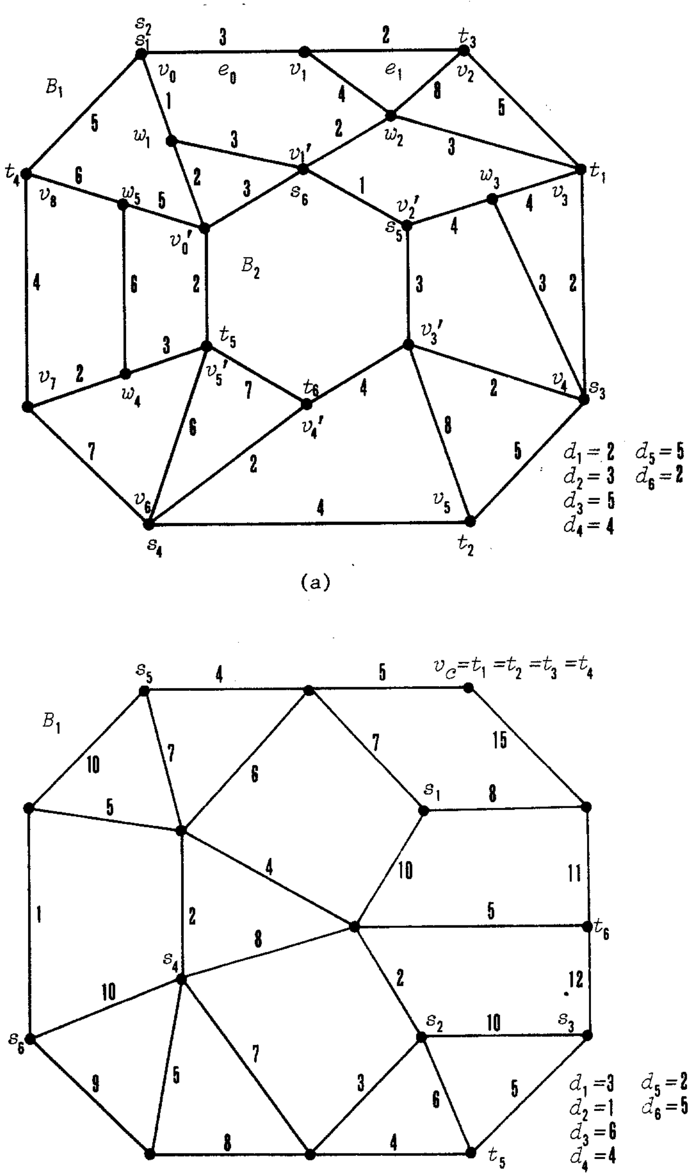

Fig. 1. (a) A network in C_{12} and (b) a network in C_{01}. (s_i is a source, t_i is a sink, and d_i is a demand.)

obtained from the dual of G. The reduction yields simple polynomial-time algorithms for the two classes of planar graphs. More precisely, we can find multicommodity flows in a graph belonging to C_{12} or C_{01} by solving the single-source shortest path problem $O(n)$ times, where n is the number of vertices in a graph. Furthermore, we can find half-integral flows for C_{12} and C_{01} if the capacities of edges and the demands of source-sink pairs are all integers. However, our algorithms do not work for directed graphs. A preliminary version of this paper appeared as [17].

2. Preliminaries. In this section we first define some terms, and then present known results.

A flow *network* $N=(G, P, c)$ is a triplet, where:

(1) $G=(V, E)$ is a finite undirected simple connected graph with vertex set V and edge set E.
(2) P is the set of source-sink pairs (s_i, t_i), where *source* s_i and *sink* t_i are distinguished vertices in V. Both source and sink are often called *terminals.*
(3) c: $E \to R^+$ is the *capacity* function. (R (or R^+) denotes the set of (nonnegative) real numbers.)

A network $N=(G, P, c)$ is *planar* if G is planar. In what follows, we assume that G has n vertices and P contains k source-sink pairs, i.e., $|V|=n$ and $|P|=k$. Each source-sink pair (s_i, t_i) of N is associated with a positive *demand* d_i. Although G is undirected, we orient the edges of G arbitrarily so that the sign of a value of a flow function can indicate the real direction of the flow through an edge. A set of functions $\{f_1, f_2, \ldots, f_k\}$ with each f_i: $E \to R$ is *k-commodity flows* of demands $d_1, d_2, \ldots, d_k$ if it satisfies:

(a) For each $e \in E$

$$\sum_{i=1}^{k} |f_i(e)| \le c(e).$$

(b) Each f_i satisfies

$$\mathrm{IN}(f_i, v) = \mathrm{OUT}(f_i, v)$$

for each $v \in V - \{s_i, t_i\}$, and

$$\mathrm{OUT}(f_i, s_i) - \mathrm{IN}(f_i, s_i) = \mathrm{IN}(f_i, t_i) - \mathrm{OUT}(f_i, t_i) = d_i,$$

where $\mathrm{IN}(f_i, v)$ is the total amount of flow f_i of commodity i entering v, and $\mathrm{OUT}(f_i, v)$ is the total amount of flow f_i emanating from v.

Classes C_1, C_{12}, and C_{01} of planar networks $N=(G, P, c)$ are formally defined as follows:

(1) Class C_1. One face boundary of G is specified, and all the source-sink pairs are located on it.

(2) Class C_{12}. Two face boundaries B_1 and B_2 of G are specified, and each of the source-sink pairs lies on B_1 or B_2. That is, the set P is partitioned into P_1 and P_2 so that

$$\text{if} \quad (s_i, t_i) \in P_1 \quad \text{then} \quad s_i, t_i \in B_1$$

and

$$\text{if} \quad (s_i, t_i) \in P_2 \quad \text{then} \quad s_i, t_i \in B_2.$$

(3) Class C_{01}. One face boundary B_1 together with a vertex v_c on B_1 is specified, and some of the source-sink pairs are located on B_1, while the sinks of all the other pairs must lie on v_c but their sources can lie anywhere in G. That is, the set P is partitioned into P_0 and P_1 so that

$$\text{if} \quad (s_i, t_i) \in P_0 \quad \text{then} \quad t_i = v_c$$

and

$$\text{if} \quad (s_i, t_i) \in P_1 \quad \text{then} \quad s_i, t_i \in B_1.$$

Note that C_1 is a subclass of C_{12} and of C_{01}. For the network of C_{12} depicted in Figure 1(a) $P_1 = \{(s_1, t_1), \ldots, (s_4, t_4)\}$ and $P_2 = \{(s_5, t_5), (s_6, t_6)\}$. For the network of C_{01} depicted in Figure 1(b) $P_0 = \{(s_1, t_1), \ldots, (s_4, t_4)\}$ and $P_1 = \{(s_5, t_5), (s_6, t_6)\}$.

We have already given a polynomial-time algorithm MULTIFLOW which finds multicommodity flows in a network belonging to class C_1 and runs in $O(n(k + T_+(n)))$ time [7]. Throughout the paper $T_+(n)$ denotes the time required for finding the single-source shortest paths in a planar graph with nonnegative edge weights having n vertices, while $T_-(n)$ denotes the time for a planar directed graph with edge weights of real numbers.

We may assume without loss of generality that B_1 is the boundary of the *outer* face of a given plane graph G. The face boundaries B_1 and B_2 are not always simple cycles, but are closed walks (that is, some vertices or edges may appear twice or more). We denote by b_1 (resp. b_2) the number of edges on B_1 (resp. B_2), and denote by B_1 (resp. B_2) the set of vertices and also the set of edges on B_1 (resp. B_2). Let $v_0, v_1, \ldots, v_{b_1-1}$ be the sequence of vertices appearing on B_1 in clockwise order, and let $e_i = (v_i, v_{i+1})$, $i = 0, 1, \ldots, b_1 - 1$, where $v_{b_1} = v_0$. Let $v'_0, v'_1, \ldots, v'_{b_2-1}$ be the vertices on B_2 appearing in clockwise order, and let $e'_i = (v'_i, v'_{i+1})$, $i = 0, 1, \ldots, b_2 - 1$, where $v'_{b_2} = v'_0$.

We denote by $E(X, Y)$ the set of edges with one end in $X \subset V$ and the other in $Y \subset V$. If $X \subset V$, then $E(X) = E(X, V - X)$ is called a *cut*. $E(X)$ is called a *cutset* if the graph $G - E(X)$ obtained from G by deleting the edges in $E(X)$ has one more connected components than G. Define

$$c(X, Y) = \sum \{c(e) | e \in E(X, Y)\}$$

and

$$c(X) = c(X, V - X).$$

Denote by $D(X, Y)$ the set of source-sink pairs with one terminal in X and the other in Y. Define

$$D(X) = D(X, V-X),$$
$$d(X, Y) = \sum \{d_i | (s_i, t_i) \in D(X, Y)\},$$
$$d(X) = d(X, V-X),$$

and

$$m(X) = c(X) - d(X) \qquad \text{(the } \textit{margin} \text{ of a cut)}.$$

We say that a network N satisfies the *cut condition* for given demands if $m(X) \geq 0$ for every $X \subset V$. The cut condition is necessary for the existence of k-commodity flows of given demands in a network, but not necessarily sufficient. However, Okamura [10] has proved the following theorem.

THEOREM 1 [10]. *Let $N = (G, P, c)$ be a planar network in class C_{12} or C_{01}. Then N has multicommodity flows of given demands if and only if N satisfies the cut condition.*

Our algorithms are based on Theorem 1. The Max Flow-Min Cut theorem does not always hold for general undirected planar networks having source-sink pairs on three or more face boundaries or for directed planar networks [7], [10]. Therefore our algorithms do not work for these networks.

The following lemmas have been known.

LEMMA 1 [7], [11]. *A network $N = (G, P, c)$ satisfies the cut condition if and only if $m(X) \geq 0$ for every cutset $E(X)$.*

LEMMA 2 [10], [11]. *Let $N = (G, P, c)$ be a network satisfying the cut condition. If $m(X) = m(Y) = 0$ and $d(X-Y;\ Y-X) = 0$ for $X, Y \subset V$, then $m(X \cap Y) = m(X \cup Y) = 0$ and $c(X-Y;\ Y-X) = 0$.*

PROOF. By simple counting we have the following two equations:

$$c(X) + c(Y) = c(X \cup Y) + c(X \cap Y) + 2c(X-Y;\ Y-X)$$

and

$$d(X) + d(Y) = d(X \cup Y) + d(X \cap Y) + 2d(X-Y;\ Y-X).$$

Subtracting the latter from the former we have

$$m(X) + m(Y) = m(X \cup Y) + m(X \cap Y) + 2c(X-Y;\ Y-X) - 2d(X-Y;\ Y-X).$$

The claim follows immediately from the above equation. □

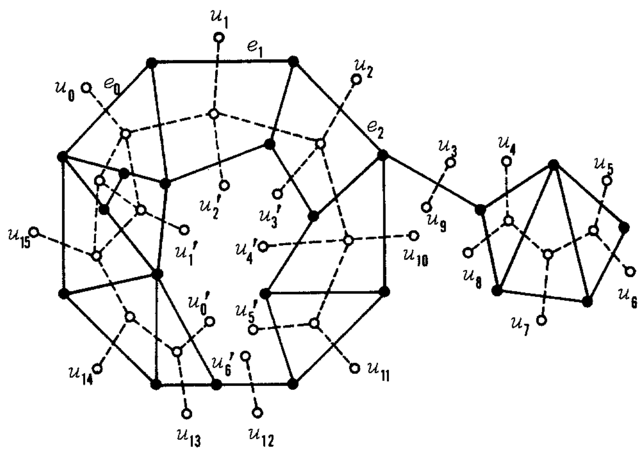

Fig. 2. Graph G in C_{12} and its G^*.

3. Testing Feasibility for Class C_{12}. In this section we give an algorithm to test the feasibility of a given network $N=(G, P, c)$ belonging to class C_{12}, that is, to determine whether there are multicommodity flows in N. A new graph G^* is constructed from G as follows (see Figure 2):

(1) Replace each edge $e \in B_1 \cup B_2$ of G with two multiple edges (if either $e \in B_1 \cap B_2$ or e is a bridge (i.e., a cutset of a single edge) then replace e with three edges).
(2) Construct a dual of the resulting (multi)graph.
(3) Remove from the dual the two vertices corresponding to B_1 and B_2.

Figure 2 illustrates a plane graph G in C_{12} and the corresponding G^*, where G is drawn by solid lines and G^* by dashed lines. Let u_i be the vertex of G^* corresponding to edge e_i on B_1, $i=0, 1, \ldots, b_1-1$, and let $U_1=\{u_0, u_1, \ldots, u_{b_1-1}\}$. Similarly define u'_i, $i=0, 1, \ldots, b_2-1$, and U_2 with respect to B_2. Each edge of G^* has length equal to the capacity of the corresponding edge of G.

We may assume that $b_1, b_2 \leq 2k$: otherwise, new edges of capacity zero can be added to G to yield $b_1, b_2 \leq 2k$, as discussed in [5] and [7]. Theorem 1 and Lemma 1 together imply that we can test the feasibility by verifying whether $m(X) \geq 0$ for every cutset $E(X)$. Since G is planar, $|E(X) \cap B_1| = 0$, 1, or 2 and $|E(X) \cap B_2| = 0$, 1, or 2 for every cutset $E(X)$. Therefore the cutsets are classified into four types:

(0) $E(X) \cap B_1 = \varnothing$ and $E(X) \cap B_2 = \varnothing$;
(1) $|E(X) \cap B_1| = 1, 2$ and $E(X) \cap B_2 = \varnothing$;
(2) $E(X) \cap B_1 = \varnothing$ and $|E(X) \cap B_2| = 1, 2$; or
(3) $|E(X) \cap B_1| = 1, 2$ and $|E(X) \cap B_2| = 1, 2$.

We now show how to compute margins of cutsets, separating these four types.

Type (0). Since every terminal lies on B_1 or B_2, $D(X) = \varnothing$ for any cutset $E(X)$

of type (0). Thus all cutsets of type (0) have nonnegative margins, and consequently it is not necessary to compute them.

Type (1). If two edges $e_g, e_h \in B_1$ are fixed, then all cutsets $E(X)$ with $E(X) \cap B_1 = \{e_g, e_h\}$ have the same $D(X) \cap P_1$; let $d_1(e_g, e_h) = \sum \{d_i | (s_i, t_i) \in D(X) \cap P_1\}$. Define the following terms:

$$c_1(e_g, e_h) = \text{MIN}\{c(X) | E(X) \text{ is a cutset of } G,\ E(X) \cap B_1 = \{e_g, e_h\},\ E(X) \cap B_2 = \varnothing\}$$

and

$$m_1(e_g, e_h) = c_1(e_g, e_h) - d_1(e_g, e_h).$$

For a fixed edge $e_g \in B_1$ and all edges $e_h \in B_1$ we can compute the values $d_1(e_g, e_h)$ in $O(b_1 + k)$ time. These values can be updated for the edge $e_{g+1} \in B_1$ clockwise next to e_g on B_1 in $O(b_1)$ time. Thus we can compute $d_1(e_g, e_h)$ for all edges $e_g, e_h \in B_1$ in $O(b_1^2)$ time [5].

On the other hand, we compute $c_1(e_g, e_h)$ as follows. Clearly, the cutset of G attaining the value $c_1(e_g, e_h)$ corresponds to the shortest path between vertices u_g and u_h in G^*. Therefore, applying a single-source shortest path algorithm to G^* once, choosing u_g as the starting point, we can compute in $O(T_+(n))$ time $c_1(e_g, e_h)$ for a fixed edge $e_g \in B_1$ and all edges $e_h \in B_1$.

Repeating the computation for each $e_g \in B_1$, we can find the minimum of $m_1(e_g, e_h)$ over all $e_g, e_h \in B_1$ in $O(b_1 T_+(n))$ time. Thus we can check the cut condition for cutsets of type (1) in $O(b_1 T_+(n))$ time.

Type (2). If two edges $e'_p, e'_q \in B_2$ are fixed, then all cutsets $E(X)$ with $E(X) \cap B_2 = \{e'_p, e'_q\}$ have the same $D(X) \cap P_2$; let $d_2(e'_p, e'_q) = \sum \{d_i | (s_i, t_i) \in D(X) \cap P_2\}$. Define

$$c_2(e'_p, e'_1) = \text{MIN}\{c(X) | E(X) \text{ is a cutset of } G,\ E(X) \cap B_1 = \varnothing,\ E(X) \cap B_2 = \{e'_p, e'_q\}\}$$

and

$$m_2(e'_p, e'_q) = c_2(e'_p, e'_q) - d_2(e'_p, e'_q).$$

As in the case of type (1) above, we can check the cut condition for cutsets of type (2) in $O(b_2 T_+(n))$ time.

Type (3). If four edges $e_g, e_h \in B_1$ and $e'_p, e'_q \in B_2$ are fixed, then $d(X)$ is constant for all cutsets $E(X)$ such that $E(X) \cap B_1 = \{e_g, e_h\}$ and $E(X) \cap B_2 = \{e'_p, e'_q\}$; the constant is denoted by $d_{12}(e_g, e_h; e'_p, e'_q)$. (See Figure 3.) Then we can easily verify

$$d_{12}(e_g, e_h; e'_p, e'_q) = d_1(e_g, e_h) + d_2(e'_p, e'_q).$$

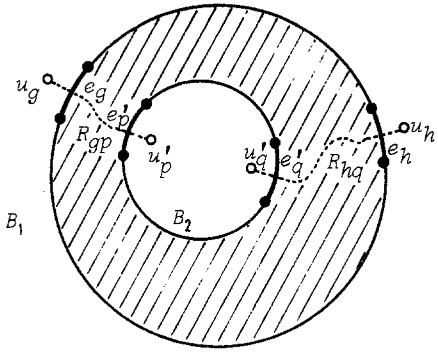

Fig. 3. Illustration for $c_{12}(e_g, e_h; e'_p, e'_q)$.

We now define

$$c_{12}(e_g, e_h; e'_p, e'_q) = \text{MIN}\{c(X) | E(X) \text{ is a cutset of } G, \\ E(X) \cap B_1 = \{e_g, e_h\}, E(X) \cap B_2 = \{e'_p, e'_q\}\}$$

and

$$m_{12}(e_g, e_h; e'_p, e'_q) = c_{12}(e_g, e_h; e'_p, e'_q) - d_{12}(e_g, e_h; e'_p, e'_q).$$

Clearly, c_{12} is equal to the length of a shortest pair of vertex-disjoint paths, each from u_g or u_h to u'_p or u'_q. Such a pair can be found by the sophisticated algorithm of Suurballe and Tarjan [15]. However, we can check more efficiently the cut condition for cutsets of type (3) simply by applying an ordinary shortest-path algorithm. The key point to notice is that we need not compute $m_{12}(e_g, e_h; e'_p, e'_q)$ itself. Instead we compute $m'_{12}(e_g, e_h; e'_p, e'_q)$ defined as follows:

$$m'_{12}(e_g, e_h; e'_p, e'_q) = \text{dist}(u_g, u'_p) + \text{dist}(u_h, u'_q) - d_{12}(e_g, e_h; e'_p, e'_q),$$

where $\text{dist}(u, u')$ denotes the distance between vertices u and u' in G^*, i.e., the length of the shortest path from u to u'. Although the two paths of length $\text{dist}(u_g, u'_p)$ and $\text{dist}(u_h, u'_q)$ in G^* may not be disjoint, the following lemma holds.

LEMMA 3. *A network N in C_{12} satisfies the cut condition if and only if $m_1(e_g, e_h) \geq 0$, $m_2(e'_p, e'_q) \geq 0$, and $m'_{12}(e_g, e_h; e'_p, e'_q) \geq 0$ for all $e_g, e_h \in B_1$ and $e'_p, e'_q \in B_2$.*

PROOF. Clearly, $\text{MIN}\{m'_{12}(e_g, e_h; e'_p, e'_q), m'_{12}(e_h, e_g; e'_p, e'_q)\} \leq m_{12}(e_g, e_h; e'_p, e'_q)$ for all $e_g, e_h \in B_1$ and $e'_p, e'_q \in B_2$. Thus the "if" part is trivial, and we shall prove the "only if" part. Assume that N satisfies the cut condition. Then $m_1(e_g, e_h) \geq 0$, $m_2(e'_p, e'_q) \geq 0$, and $m_{12}(e_g, e_h; e'_p, e'_q) \geq 0$ for all $e_g, e_h \in B_1$ and $e'_p, e'_q \in B_2$. Thus we shall verify $m'_{12}(e_g, e_h; e'_p, e'_q) \geq 0$ for all $e_g, e_h \in B_1$ and $e'_p, e'_q \in B_2$. Let R_{gp} be a shortest path from u_g to u'_p in G^*, and let R_{hq} be a shortest path from u_h

to u'_q in G^*. If paths R_{gp} and R_{hq} are vertex-disjoint, then there exists a cutset of G which consists of the edges corresponding to those of R_{gp} and R_{hq}. Therefore

$$m'_{12}(e_g, e_h; e'_p, e'_q) \geq m_{12}(e_g, e_h; e'_p, e'_q) \geq 0.$$

Thus we may assume that R_{gp} and R_{hq} are not vertex-disjoint. Then $R_{gp}+R_{hq}$ contains two edge-disjoint paths: a path Q_{gh} between u_g and u_h and a path Q_{pq} between u'_p and u'_q. Let $E(X)$ and $E(Y)$ be the cutsets of G corresponding to Q_{gh} and Q_{pq}, respectively. Then we have

$$m(X) = \text{leng}(Q_{gh}) - d_1(e_g, e_h) \geq m_1(e_g, e_h)$$

and

$$m(Y) = \text{leng}(Q_{pq}) - d_2(e'_p, e'_q) \geq m_2(e'_p, e'_q),$$

where leng(Q) is the length of path Q in G^*. Clearly,

$$\text{leng}(Q_{gh}) + \text{leng}(Q_{pq}) \leq \text{leng}(R_{gp}) + \text{leng}(R_{hq}).$$

Therefore

$$\begin{aligned} m'_{12}(e_g, e_h; e'_p, e'_q) &= \text{leng}(R_{gp}) + \text{leng}(R_{hq}) - d_{12}(e_g, e_h; e'_p, e'_q) \\ &\geq m_1(e_g, e_h) + m_2(e'_p, e'_q) \geq 0. \end{aligned}$$

□

Thus it suffices to check whether $m'_{12}(e_g, e_h; e'_p, e'_q) \geq 0$ for all $e_g, e_h \in B_1$ and $e'_p, e'_q \in B_2$. The checking can be done as follows. First compute for each $e_g \in B_1$ and $e'_q \in B_2$

$$m^*_{12}(e_g, e'_q) = \text{MIN}\{-d_1(e_g, e_h) + \text{dist}(u_h, u'_q) | e_h \in B_1\}$$

and

$$m^*_{21}(e_g, e'_q) = \text{MIN}\{\text{dist}(u_g, u'_p) - d_1(e'_p, e'_q) | e'_p \in B_2\}.$$

Note that for $e_g \in B_1$ and $e'_q \in B_2$

$$m^*_{12}(e_g, e'_q) + m^*_{21}(e_g, e'_q) = \text{MIN}\{m'_{12}(e_g, e_h; e'_p, e'_q) | e_h \in B_1, e'_p \in B_2\}.$$

Then compute

$$A = \text{MIN}\{m^*_{12}(e_g, e'_q) + m^*_{21}(e_g, e'_q) | e_g \in B_1, e'_q \in B_2\}.$$

Clearly, $A \geq 0$ if and only if $m'_{12}(e_g, e_h; e'_p, e'_q) \geq 0$ for all $e_g, e_h \in B_1$ and $e'_p, e'_q \in B_2$.

We now show that the computation above can be done in $O((b_1+b_2)T_+(n))$ time. We can compute $m_{12}^*(e_g, e'_q)$ for a fixed $e_g \in B_1$ and all $e'_q \in B_2$ in $O(T_+(n))$ time: construct a planar graph from G^* by adding a new vertex v and edges (v, u_h) of length $L - d_1(e_g, e_h)$ for all $u_h \in U_1 - u_g$, and find shortest paths from v to all $u'_q \in U_2$, where L is a sufficiently large positive number. Thus we can compute all m_{12}^* in $O(b_1 T_+(n))$ time. Similarly we can compute all m_{21}^* in $O(b_2 T_+(n))$ time. From these m_{12}^* and m_{21}^*, value A can be computed in time $O(b_1 b_2) \leq O((b_1+b_2)T_+(n))$.

From the discussions in (0), (1), (2), and (3) above, we can conclude:

THEOREM 2. *The feasibility of a network N in C_{12} can be tested in $O((b_1+b_2)T_+(n))$ time if N, B_1, and B_2 have n vertices, b_1 edges, and b_2 edges, respectively.*

4. Finding Flows for C_{12}.

In this section we give an algorithm MFLOW12 which finds multicommodity flows in a network N belonging to C_{12} and satisfying the cut condition. The algorithm spends $O(kn + nT_+(n))$ time. In this section we assume that all edges of capacity zero are deleted and consequently all edges have positive capacities.

4.1. *Basic Procedure PUSH.*

In this subsection we present a basic procedure PUSH which our algorithms repeatedly use to find flows in a network in C_{12} and C_{01}.

For an edge $e = (v, w) \in E$ and set P_l $(l = 0, 1, 2)$, procedure PUSH(N, P_l, e) repeats the following operation for each pair $(s_i, t_i) \in P_l$ having a terminal, s_i say, on v:

(a) Push an appropriate unit D of flow f_i through e.
(b) Decrease the capacity of e by D.
(c) Split the single demand d_i of f_i into two, demand D of a new pair (w, t_i) and the residual demand $d_i - D$ of pair (s_i, t_i). (Two flows realizing split demands will be superimposed to realize the original single flow f_i.)

We choose D so that the resulting network N' satisfies the cut condition. Furthermore, we choose e so that N' also belongs to C_{12} (resp. C_{01}) if N belongs to C_{12} (resp. C_{01}). Our algorithm repeatedly applies procedure PUSH until the network is eventually transformed into one belonging to C_1.

We now present the details of procedure PUSH. Of course, D cannot exceed d_i or $c(e)$, but we wish to choose D as large as the resulting network will allow before violating the cut condition. If pushing D units of f_i changes the margin $m(X)$ of a cutset $E(X)$, then $e \in E(X)$, $s_i, t_i \notin X$, and $m(X)$ is decreased by $2D$. Therefore D cannot exceed one-half of the minimum margin of these cutsets. Thus we choose D as follows:

$$D = \text{MIN}\{c(e), d_i, m(e, (s_i, t_i))/2\},$$

where

$$m(e, (s_i, t_i)) = \text{MIN}\{m(X) | E(X) \text{ is a cutset of } G, e \in E(X), s_i, t_i \notin X\}.$$

To keep track of the origin we assign to each pair (s_i, t_i) a commodity number commodity(i); clearly, commodity(i) $= i$ if $i \le k$; and commodity(i) $=$ commodity(j) if $i > k$ and (s_i, t_i) raises from (s_j, t_j). Then the superimposition in operation (c) above can be done mechanically using this numbering. We now describe procedure PUSH in pidgin ALGOL.

```
procedure PUSH(N, P_l, e);
  begin
    { l=0, 1 or 2. edge e=(v, w) is suitably chosen }
    for each terminal s_i (not necessarily source) on v belonging to P_l do
      begin
        D := MIN{c(e), d_i, m(e, (s_i, t_i))/2};
        { push D units of flow through e }
        if D > 0 then
          begin
            j := commodity(i);   {1 ≤ j ≤ k }
            f_j(e) := f_j(e) ± D;   { the sign ± depend on the orientation
                                      of e and whether s_i is a source or
                                      sink }
            c(e) := c(e) − D;   { residual capacity }
            if D = d_i then   { flow of (s_i, t_i) has been entirely pushed
                                through e }
              begin
                s_i := w;   { move terminal s_i from v to w }
                if s_i = t_i then P_l := P_l − (s_i, t_i)   { flow of (s_i, t_i) has
                                                            been realized }

              end
            else   { D < d_i, flow of (s_i, t_i) has been partly pushed
                     through e }
              begin
                d_i := d_i − D;   { residual demand }
                if t_i ≠ w then
                { add a surrogate (s_{k+1}, t_{k+1}) of pair (s_i, t_i) }
                  begin
                    s_{k+1} := w;
                    t_{k+1} := t_i;
                    P_l := P_l ∪ {(s_{k+1}, t_{k+1})};
                    commodity(k+1) := j;
                    d_{k+1} := D;   { split demand }
                    k := k+1
                  end
              end
          end
      end
  end;
```

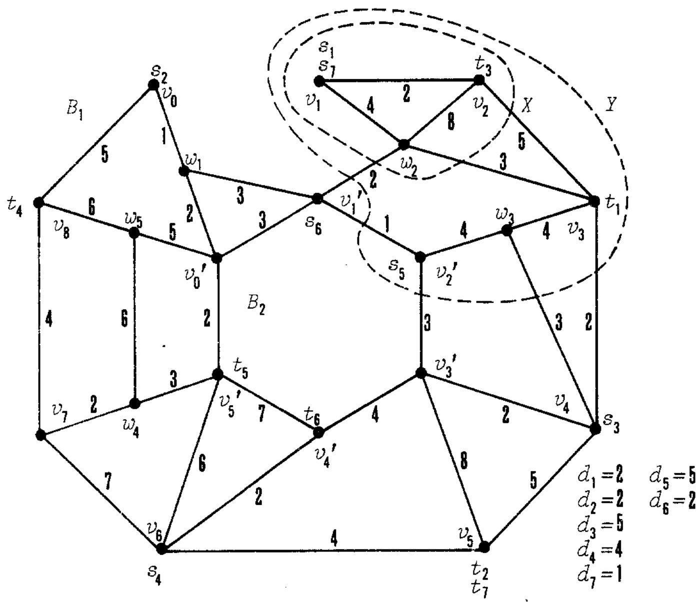

Fig. 4. The network after the execution of PUSH(N, P_1, e_0).

EXAMPLE. Suppose that PUSH(N, P_1, e) is executed for a network N in Figure 1(a), choosing an edge $e_0 = (v_0, v_1)$ on B_1 as e. In this case there are two terminals s_1 and s_2 on v_0. We can verify that $m(e_0, (s_1, t_1)) = m(X) = 8$ with $X = \{v_1, v_2, w_2\}$. Therefore $D = \text{MIN}\{3, 2, 8/2\} = 2$ for s_1. Since $D = d_1$, terminal s_1 is moved to v_1 and $c(e_0)$ is reduced to $3 - D = 1$ when D units of f_1 are pushed through e. Then we know that $m(e_0, (s_2, t_2)) = m(Y) = 2$ with $Y = \{v_1, v_2, v_3, w_2, w_3, v_2'\}$. Therefore $D = \text{MIN}\{1, 3, 2/2\} = 1$ for s_2. Since $D < d_2$, a surrogate (s_7, t_7) of pair (s_2, t_2) is introduced with s_7 on v_1 and t_7 on the same vertex as t_2 when D units of f_2 are pushed through e. Note that $c(e)$ becomes zero. Thus network N becomes as shown in Figure 4 when PUSH terminates.

4.2. Algorithm MFLOW12. Algorithm MFLOW12 first realizes the flows between source-sink pairs in P_1, and then realizes all the remaining flows in the resulting network belonging to C_1 simply by using the known algorithm MULTIFLOW [7].

MFLOW12 realizes flows of P_1 by repeating the following: choose an appropriate edge e on the outer boundary B_1 and push flows of P_1 through e by procedure PUSH(N, P_1, e). The algorithm initially chooses as e an arbitrary edge (v, w) on B_1, and pushes flows of P_1 having terminals on v through e in the clockwise direction by PUSH(N, P_1, e), where w is the vertex clockwise next to v on B_1. Then the algorithm chooses as a new e the edge on B_1 clockwise next to e, and repeats the same operation. When the capacity $c(e)$ of e is decreased to zero, MFLOW12 deletes edge e from the graph G and chooses as a new e an edge on the new boundary B_1 of G. When a connected graph G is disconnected into two components, the algorithm recurses to each subnetwork. Since only edges on B_1 are chosen as e, the network always belongs to C_{12} during the execution of MFLOW12.

The algorithm is formally described as follows:

```
procedure MFLOW12(N);
  begin
    for each edge e ∈ E and i (1 ≤ i ≤ k) do f_i(e) := 0;   { initialization }
    for each i (1 ≤ i ≤ k) do commodity(i) := i;
    e := an arbitrary edge on B_1;   { e = (v, w) }
    ROTATE(N, e)
  end;

procedure ROTATE(N, e);
  begin
    if N ∈ C_1 then MULTIFLOW(N)   { MULTIFLOW is given in [7] }
    else   { N ∈ C_12 − C_1 }
      begin
        PUSH(N, P_1, e);   { c(e) may be decreased }
        e' := the edge clockwise next to e on B_1;
        e'' := the edge clockwise next to e around v among the edges
        incident with v; (see Figure 5)
        { either e' or e'' is chosen as new e below }
        if c(e) > 0 then ROTATE(N, e')   { proceed to e' }
        else   { c(e) = 0 }
          begin
            G := G − e;   { delete edge e }
            if G is connected then ROTATE(N, e'')   { proceed to e'' }
            else   { e was a bridge, and new G is disconnected }
              begin
                let G_a and G_b be the two connected components in G;
                let N_a and N_b be the subnetworks of N with graphs
                G_a and G_b, respectively;
                { either N_a or N_b belongs to C_1 }
                assume that v is in G_a and w in G_b;
                ROTATE(N_a, e'');
                ROTATE(N_b, e')
              end
          end
      end
  end;
```

EXAMPLE. Figure 6 illustrates a partial traversal of variable e in the network N of Figure 1(a). The deleted edges are drawn in dashed lines. Number i in a circle and an arrow next to an edge indicate that MFLOW12(N) assigns the edge

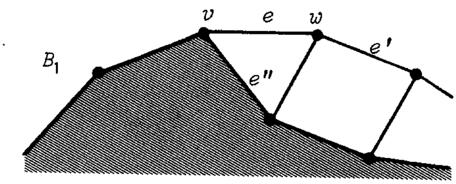

Fig. 5. Edges e, e', and e''.

to variable e for the orientation of the arrow in the ith execution of PUSH. The edge (v_1, w_2) has been assigned to e once for each of the two orientations.

For simplicity MFLOW12 above uses the known procedure MULTIFLOW [7] when a network is reduced to one in C_1. However, MFLOW12 itself can substitute for MULTIFLOW. Note that the computation of margins for networks in C_1 is easier than for C_{12}.

4.3. Polynomial Boundedness. In this subsection we show that MFLOW12 correctly finds multicommodity flows in polynomial time. Since the time required by PUSH is dominating in the running time, we bound the number of times PUSH is executed, that is, the number of edges variable e traverses. We claim the following Lemma 4.

LEMMA 4. PUSH *is executed* $O(n)$ *times during one execution of* MFLOW12.

In order to prove Lemma 4 we need some terms and lemmas. Let $s_i \in B_1$ be a terminal of P_1, and let e_j be an edge joining s_i and a clockwise next vertex on

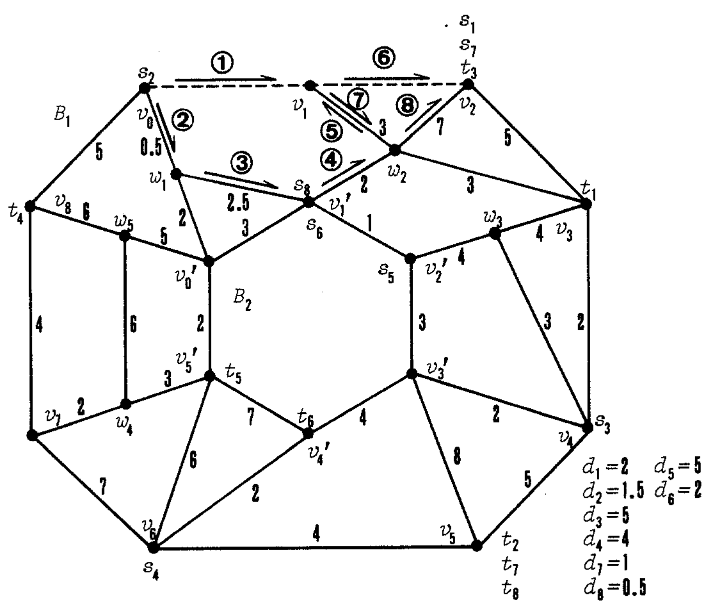

Fig. 6. The network after eight executions of PUSH.

B_1. (Note that there exist two or more such edges if s_i is a cutvertex of G.) Edge e_j *blocks* terminal s_i if there exists a cutset $E(X)$ such that $m(X)=0$, $e_j \in E(X)$, and $(s_i, t_i) \notin D(X)$. Clearly, only the flows of pairs in $D(X)$ can pass through edges in such a cutset $E(X)$, and hence flow f_i cannot pass through edge e_j. Terminal s_i is *blocked* if there is such an edge blocking s_i. When the execution of PUSH(N, P_1, (v, w)) does not reduce the capacity of edge (v, w) to zero, each of the terminals of P_1 remaining on v is blocked. The following two lemmas hold.

LEMMA 5. *If a network $N=(G, P, c)$ in C_{12} satisfies the cut condition, then there exists no cut $E(X)$ satisfying the following conditions*:

(5a) $|E(X) \cap B_1| = 1$ *or* 2;
(5b) *X contains no terminal of P_2*;
(5c) $m(X)=0$; *and*
(5d) *every terminal of P_1 in X is blocked.*

PROOF. Suppose that a cut $E(X)$ satisfies conditions (5a)–(5d), and that $|X \cap B_1|$ is minimum among such cuts. We may assume that $(v_{b_1-1}, v_0) \in E(X) \cap B_1$ and $v_0 \in X$. Furthermore, interchanging the roles of sources and sinks if necessary, we may assume that every source of P_1 precedes the corresponding sink on B_1 clockwise going from v_0 to v_{b_1-1}. Clearly, $D(X) \neq \varnothing$ because $c(X)>0$ and $m(X)=0$. Let (s_i, t_i) be a pair in $D(X)$ such that sink t_i appears first on B_1 clockwise going from v_0. Since source s_i lies in X, s_i is blocked by an edge e_j joining s_i and a clockwise next vertex on B_1. Thus there is a cutset $E(Y)$ such that $s_i, t_i \notin Y$, $m(Y)=0$, and $e_j \in E(Y)$. Condition (5b) and the selection of (s_i, t_i) imply that there is no source-sink pair having one terminal in $X-Y$ and the other in $Y-X$, and hence $d(X-Y;\ Y-X)=0$. Thus Lemma 2 implies $m(X \cap Y)=0$ and $c(X-Y;\ Y-X)=0$. If $(X \cap Y) \cap B_1 \neq \varnothing$, then the cut $E(X \cap Y)$ satisfies conditions (5a)–(5d) and $|(X \cap Y) \cap B_1| \leq |X \cap B_1| - 1$, contradicting the minimality of $|X \cap B_1|$. If $(X \cap Y) \cap B_1 = \varnothing$, then $e_j \in E(X-Y;\ Y-X)$, contrary to $c(X-Y;\ Y-X)=0$. □

LEMMA 6. *If network $N=(G, P, c)$ in C_{12} satisfies the cut condition and $P_1 \neq \varnothing$, then at least one terminal of P_1 is unblocked.*

PROOF. Suppose for a contradiction that every terminal of P_1 is blocked. Then a source s_i of P_1 is blocked by an edge $e_j \in B_1$, that is, there is a cutset $E(X)$ such that $s_i, t_i \notin X$, $m(X)=0$, and $e_j \in E(X)$. The sink t_i is also blocked by an edge $e_l \in B_1$, that is, there is a cutset $E(Y)$ such that $s_i, t_i \notin Y$, $m(Y)=0$, and $e_l \in E(Y)$. Since every terminal of P_1 is blocked, Lemma 5 implies that $X \cap B_2 \neq \varnothing$ and $Y \cap B_2 \neq \varnothing$. Since $m(X)=m(Y)=0$ and $(s_i, t_i) \notin D(X) \cup D(Y)$, all the edges in $E(X) \cup E(Y)$ are occupied by flows other than f_i. Furthermore, terminals s_i and t_i lie in distinct components in $G-E(X) \cup E(Y)$ (see Figure 7). Therefore flow f_i cannot exist. However, since N satisfies the cut condition, by Theorem 1 N has multicommodity flows, a contradiction. □

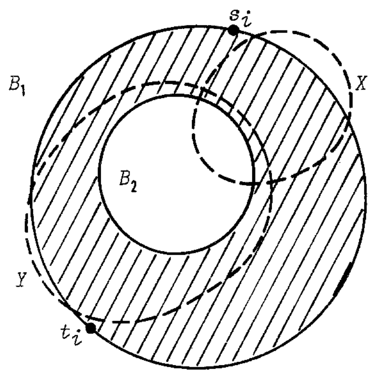

Fig. 7. Two saturated cutsets $E(X)$ and $E(Y)$.

Lemma 6 implies that MFLOW12 pushes a positive amount of a flow through at least one edge on B_1 while variable e traverses edges on B_1 once. Thus MFLOW12 correctly finds flows if it terminates finitely.

Intuitively we claim that variable e traverses each edge of G no more than twice. We may assume that edge e_0 is initially chosen as variable e. As shown later in Lemma 9, if an edge is deleted before e proceeds to the last edge e_{b_1-1} on B_1, then the claim can be rather easily verified. Otherwise, all the unblocked terminals of P_1 lie on vertex v_{b_1-1}, and there must exist a "saturated" cutset intersecting with B_1 and B_2. More precisely we have the following lemma.

LEMMA 7. *Let network $N=(G, P, c)$ in C_{12} satisfy the cut condition. If all the unblocked terminals of P_1 lie on v_{b_1-1}, then there is a cutset $E(X)$ satisfying the following conditions*:

(7a) $E(X) \cap B_1 \neq \varnothing$;
(7b) $X \cap B_2 \neq \varnothing$;
(7c) $m(X)=0$;
(7d) $v_{b_1-1}, v_0 \notin X$;
(7e) *X induces a connected subgraph; and*
(7f) *X contains no terminal of P_1.*

PROOF. Suppose that all the unblocked terminals of P_1 lie on v_{b_1-1}. We may assume that every sink of P_1 precedes the corresponding source on B_1 counterclockwise going from v_{b_1-1} to v_0. Let s_i be the source that first appears on B_1 counterclockwise going from v_{b_1-1} to v_0. Since s_i is not on v_{b_1-1}, s_i is blocked by an edge $e_j \in B_1$ and hence there is a cutset $E(X)$ such that s_i, $t_i \notin X$, $m(X)=0$, $e_j \in E(X)$, and v_{b_1-1}, $v_0 \notin X$. Since every terminal of P_1 in X is blocked, by Lemma 5 X contains a terminal of P_2 and hence $X \cap B_2 \neq \varnothing$. Clearly, $E(X)$ satisfies conditions (7a)-(7e). Thus it remains to show that $E(X)$ satisfies condition (7f).

Suppose there is a terminal of P_1 in X. Such a terminal must be a sink. Let (s_l, t_l) be the pair in $D(X) \cap P_1$ such that source s_l first appears on B_1 clockwise going from v_0. Since $t_l \in X$, t_l is not on v_{b_1-1}. Therefore sink t_l is blocked by an edge e_p, and hence there is a cutset $E(Y)$ such that s_l, $t_l \notin Y$, $e_p \in E(Y)$, and $m(Y) = 0$. We consider the following two cases:

Case 1: $Y \cap B_2 = \emptyset$. In this case clearly $D(X - Y;\ Y - X) \cap P_2 = \emptyset$. Furthermore, $D(X - Y;\ Y - X) \cap P_1 = \emptyset$, due to the selection of pairs (s_i, t_i) and (s_l, t_l). Therefore $d(X - Y;\ Y - X) = 0$ and, consequently, Lemma 2 implies $m(X \cap Y) = 0$ and $c(X - Y;\ Y - X) = 0$. If $X \cap Y \neq \emptyset$, then the cut $E(X \cap Y)$ satisfies conditions (5a)–(5d), a contradiction. If $X \cap Y = \emptyset$, then $e_p \in E(X - Y;\ Y - X)$, contradicting $c(X - Y;\ Y - X) = 0$.

Case 2: $Y \cap B_2 \neq \emptyset$. An edge $e_q \in B_1$ blocks s_l, and hence there is a cutset $E(X')$ such that s_l, $t_l \notin X'$, $e_q \in E(X')$, and $m(X') = 0$. Since every terminal in X' is blocked, $X' \cap B_2 \neq \emptyset$ by Lemma 5. Thus a contradiction can be easily derived with respect to X', Y, and (s_l, t_l) as in the proof of Lemma 6. □

If there exists a "saturated" cutset satisfying conditions (7a)–(7f), then variable e will traverse each edge no more than once, as we claim in the following lemma.

LEMMA 8. *Assume that*:

(1) *Network* $N = (G, P, c) \in C_{12}$ *satisfies the cut condition.*
(2) *A cut* $E(X)$ *satisfies conditions* (7a)–(7f).
(3) *For two distinct vertices* $v_h \in B_1$ *and* $v_g \in X \cap B_1$, *all the unblocked terminals of* P_1 *lie only on the vertices* $v_h, v_{h+1}, \ldots, v_g$.

If MFLOW12(N) *is executed with choosing edge* e_h *as initial e, then a single edge is not assigned to the variable e more than once for each of its two orientations and none of edges* $e_g, e_{g+1}, \ldots, e_{h-1}$ *is assigned to e for the orientation from* v_i *to* v_{i+1}, $g \le i \le h-1$, *before N is reduced to subnetworks all belonging to* C_1. (*See Figure* 8.)

PROOF. Assume that $N = (G, P, c)$ is a network for which the lemma is not true, and that G has a minimum number of edges among such networks; clearly, the number is positive. Since X contains no terminal of P_1, procedure PUSH(N, P_1, e)

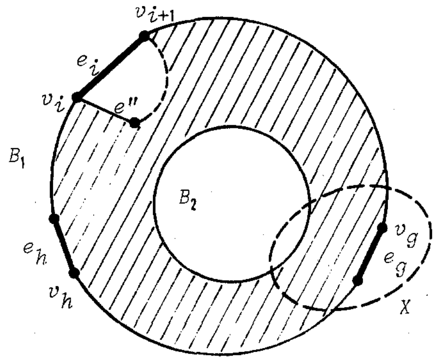

Fig. 8. Illustration for proof of Lemma 8.

does not push flows through any edge incident with a vertex in X. Therefore the cut $E(X)$ continues to satisfy conditions (7a)–(7f) before N is reduced to subnetworks all belonging to C_1.

An edge must be deleted before edge e_g is assigned to e. Otherwise, just before the algorithm assigns edge e_g to the variable e, every terminal of P_1 is blocked, contrary to Lemma 6.

Let e_i be the first edge deleted from the graph G. Assume that network N results in $N' = (G', P', c')$ when procedure PUSH for the edge e_i finishes, where $G' = G - e_i$. Note that MFLOW12(N) has assigned each of edges $e_h, e_{h+1}, \ldots, e_i$ to the variable e once so far. Furthermore, in N' all the unblocked terminals of P_1 lie on $v_i, v_{i+1}, \ldots, v_g$. We consider the following two cases:

Case 1: G' is Connected. Let e'' be the edge clockwise next to e_i around v_i among the edges incident with v_i. (See Figure 8.) Since G' has fewer edges than G, MFLOW12(N'), choosing e'' as initial e, assigns no single edge of G' to e more than once for each of its two orientations and assigns none of edges e_g, $e_{g+1}, \ldots, e_{h-1}, \ldots, e_{i-1}$ to e for the orientation before N' is reduced to subnetworks all belonging to C_1. The behavior of MFLOW12(N) after edge (v_i, v_{i+1}) is deleted is identical with that of MFLOW12(N'). Hence the lemma must hold for N, a contradiction.

Case 2: G' is Disconnected. Let G_a and G_b be the two connected components of G', and let N_a and N_b be the corresponding networks. We may assume that G_a contains B_2, and hence $N_a \in C_{12} - C_1$ and $N_b \in C_1$. Thus we consider the behavior of MFLOW12(N_a). The cut $E(X)$ is in N_a because $E(X)$ satisfies conditions (7b) and (7e). Since G_a has fewer edges than G, we can derive a contradiction as in Case 1. □

We are now ready to prove the following lemma.

LEMMA 9. *Algorithm* MFLOW12(N) *assigns no single edge to the variable e more than twice for each of its two orientations before network N is reduced to subnetworks all belonging to* C_1.

PROOF. Assume that $N = (G, P, c)$ is a network for which the lemma is not true, and that G has a minimum number of edges among such networks. We may assume without loss of generality that edge $e_0 = (v_0, v_1)$ is first assigned to e.

If at least one edge e_i other than the last edge e_{b_1-1} on B_1 is deleted on the first traversal of B_1, then we can derive a contradiction as in the proof of Lemma 8. (Note: If only the edge e_{b_1-1} is deleted on the first traversal of B_1, then there may appear an unblocked terminal on v_0, and consequently an argument such as the one in the proof of Lemma 8 does not work.)

Thus we may assume that no edge is deleted before the algorithm assigns edge e_{b_1-1} to the variable e. Assume that network N results in $N' = (G', P', c')$ when procedure PUSH for edge e_{b_1-2} finishes. Since all the unblocked terminals of P_1 lie on v_{b_1-1}, by Lemma 7 network N' has a cutset satisfying conditions (7a)–(7f).

Then Lemma 8 implies that, after assigning e_{b_1-1} to e, the algorithm assigns no single edge to e more than once for each of its two orientations. Thus MFLOW12(N), choosing e_0 as initial e, assigns no single edge to the variable e_0 more than twice for each of its two orientations, contrary to the assumption. □

Flows in subnetworks belonging to C_1 can be found by procedure MULTIFLOW [7]. Like MFLOW12, MULTIFLOW repeats pushing flows through edges on a face boundary in clockwise order by using procedure PUSH. It has been shown that MULTIFLOW assigns no single edge to the variable e more than once for each of its two orientations (Lemma 6 of [7]). Combining this result with Lemma 9, we can conclude that the PUSH is executed at most six times the number of edges of G. Thus we have Lemma 4 because G is planar and has $O(n)$ edges.

4.4. Complexity of MFLOW12. In this subsection we show that we can implement MFLOW12 to run in $O(kn+nT_+(n))$ time. The running time of MFLOW12 is dominated by the time for PUSH, and PUSH is executed $O(n)$ times. Therefore we bound the time for one execution of PUSH.

The execution time of PUSH(N, P_1, e) is dominated by the time for computing $m(e, (s_i, t_i))$ for all pairs having a terminal on v. We claim that the computations can be done in $O(k+T_+(n))$ time. We may assume that $e=e_0$. Assume that exactly l terminals of P_1 lie on v_0. Since one execution of PUSH introduces at most one new source-sink pair, the number of source-sink pairs is at most $k+O(n)$ throughout the execution of MFLOW12. Therefore $l=O(k+n)$. We may assume that $(s_1, t_1), (s_2, t_2), \ldots, (s_l, t_l) \in P_1$, all sources s_i, $1 \le i \le l$, lie on v_0, and $t_1, t_2, \ldots, t_l$ appear in that order on B_1 clockwise going from v_0. For each edge $e_g \in B_1 - e_0$, define

$$m(e_g) = \mathrm{MIN}\{m(X) | E(X) \text{ is a cutset of } G,\ e_0,\ e_g \in E(X)\}.$$

If sink t_i, $1 \le i \le l$, lies on $v_h \in B_1$, then $m(e_0, (s_i, t_i)) = \mathrm{MIN}\{m(e_g) | 1 \le g < h\}$. Moreover, if D units of flow f_i are pushed through e_0, then all $m(e_g)$ with $1 \le g < h$ decrease by the same units $2D$ and the remaining $m(e_g)$ do not change. Therefore, once all $m(e_g)$ have been computed before flows are pushed through e_0, the values $m(e_g)$ can be effectively updated if the flows $f_1, f_2, \ldots, f_l$ are pushed in that order. The update can be done in $O(l+b_1)$ time. Thus it suffices to show that we can compute $m(e_g)$ for all $e_g \in B_1 - e_0$ in $O(k+T_+(n))$ time.

As in the proof of Lemma 3, we have

$$m(e_g) = \mathrm{MIN}\{m_1(e_0, e_g),\ m'_{12}(e_0, e_g)\},$$

where $m'_{12}(e_0, e_g) = \mathrm{MIN}\{m'_{12}(e_0, e_g;\ e'_p, e'_q) | e'_p, e'_q \in B_2\}$. As in Section 3 we can compute $m_1(e_0, e_g)$ for all $e_g \in B_1 - e_0$ in $O(k+T_+(n))$ time. On the other hand, we compute $m'_{12}(e_0, e_g)$ for all $e_g \in B_1 - e_0$ in $O(k+T_+(n))$ time as follows:

Step 1. Compute $\mathrm{dist}(u_0, u'_p)$ for all $u'_p \in U_2$.

Step 2. Compute $m_{21}^*(e_0, e'_q) = \text{MIN}\{\text{dist}(u_0, u'_p) - d_2(e'_p, e'_q) | e'_p \in B_2\}$ for all $e'_q \in B_2$.

Step 3. Compute $c_1^*(e_0, e_g) = \text{MIN}\{m_{21}^*(e_0, e'_q) + \text{dist}(u'_q, u_g) | e'_q \in B_2\}$ for all $e_g \in B_1 - e_0$.

Step 4. Compute $m'_{12}(e_0, e_g) = c_1^*(e_0, e_g) - d_1(e_0, e_g)$ for all $e_g \in B_1 - e_0$.

Clearly, step 1 can be done in $O(T_+(n))$ time. Using an appropriate data structure, we can execute step 2 in $O(k+n)$ time, as shown in the Appendix. Step 3 can be done in $O(T_+(n))$ time as in the computation of m_{12}^* in Section 3. Clearly, step 4 can be done in $O(k+n)$ time.

Thus one execution of PUSH can be done in $O(k+T_+(n))$ time. The other steps in MFLOW12, such as the initialization of flows, can be done in $O(n(k+n))$ time. Thus we can conclude:

THEOREM 3. *If a network $N \in C_{12}$ has n vertices and k source-sink pairs, then Algorithm* MFLOW12 *finds flows in $O(kn + nT_+(n))$ time.*

5. Testing the Feasibility for Class C_{01}. In this section we give an algorithm for testing the feasibility of a network in class C_{01}. We first construct a planar digraph G^* from a given planar undirected graph G as follows:

(1) Replace each edge of G on B_1 by two multiple edges (replace each bridge on B_1 by three multiple edges).
(2) Construct the dual of the resulting graph.
(3) Remove the vertex corresponding to B_1 from the dual.
(4) Replace each edge e of the resulting graph with two directed edges, e^+ and e^-, one in each direction.

Figure 9 illustrates a pair G and G^*, where G is drawn in solid lines and G^* in dashed lines. Let $U_1 = \{u_0, u_1, \ldots, u_{b_1-1}\}$ be the set of vertices in G^* corresponding to edges in B_1.

Choose an arbitrary spanning tree T of G, and regard T as a rooted tree with root v_c. Remember that v_c is the vertex on B_1 on which all sinks of P_0 lie. In Figure 9 T is drawn by thick lines. In this section we orient the edges in T in the direction going from root v_c to leaves, and orient the other edges of G arbitrarily, as illustrated in Figure 9. Denote by e^+ and e^- the two directed edges of G^* corresponding to e of G assuming that the arrowhead of the oriented e first touches the arrowhead of e^+ and then e^- when e is rotated clockwise in the plane. One example is illustrated in Figure 9. The lengths of edges e^+ and e^- corresponding to an oriented edge $e = (u, v)$ are defined as follows:

(1) If $e \notin T$, then $\text{leng}(e^+) = \text{leng}(e^-) = c(e)$.
(2) If $e \in T$, then

$$\text{leng}(e^+) = c(e) + \sum \{d_i | (s_i, t_i) \in P_0, \text{ and } s_i \text{ is a descendant of } v \text{ in } T\}$$

and

$$\text{leng}(e^-) = c(e) - \sum \{d_i | (s_i, t_i) \in P_0, \text{ and } s_i \text{ is a descendant of } v \text{ in } T\}.$$

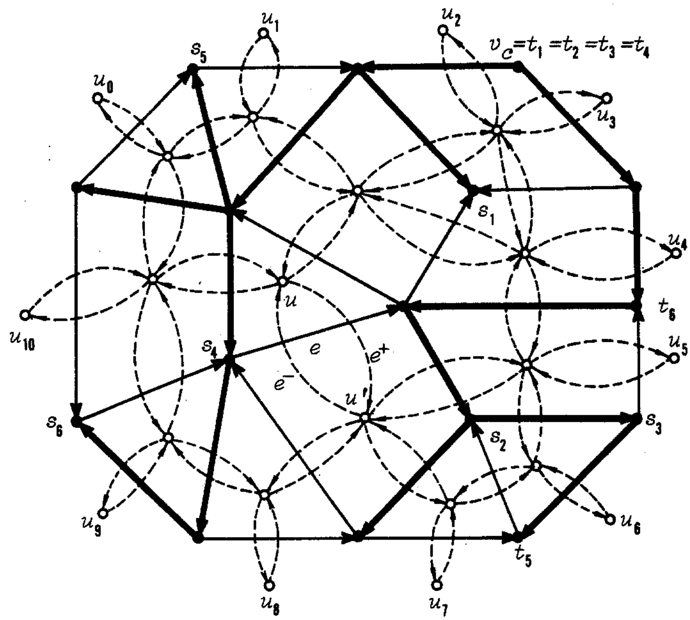

Fig. 9. Graph G and G^* of a network in C_{01}.

We now have the following lemma.

LEMMA 10. *Let $E(X)$ be a cutset of G with $E(X) \cap B_1 = \varnothing$, let Z_c be the clockwise cycle in digraph G^* corresponding to $E(X)$, and let Z_{cc} be the counterclockwise cycle. Then the lengths* $\text{leng}(Z_c)$ *and* $\text{leng}(Z_{cc})$ *of Z_c and Z_{cc} satisfy*

$$\text{leng}(Z_c) = m(X), \textit{ and}$$

$$\text{leng}(Z_{cc}) = m(X) + 2d(X).$$

PROOF. For $(s_i, t_i) \in P_0$ let Q_i be the unique path in tree T from sink t_i $(=v_c)$ to source s_i. We may assume that $v_c \notin X$ and hence $X \cap B_1 = \varnothing$. Let $Q_i^+ = \{e^+ | e \in Q_i\}$ and $Q_i^- = \{e^- | e \in Q_i\}$, and let $q_i^+ = |Z_c \cap Q_i^+|$ and $q_i^- = |Z_c \cap Q_i^-|$. Clearly, $q_i^- - q_i^+ = 0$ or 1, and $q_i^- - q_i^+ = 1$ if and only if $(s_i, t_i) \in D(X)$ (see Figure 10). Since in $\text{leng}(Z_c)$ each d_i is subtracted $(q_i^- - q_i^+)$ times from $c(X)$, we have

$$\text{leng}(Z_c) = c(X) - \sum \{(q_i^- - q_i^+)d_i | (s_i, t_i) \in P_0\}$$

$$= c(X) - d(X) = m(X).$$

Similarly, for the counterclockwise cycle Z_{cc}, we have

$$\text{leng}(Z_{cc}) = c(X) + d(X) = m(X) + 2d(X). \qquad \square$$

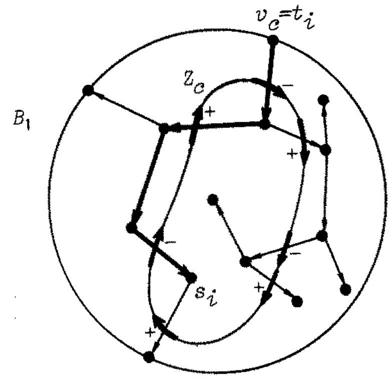

Fig. 10. Illustration for the proof of Lemma 10 (Q_i is drawn in thick lines, $q_i^- = 2$ and $q_i^+ = 1$).

As in Section 3, we may assume that $b_1 \le 2k$. Theorem 1 and Lemma 1 imply that we can test the feasibility by checking whether $m(X) \ge 0$ for every cutset $E(X)$ such that $|E(X) \cap B_1| = 0$, 1, or 2. We now show how to compute margins, separating two cases.

Case 1. $E(X) \cap B_1 = \emptyset$. We can verify the following lemma using Lemma 10.

LEMMA 11. *Every cutset $E(X)$ with $E(X) \cap B_1 = \emptyset$ has a nonnegative margin in network N if and only if G^* contains no negative directed cycle.*

PROOF. If network N has a cutset $E(X)$ such that $E(X) \cap B_1 = \emptyset$ and $m(X) < 0$, then by Lemma 10 G^* contains a negative clockwise cycle.

Suppose conversely that G^* contains a negative directed cycle. Then there must exist a simple negative cycle Z in G^*. If $Z = \{e^+, e^-\}$ for an edge $e \in E$, then $\text{leng}(Z) = 2c(e) \ge 0$, a contradiction. Thus the cycle Z of G^* corresponds to a cutset $E(X)$ of G. If Z is clockwise, then by Lemma 10 $m(X) = \text{leng}(Z) < 0$. If Z is counterclockwise, $m(X) = \text{leng}(Z) - 2d(X) < 0$. In either case $m(X) < 0$. □

We can detect a negative cycle in G^* by applying a shortest-path algorithm to G^* [6]. Thus the cut condition for these cutsets can be checked in $O(T_-(n))$ time. Note that some edges of G^* may have negative length.

Case 2. $|E(X) \cap B_1| = 1$ or 2. Define $d_1(e_g, e_h)$ and $m_1(e_g, e_h)$ for edges e_g and e_h on B_1 as in Section 3. Then we can verify as in Lemma 10,

$$m_1(e_g, e_h) = \text{MIN}\{\text{dist}(u_g, u_h), \text{dist}(u_h, u_g)\} - d_1(e_g, e_h),$$

where $\text{dist}(x, y)$ denotes the length of the shortest directed path going from x to y in G^*. Therefore we can compute $\text{MIN}\{m_1(e_g, e_h) | e_g, e_h \in B_1\}$ in $O(b_1 T_-(n))$ time simply by applying b_1 times a single source shortest-path algorithm to G^*, choosing vertices in U_1 as the starting point. However, a standard technique of shortest-path computation [20, p. 94] can improve the bound: once computation of shortest paths from a starting point is done for digraph G^* with negative edges, all the remaining computations can be done for a digraph with nonnegative edges. Thus we have:

THEOREM 4. *The feasibility for a network in C_{01} can be tested in $O(T_-(n)+b_1T_+(n))$ time.*

6. Finding Flows for C_{01}. In this section we first give an algorithm MFLOW01 which finds multicommodity flows in a network $N \in C_{01}$ satisfying the cut condition, and then show that MFLOW01 runs in $O(kn+nT_+(n))$ time.

6.1. Algorithm MFLOW01. A network N satisfying the cut condition is *minimal* if for every edge e with $c(e)>0$ there is a cutset $E(X)$ such that $e \in E(X)$ and $m(X)=0$. Clearly, a minimal network has no edge of surplus capacity: multicommodity flows $\{f_1, f_2, \ldots, f_k\}$ in N must satisfy

$$\sum_{i=1}^{k} |f_i(e)| = c(e)$$

for each $e \in E$. Furthermore, we have:

LEMMA 12. *If a minimal network has multicommodity flows, then each of the flows is acyclic, that is, the edges through which a single flow passes induce an acyclic digraph.*

First we reduce a given network N satisfying the cut condition into a minimal one by the following procedure:

procedure MIN(N);
  begin
    for each edge $e \in E$ **do**
      begin { reduce surplus capacity }
        $m(e) = \text{MIN}\{m(X) \mid X \subset V, e \in E(X)\}$;
        $c(e) := c(e) - \text{MIN}\{c(e), m(e)\}$
      end
  end;

Next we decide the direction of flows of P_1 in edges of the resulting minimal network N. By Lemma 12 each of the multicommodity flows is acyclic in N. Let $e \in E$ be any edge with $c(e)>0$ in N. Then there exists $X \subset V$ such that $m(X)=0$ and $e \in E(X)$. We may assume that $v_c \notin X$. The cut $E(X)$ is "saturated" by the demands of pairs in $D(X)$, and all terminals of P_0 lying in X are sources. Therefore if v is the end of e in X and w the other, then we know that no flow for P_0 passes through e in a direction from w to v. Thus for each edge $e \in E$ we can determine its direction in which flows for P_0 can pass through e. A digraph G_{acy} indicating these directions is constructed from a minimal network $N = (G, P, c)$ by the following procedure:

```
procedure ACYCLIC(N);
  begin
    let G_acy be the digraph obtained from G by replacing each edge
    by two multiple directed edges, one in each direction;
    for each edge e of G do
      begin
        if c(e) = 0 then remove from G_acy the two directed edges
                         corresponding to e
        else  { c(e) > 0 and m(e) = 0 }
          begin
            find a cutset E(X) such that m(X) = 0, e ∈ E(X), and
            v_c ∉ X;
            for each edge e' ∈ E(X) do
              begin
                let v and w be the ends of e' such that v ∈ X and
                w ∉ X;
                remove directed edge (w, v) from G_acy
              end
          end
      end
  end;
```

LEMMA 13. *If N is a minimal network, then procedure* ACYCLIC(N) *produces an acyclic digraph G_{acy}.*

PROOF. Let Z be an arbitrary undirected cycle in G. When the outer **for** statement of procedure ACYCLIC is executed for an edge e of Z, at least one forward edge and one backward edge in the clockwise direction of Z are deleted. Thus in G_{acy} there is no directed cycle corresponding to Z. □

Since G_{acy} is acyclic, the vertices can be numbered in topological order. The following algorithm MFLOW01 first finds all the flows for P_0 by repeatedly applying procedure PUSH for each of the vertices in that order, and then finds the flows for P_1 by applying MULTIFLOW once for the resulting network belonging to class C_1.

```
procedure MFLOW01(N);
  begin
    for each edge e and i (1 ≤ i ≤ k) do f_i(e) := 0;
    for each i(1 ≤ i ≤ k) do commodity(i) := i;
    MIN(N);  { reduce N to a minimal network }
    ACYCLIC(N);  { construct G_acy }
```

for each vertex w_i of G_{acy} in the topological order **do**
  for each edge e of G_{acy} emanating from w_i **do** PUSH(N, P_0, e);
{ flows for P_0 have been realized, and N belongs to C_1 }
MULTIFLOW(N)
end;

We now verify the correctness of algorithm MFLOW01. Throughout the execution of MFLOW01(N), network N continues to satisfy the cut condition and belongs to C_{01}. Thus we show that the realization of flows for P_0 is completed when the two nested **for** statements in MFLOW01 terminate. This is a direct consequence of the following lemma.

LEMMA 14. *No terminal of P_0 remains on $w_i \in V$ just after* PUSH(N, P_0, e) *is executed for all edges emanating from w_i. Furthermore, no terminal of P_0 is moved to w_i thereafter.*

PROOF. Suppose that a terminal s_j remains on w_i just after the executions of PUSH for all edges emanating from w_i. Since the network satisfies the cut condition and belongs to C_{01}, Theorem 1 implies that there exist multicommodity flows, including flow f_j of pair (s_j, t_j). The flow f_j must pass through edges emanating from w_i. Let e be one of these edges. Then, just after PUSH(N, P_0, e) is executed, either $c(e)=0$ or there exists a "blocking" cutset $E(X)$ such that $e \in E(X)$, $m(X)=0$, and $s_j, t_j \notin X$. Since the margin of any cut does not increase during the execution of MFLOW01, $m(X)$ remains zero thereafter. Thus f_j cannot pass through e, a contradiction.

Since MFLOW01 repeats pushing flows from vertices in the topological order, no terminal of P_0 is moved to w_i thereafter. □

6.2. Complexity of MFLOW01. In this subsection we show that MFLOW01 runs in $O(kn + nT_+(n))$ time. It has been known that MULTIFLOW runs in that time [7]. Therefore we shall show that the remaining part of MFLOW01 terminates in that time.

Since PUSH is executed at most once for each edge and G is planar, PUSH is executed $O(n)$ times in total. Since one execution of PUSH introduces at most one new pair, there exist $O(n)$ pairs of P_0 throughout the execution of MFLOW01.

We show below that both $m(e)$ and $m(e, (s_i, t_i))$ can be computed in $O(k + T_-(n))$ time. Let u be the tail of e^+ of G^*, and let u' be the head (see Figure 9). In what follows, shortest paths are computed in graph $G^* - \{e^+, e^-\}$.

(a) *Computation of m(e).* Define

$$m_0(e) = \mathrm{MIN}\{m(X) | e \in E(X),\ E(X) \cap B_1 = \emptyset,\ E(X) \text{ is a cutset}\}$$

and

$$m_1(e) = \mathrm{MIN}\{m(X) | e \in E(X),\ E(X) \cap B_1 \neq \emptyset,\ E(X) \text{ is a cutset}\}.$$

Then $m(e) = \text{MIN}\{m_0(e), m_1(e)\}$. These $m_0(e)$ and $m_1(e)$ are computed as follows.

(a1) $m_0(e)$. Lemma 10 implies that $m_0(e)$ is equal to the length of the minimum directed cycle in G^* passing through e^+ or e^-. Let

$$m_0(e^-) = \text{dist}(u, u') + \text{leng}(e^-)$$

and

$$m_0(e^+) = \text{dist}(u', u) + \text{leng}(e^+),$$

where dist(x, y) denotes the length of a shortest directed path from x to y in $G^* - \{e^+, e^-\}$. Then $m_0(e) = \text{MIN}\{m_0(e^-), m_0(e^+)\}$. Thus $m_0(e)$ can be computed by solving twice the single source shortest-path problem in $G^* - \{e^+, e^-\}$.

(a2) $m_1(e)$. The cutset $E(X)$ of margin $m_1(e)$ corresponds to a path in G^* connecting two vertices of U_1 through e^+ or e^-. Define

$$m_1(e^+) = \text{MIN}\{\text{dist}(u_g, u) + \text{leng}(e^+) + \text{dist}(u', u_h) - d_1(e_g, e_h) | e_g, e_h \in B_1\}$$

and

$$m_1(e^-) = \text{MIN}\{\text{dist}(u_g, u') + \text{leng}(e^-) + \text{dist}(u, u_h) - d_1(e_g, e_h) | e_g, e_h \in B_1\}.$$

Although the two paths of lengths dist(u_g, u) and dist(u', u_h) (or dist(u_g, u') and dist(u, u_h)) may not be vertex-disjoint, we can use $m_1'(e) = \text{MIN}\{m_1(e^+), m_1(e^-)\}$ instead of $m_1(e)$ to compute $m(e)$, as the next lemma claims.

LEMMA 15. *If a network N satisfies the cut condition, then*

$$m(e) = \text{MIN}\{m_0(e), m_1'(e)\}.$$

PROOF. Clearly, $m_1'(e) \le m_1(e)$. Therefore we shall show that $m_1'(e) < m_1(e)$ implies $m_0(e) \le m_1'(e)$. Suppose that $m_1'(e) < m_1(e)$. We may assume that $m_1(e^+) \le m_1(e^-)$ and hence $m_1'(e) = m_1(e^+)$. Let path Q from u_g to u_h through edge e^+ in G^* have length $m_1(e^+) + d_1(e_g, e_h)$. Since $m_1'(e) < m_1(e)$, Q is not a simple path, but Q is an edge-disjoint union of a simple path Q_s from u_g to u_h not passing through edge e^+ and some simple cycles. One of these cycles, say Z, passes through edge e^+ and, clearly, leng$(Z) \ge m_0(e)$. Since the cut condition is satisfied, each of the other cycles has a nonnegative length and the length of path Q_s satisfies leng$(Q_s) - d_1(e_g, e_h) \ge 0$. Therefore

$$m_1(e^+) \ge \text{leng}(Z) + \text{leng}(Q_s) - d_1(e_g, e_h) \ge m_0(e). \qquad \square$$

We can compute $m_1(e^+)$ as follows. First solve twice the single source shortest-path problem, once from u' in $G^* - \{e^+, e^-\}$ and once from u in the graph obtained from $G^* - \{e^+, e^-\}$ by reversing the direction of all the edges. Then from the found distances we can compute $m_1(e^+)$ in $O(b_1^2)$ time by a straightforward method. Using the variable priority queue in the Appendix, we can compute in $O(k + b_1)$ time. Thus $m_1(e^+)$ can be computed in $O(k + T_-(n))$ time. Similarly $m_1(e^-)$ can be computed in that time.

Since $m(e)$ can be computed immediately from $m_0(e)$ and $m_1'(e)$, the computation of $m(e)$ spends $O(k + T_-(n))$ time.

(b) *Computation of* $m(e, (s_i, t_i))$. Procedure PUSH always pushes D units of a flow f_i through an edge e where $D = \text{MIN}\{d_i, c(e), m(e, (s_i, t_i))/2\}$. We compute D without explicitly computing $m(e, (s_i, t_i))$. Instead we compute $m'(e, (s_i, t_i))$ defined as follows: if edge e is oriented to emanate from s_i, then

$$m'(e, (s_i, t_i)) = \text{MIN}\{m_0(e^-), m_1(e^-)\}; \tag{1}$$

otherwise,

$$m'(e, (s_i, t_i)) = \text{MIN}\{m_0(e^+), m_1(e^+)\}.$$

The following lemma justifies it.

Lemma 16. $D = \text{MIN}\{d_i, c(e), m'(e, (s_i, t_i))/2\}$.

Proof. We verify the equation only for the case e is oriented to emanate from s_i because the other case can be treated similarly. By definition $m(e, (s_i, t_i))$ is the minimum $m(X)$ over all the cutsets $E(X)$ such that $X \subset V$, s_i, $t_i \notin X$, and $e \in E(X)$. Let $m(X')$ be minimum among all these cutsets with $E(X') \cap B_1 = \varnothing$, while let $m(X'')$ be minimum among these with $E(X'') \cap B_1 \neq \varnothing$. Thus

$$m(e, (s_i, t_i)) = \text{MIN}\{m(X'), m(X'')\}. \tag{2}$$

Since e is oriented to emanate from s_i, cutset $E(X')$ of G corresponds to a minimum clockwise cycle through e^- in G^*. Since $m_0(e^-) = \text{dist}(u, u') + \text{leng}(e^-)$, $m_0(e^-) \leq m(X')$. Let Q be the shortest path (of length $\text{dist}(u, u')$) from u to u' in $G^* - \{e^+, e^-\}$, then the cycle $Q' = Q \cup \{e^-\}$ in G^* corresponds to a cutset $E(Y)$ of G. We may assume $t_i \notin Y$; otherwise replace Y with the complement $V - Y$. By Lemma 10, if cycle Q' is clockwise, then

$$m(Y) = m_0(e^-) = m(X').$$

On the other hand, if Q' is counterclockwise, then $s_i \in Y$, $d_i \leq d(Y)$ and hence

$$2d_i \leq m(Y) + 2d(Y) = m_0(e^-) \leq m(X').$$

Thus either

$$m_0(e^-) = m(X') \tag{3}$$

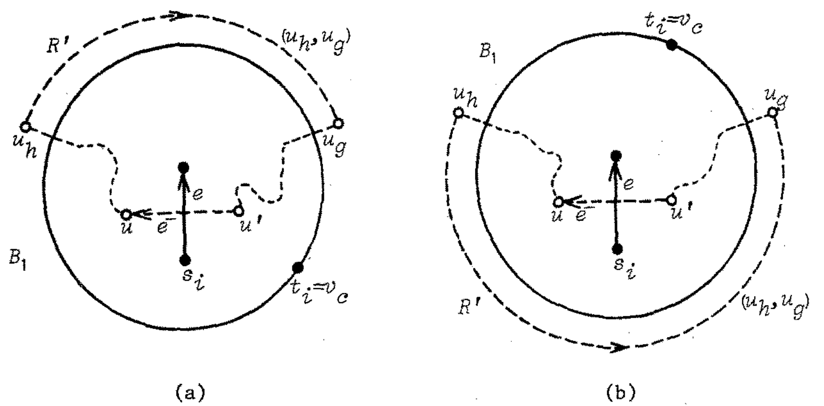

Fig. 11. (a) Clockwise cycle R' and (b) counterclockwise cycle R'.

or

(4) $$2d_i \leq m_0(e^-) < m(X').$$

Let $E(X'') \cap B_1 = \{e_g, e_h\}$. $E(X'')$ corresponds to a path R in G^* through e^- either from u_g to u_h or from u_h to u_g. We may assume that path R is from u_g to u_h. Add edge (u_h, u_g) to R so that v_c lies outside the resulting plane directed cycle $R' = R \cup (u_h, u_g)$. Then the cycle R' must be clockwise; otherwise, X'' would contain s_i. (See Figure 11.) Remember $m_1(e^-)$ is defined as

$$m_1(e^-) = \text{MIN}\{\text{dist}(u_p, u') + \text{leng}(e^-) + \text{dist}(u, u_q) - d_1(e_p, e_q) | e_p, e_q \in B_1\}.$$

Clearly, $m_1(e^-) \leq m(X'')$. Let R_{pq} be the path in G^* from u_p to u_q through e^- of length $m_1(e^-) + d_1(e_p, e_q)$. First consider the case R_{pq} is a simple path. Add edge (u_q, u_p) to R_{pq} so that v_c lies outside the resulting plane cycle $R'_{pq} = R_{pq} \cup (u_q, u_p)$. If cycle R'_{pq} is clockwise, then $m_1(e^-) = m(X'')$. Otherwise, $2d_i \leq m_1(e^-)$. Next consider the case path R_{pq} is not simple. Then R_{pq} contains a cycle Z through e^- in G^*. If Z is clockwise, then $m_0(e^-) \leq \text{leng}(Z) \leq m_1(e^-)$. Otherwise, $2d_i \leq \text{leng}(Z) \leq m_1(e^-)$. Thus in either case we have either

(5) $$m_1(e^-) = m(X'')$$

or

(6) $$\text{MIN}\{2d_i, m_0(e^-)\} \leq m_1(e^-) \leq m(X'').$$

Equations (1)–(6) imply that either

$$m'(e, (s_i, t_i)) = m(e, (s_i, t_i))$$

or

$$2d_i \leq m'(e, (s_i, t_i)) < m(e, (s_i, t_i)).$$

This immediately implies the claimed equation

$$D = \mathrm{MIN}\{d_i, c(e), m'(e, (s_i, t_i))/2\}. \qquad \square$$

We can compute $m'(e, (s_i, t_i))$ in $O(k + T_-(n))$ time. The computation occurs $O(n)$ times, and so spends $O(kn + nT_-(n))$ time in total. The other tasks can be done in $O(kn + nT_+(n))$ time as in Section 4.4. Thus algorithm MFLOW01 runs in $O(kn + nT_-(n))$ time. Edge weights of G^* may be negative and, to make matters worse, edge weights may decrease during the execution of MFLOW01. However, using the same standard technique as in Section 5, we may compute the shortest paths in graphs without negative edges except the first computation. Thus we can conclude:

THEOREM 5. *Algorithm* MFLOW01 *finds multicommodity flows for networks belonging to* C_{01} *in* $O(kn + nT_+(n))$ *time.*

7. Conclusion. We have presented simple efficient algorithms for the multicommodity flow problems for two classes C_{12} and C_{01} of planar undirected networks. C_{12} consists of networks in which every source-sink pair lies on one of the two specified face boundaries B_1 and B_2. C_{01} consists of networks in which some pairs lie on the specified boundary B_1 and all other pairs share a common sink on B_1. The feasibility can be checked by solving the single-source shortest-path problem $O(b_1 + b_2)$ times for C_{12} and b_1 times for C_{01}, where b_1 and b_2 are the number of edges on B_1 and B_2, respectively. On the other hand, multicommodity flows can be found by solving the shortest-path problem $O(n)$ times for C_{12} and C_{01}. More precisely the feasibility can be checked in $O((b_1 + b_2)T_+(n))$ time for C_{12} and in $O(T_-(n) + b_1 T_+(n))$ time for C_{01}, while k-commodity flows can be found in $O(kn + nT_+(n))$ time for C_{12} and C_{01}.

If the usual Dijkstra's algorithm [1], [20] is used, then $T_+(n) = O(n \log n)$. Frederickson [3] shows that if a planar separator algorithm is used then $T_+(n) = O(n)$ assuming the preprocessing is done in $O(n \log n)$ time. It is well known that $T_-(n) = O(n^2)$ [1], [20] if an ordinary shortest-path algorithm is used, while $T_-(n) = O(n^{3/2})$ if a planar separator algorithm is used [6].

If the capacities and demands are all integers, then our algorithms find half-integral flows for C_{12} and C_{01}. Note that $m(X)$ is an integer for any $X \subset V$ and D is always a half-integer throughout the execution of algorithms. A network is *even* if the capacities and demands are all integers and for each vertex v the capacities of edges incident with v and the demands of terminals lying on v total to an even integer. MFLOW12 finds integral flows for an even network in C_{12}. This is not the case for MFLOW01 because procedure MIN may reduce an even network to a minimal one which is not even. However, we can modify MFLOW01 so that it finds integral flows for an even network in C_{01} (the details are left to the reader). Thus our algorithms can be used to find edge-disjoint paths in plane grids [16], and are expected to be useful for VLSI routing problems.

Appendix. Detail of Step 2. We now show that we can execute step 2 (in Section 4.4) in $O(k+n)$ time. Clearly, we can compute $m_{21}^*(e_0, e'_q)$ for a single edge $e'_q \in B_2$ in $O(k+n)$ time. The key point to notice is that $m_{21}^*(e_0, e'_q)$ can be updated from $m_{21}^*(e_0, e'_{q-1})$. In order to perform the update efficiently, we need a data structure called a variable priority queue Q [18]. Q is a sequence of elements ordered from left to right, and each element e in Q is associated with a real number key(e). The following instructions are permitted:

1. INJECT(Q, e, key(e)): insert a new element e with key(e) into Q as the rightmost element.
2. POP(Q): delete the leftmost element in Q.
3. DECREASE(Q, e, D): given an element e in Q together with a nonnegative number D, decrease by D all the keys of element e and those on e's right.
4. UPDATE(Q, D): add some real number D to all the keys of elements in Q.
5. MIN(Q): return the minimum key in Q.

Let $R_p = \{v'_{q+1}, v'_{q+2}, \ldots, v'_p\}$, then

$$-d_2(e'_p, e'_q) = -d_2(e'_p, e'_{q-1}) + \sum \{d_i \mid (s_i, t_i) \in P_2 \cap D(\{v'_q\})\}$$
$$-2 \sum \{d_i \mid (s_i, t_i) \in P_2 \cap D(\{v'_q\}, R_p)\}.$$

Therefore, using the queue Q, we can compute $m_{21}^*(e_0, e'_q)$ for all $e'_q \in B_2$ as follows:

```
procedure M21*;
  begin
    prepare an empty queue Q;
    for each edge e'_p, p = 1, 2, ..., b_2 - 1 do
      INJECT(Q, e'_p, dist(u_0, u'_p) - d_2(e'_p, e'_0));
    { the key of edge e'_p in Q is dist(u_0, u'_p) - d_2(e'_p, e'_0) }
    m*_21(e_0, e'_0) := MIN(Q);
    for each q, q = 1, 2, ..., b_2 - 1, do
      begin
        { Q contains edges e'_p, p = q, q+1, .., q-2, having keys
          dist(u_0, u'_p) - d_2(e'_p, e'_{q-1}) }
        POP(Q);  { delete e'_q from Q }
        INJECT(Q, e'_{q-1}, dist(u_0, u'_{q-1}));
        { Q contains e'_{q+1}, e'_{q+2}, ..., e'_{q-1} }
        UPDATE(Q, Σ {d_i | (s_i, t_i) ∈ P_2 ∩ D({v'_q})});
        for each pair (s_i, t_i) ∈ P_2 ∩ D({v'_q}) do
          begin
            assume that s_i lies on v'_q; let e'_p be the edge clockwise
                incident with t_i on B_2;
            DECREASE(Q, e'_p, 2d_i)
          end;
        { each edge e'_p in Q has key dist(u_0, u'_p) - d_2(e'_p, e'_q) }
        m*_21(e_0, e'_q) := MIN(Q)
      end
  end;
```

Thus during the execution of procedure M21* instructions 1–5 occur $O(k+n)$ times in total. If the queue Q is realized by a 2-3 tree [1], then each instruction is executed in $O(\log n)$ time, and hence the execution of M21* spends $O((k+n)\log n)$ time. Using a disjoint set union algorithm [4], we can realize Q in a more sophisticated way so that M21* runs in time linear in the number of instructions [18]. Therefore step 2 can be done in $O(k+n)$ time.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[2] H. Diaz and G. de Ghellinck, Multicommodity maximum flow in planar networks (the *D*-algorithm approach), CORE Discussion Paper No. 7212, Center for Operations Research and Econometrics, Louvain-la-Neuve, 1972.

[3] G. N. Frederickson, Shortest-path problems in planar graphs, *Proc. 24th IEEE Symp. on Foundations of Computer Science*, Tucson, 1983, pp. 242–247.

[4] H. Gabow and R. E. Tarjan, A linear-time algorithm for a special case of disjoint set union, *J. Comput. System Sci.*, **30** (1985), pp. 209–221.

[5] R. Hassin, On multicommodity flows in planar graphs, *Networks*, **14** (1984), pp. 225–235.

[6] R. J. Lipton, D. J. Rose, and R. E. Tarjan, Generalized nested dissection, *SIAM J. Numer. Anal.*, **16**, 2 (1979), pp. 346–358.

[7] K. Matsumoto, T. Nishizeki, and N. Saito, An efficient algorithm for finding multicommodity flows in planar networks, *SIAM J. Comput.*, **14**, 2 (1985), pp. 289–302.

[8] K. Matsumoto, T. Nishizeki, and N. Saito, Planar multicommodity flows, maximum matchings, and negative cycles, *SIAM J. Comput.*, **15**, 2 (1985), pp. 495–510.

[9] T. Nishizeki, N. Saito, and K. Suzuki, A linear-time routing algorithm for convex grids, *IEEE Trans. Computer-Aided Design*, **4**, 1 (1985), pp. 68–76.

[10] H. Okamura, Multicommodity flows in graphs, *Discrete Appl. Math.*, **6** (1983), pp. 55–62.

[11] H. Okamura and P. D. Seymour, Multicommodity flows in planar graphs, *J. Combin. Theory Ser. B*, **31** (1981), pp. 75–81.

[12] M. Sakarovitch, The Multicommodity Flow Problem, Doctoral Thesis, Operations Research Center, University of California, Berkeley, 1966.

[13] M. Sakarovitch, Two commodity network flows and linear programming, *Math. Programming*, **4** (1973), pp. 1–20.

[14] D. D. Sleator and R. E. Tarjan, A data structure for dynamic trees, *J. Comput. System Sci.*, **26** (1983), pp. 362–390.

[15] J. W. Suurballe and R. E. Tarjan, A quick method for finding shortest pairs of disjoint paths, *Networks*, **14** (1984), pp. 325–336.

[16] H. Suzuki, A. Ishiguro, and T. Nishizeki, Edge-disjoint paths in a region bounded by nested rectangles, Technical Report AL85-28, Institute of Electrical and Communication Engineers of Japan, 1985, pp. 13–22 (in Japanese).

[17] H. Suzuki, T. Nishizeki, and N. Saito, Multicommodity flows in planar undirected graphs and shortest paths, *Proc. 17th Annual ACM Symp. on Theory of Computing*, 1985, pp. 195–204.

[18] H. Suzuki, T. Nishizeki, and N. Saito, A variable priority queue and its applications, Technical Report CAS86-131, Institute of Electrical and Communication Engineers of Japan, 1986, pp. 23–33.

[19] É. Tardos, A strongly polynomial algorithm to solve combinatorial linear programs, Report 84360-OR, Institut Ökonometrie und Operations Research, Rheinishe Friedrich-Wilhelms Universität, Bonn.

[20] R. E. Tarjan, *Data Structures and Network Algorithms*, SIAM, Philadelphia, PA, 1983.