SIAM J. COMPUT.
Vol. 14, No. 3, August 1985

008

AN $O(n \log^2 n)$ ALGORITHM FOR MAXIMUM FLOW IN UNDIRECTED PLANAR NETWORKS*

REFAEL HASSIN† AND DONALD B. JOHNSON‡

Abstract. A new algorithm is given to find a maximum flow in an undirected planar flow network in $O(n \log^2 n)$ time, which is faster than the best method previously known by a factor of $\sqrt{n}/\log n$. The algorithm constructs a transformation of the dual of the given flow network in which differences between shortest distances are equal, under suitable edge correspondences, to edge flows in the given network. The transformation depends on the value of a maximum flow. The algorithm then solves the shortest distances problem efficiently by exploiting certain structural properties of the transformed dual, as well as using a set of cuts constructible in $O(n \log^2 n)$ time by a known method which is also used to find the requisite flow value. The main result can be further improved by a factor of $\log n/\log^* n$ if a recently developed shortest path algorithm for planar networks is used in place of Dijkstra's algorithm in each step where shortest paths are computed.

Key words. flow, maximum flow, planar, network, duality, graph algorithm

1. Introduction. The best algorithms known for solving the maximum flow problem in capacitated networks with n vertices and m edges run in $O(\min\{n^{5/3}m^{2/3}, nm \log n\})$ computational steps [4], [12], [13]. On planar networks this bound reduces to $O(n^2 \log n)$, a bound known earlier for undirected networks [6], since $m = O(n)$.

The best bound for general planar networks is $O(n^{3/2} \log n)$. This bound is achieved by a divide-and-conquer algorithm, due to Johnson and Venkatesan [8], that operates on recursively subdivided regions of a planar representation of the given network. More efficient algorithms exist for (s, t)-planar networks, those that can be drawn in the plane with the source and the sink on a common face. These algorithms run in $O(n \log n)$ steps. One, due to Itai and Shiloach [6], derives from the "uppermost path" method of Ford and Fulkerson [3]. The other, due to Hassin [5], makes use of the properties of shortest paths in a planar dual network.

The algorithm of Itai and Shiloach [6] for flows in (general) undirected planar networks consists of two phases, each of which runs in $O(n^2 \log n)$ time. In the first phase a minimum (s, t)-cut is found; in the second a flow with value equal to the capacity of this cut is constructed. Reif [11] has shown how to find a minimum (s, t)-cut in an undirected planar network in $O(n \log^2 n)$ time. In this paper we show how to extend the ideas of [5] so that a shortest path computation in a derived network that depends on a given feasible flow value yields a flow function of this value for a general undirected planar network. We then show how to solve this shortest path problem quickly using a set of cuts which can be constructed by a modification of Reif's algorithm. Given a flow value and these cuts, our algorithm runs in $O(n \log n)$ time, thus giving a combined algorithm which solves the maximum flow problem in undirected planar networks in $O(n \log^2 n)$ time.

Our bounds cited above are derived using Dijkstra's shortest path algorithm (see [1], [7]) as a subroutine. If the algorithms of Frederickson [2] are used, these bounds can be improved as will be discussed later. It is interesting to observe that, in the case of finding flows in (s, t)-planar networks, the algorithm of reference [5] is amenable

* Received by the editors December 7, 1982, and in final revised form May 9, 1984.

† Department of Statistics, Tel Aviv University, Tel Aviv 69978, Israel.

‡ Department of Computer Science, Pennsylvania State University, University Park, Pennsylvania 16802.
The work of this author was partially supported by the National Science Foundation under grant MCS 80-02684.

to improvement using Frederickson's algorithm, but the "uppermost path" algorithm is not since sorting can be reduced to the uppermost path computation [6].

2. Definitions and assumptions.

A *flow network* N is a quadruple (G, s, t, c) where

(i) $G=(V, E)$ is an undirected graph with n vertices and m edges and, throughout this paper, is assumed to be given with a fixed planar embedding,

(ii) s and t are distinct vertices, the *source* and *sink*, respectively, and

(iii) $c: E \to \mathrm{R}^+$ is a capacity function assigning a positive real to each edge.

We denote an (undirected) edge with endvertices v and w as $(v-w)$. A flow is a function $f: V \times V \to \mathrm{R}^+ \cup \{0\}$ satisfying $f(v, w)=0$ whenever $(v-w) \notin E$, $0 \leqq f(v, w)+f(w, v) \leqq c(e)$ for every edge $e=(v-w) \in E$, and $\sum_{(v-w) \in E}[f(v, w)-f(w, v)]=0$ for every vertex $v \in V-\{s, t\}$. The *value* of a flow f is defined by $v(f)=\sum_{(v-t) \in E}[f(v, t)-f(t, v)]$. A flow f is a *maximum flow* if $v(f) \geqq v(f')$ for every other flow f'. We denote the value of a maximum flow by $v_{\max}$ where the network referred to is understood.

A *cut* $C \subseteq E$ is a minimal set of edges that disconnects t from s. The capacity of a cut is the sum of the capacities of its edges. A classical result is that the value of a maximum flow is equal to the minimum over the capacities of all cuts [3].

Without loss of generality we assume that G is triconnected. (If G were not, it could be triangulated in linear time.) Therefore G has a unique dual $G^{\mathrm{d}}=(V^{\mathrm{d}}, E^{\mathrm{d}})$ which is a graph without loops and multiple edges. (The uniqueness is by virtue of the fixed embedding.) Let F and F^{d} denote the set of faces of G and G^{d}, respectively.

The following one-to-one correspondences exist: $V \leftrightarrow F^{\mathrm{d}}$, $F \leftrightarrow V^{\mathrm{d}}$, and $E \leftrightarrow E^{\mathrm{d}}$. For corresponding edges $(v'-w') \in E$ and $(v-w) \in E^{\mathrm{d}}$, v' is taken to correspond to v and w' to w when w' follows v' in the clockwise direction in the face corresponding to v. For each $e \in E^{\mathrm{d}}$ we define its *length* $l(e)$ to be equal to the capacity $c(e')$ of the corresponding primal edge $e' \in E$. These definitions give us a distance network $N^{\mathrm{d}}=(G^{\mathrm{d}}, l)$ corresponding to the given capacitated network N. The procedure for dualizing planar graphs, including the correspondence between the endpoints of the edges, is described, for instance, in [9].

3. Finding a minimum (s, t)-cut. We start with a brief description of Itai and Shiloach's algorithm.

Let ϕ^s and ϕ^t denote the faces in N^{d} which correspond to s and t, respectively. Without loss of generality we assume that ϕ^s is the exterior face of N^{d}. A minimum cut in N corresponds therefore to a cycle of minimum length enclosing ϕ^t in N^{d}.

Let $\Pi=(\xi^s=\xi_1, \cdots, \xi_k=\xi^t)$ be a shortest (ξ^s, ξ^t)-path in N^{d} where ξ^s is a dual vertex on ϕ^s and ξ^t is a dual vertex on ϕ^t, both chosen so as to minimize the length of Π over all shortest paths between such pairs. Call an edge $(\xi-\xi_i)$ Π-*left* if $\xi \notin \Pi$ and when traversing Π from ξ^s to ξ^t it is incident with ξ_i on the left. Define Π-*right* edges similarly. These definitions are extended to the edges incident with ξ^s and ξ^t by viewing Π as extended at each of its two ends by an edge to a new vertex situated properly within ϕ^s and ϕ^t, respectively. Figure 1 illustrates these concepts. Since N^{d} is triconnected and has a fixed embedding, every edge incident with Π is either Π-left or Π-right but not both.

Since Π is a shortest (ξ^s, ξ^t)-path, there exists a cycle of minimum length enclosing ϕ^t which intersects Π exactly once, and uses exactly one Π-right and one Π-left edge. The algorithm of Itai and Shiloach finds such a cycle and the corresponding minimum cut in the primal network as follows.

ALGORITHM MIN-CUT (N) [6]
{N is a flow network with dual N^d}
{MIN-CUT (N) is a minimum (s, t)-cut of N}
for $i = 1, \cdots, k$ **do**
Direct every Π-left edge $(\xi_i - \xi)$ in N^d from ξ_i to ξ and every Π-right edge $(\xi_i - \xi)$ from ξ to ξ_i.
endfor
for $i = 1, \cdots, k$ **do**
Let C_i^d in N^d be a *minimum ξ_i-cycle*, a shortest cycle that uses exactly one Π-left and one Π-right edge, and its Π-left edge is incident with ξ_i. {It is easy to see that such a cycle encloses ϕ^t}
endfor
return (the minimum (s,t)-cut corresponding to C_j^d for which
$$l(C_j^d) = \min \{l(C_i^d) | i = 1, \cdots, k\})$$
end MIN-CUT

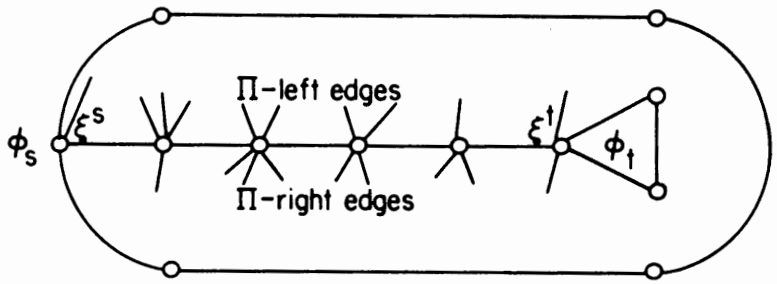

FIG. 1. *A partial rendition of the dual N^d of some given network N showing a shortest (ξ^s, ξ^t)-path Π and Π-left and Π-right edges.*

Given the directing of the edges in the first step, each cycle C_i^d in the second step can be found by splitting vertex ξ_i into ξ_i', with the Π-left edges, and ξ_i'' with the other edges incident with ξ_i, and then finding a shortest (ξ_i', ξ_i'')-path. When Dijkstra's algorithm (see [1], [7]) is used, the time required for finding each of these paths is bounded by $O(n \log n)$. Thus, since $k \leqq n$, it follows that MIN-CUT($\cdot$) terminates in $O(n^2 \log n)$ time.

Dijkstra's algorithm has been improved upon recently by Frederickson [2] in the case where the shortest path problem to be solved is on a planar network. As with Dijkstra's algorithm, Frederickson's results apply when all edge lengths are nonnegative. He shows that a single shortest path computation can be done in $O(n\sqrt{\log n})$ time. With $O(n \log n)$ preprocessing, each of any number of single source computations can be done in $O(n \log^* n)$ time. Thus Itai and Shiloach's minimum-cut algorithm can be implemented to run in $O(n^2 \log^* n)$ time. Similar improvements are obtainable in Reif's algorithm (discussed below) and in ours.

For simplicity of presentation we shall assume the use of Dijkstra's algorithm in each of the results in what follows and then, where appropriate, we shall indicate how Frederickson's algorithms can be employed. (As is well known, when computing with edge lengths from some restricted domains, the running time of Dijkstra's algorithm can also be improved. We omit discussion of what can be done in such special cases except to note that each of our bounds can be improved when finding shortest paths from a single source to all other vertices is $o(n \log n)$.)

Reif [11] describes a more efficient implementation of Itai and Shiloach's minimum-cut algorithm. He observes that if C^d is a minimum ξ_i-cycle then, for $j = 1, \cdots, i-1$,

there exist minimum ξ_j-cycles that enclose $C^{\mathbf{d}}$ (that is, have no vertices strictly within $C^{\mathbf{d}}$) and, for $j = i+1, \cdots, k$, there exist minimum ξ_j-cycles which are enclosed by $C^{\mathbf{d}}$. This observation allows the following divide-and-conquer algorithm, which we state for our purposes so that it generates a representation for each of the cuts corresponding to a minimum ξ_i-cycle for each $i = 1, \cdots, k$.

ALGORITHM CUTS (N_j) [11]
{N_j is an undirected planar flow network with n_j vertices}
{CUTS (N_j) is the set of cuts in N_j that correspond to the cycles in a set of minimum ξ_i-cycles in $N_j^{\mathbf{d}}$, one for each $i = 1, \cdots, k_j$}
if $k_j = 1$ **then return**({HIMID (N_j)})
else if $k_j = 2$ **then return**({LOMID (N_j)} $\cup$ {HIMID (N_j)})
else return({*HIMID* (N_j)} $\cup$ CUTS ($N_s(N_j)$) $\cup$ CUTS ($N_t(N_j)$))
end CUTS

Here

(i) HIMID (N_j) returns in $O(n_j \log n_j)$ time the cut corresponding to a minimum ξ_{mid}-cycle of N_j where $\Pi_j = (\xi^s = \xi_1, \cdots, \xi_{k_j} = \xi^t)$ and mid $= \lceil k_j/2 \rceil$. (An algorithm to do this is obtained from Algorithm MIN-CUT by replacing "**for** $i = 1, \cdots, k$" with "**for** $i = \lceil k/2 \rceil$".) LOMID (N_j) is similarly defined with mid $= \lfloor k_j/2 \rfloor$. (The introduction of the two "MIDs" corrects a minor error in the original presentation where, in fact, termination is not assured.)

(ii) The networks $N_s(N_j)$ and $N_t(N_j)$ are obtained from $N_j - \text{HIMID}(N_j)$ by adding a second source vertex s_t and a second sink vertex t_s and then, for each edge $(v-w) \in \text{HIMID}(N_j)$ where v is connected by some path to s in $N_j - \text{HIMID}(N_j)$, adding $(v - t_s)$ and an edge $(s_t - w)$, replacing multiple edges with a single "super" edge of capacity equal to the sum of the capacities of the replaced edges. The resulting network has two connected components; the one containing s is the (s, t_s)-flow network $N_s(N_j)$ and the other, containing t, is the (s_t, t)-flow network $N_t(N_j)$.

Let a network N_j which is an argument to CUTS ($\cdot$) be at level l if it is generated as a result of l prior calls to CUTS ($\cdot$) applied to N, that is, if it is generated at level l in the recursion. Thus the given network N is at level 0 and no network is at a level greater than $\lceil \log(k-1) \rceil$, since no (ξ^s, ξ^t)-path at level l has more than $|V(\Pi_{l-1})|+1$ vertices, where $|V(\Pi_{l-1})|$ is the number of vertices in the longest such path at level $l-1$, if the simple expedient is employed of inheriting, rather than recomputing, Π for each of $N_s(N_j)$ and $N_t(N_j)$ from Π for N_j.

From the construction it is evident that σ_l, the total number of vertices of networks at level l, must satisfy

$$\sigma_l \leqq \sigma_0 + \sum_{q=1}^{l} 2^q < n + 2 \cdot 2^{\log n} = 3n.$$

Thus, since both LOMID (N_j) and HIMID (N_j) run in $O(n_j \log n_j)$ time, where a network N_j has n_j vertices, the running time of Algorithm CUTS ($\cdot$) on a given network N with n vertices, is

$$O\left(\sum_{l=0}^{\lceil \log(k-1) \rceil} \sigma_l \log n\right) = O(n \log n \log k) = O(n \log^2 n).$$

Then, to find a minimum cut of the given network N takes time equal to $O(|\text{CUTS}(N)|)$ which is surely $O(n \log n \log k)$ by the timing analysis above. The reader may find a more thorough exposition of a more complicated proof in the original reference [11].

As indicated above, the result can be improved to $O(\min\{n \log n + n \log^* n \log k, n\sqrt{\log n}\, k\})$ when Frederickson's shortest path algorithms are used.

In the original reference, the set CUTS (N) is not constructed. Instead, only a minimum cut is found. With respect to our generalization, it must be noticed that in general some cuts in CUTS (N) are described in terms of "super" edges that represent sets of edges in some network nearer the root in the execution tree, and not explicitly in terms of the original edges of N. However, we may keep in CUTS (N) the information necessary to expand any cut to a description in terms of the edges of N.

It in fact is possible to obtain a representation for CUTS (N), which is in terms of the original edges of N and is $O(n)$ in size, at no asymptotically significant increase in running time and from which one minimum cut corresponding to ξ_i-cycle $C_i^{\mathbf{d}}$ can be recovered for each i in the order $i = 1, \cdots, k$, where a shortest (ξ^s, ξ^t)-path in the dual of the given network is $\Pi = (\xi^s = \xi_1, \cdots, \xi_k = \xi^t)$. The first step is to record the execution tree of CUTS $(\cdot)$ applied to N, assigning to the root the cut HIMID (N), to each of the two tree edges from the root the *changes* that need to be made to HIMID (N) to produce HIMID $(N_s(N))$ and HIMID $(N_t(N))$, respectively, etc., recording no cuts themselves at tree vertices other than the root. The edges to some leaves will need changes for both LOMID and HIMID.

This information can be recorded in terms of the "super" edges during the execution of CUTS $(\cdot)$ applied to N. Then the second step is an inorder traversal of the labeled execution tree to produce the changes, in Π order, in terms of the original edges of N. This step can be done within the same running time as CUTS $(\cdot)$ since "super" edges need be expanded at most twice, once when they enter some cut and once when they leave it or a later cut. If the edges of the cut C_1, corresponding to the minimum ξ_1-cycle, are placed initially in a search tree, ordered lexicographically on the edges taken as ordered pairs of vertices, the cuts can be constructed (though not output) in Π order in $O(n \log n)$ time by using the changes to modify the search tree containing the edges of one cut to obtain the next.

4. Finding a maximum (s, t) flow. An algorithm for constructing a flow of value D, if one exists, in a general (directed or undirected) planar network is described in [6]. This algorithm runs in $O(n^2 \log n)$ time and can be used to construct a maximum flow whenever the flow's value $v_{\max}$ is known.

In this section we describe an alternative algorithm which can be applied to undirected networks if $v_{\max}$ is known.

We first define a transformation of distance network $N^{\mathbf{d}}$ as follows:

(i) For each vertex $\xi_i \in \Pi$, two vertices ξ_i' and ξ_i'' are created.

(ii) Each edge $(\xi_i - \xi_{i+1})$ on Π is replaced by two edges $(\xi_i' - \xi_{i+1}')$ and $(\xi_i'' - \xi_{i+1}'')$ each with length equal to $l(\xi_i - \xi_{i+1})$.

(iii) Edges directed from ξ_i'' to ξ_i' with length $-v_{\max}$ are added for $i = 1, \cdots, k$. (These are the only directed edges.)

(iv) Every Π-left edge $(\xi_i - \xi)$ is replaced by $(\xi_i' - \xi)$ with length equal to $l(\xi_i - \xi)$. Every Π-right edge $(\xi_i - \xi)$ is replaced by $(\xi_i'' - \xi)$ with length equal to $l(\xi_i - \xi)$.

(v) Every vertex $\xi_i \in \Pi$ is now isolated and may be removed.

The transformation is illustrated in Fig. 2. We denote the transformed graph by $G^{\mathbf{t}} = (V^{\mathbf{t}}, E^{\mathbf{t}})$ and the transformed distance network by $N^{\mathbf{t}}$.

Let $1 \leqq r \leqq k$ be an index for which it was found that the length of the minimum ξ_r-cycle was $v_{\max}$. For every vertex $v \in V^{\mathbf{t}}$ let $u(v)$ be the length of a shortest (ξ_r', v)-path in $N^{\mathbf{t}}$. Since the length of every minimum ξ_i-cycle in $N^{\mathbf{d}}$ was found to be at least $v_{\max}$, $N^{\mathbf{t}}$ has no negative cycles and $u(v)$ is defined for every v.

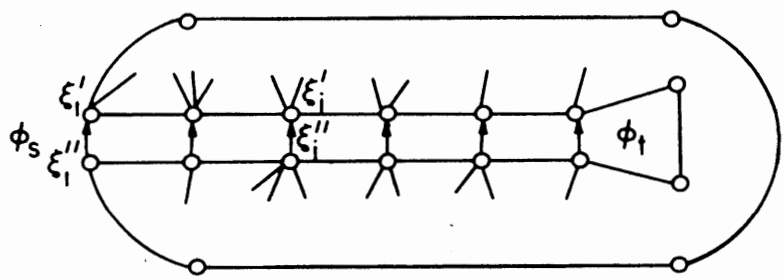

FIG. 2. *A partial rendition of the transformed dual N^t of some given network N showing how the vertices ξ_i are split into ξ_i' and ξ_i'' and new directed edges are introduced.*

LEMMA 1. $u(\xi_i') = u(\xi_i'') - v_{\max}$ for $i = 1, \cdots, k$.

Proof. By construction, $u(\xi_r') = 0$ and $u(\xi_r'') = v_{\max}$. Since Π is a shortest (ξ^s, ξ^t)-path there are only two possibilities concerning a shortest (ξ_r', ξ_i')-path:

(i) It reaches ξ_i' from ξ_i''. In this case the lemma holds immediately.

(ii) It terminates in a sequence of vertices $(\xi_j' | j \in J)$. If ξ_p' is the first vertex in this sequence, then $u(\xi_p') = u(\xi_p'') - v_{\max}$, since either $p = r$ or a shortest (ξ_r', ξ_p')-path reaches ξ_p' from ξ_p''. This establishes that $u(\xi_i'') \leqq u(\xi_i') + v_{\max}$.

On the other hand, ξ_i' can be reached from ξ_i'' along edge $(\xi_i'' - \xi_i')$ and, since $l(\xi_i'' - \xi_i') = -v_{\max}$, it follows that $u(\xi_i') \leqq u(\xi_i'') - v_{\max}$. The lemma in this case follows from the last two inequalities. □

For every edge $(v - e) \in E^t$, let us define $u^t_{vw} = -u^t_{wv} = u(w) - u(v)$. By Lemma 1, $u(\xi_i') - u(\xi_j') = u(\xi_i'') - u(\xi_j'')$, so that u^d_{vw} is well defined for each edge $(v - w) \in E^d$ as follows.

(i) For every Π-left edge $(\xi_a - w) \in E^d$, $u^d_{\xi_a w} = u^t_{\xi_a' w}$.

(ii) For every Π-right edge $(v - \xi_a) \in E^d$, $u^d_{v\xi_a} = u^t_{v\xi_a''}$.

(iii) For every Π-edge $(\xi_a - \xi_b) \in E^d$, $u^d_{\xi_a\xi_b} = u^t_{\xi_a'\xi_b'} = u^t_{\xi_a''\xi_b''}$.

(iv) For every other edge $(v - w) \in E^d$, $u^d_{vw} = u^t_{vw} = u(w) - u(v)$.

THEOREM 1. *A maximum flow f can be constructed as follows.*

For each edge $(v - w) \in E^d$ and associated edge $(v' - w') \in E$ where v corresponds with v' and w corresponds with w', let

$$f(v', w') = \max\{0, u_{vw}\} \text{ and } f(w', v') = \max\{0, u_{wv}\}.$$

Proof. The following observations show that f is a flow with value $v_{\max}$ and thus a maximum flow.

(i) $0 \leqq \max\{0, u_{vw}\} \leqq l(v, w) = c(v', w')$.

(ii) For every dual face $\phi \in F^d - \{\phi^s, \phi^t\}$, with dual vertices $v_1, \cdots, v_q, v_{q+1} = v_1$ in clockwise order and primal vertex $a \in V - \{s, t\}$ associated with ϕ,

$$\sum_{i=1}^{q} u_{v_i v_{i+1}} = \sum_{(a-b) \in E} (f(b, a) - f(a, b)) = 0.$$

See Fig. 3.

(iii) Let $\xi_k' = v_1, \cdots, v_q = \xi_k''$ be the vertices belonging to ϕ^t in N^t in clockwise order (see Fig. 4). By Lemma 1,

$$u(v_1) - u(v_q) = u(\xi_k') - u(\xi_k'') = -v_{\max},$$

so that

$$\sum_{i=1}^{q-1} u^t_{v_i v_{i+1}} - v_{\max} = 0.$$

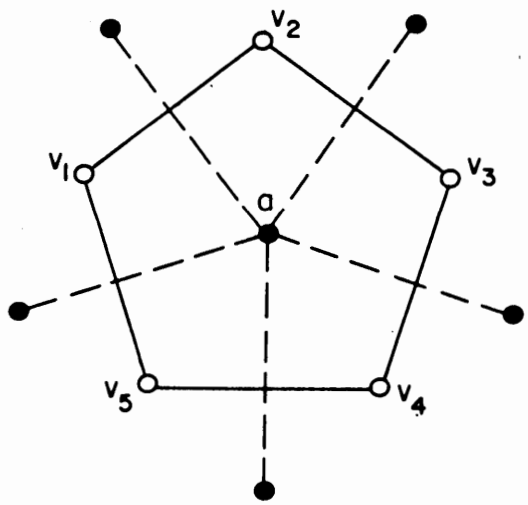

FIG. 3. *Dual face* $\phi = (v_1, v_2, v_3, v_4, v_5)$ *corresponding to primal vertex a, for the case* $\phi \in F^d - \{\phi^s, \phi^t\}$, *where* $q = 5$. *Primal edges are shown dashed. As described in* § 2, *primal and dual edges that cross correspond.*

This implies

$$\sum_{(t-v')\in E} (f(v', t) - f(t, v')) = v_{\max}. \qquad \square$$

Once the labeling function $u(V^t)$ is obtained, the above construction can easily be seen to yield a flow function in $O(n)$ time. Thus, it remains to show how to compute $u(V^t)$ efficiently.

We note before proceeding to this discussion that, when $k = O(1)$, the bound in § 3 for finding CUTS (N) reduces to $O(n \log n)$. In the case when the network is (s, t)-planar (*i.e.* $k = 1$), the dual faces ϕ^s and ϕ^t have a common dual vertex which can be chosen as $\xi^s = \xi^t$ so that only one cycle need be found. In this case the algorithm is essentially the one described in [5] and the time needed to find both a minimum cut and a maximum flow is $O(n \log n)$ when Dijkstra's algorithm is used and $O(n\sqrt{\log n})$ when Frederickson's algorithm is used.

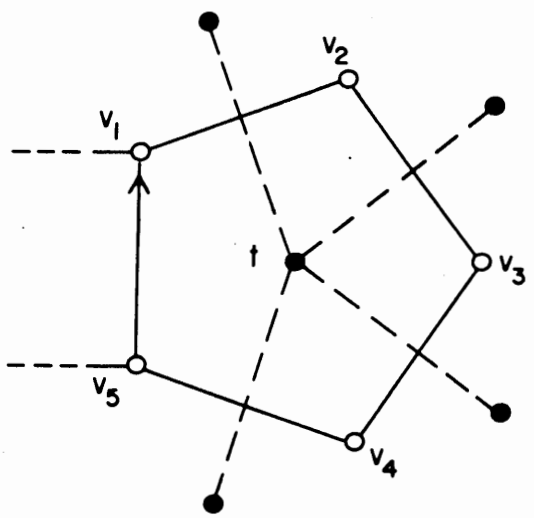

FIG. 4. *Dual face* $\phi = (v_1, v_2, v_3, v_4, v_5)$ *corresponding to primal vertex t. Notice that there is no primal edge corresponding to dual edge* $(v_5 - v_1)$, *since this directed edge was introduced in the transformation of* N^d.

5. Computing distances in the transformed dual network. As above, let a minimum cut cycle of N^d be a ξ_r-cycle for some r, $1 \leqq r \leqq k$. Shortest (ξ_r', v)-paths in N^t can be computed for every $v \in V^t$ in $O(n^{3/2})$ time [10] and thus a maximum flow can be obtained, as described in the previous section, within this bound. It is not known how to solve the shortest path problem faster in general in planar networks when $\Omega(n)$ of the edges have negative lengths. However, our network N^t has a structure which we exploit to obtain the required shortest distances in $O(n \log n)$ time, using repeated applications of Dijkstra's algorithm, when given a suitable representation of the minimum ξ_i-cycles in N^t that correspond to the (noncrossing) cuts in the set CUTS (N).

We denote the minimum ξ_i-cycles that we obtain in § 3 as C_i^t where, for each $i = 1, \cdots, k$, cycle C_i^t corresponds with the primal cut C_i. We now define the sets Δ_i^+ and Δ_i^- for $i = 1, \cdots, k-1$ by the relation

$$V(C_{i+1}^t) = (V(C_i^t) - \Delta_i^-) \cup \Delta_i^+,$$

where

(i) $V(C_i^t)$ is the vertex set of C_i^t for $i = 1, \cdots, k$, and

(ii) the intersection $\Delta_i^+ \cap \Delta_i^-$ contains only those vertices in $C_i^t \cap C_{i+1}^t$ connected by an edge to some vertex in $C_i^t - C_{i+1}^t$. See Fig. 5. We note that the inverse relation also holds,

$$V(C_i^t) = (V(C_{i+1}^t) - \Delta_i^+) \cup \Delta_i^-.$$

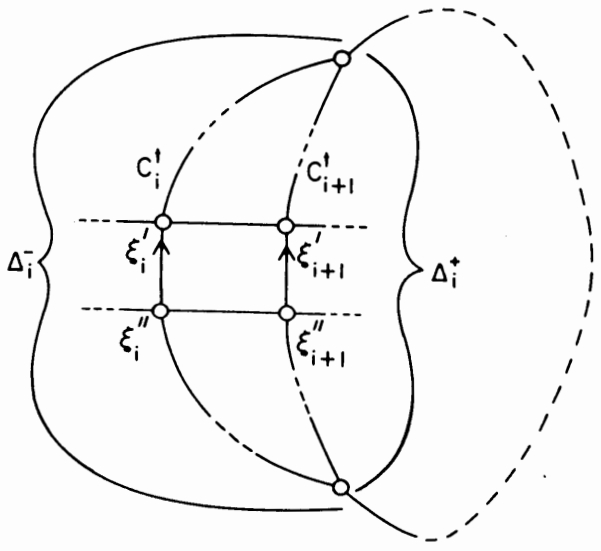

FIG. 5. *Example of C_i^t and C_{i+1}^t in the general case where there may be vertices in common. The set Δ_i^- is comprised of all vertices of $C_i^t - C_{i+1}^t$ plus the first and last vertices on $C_i^t \cap C_{i+1}^t$. The set Δ_i^+ is comprised of all vertices of $C_{i+1}^t - C_i^t$ plus the first and last vertices on $C_i^t \cap C_{i+1}^t$.*

The sets Δ_i^+ and Δ_i^- for $i = 1, \cdots, k-1$ can be generated in order $i = 1, \cdots, k-1$ in $O(n)$ time by a traversal of the Reif execution tree as described in § 3. This fact would be immediate if Δ_i^+ and Δ_i^- were defined so that $\Delta_i^+ \cap \Delta_i^- = \varnothing$. However, even though there is overlap, the bound of $O(n)$ can be seen to hold by observing that the sum over all vertices of the number of times a vertex can repeat within either all the Δ^+ sets or all the Δ^- sets is bounded by the number of edges in N^t, which is $O(n)$. Not only can these sets be used, as described in § 3, to recover the minimum length

cycles C_i^t, they are also of essential use in reducing the complexity of the computation of the labels $u(v)$ for $v \in V^t$ to $O(n \log n)$.

The cycles C_i^t for $i = 1, \cdots, k$ divide N^t into $k+1$ subnetworks $N_0^t, \cdots, N_k^t$ where, for $i = 1, \cdots, k-1$, N_i^t is the subnetwork bounded by and including C_i^t and C_{i+1}^t, N_0^t is everything outside and including C_1^t, and N_k^t is everything within and including C_k^t. See Fig. 6. Since the intersection of two adjacent subnetworks is a shortest cycle we obtain the following result.

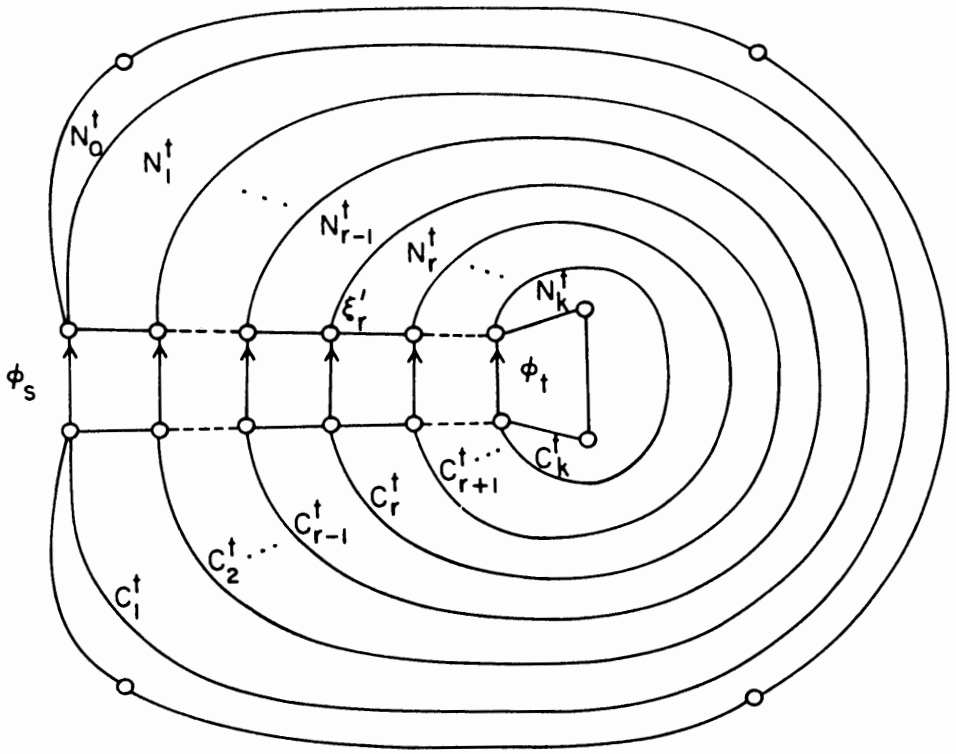

FIG. 6. *Example of subnetworks of N^t induced by the cycles C_i^t, $i = 1, \cdots, k$. It may be that adjacent cycles have subpaths in common as shown in Fig. 5.*

LEMMA 2. *Let v be a vertex in N_i^t. Then if $i < r$ there exists a shortest (ξ_r', v)-path in N^t which is contained in $\cup_{j=0,r-1} N_j^t$. Similarly, if $i \geqq r$ there exists a shortest (ξ_r', v)-path in N^t which is contained in $\cup_{j=r,k} N_j^t$.*

This lemma implies that the computation of $u(v)$ for $v \in V^t$ can be restricted to the subnetwork $\cup_{j=0,r-1} N_j^t$ for v in this subnetwork, and similarly for v in $\cup_{j=r,k} N_j^t$. We confine our detailed discussion to the latter case. The former case is treated similarly.

Let $P(v)$ be a shortest (ξ_r', v)-path. An example is given in Fig. 7, where P_v is shown as a concatenation of subpaths P_1, P_2, P_3, P_4, and P_5. In general, for any vertex $v \in V^t$ where v is in N_i^t let a *normal* path be a simple (ξ_r', v)-path $P(v) = (P_r, \cdots, P_q, \cdots, P_{2q-i})$ such that, for $j = r, \cdots, q$, subpath P_j is in N_j^t, and, for $j = q+1, \cdots, 2q-i$, subpath P_j is in N_{2q-j}^t and uses no edges of negative length. We require also that q be minimal subject to these conditions. Call q the *index of reversal* of $P(v)$. As the following lemma states, for every v there exists a shortest (ξ_r', v)-path that is normal.

LEMMA 3. *For any $i = r, \cdots, k$, for every vertex v in N_i^t there exists a normal shortest path $P(v)$.*

Proof. (It may be helpful to consider the example in Figure 7.) Assume that a shortest path P touches some shortest cycle C^t corresponding to some member of CUTS (N). If the last shortest cycle it touched was also C^t, then there is a subpath of the cycle that can be used to replace the subpath of P whose endpoints are on the cycle. Then, once P departs from some cycle C_j^t into N_{j-1}^t, P cannot use a negative edge because, to do so, it would cross itself and the embedded cycle thus created (which could not have negative length) could be removed. □

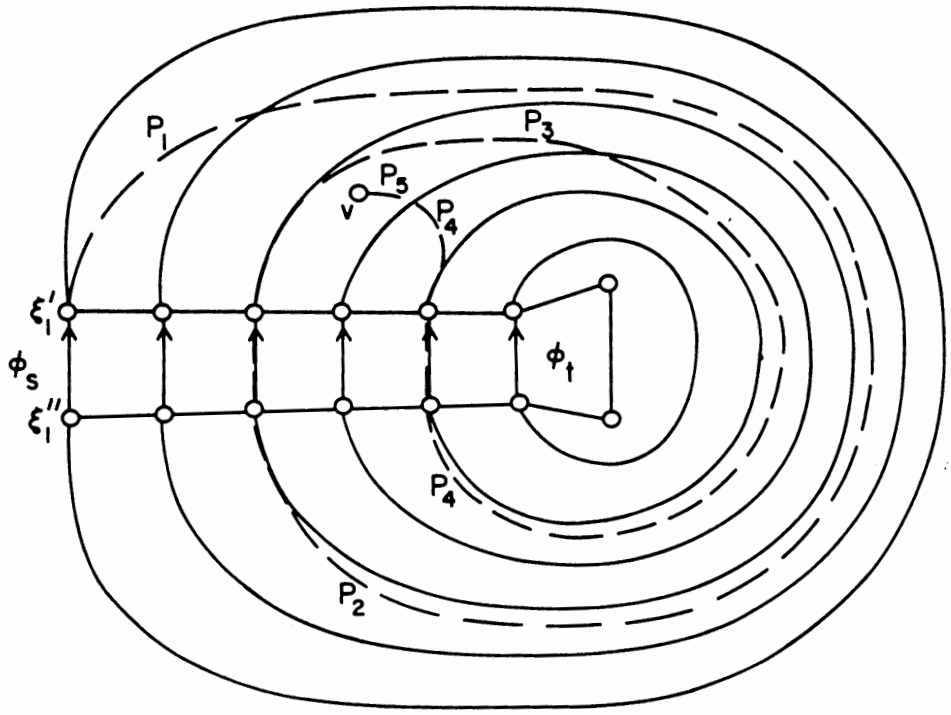

FIG. 7. *Example of shortest* (ξ_r', v)*-path* $P(v) = (P_1, P_2, P_3, P_4, P_5)$. *In this example* $r = 1$, *the index of reversal* $q = 4$, *and* v *is in* N_3^t.

We now give an algorithm to compute $u(v)$ for all v in $\cup_{i=r,k} N_i^t$.

ALGORITHM INSIDE-LABELS (N^t)
  {Given the transformed dual N^t, INSIDE-LABELS $(\cdot)$ produces the labels $u(v)$ for all the vertices v in $\cup_{i=r,k} N_i^t$}
  {Initialize the labels}
  for v in $\cup_{i=r,k} N_i^t$ **do** $u(v) \leftarrow \infty$ **endfor**
  $u(\xi_r') \leftarrow 0$
  {Compute shortest paths in C_r^t}
  SP $(C_r^t, \{\xi_r'\})$
  {Compute shortest paths in the forward direction}
  for $i \leftarrow r$ **until** k **do**
    {Compute shortest paths in N_i^t from start vertices in Δ_i^-}
    SP (N_i^t, Δ_i^-)
  endfor
  {Extend shortest paths in the backward direction}
  SP $((\cup_{i=r,k} N_i^t), \Delta_{k-1}^+)$
end INSIDE-LABELS

In Algorithm INSIDE-LABELS $(\cdot)$, SP (X, W) is a two-step algorithm. The first step is Dijkstra's shortest path algorithm applied to network X from which all edges of negative length have been removed and with whatever u-labels its vertices have, starting the candidate set with the vertices in W. This is equivalent to running the usual version of the algorithm from w after deleting the edges of negative length and augmenting X with a new vertex w and edges $(w-v)$ of length $u(v)$ for each $v \in W$. The second step is to treat the edges of negative length individually as follows. For edge $(\xi''-\xi')$ execute the assignment

$$u(\xi') = \min\{u(\xi'), u(\xi'') - v_{\max}\}.$$

LEMMA 4. *Algorithm* INSIDE-LABELS $(\cdot)$ *computes* $u(v)$ *correctly for all* $v \in \cup_{i=r,k} N_i^t$. *Whenever a vertex* $v \in V(C_i^t) - \Delta_i^-$ *is expanded in a shortest path computation on* N_i^t *in the forward loop, no labels change.*

Proof. Observe that Dijkstra's algorithm is applied always to subnetworks without edges of negative length. The effect of the edges of negative length is obtained explicitly following each application of Dijkstra's algorithm.

Give C_r^t the name N_{r-1}^t. First, it can be seen that all labels $u(v)$ for $v \in N_{r-1}^t$ are computed correctly by the first three lines of the algorithm. Then, let V_h be the set of all vertices v for which there is a normal shortest (ξ_r', v)-path $P(v) = (P_r, \cdots, P_{h-1})$.

Consider first such paths for which $h-1$ is the index of reversal (that is, there is no reversal). Assume that $u(v)$ has been correctly computed by INSIDE-LABELS $(\cdot)$ for all $v \in V_h$ when $i = h \leqq k$ before the beginning of some iteration of the loop that starts with "**for** $i \leftarrow r$ **until** k **do**". Consider a vertex $w \in N_h^t$ for which there is a normal shortest (ξ_r', w)-path $P(w) = (P_r, \cdots, P_h)$. If $w \in C_h^t$, then $u(w)$ is already correct by assumption. Otherwise, $w \notin C_h^t$, and the last vertex x on $P(w)$ that is in N_{h-1}^t and therefore with correct label $u(x)$, is in Δ_h^-. It follows that the iteration with $i = h$ must compute $u(w)$ correctly and that no label is changed by an expansion of a vertex in $V(C_i^t) - \Delta_i^-$. By induction, then, it is shown that $u(v)$ is computed correctly for all $v \in V_h$, $r \leqq h \leqq k$.

A simpler argument is applicable to the segment $(P_q, \cdots, P_b)$, of any normal shortest path $P(v) = (P_r, \cdots, P_q, \cdots, P_b)$ with index of reversal $q < b$, given as we have just proved that $u(w)$ is correct for all w for which there exists a normal shortest path $P(w) = (P_r, \cdots, P_q)$ upon completion of the loop that starts with "**for** $i \leftarrow r$ **until** k **do**". These segments contain no edges of negative length. Thus a single application of Dijkstra's algorithm suffices and, in fact, the second step in SP can be omitted. The desired results follow from Lemma 2. □

When INSIDE-LABELS $(\cdot)$ is combined with a similar procedure to calculate labels on $\cup_{i=1,r-1} N_i^t$ we have a correct procedure for the entire problem. Call this combined procedure LABELS $(\cdot)$.

From earlier discussions it follows that the running time of LABELS $(\cdot)$ is $O(s \log n)$, where $s < n^t + \sum_{i=1,k} |C_i^t|$ for $n^t = |V(N^t)| = O(n)$. There exist networks N for which $\sum_{i=1,k} |C_i^t| = \Omega(n^2)$, so LABELS $(\cdot)$ as stated above has a running time of $O(n^2 \log n)$.

To improve this bound we give the following refinements in which shortest paths are computed on the subnetworks $N_i^t - (V(C_i^t) - \Delta_i^-)$, as Lemma 4 permits, and these subnetworks are produced explicitly by removing and replacing edges in N^t. In INSIDE-LABELS $(\cdot)$ replace

{Compute shortest paths in C_r^t}
SP $(C_r^t, \{\xi_r'\})$

with

{Isolate C_r^t in N^t}
$N^t \leftarrow N^t - \{(v-w) | v \in C_r^t, w \notin C_r^t\}$
{Compute shortest paths in C_r^t}
SP $(N^t, \{\xi_r'\})$
{Disconnect C_r^t}
$N^t \leftarrow N^t - \{(v-w) | v, w \in C_r^t\}$,

and replace

for $i \leftarrow r$ **until** k **do**
  {Compute shortest paths in N_i^t from start vertices in Δ_i^-}
  SP (N_i^t, Δ_i^-)
endfor

with

for $i \leftarrow r$ **until** k **do**
  {At each iteration, C_i^t is disconnected}
  {Put into N^t the edges that enter the proper interior of N_i^t from C_i^t, and delete the edges that are incident on some vertex properly within the region bounded by C_{i+1}^t. Thus, since no edges come into N_i^t from the two regions surrounding it, SP $(\cdot)$ will compute on N_i^t}
  $N^t \leftarrow (N^t \cup \{(v-w) | v \in \Delta_i^-, w \notin C_i^t$ and within $C_i^t\})$
    $-\{(v-w) | v \in \Delta_i^+, w \notin C_{i+1}^t$ and within $C_{i+1}^t\}$
  {Compute shortest paths in N_i^t from start vertices in Δ_i^-}
  SP (N^t, Δ_i^-)
  {Maintain C_{i+1}^t disconnected}
  $N^t \leftarrow N^t - \{(v-w) | v, w \in \Delta_i^+\}$
endfor.

Now, the sum over all forward shortest path computations of the number of vertices processed in each computation is $O(n)$ since $\sum_{i=r,k} |\Delta_i^-|$ is $O(n)$, so these computations are of complexity $O(n \log n)$. The manipulations of N^t can also be implemented to run in $O(n \log n)$ time over all. This gives us our theorem.

THEOREM 2. *Shortest (ξ_r', v)-paths can be computed for every $v \in V^t$ in $O(n \log n)$ time, given the set* CUTS (N).

Frederickson's shortest path algorithm without preprocessing [2] can be used in each instance where Dijkstra's algorithm is used in our algorithm, giving a bound of $O(n\sqrt{\log n})$ overall when given the set CUTS (N).

Theorems 1 and 2, together with the results of § 3, imply an algorithm for maximum flows in undirected planar networks that runs in $O(n \log^2 n)$ time and $O(n)$ space. With Frederickson's improvements, the time bound is $O(n \log n \log^* n)$ when $k = \Omega(n)$ and as small as $O(n\sqrt{\log n} \log k)$ when k is small.

6. Acknowledgment. We are grateful to a referee for pointing out an error, now corrected, in the result of § 5.

REFERENCES

[1] E. W. DIJKSTRA, *A note on two problems in connexion with graphs*, Numer. Math., 1 (1959), pp. 269-271.
[2] G. N. FREDERICKSON, *Shortest path problems in planar graphs*, Proc. 24th Annual Symposium on Foundations of Computer Science, 1983, pp. 242-247.

[3] L. R. FORD AND D. R. FULKERSON, *Maximal flow through a network*, Canad. J. Math., 8 (1956), pp. 399-404.

[4] Z. GALIL, *A new algorithm for the maximal flow problem*, Proc. 19TH Annual Symposium on Foundations of Computer Science, 1978, pp. 231-245.

[5] R. HASSIN, *Maximum flow in (s, t) planar networks*, Inform. Proc. Lett., 13 (1981), p. 107.

[6] A. ITAI AND Y. SHILOACH, *Maximum flow in planar networks*, this Journal, 8 (1979), pp. 135-150.

[7] D. B. JOHNSON, *Efficient algorithms for shortest paths in sparse networks*, J. Assoc. Comput. Mach., 24 (1977), pp. 1-13.

[8] D. B. JOHNSON AND S. M. VENKATESAN, *Using divide and conquer to find flows in directed planar networks in $O(n^{3/2} \log n)$ time*, Proc. Twentieth Annual Allerton Conference on Communication, Control, and Computing, Univ. Illinois, Urbana, October 1982, pp. 898-905.

[9] E. L. LAWLER, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart, and Winston, New York, 1976.

[10] R. J. LIPTON, D. J. ROSE, AND R. E. TARJAN, *Generalized nested dissection*, SIAM J. Numer. Anal., 16 (1979), pp. 346-358.

[11] J. H. REIF, *Minimum s-t cut of a planar undirected network in $O(n \log^2 (n))$ time*, this Journal, 12 (1983), pp. 71-81.

[12] D. D. SLEATOR, *An $O(nm \log n)$ algorithm for maximum network flow*, Ph.D. Dissertation, Comp. Sci. Dept., Stanford Univ., Stanford, CA, 1980.

[13] D. D. SLEATOR AND R. E. TARJAN, *A data structure for dynamic trees*, Proc. 13th Annual ACM Symposium on Theory of Computing, 1981, pp. 114-122; J. Comput. System Sci., 26 (1983), pp. 362-391.