Parallel Algorithms for Minimum Cuts and Maximum Flows in Planar Networks

DONALD B. JOHNSON

Dartmouth College, Hanover, New Hampshire

Abstract. Algorithms are given that compute maximum flows in planar directed networks either in $O((\log n)^3)$ parallel time using $O(n^4)$ processors or $O((\log n)^2)$ parallel time using $O(n^6)$ processors. The resource consumption of these algorithms is dominated by the cost of finding the value of a maximum flow. When such a value is given, or when the computation is on an undirected network, the bound is $O((\log n)^2)$ time using $O(n^3)$ processors. No efficient parallel algorithm is known for the maximum flow problem in general networks.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems; G.2.2 [**Discrete Mathematics**]: Graph Theory—*graph algorithms*; *network problems*

General Terms: Algorithms, Theory, Verification

Additional Key Words and Phrases: Directed networks, network flow, parallel computation, planarity, planar duality

1. *Introduction*

An important question in parallel computation is what combinatorial problems are suited to parallel computation, in the sense that efficient use can be made of parallel resources, and what problems are not suitable in this sense. If we confine ourselves to problems in **P**, then we may adopt the criterion of whether or not there exists an algorithm that runs in polylogarithmic time, that is, in $O((\log n)^k)$ time for some k, using a polynomial number of processors.

The maximum flow problem in networks is a significant problem in the context of this question as well as an important problem in operations research. The general problem is presumably unsuited to parallel computation in the sense indicated above, since it has been shown [6, 7] that finding even the value of a maximum flow is log space complete for **P**. The existence of a polylogarithmic-time parallel algorithm for any problem in this class implies such an algorithm for every problem in **P**. However, our results show that, when networks are planar, maximum flows

This research was partially supported by the National Science Foundation under grant MCS 80-02684 and by sabbatical leave from The Pennsylvania State University.

A preliminary version of this work appears in *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science* (Nov.). IEEE, New York, 1982, pp. 244–254. © IEEE 1982. Any portion of this paper that appeared in the preliminary version is reprinted with permission.

Author's address: Department of Mathematics and Computer Science, Bradley Hall, Dartmouth College, Hanover, NH 03755.

can be found in polylogarithmic time using a polynomial number of processors with a common memory, in both the case of undirected and of directed networks.

The first step in our solution of the maximum flow problem is to obtain a planar dual in which distances relate to the flows to be computed on the primal network. Whereas efficient applications of duality theory are known in the serial model for the undirected case [9–11, 16], no such result has been found previously for the directed case in either serial or parallel models. A parallel algorithm for maximum flows that runs in $O(n^2 \log n)$ time using n processors is known [17], but as noted below this bound can be achieved for planar networks by known serial algorithms.[1]

To find a maximum flow in a planar directed network, we first compute its value using the dual network alluded to above. This algorithm runs in $O((\log n)^2)$ time using $O(n^6)$ processors. Additional results for computing path lengths give a modified algorithm that runs in $O((\log n)^3)$ time using $O(n^4)$ processors. When the flow value is known, the dual network can be augmented so that flows in primal edges are equal to differences of shortest distances in the dual network. The algorithm to compute the flow runs in $O((\log n)^2)$ time using $O(n^3)$ processors. We have no better bounds for computing flows in undirected networks given an embedding and a flow value (even in the (s, t)-planar case).

Our results employ the model of unbounded parallelism where it is assumed that an unbounded number of processors is available, memory is common to all processors, and a memory location can be read from, but not written into, simultaneously by more than one processor.

On one processor, that is in the serial model, the best results known are $O(nm \log(n^2/m))$ for general directed or undirected networks with n vertices and m edges [4, 5, 18, 19], which is $O(n^2 \log n)$ on any network in which $m = O(n)$. On planar networks, the best serial bounds are $O(n^{3/2} \log n)$ in the directed case [13] and $O(n(\log n)^2)$ in the undirected case [10]. (These bounds are improvable by almost a factor of $\log n$ when Frederickson's algorithms for shortest paths in planar networks [3] are used in place of Dijkstra's algorithm.) These last two flow algorithms are quite different from each other and from the augmentation algorithms that give the best running times on general, not necessarily planar, networks. The $O(n^{3/2} \log n)$ algorithm for the directed case is a divide-and-conquer algorithm that operates on successively subdivided regions of a planar embedding of the given network. The $O(n(\log n)^2)$ algorithm for the undirected case solves a distan9e problem in a planar dual of the given network. To this extent it employs the approach we use in the present paper.

2. *Preliminaries*

Let $G = (V, E)$ be a directed graph on n vertices with no vertex with exactly one entering and one leaving edge. For the graph of a flow network, this limitation causes no loss of generality. In order to simplify our presentation, we assume at the outset that G has at most one edge between any pair of vertices and that G is given with an arbitrary, fixed embedding in the plane. The assumption of a planar embedding has no significant effect on our results since such an embedding can be obtained using the method of Klein and Reif [14] in $O((\log n)^2)$ time using $O(n)$ processors, bounds that do not dominate the requirements of our algorithms.

[1] An alternative presentation of our results for the undirected case has been given by Janiga and Koubek in a note entitled "A Note on Finding Minimum Cuts in Directed Planar Networks by Parallel Computation" [12], which, the title notwithstanding, deals exclusively with the undirected case.

A *path* p of edge-length k in graph G is a connected subgraph induced by an ordered edge set $X = \{(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)\}$ (under a suitable indexing of vertices). A path p is a *cycle* if $v_0 = v_k$. A path or cycle p is *simple* if no vertex on it has more than one entering edge. We employ the notation $v_i \in p$ and $(v_i, v_{i+1}) \in p$ to express occurrence of vertices and edges, respectively, on paths. We also find it convenient to denote paths as sequences of vertices.

A *distance network* $N = (G, l)$ has a *length function* l: $V \times V \rightarrow \mathbf{R} \cup \{\infty\}$, where $l(v, w) = \infty$ if and only if $(v, w) \notin E$. The *length* of a path p in a distance network is the sum of the values of the length function on its edges. The *distance* from vertex v to vertex w in a distance network is the minimum of the lengths of all paths from v to w, or α if there is no path.

A *flow network* $N = (G, c, s, t)$ has a *capacity function* c: $V \times V \rightarrow \mathbf{R}^+ \cup \{0\}$ and distinguished vertices s and t, the *source* and *sink*, respectively. It is required that $c(v, w) = 0$ for all $(v, w) \notin E$. We also refer to networks $N = (G, c)$ in which a source and sink are not identified. A set $X \subseteq E$ is a *disconnecting set* if the undirected version of $(V, E - X)$ has no path from s to t. A disconnecting set $X \subseteq E$ is an (s, t)*-cut* if no proper subset of X is disconnecting. Let (V_s, E_s) and (V_t, E_t) be the weakly connected components of $(V, E - X)$ for some (s, t)-cut X where $s \in V_s$ and $t \in V_t$. The *capacity* $c(X)$ of an (s, t)-cut X is the sum of the capacities of the edges in $\{e \mid e = (v, w) \in X \text{ and } v \in V_s, w \in V_t\}$. An (s, t)-cut X is minimum if $c(X)$ is minimized over all (s, t)-cuts. An (s, t)*-flow* is a function f: $V \times V \rightarrow \mathbf{R}$ for which $0 \leq f(e) \leq c(e)$ for each $e \in V \times V$. Additionally,

$$val(v) \equiv \sum_w f(w, v) - \sum_w f(v, w) = 0$$

for all vertices $v \neq s, t$. We call $val(v)$ the *value of f at vertex v* and $val(f) \equiv val(t)$ the *value* of f. Additional definitions and terminology in graph and network flow theory may be found in standard references [1, 8, 15].

3. *Computing a Minimum-Capacity Cut in a Directed Network*

The following fundamental result in network flow theory states a duality between cuts and flows:

LEMMA 1 [2]. *The value of a maximum flow is equal to the capacity of a minimum (s, t)-cut in any flow network.*

Our algorithm exploits this duality explicitly in the framework of a graph theoretic duality that exists for planar graphs. The subject of this section is the first step of the algorithm, computing minimum capacity cuts. Section 4 presents additional results for computing paths that are shortest subject to constraints, and Section 5 shows how to find a flow of a given value.

The *plane dual* $D = ((\phi, D(E)), l)$ of N is a planar distance network defined in terms of N and its planar representation. The set ϕ is a set of dual vertices F that correspond to the faces of G. In the planar representation, each dual vertex F is located in the face of G to which it corresponds. We also use lowercase letters for dual vertices, for example, $v, w \in \phi$. As shown in Figure 1, for each edge $e \in E$, let $D(e)$ be a directed dual edge that crosses e from left to right (from the dual vertex in the face on the left of e to the dual vertex in the face on the right of e) and has length $l(D(e)) = c(e)$. The edge set $D(E)$ of the dual network is then defined as follows:

$$D(E) = \{D(e) \mid e \in E\} \cup \{(w, v) \mid D^{-1}(v, w) \in E \text{ and } D^{-1}(w, v) \notin E\}.$$

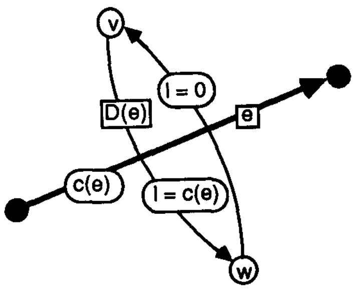

FIG. 1. Primal edge e, with capacity $c(e)$, and dual edges $D(e)$ of length $c(e)$ and back edge of length 0.

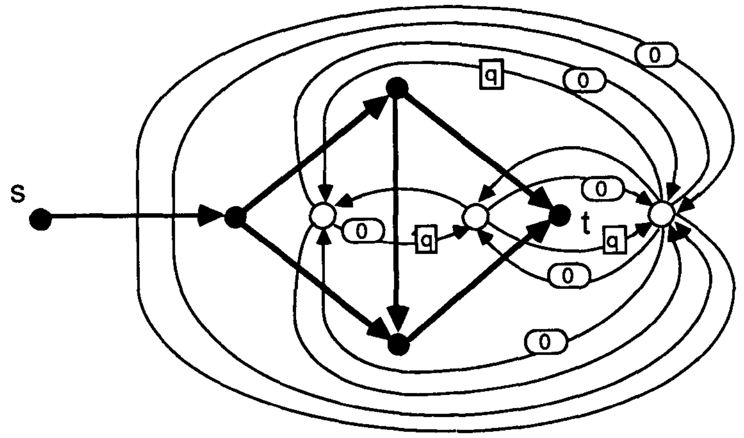

FIG. 2. Network N and dual D showing back edges of length 0 and a forward cut-cycle q.

The additional edges (w, v) are called *back edges* and are assigned $l(w, v) = 0$. Back edges are embedded in the planar representation so that they cross the edge from which they are derived from right to left. The length function l on $D(E)$ extends to sets of edges (paths in particular) as the sum. The graph $(\phi, D(E))$ of the dual network D may have edges that are loops.

Let q be a simple cycle in D, as shown in Figure 2, for example. If q (taken as a Jordan curve) separates primal vertices s and t in the plane, then q is a *cut-cycle.* A cut-cycle q that is directed clockwise around s (or anticlockwise around t) is a *forward* cut-cycle. There is a one-to-one correspondence between forward cut-cycles q in D and (s, t)-cuts X in N and, for any such pair (q, X), $l(q) = c(X)$ since each forward edge e in X corresponds to cycle edge $D(e)$ which, therefore, is of length $l(D(e)) = c(e)$, while each backward edge e in X corresponds to a back edge (crossing e from right to left) of length zero. Cut-cycles with opposite orientation are called *reverse* cut-cycles. We summarize these observations in the following lemma, which is the analog for the directed case of a previous result [16]:

LEMMA 2. *The length of a minimum length forward cut-cycle in D is equal to the capacity of a minimum capacity cut in N.*

A cut cycle that is of minimum length subject to the constraint that it contain a specified vertex v is called a *v-minimum cut cycle.*

We now establish a convenient characterization of cycles which are cut-cycles. (See Figure 3.) A *$\mu(s, t)$-path*, *μ-path*, or μ for short, is a path of minimum

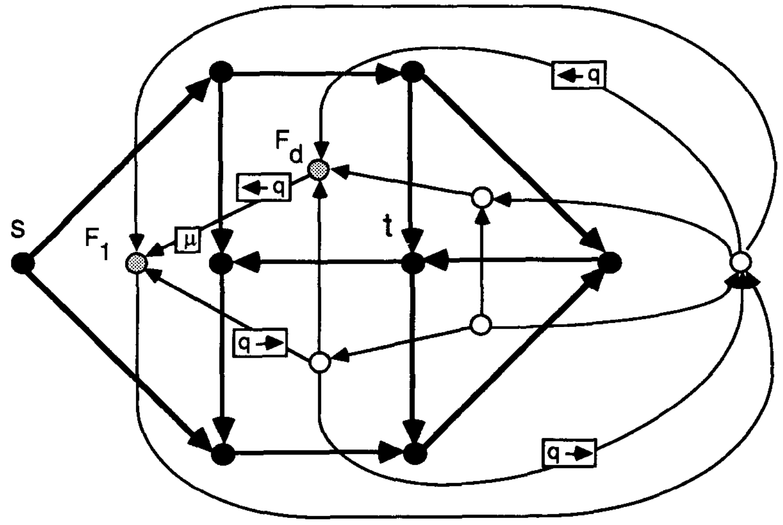

FIG. 3. Dual D, embedded with N, showing a μ-path with two vertices, and one of several possible forward cut-cycles q. There is one back edge of zero length in q. (Zero-length edges are not shown explicitly.)

edge-length in D from any vertex of D adjoining s to any vertex of D adjoining t. A vertex F of D is said to *adjoin* a vertex v of N if v is on the perimeter of the face in N corresponding to F. Let μ traverse the vertices $F_1, F_2, \ldots, F_d$ (indexed without loss of generality) in D. Edges not on μ but with endpoints on μ are said to be incident either on the left or the right depending on whether they are incident on the specified side when encountered while traversing μ from F_1 to F_d.

Let a path μ be defined as above, and consider the edges incident on μ on the left. Of these edges, those entering μ comprise the set IN and those leaving μ comprise the set OUT. Let p be any path in D. We define

$$wrap(p) = |p \cap IN| - |p \cap OUT|.$$

The function *wrap* is additive over concatenation of paths.

LEMMA 3. *If p is a simple cycle, then $wrap(p) \in \{-1, 0, 1\}$. Furthermore, $wrap(p) = 1$ if and only if p is a forward cut-cycle, and $wrap(p) = -1$ if and only if p is a reverse cut-cycle.*

PROOF. For the purposes of this proof, let μ be extended to additional dual vertices F_s, located coincident in the plane with primal vertex s, and F_t, located coincident with primal vertex t, so that $\mu = (F_s = F_0, F_1, \ldots, F_d, F_{d+1} = F_t)$. Thus p has the form $q_1 p_1 q_2 p_2 \cdots q_j p_j$ where, when there is a vertex common with μ, each subpath p_i begins and ends in a μ-vertex but contains no internal vertex from μ.

Each subpath $q_i = F_j, \ldots, F_k$ (where $j = k$ in the trivial case) contributes a value from $\{-1, 0, 1\}$ to the value of $wrap(p)$ depending on whether p crosses μ (contribution is -1 or 1 depending on direction) or enters and leaves on the same side (contribution is 0) at the ends of q_i. The contributions of the q_i computed in this way sum to $wrap(p)$. A subpath q_i that makes a nonzero contribution is called a *crossing*, and its vertices are *crossing vertices*, *forward* when the contribution is 1, and *backward* when the contribution is -1. Figure 4 illustrates the contribution of crossings to the function *wrap*.

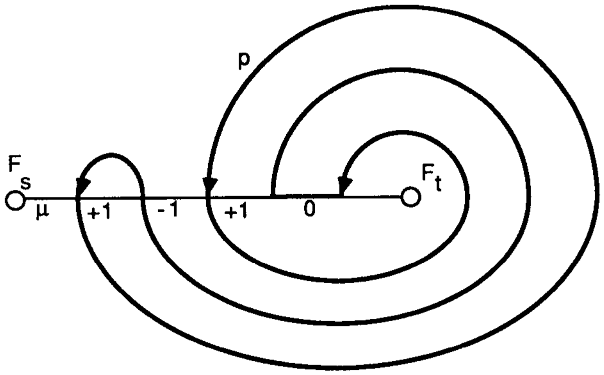

FIG. 4. Path μ and forward cut-cycle p in D showing increments to wrap(p) at crossings of μ. (Not all vertices are shown.) The path μ can be shortest from F_s to F_t, and p can be of minimum length in the direction shown.

Taking the cycle p as a Jordan curve and μ as a line segment, it follows from the fact that a Jordan curve partitions the plane into two regions that these crossings must alternate in direction when they are taken in the order in which they are encountered by traversing μ from F_s to F_t. Furthermore, dual vertices F_s and F_t either lie in the same region of the plane relative to p or thev do not. When F_s and F_t lie in the same region, then μ crosses p an even number of times. In this case, p is not a cut-cycle and $wrap(p) = 0$. When F_s and F_t do not lie in the same region, then μ crosses p an odd number of times; p is a cut-cycle, and $wrap(p) \in \{-1, 1\}$.

To distinguish the cases of forward and reverse cut-cycles, we notice that the lemma, if true, must hold independent of how μ is defined provided only that it begin adjoining s and end adjoining t. Therefore, μ could be chosen, without regard to its edge-length, so that there is exactly one subpath in common with p and therefore one crossing. This one crossing contributes 1 to $wrap(p)$ exactly when the crossing is forward, in which case p is a forward cut-cycle. □

It would be convenient if μ could be chosen in advance so that $wrap(p)$ could be bounded between -1 and $+1$ or, alternatively, if the *wrap* of cut-cycles could be reduced by replacing subpaths in a cycle with subpaths of μ. We do not know how to choose a good μ efficiently in advance, nor do we know how to make μ shortest in both directions so that the *wrap* of cut-cycles could be reduced by a replacement such as the one just suggested. For example, Figure 4 shows a case where a straightforward reduction is not possible under the assumption that μ is shortest from F_s to F_t.

Consequently, in order to keep track of the *wrap* of paths, we construct a new network from multiple copies of a closely related plane network D^i in a bounded region of the plane (which it is convenient to view as a rectangle) for some arbitrary index i. The construction of D^i, illustrated in Figure 5, begins with an embedding of D on the sphere. A μ-path $\mu = (F_1, \ldots, F_d)$ is identified in D. The sphere is then pierced at s and t, yielding an embedding of D on the cylinder which is then cut along μ, giving two copies of μ, one on each of two opposite sides of the resulting rectangle. Exactly one copy of μ will have edges from the sets IN and OUT incident on it. The μ-vertices $\{F_j\}$ in this copy are relabeled $\{F_j^{i+1}\}$. The remaining vertices $\{v\}$ of D, including the other μ-vertices $\{F_j\}$, are relabeled $\{v^i\}$. This construction gives the plane network D^i. Every path $p' = (F_j^i, \ldots, F_l^{i+1})$ in

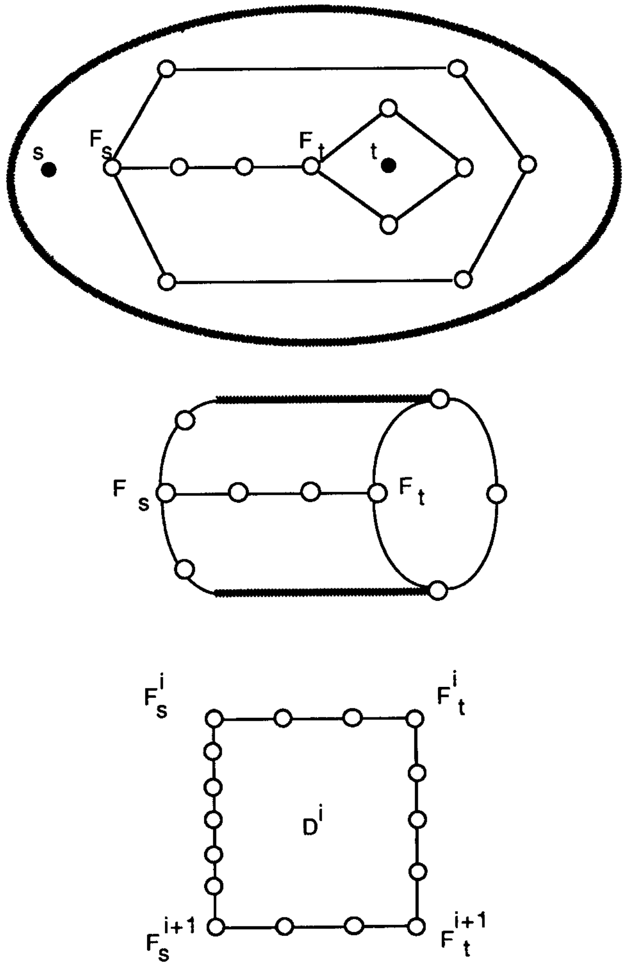

FIG. 5. Construction of D^i from D.

D^i maps to a unique path $p = (F_j, \ldots, F_l)$ in D. For any such p, $wrap(p) = 0$. Provision will now be made for paths that cross μ.

Let D^* be an infinite network formed by concatenating the members of $\{D^i \mid i = 0, 1, -1, 2, -2, \ldots\}$ so that vertices F_j^i in D^{i-1} and D^i are identified. See Figure 6. This construction may be denoted as $\bigcup_{\text{all } i} D^i$. Under the convention that the first vertex in a path p in D maps to a vertex in D^0, there is a one-to-one correspondence between paths $p = (F_j, \ldots, F_m)$ in D and $p^* = (F_j^0, \ldots, F_m^{wrap(p)})$. In particular, if we let $h(p^*)$ be the extension to paths of the homomorphism $h(u^i) = u$ for vertices u^i in D^* and u in D and any superscript index i, then $h(p^*) = p$.

As will be described shortly, our algorithm constructs a subgraph of D^*. The utility of D^* is that it forbids a class of paths p with $wrap(p) = 1$ that are not simple and contain no subgraph that is a forward cut-cycle, yet contain a subgraph that is a backward cut-cycle of zero length. Any flow network with no paths from t to s produces backward cut-cycles of zero length in D. Figure 7 illustrates such a nonsimple cycle that a dynamic program, parameterized with the values of $wrap$, would accept when run on D. As Figure 7 illustrates, its image in D^* is disconnected.

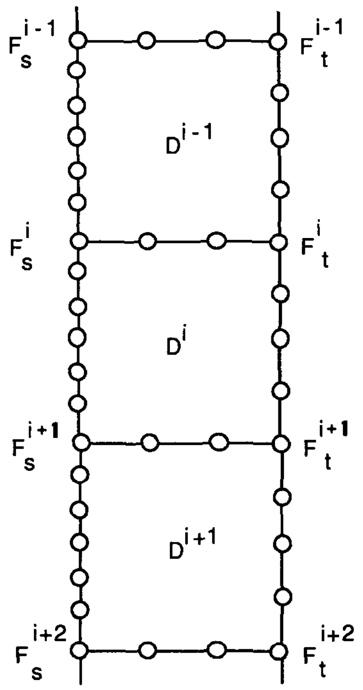

FIG. 6. Construction of D^* by concatenation of networks D^i.

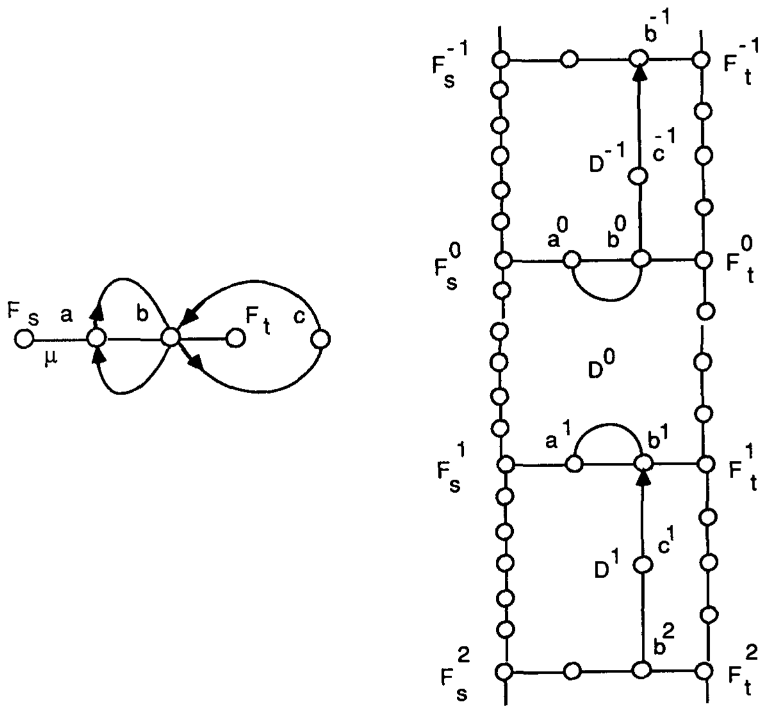

FIG. 7. If $l(b, c, b) = 0$, path $p = (a, b, c, b, a)$ may erroneously appear to be a forward cut-cycle of minimum length in D. However, the image of p in D^* is disconnected and will not be constructed by our algorithm.

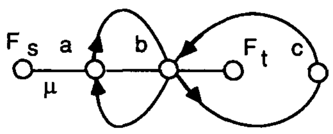

FIG. 8. Path $p = (a, b, c, b, a)$ is not a forward cut-cycle and may be shorter than any forward cut-cycle containing vertex a. The path p, however, contains a forward cut-cycle of no greater length than can be found at b.

Let $\Psi_l(n)$ be any set of simple paths $(F_l^0, \ldots, F_l^1)$ in D^* that contains no path with an internal vertex from the set $\{F_j^i \mid 1 \le j \le l$ and all $i\}$ and, subject to this restriction on internal vertices, contains (but is not necessarily restricted to) all simple paths $(F_l^0, \ldots, F_l^1)$ in D^* with n or fewer vertices. (A set $\Psi_l(n)$ may be described alternatively in terms of $D^* - \{F_j^i \mid 1 \le j < l$ or $(j = l$ and $i \ne 0, 1)\}$.) Our algorithm will require paths p, from some set $\Psi_l(n)$, for which $h(p)$ is simple.

LEMMA 4. *Given a fixed l, let p be a path in any $\Psi_l(n)$ that is shortest subject to the constraint that $h(p)$ is simple. Then $h(p)$ is an F_l-minimum cut-cycle in D.*

PROOF. For any $\Psi_l(n)$, let p' be any path in $\Psi_l(n)$ for which $h(p')$ is simple. Thus $wrap(h(p')) = 1$ and, by Lemma 3, $h(p')$ is a forward cut-cycle.

Since every edge in D is the image of edges in D^*, it may be shown by induction on the subpaths of q that, for fixed l, any forward cut-cycle $q = (F_l, \ldots, F_l)$ in D is the image of some path p that is in every $\Psi_l(n)$. It follows that if p is of minimum length in any $\Psi_l(n)$, subject to the restriction that $h(p)$ is simple, then $h(p)$ is of minimum length among all forward cut-cycles with minimum μ-vertex F_l. □

The preceding lemma does not as yet yield our result because, while it is relatively easy to find shortest paths over some $\Psi_l(n)$, the requirement that the image $h(p)$ must be simple is difficult to enforce. If this condition is not enforced, then a shortest path we find at F_l may be shorter than an F_l-minimum cut-cycle. (See Figure 8 for an example.) We attack this problem globally over all $l = 1, \ldots, d$ by means of the next lemma.

For any closed path q denoted as $(v_1, v_2, \ldots, v_m)$, $v_m = v_1$, we define an *initial cycle* q_I to be any closed path $(v_1, \ldots, v_m)$ that is simple and is a subgraph of q. For example, one initial cycle is the path generated by a traversal of q from vertex v_1 where, at each ith path vertex v_i at which q intersects itself, the edge $(v_j, v_{j+1}) \in q$ is taken, where j is the largest index for which $v_i = v_j$ in the vertex set of the graph. It is clear that the closed path $q_I = (v_1, \ldots, v_m)$, $v_m = v_1$, so generated is simple. Thus at least one initial cycle q_I exists for any q. We note that the definition of initial cycle depends on the initial vertex in the denotation of the path and, consequently, different initial cycles of the same path may have different start vertices. However, the initial cycles of a path in any $\Psi_l(n)$ denoted as such all have the same start vertex.

LEMMA 5. *For fixed l and any $\Psi_l(n)$, let p be a shortest path in $\Psi_l(n)$. If $h(p)$ is not simple, then one of two cases occurs:*

(i) all initial cycles q_I of $h(p)$ are forward cut-cycles, or
(ii) there exists an index $h > l$ and a path p' that belongs to every $\Psi_h(n)$ and that is no longer than p.

PROOF. When an initial cycle q_I of $h(p)$ is not a forward cut-cycle, then by Lemma 3 it has wrap equal to 0 or -1. It follows by additivity of *wrap* over concatenation of paths that the sum of the wraps of the paths in $h(p) - q_I$, which is a collection of directed closed paths, is greater than or equal to 1. Therefore at least one closed path in the collection has a strictly positive wrap. Repeated application of this construction to such a closed path, choosing any initial cycle at each step, will eventually yield a closed path q_1' with wrap equal to 1. Since all edge lengths are nonnegative and a simple path can have no more than n vertices, it follows that $h(q_1')$ satisfies the conditions set forth in (ii). □

LEMMA 6. *For each $l = 1, \ldots, d$, let p_l be shortest in some $\Psi_l(n)$, let Q be any set containing one arbitrary initial cycle from each path in the set $\{p_l \mid l = 1, \ldots, d\}$, and let $Q_C \subseteq Q$ be the forward cut-cycles in Q. Any cycle of minimum length in Q_C corresponds to a minimum capacity cut in N.*

PROOF. Since all edge weights are nonnegative, subpaths are never longer than the paths in which they are contained. Thus Lemmas 4 and 5 guarantee that the minimum cycle is a minimum forward cut-cycle for D, which in turn by Lemmas 2 and 3 gives our result. □

Lemma 6 establishes the correctness of the following algorithm:

```
Algorithm DIRECTED-MIN-CUT
1. Construct $D^0$.
2. For each $l = 1, \ldots, d$, find a shortest path in some $\Psi_l(n)$.
3. In each path obtained in Step 2, find an initial cycle. Find the shortest cut-cycle among
   the subset $Q_C$ of these initial cycles that happen to be forward cut-cycles.
```

THEOREM 1. *Algorithm DIRECTED-MIN-CUT can be implemented to run in $O((\log n)^2)$ parallel time using $O(n^6)$ processors, and in $O((\log n)^3)$ time using $O(n^4)$ processors.*

PROOF. Given a planar embedding of D, it is possible to construct D^0 in $O(\log n)$ parallel time using $O(n)$ processors.

We proceed as follows with Step 2: For a given l, let us take D_l^* to be $D^* - \{F_j^i \mid j = 1, \ldots, l \text{ and all } i\}$. Let Δ_l be the matrix of shortest distances in D_l^* between vertices in the set $\mathbf{F} = \{F_j^{-1}, F_j^0, F_j^1, F_j^2 \mid j = l + 1, \ldots, d\}$, let A_l be a $1 \times 4(d - l)$ matrix of shortest distances from F_l^0 to $\mathbf{F}$ in $D^{-1} \cup D^0 \cup D^1$, and let B_l be a $4(d - l) \times 1$ matrix of shortest distances from $\mathbf{F}$ to F_l^1 in $D^{-1} \cup D^0 \cup D^1$. In $O((\log n)^2)$ time and $O(n^3)$ processors construct the distance matrices A_l and B_l and evaluate the "plus-min" matrix product $A_l\ \Delta_l\ B_l$ which yields (by the standard bookkeeping techniques for dynamic programming) a shortest path $(F_l^0, \ldots, F_l^1)$ with no internal vertices from the set $\{F_j^i \mid 1 \le j \le l \text{ and all } i\}$. We now show how to obtain Δ_l from D^0.

Let M_l be a matrix of numbers on $S \times S$, where $S = \{F_j^i \mid j = l + 1, \ldots, d, \text{ and all } i\}$, a subset of the vertices of D_l^*. Let S be ordered so that F_j^i precedes F_k^i when $j < k$, and F_j^i precedes F_k^m when $i < m$. With this ordering, square submatrices M_l^{ij} of M_l are induced by $\{F_{l+1}^i, \ldots, F_d^i\} \times \{F_{l+1}^j, \ldots, F_d^j\}$. These $(d - l) \times (d - l)$ submatrices are *blocks* of M_l, and we may represent M_l as a *block matrix*, the elements of which are not numbers but $(d - l) \times (d - l)$ matrices of numbers. A block M_l^{ij} is on the *main block diagonal* when $i = j$, and is on the same *minor* block diagonal whenever $i - j = c$, for a fixed $c \neq 0$. We determine the elements of M_l by assigning to blocks $M_l^{i,(i-1)}$, M_l^{ii}, and $M_l^{i,(i+1)}$ the shortest distances from $\{F_{l+1}^i, \ldots, F_d^i\}$ to $\{F_{l+1}^{i-1}, \ldots, F_d^{i-1}\}$ in D^i, to $\{F_{l+1}^i, \ldots, F_d^i\}$ in $D^i \cup D^{i+1}$, and to $\{F_{l+1}^{i+1}, \ldots, F_d^{i+1}\}$ in D^{i+1}, respectively (which are over paths with at most n vertices). Any M_l^{ij} not so assigned for some i and for some j is set to ∞. The resulting matrix M_l does not represent the edges in D_l^*, but rather the length of certain paths in D_l^*. We observe that M_l is block tridiagonal, with identical blocks throughout each of the diagonals. Since $d - l = O(n)$, the results of Section 5 therefore show that we may obtain Δ_l in $O((\log n)^2)$ time using $O(n^5)$ processors or in $O((\log n)^3)$ time using $O(n^3)$ processors. For all $l = 1, \ldots, n$, computations of Δ_l, A_l, B_l, and then of $A_l\, \Delta_l\, B_l$, are done in parallel, yielding one shortest path $(F_l^0, \ldots, F_l^l)$ for each l. Each of these paths is on M_l and contains $O(n^2)$ vertices. As indicated previously, we may therefore apply the homomorphism h to each path and then apply the *lastmate* construction (defined below), yielding paths of vertex-length $O(n)$. Then these paths can be expanded by replacing the edges of M_l with the shortest paths on sets of μ-vertices from which M_l was obtained. This gives paths of vertex-length $O(n^2)$ from which one more application of the *lastmate* construction yields an initial cycle for the true shortest paths in D_l^*.

To implement Step 3, it is necessary in each of d paths to find an initial cycle and test if it is a forward cut-cycle. This can be done in $O((\log n)^2)$ time and $O(n^4)$ processors, as we now describe.

Let $p = (v_1, v_2, \ldots, v_m)$ be a closed path where $v_1 = v_m \neq v_\alpha$ for $\alpha = 2, \ldots, m - 1$. Let the *last mate* of a vertex v_α in p be the vertex defined as $lastmate(v_1) = v_2$ and, for $\alpha = 2, \ldots, m - 1$, $lastmate(v_\alpha) = v_\beta$ where $\beta = \max\{\gamma + 1 \mid v_\alpha = v_\gamma\}$. Given p, last mates for all vertices v_α, $\alpha < m$, in p can be found in $O(\log n)$ parallel time and $O(mn)$ processors, when there are no more than n distinct vertices in p (which is the case in our application). The graph induced on $\{v_1, \ldots, v_m\}$ by directed edges $(v_\alpha, lastmate(v_\alpha))$ has exactly one path p_1 of the form $(v_1, \ldots, v_m)$, and this path p_1 is an initial cycle of p. This initial cycle p_1 may be found in an additional $O(\log n)$ parallel time using $O(m)$ processors.

To find one initial cycle for each of d paths p_l in $O(\log n)$ parallel time using $O(n^4)$ processors we must therefore guarantee that $m = O(n^2)$. As will be seen, we may only be able to guarantee in our application that $m = O(n^3)$, but when $m = \omega(n^2)$ we are able to apply the *lastmate* reduction twice, first to paths with $O(n^2)$ vertices, reducing the number of vertices on these paths to $O(n)$, then expanding these paths again to $O(n^2)$ vertices, and applying the *lastmate* reduction once more. Any initial cycle of any closed subgraph $(v_1, \ldots, v_m)$ of some closed path $p = (v_1, \ldots, v_m)$ is also an initial cycle of p. □

4. *Shortest Paths*

Shortest paths between all pairs of vertices in a network, directed or undirected, can be computed in $O((\log n)^2)$ parallel time by straightforward parallelization of the $O(n^3 \log n)$ serial-time shortest path algorithm that is based on repeated *plus-min* multiplication of the edge-length matrix. To do so, initialize the matrix $K^{(1)}$ to zero, and $U^{(1)}$ to the edge-length matrix, and compute as follows:

```
for $k = 1$ to $p = \lceil \log(n-1) \rceil$ do
    $U_{ij}^{(2^k)} = \min_m \{U_{im}^{(2^{k-1})} + U_{mj}^{(2^{k-1})}\}$
    if $U_{ij}^{(2^k)} < U_{ij}^{(2^{k-1})}$ then
        set $K_{ij}^{(2^k)} = m$ for $m$ satisfying
            $U_{ij}^{(2^k)} = U_{im}^{(2^{k-1})} + U_{mj}^{(2^{k-1})}$
    else $K_{ij}^{(2^k)} = K_{ij}^{(2^{k-1})}$
endfor
```

There is a cycle of negative length and containing no more than 2^l edges if and only if, for some i, $U_{ii}^{(2^l)} < 0$. When there are no cycles of negative length, $U_{ij}^{(2^l)}$ may be interpreted as the length of a shortest path from i to j such that the path contains no more than 2^l edges. Thus the construction below can be used to identify all the edges in a path from i to j.

```
**IDENTIFY** $(i, j, K^{(2^p)})$
    Let $m = K_{ij}^{(2^p)}$
    **if** $m = 0$ **then** edge $(i, j)$ is in the shortest path
    **else IDENTIFY** $(i, m, K^{(2^p)})$ **and IDENTIFY** $(m, j, K^{(2^p)})$
**END_IDENTIFY**
```

Shortest paths between all pairs of vertices in a general directed graph with n vertices or, when it occurs, the existence of a cycle of negative length can be found in $O((\log n)^2)$ parallel time using $O(n^3)$ processors by means of the above procedures.

It is convenient to observe that plus–min multiplication may be extended to block matrices, matrices in which each element is a square matrix, say, $m \times m$. In the extension, *plus* is plus–min multiplication on the blocks, and *min* is elementwise minimization. When two matrices that are arguments for plus-min multiplication can be viewed as block matrices, the product is the same whether it is found by plus-min multiplication on the matrices or by extended plus–min multiplication on the block matrices.

Now, let a shortest path problem be posed as a nonnegative edge-length matrix M of infinite dimensions that is block tridiagonal on $m \times m$ blocks, and all blocks within any single block diagonal are identical. The problem is to find some $cm \times cm$ block, for positive integer c, in the distance matrix that is the k-fold plus–min multiple of M for some $k \geq m - 1$. For example, solving this problem for $c = 4$ and $m = n - l$ on M_l for the network D_l^* of the previous section will yield the matrix Δ_l.

To compute M^{2h} by squaring M^h, h a nonnegative power of two, we proceed as follows: It may be shown by induction that M^h has $2h + 1$ distinct blocks that are not ∞. Therefore, producing the $4h + 1$ blocks sufficient to describe M^{2h} can be done by $4h + 1$ inner product computations, each one requiring m^2 inner products over vectors of length $m(2h + 1)$. Producing M^{2h} from M^h, therefore, takes $O(\log m)$ parallel time and uses $O(hm^2mh) = O(m^5)$ processors. Since $\lceil(\log(m - 1)\rceil$ such multiplications suffice to solve the problem, the overall parallel running time is $O((\log m)^2)$ with $O(m^5)$ processors. We observe that paths represented in the k-fold product have at most $k + 1$ vertices.

A second approach to this problem produces a $cm \times cm$ block on the main diagonal of a matrix of distances, taking $O((\log m)^3)$ parallel time but using only $O(m^3)$ processors. In general, the block found is an elementwise lower bound to the same block were it to be found by the first approach. But each element is a path length, and we may therefore interpret the result as a minimization over some $\Psi_l(n)$ (even though some of the paths have $\omega(n)$ vertices). Thus Lemma 6 allows us to extract our result as before.

For this second method we consider finite submatrices of M, where the matrix containing blocks with indices (i, j) for $i_1 \leq i \leq i_2$ and $j_1 \leq j \leq j_2$ is denoted $M(i_1 : i_2, j_1 : j_2)$. (If $i = i_1 = i_2$ or $j = j_1 = j_2$, we condense the notation, for example, writing $M(i, j)$ when all these equalities hold.) Our consideration of finite submatrices is motivated by the following result:

LEMMA 7. *For any positive integer h,*

$$M^h(i, j) = M(i_a : i_b, j_a : j_b)^h(i, j),$$

for $i_a = j_a \leq \min\{i, j\} - h + 1$ and $i_b = j_b \geq \max\{i, j\} + h - 1$.

PROOF. Since the lemma holds immediately for $h = 1$, we proceed by induction. For any matrix X, $X^h = XX^{h-1} = X^{h-1}X$ for $h \geq 1$. Let X be block tridiagonal. Then

$$\begin{aligned}
&(XX^{h-1})(i,j)\\
&\quad = \min\{X(i, i-1)X^{h-1}(i-1,j), X(i, i)X^{h-1}(i,j), X(i, i+1)X^{h-1}(i+1,j)\}\\
&\quad = (X^{h-1}X)(i,j)\\
&\quad = \min\{X^{h-1}(i,j-1)X(j-1,j), X^{h-1}(i,j)X(j,j), X^{h-1}(i,j+1)X(j+1,j)\}.
\end{aligned}$$

By hypothesis,

$$M^{h-1}(r, s) = M(r_a : r_b, s_a : s_b)^{h-1}(r, s),$$

for all $r_a = s_a \leq \min\{r, s\} - (h-1) + 1$, $r_b = s_b \leq \max\{r, s\} + (h-1) - 1$. Thus

$$M^h(r, s) = M(r_a^* : r_b^*, s_a^* : s_b^*)^h(r, s)$$

where

$$\begin{aligned}
r_a^* = s_a^* &\leq \min\{i, j\} - (h-1) + 1 - 1 = \min\{i, j\} - h + 1,\\
r_b^* = s_b^* &\geq \max\{i, j\} + (h-1) - 1 + 1 = \max\{i, j\} + h - 1,
\end{aligned}$$

for $(r, s) \in \{(i-1, j), (i, j), (i+1, j), (i, j-1), (i, j+1)\}$, which gives our result. □

Our first objective is to compute $M^k(i, i)$ for k a power of two for which $k \geq m - 1$. To this end, for any h a positive power of two, let $a(h) = i - h + 1$, $b(h) = i + h - 1$ (these values are compatible with the definitions of i_a and i_b of the preceding lemma) and let $t(h)$ be the least power of two for which $t(h) \geq (2h + 1)m$. Define the 3×3 block matrix $Q_2(1:3, 1:3) = M(i-1 : i+1, i-1 : i+1)$. Then define Q_{2h} as follows:

$$\begin{aligned}
Q_{2h}(1, 1) &= Q_h^{t(h)}(1, 1),\\
Q_{2h}(1, 2) &= Q_h^{t(h)}(1, 3),\\
Q_{2h}(1, 3) &= \infty,\\
Q_{2h}(2, 1) &= Q_h^{t(h)}(3, 1),\\
Q_{2h}(2, 2) &= \min\{Q_h^{t(h)}(3, 3), Q_h^{t(h)}(1, 1)\},\\
Q_{2h}(2, 3) &= Q_h^{t(h)}(1, 3),\\
Q_{2h}(3, 1) &= \infty,\\
Q_{2h}(3, 2) &= Q_h^{t(h)}(3, 1),\\
Q_{2h}(3, 3) &= Q_h^{t(h)}(3, 3),
\end{aligned}$$

where minimization over matrices is elementwise.

LEMMA 8. *For h a positive power of two and all $s(h) \leq t(h)$,*

$$\begin{aligned}
M(a(h) : b(h), a(h) : b(h))^{s(h)}(a(h), a(h)) &\geq Q_h^{t(h)}(1, 1),\\
M(a(h) : b(h), a(h) : b(h))^{s(h)}(b(h), a(h)) &\geq Q_h^{t(h)}(3, 1),\\
M(a(h) : b(h), a(h) : b(h))^{s(h)}(a(h), b(h)) &\geq Q_h^{t(h)}(1, 3),\\
M(a(h) : b(h), a(h) : b(h))^{s(h)}(b(h), b(h)) &\geq Q_h^{t(h)}(3, 3),\\
M(a(h) : b(h), a(h) : b(h))^{s(h)}(i, i) &\geq Q_h^{t(h)}(2, 2),
\end{aligned}$$

and furthermore every entry in $Q_h^{t(h)}$ represents a path length in the graph for which $M(a(h) : b(h), a(h) : b(h))$ is the edge-length matrix.

PROOF. Since $M(a(2) : b(2), a(2) : b(2)) = Q_2$, the lemma is true for $h = 2$. Assume the lemma holds for some $h = h_0 \geq 2$. By the properties of plus-min multiplication we may view the matrix $M(a(h_0):b(h_0), a(h_0):b(h_0))^{s(h_0)}$ as containing shortest distances among all paths with no more than $s(h_0)$ edges in the directed graph for which $M(a(h_0):b(h_0), a(h_0):b(h_0))$ is the edge-length matrix. By hypothesis we may assert that the relevant submatrices in $Q_{h_0}^{t(h_0)}$ contain shortest distances among a set of paths that contains all paths with $s(h_0)$ or fewer edges.

The key observation for the induction step is that $M(a(2h_0):b(2h_0), a(2h_0):b(2h_0))$ has no more than $t(2h_0)$ vertices, and therefore every shortest distance is over a path with no more than $t(2h_0) - 1$ edges. Thus the $t(2h_0)$-fold product of Q_{2h_0} must also satisfy the lemma, as a case analysis on each $m \times m$ block shows. We omit further details. □

It follows from the above that we may generate the sequence $Q_2, Q_4, Q_8, \dots, Q_k, Q_k^{t(k)}$ in $O((\log m)^3)$ parallel time using $O(m^3)$ processors. Lemmas 7 and 8 then allow us to bound $M^k(i, i)$ elementwise from below by $Q_k^{t(k)}(2, 2)$.

Our computation of $M^k(i:i + c - 1, i:i + c - 1)$ is completed as follows. Define the infinite block tridiagonal matrix R as follows for all i:

$$\begin{aligned} R(i, i) &= Q_k^{t(k)}(2, 2) \le M^k(i, i), \\ R(i, i + 1) &= M(i, i + 1), \\ R(i + 1, i) &= M(i + 1, i), \\ R(i, j) &= \infty \quad \text{for} \quad j > i + 1 \quad \text{or} \quad j < i - 1. \end{aligned}$$

LEMMA 9

$$M^k(i:i + c - 1, i:i + c - 1) \ge R(i:i + c - 1, i:i + c - 1)^k,$$

and furthermore every entry in $R(i:i + c - 1, i:i + c - 1)^k$ *is a path length in M.*

PROOF. The k-fold multiple of $R(i:i + c - 1, i:i + c - 1)$ represents shortest paths in a set of paths that contains all paths of vertex length $k + 1$, which is sufficient for the result. □

5. *Computing Flows and Maximum Flows in Planar Networks*

In the preceding sections we have shown how to obtain the value of a maximum flow in a planar directed network. In this section we consider how to obtain a feasible flow f in a planar directed network, given the flow value $val = val(f)$. Hassin and Johnson [10] have given a reduction of the flow problem on undirected planar networks to a shortest path problem. The reduction fails on directed networks because the path $\mu = (F_1, \dots, F_d)$, which in this case is chosen to be shortest with respect to the edge-length function l, is not generally shortest in both directions in a directed network. We therefore have the following vertex-labeling algorithm, which may be viewed as a generalization of the one in the paper cited [10]. The algorithm is applicable to undirected as well as directed flow networks.

Let $N = (V, E, c, s, t)$ be a planar flow network, let D be its dual, and let μ be a simple path from the dual face containing s to the dual face containing t as before. It is not in fact necessary for μ to be minimal in any respect. Given a positive flow value val, we augment D as follows to give $aug_{val}(D)$ (see Figure 9):

(i) Construct D^0 as in Section 3.
(ii) For each $i = 1, \dots, d$, construct two new edges, (F_i^0, F_i^1) of length $-val$ and (F_i^1, F_i^0) of length val.

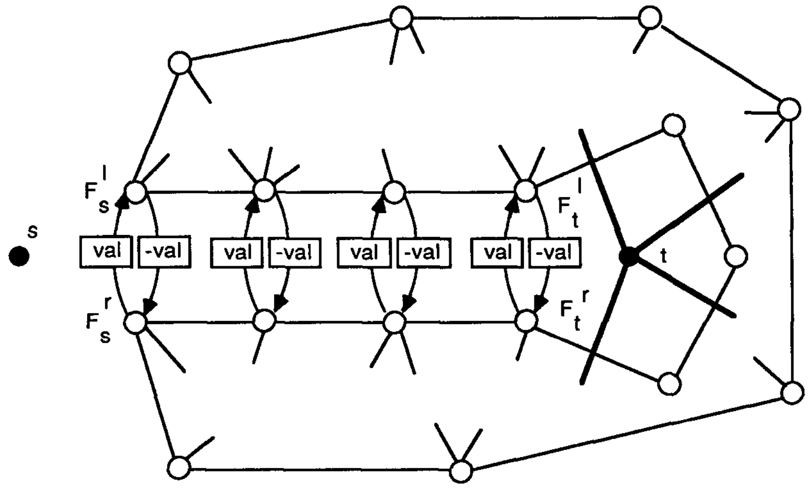

FIG. 9. The construction of $aug_{val}(D)$. Edges which are shown without direction represent edge-back edge pairs. Not all vertices are shown.

Define labelings $\lambda(\cdot)$ on dual vertices and $\lambda(\cdot, \cdot)$ on ordered pairs of dual vertices as any values from $\mathbf{R}$ that satisfy

(i) $\lambda(F_i^l) = \lambda(F_i^o) - val$,

(ii) $\lambda(u, w) = \begin{cases} \lambda(w) - \lambda(u) & \text{for } (u, w) \text{ a dual edge,} \\ 0 & \text{otherwise,} \end{cases}$

(iii) $0 \leq \lambda(u, w) \leq l(u, w)$ for $(u, w) \in \phi \times \phi$.

As will be seen, whether a labeling λ exists for $aug_{val}(D)$ depends on val. Since (i) and (ii) imply that $\lambda(F_i^o, F_{i+1}^o) = \lambda(F_i^l, F_{i+1}^l)$ for all $i = 1, \ldots, d-1$, it follows that λ, both as a vertex and an edge labeling, may be defined on D as a restriction of λ on $aug_{val}(D)$ whenever λ exists on $aug_{val}(D)$. Given λ on D, we define $\lambda(V \times V)$ as $\lambda(e) = \lambda(D(e))$, for $e \in E$ and $\lambda(V \times V - E) = 0$.

LEMMA 10. *Whenever λ exists on $aug_{val}(D)$, $\lambda(V \times V)$ is a flow of value val in N.*

PROOF. Let λ be given on $aug_{val}(D)$ for some $val \geq 0$. It follows from (iii) that $0 \leq \lambda(u, w) \leq c(u, w)$ for all $(u, w) \in V \times V$. For any $z \in V$ let $(e_1, \ldots, e_m)$ be the edges in E that are incident on z, and let $\{e_i\}_{i \in I}$ be the edges in $(e_1, \ldots, e_m)$ directed into z. Then

$$\Sigma(z) \equiv \sum_{i \in I} \lambda(e_i) - \sum_{i \in \{1, \ldots, m\} - I} \lambda(e_i) = \sum_{j=1}^{m} \lambda(u_j, u_{j+1}) \equiv \Sigma(\beta)$$

where $\beta = (u_1, \ldots, u_{m+1} = u_1)$ is a face of D (omitting the back edges in D from consideration in identifying faces). By construction of λ, we know that

$$\Sigma(\alpha) \equiv \sum_{j=1}^{k} \lambda(u_j, u_{j+1}) = \sum_{j=1}^{k} \lambda(u_j) - \sum_{j=1}^{k} \lambda(u_j) = 0$$

for every face $\alpha = (u_1, \ldots, u_{k+1} = u_1)$ in $aug_{val}(D)$. When $z \neq s, t$, each face α is identical to some face β in D, and $\Sigma(z) = 0$ as required for a flow. When $z = t$,

α corresponds to a face β, but α has an additional edge (u_{k+1}, u_1) for which $\lambda(u_{k+1}, u_1) = \lambda(u_1) - \lambda(u_{k+1}) = -val$. Therefore $\Sigma(t) = \Sigma(\beta) = \Sigma(\alpha) - (-val) = val$ in this case, as required for λ to be a flow of value *val* in *N*. □

LEMMA 11. *Let $\sigma(w)$ be the shortest distance in $aug_{val}(D)$ from some fixed vertex σ in $aug_{val}(D)$ to each vertex w in $aug_{val}(D)$. If $\sigma(w)$ is defined for all w, then $\sigma(w)$ satisfies the definition for $\lambda(w)$ for all w in $aug_{val}(D)$.*

THEOREM 2. *Let a network N and a value $val \geq 0$ be given. There is an algorithm that runs in $O((\log n)^2)$ parallel time and uses $O(n^3)$ processors that*

(*i*) *constructs a flow of value val in N if val is feasible, or*
(*ii*) *reports val is infeasible if that is the case.*

PROOF. The dual network $aug_{val}(D)$ can be constructed from N in $O((\log n)^2)$ time using $O(n)$ processors, including the cost of embedding N in the plane. Finding shortest paths or, alternatively, detecting the existence of a negative-length cycle can be done in $O((\log n)^2)$ time with $O(n^3)$ processors. Since $aug_{val}(D)$ is strongly connected, it therefore is sufficient to prove that $\sigma(w)$ is defined for all w exactly when there is no negative-length cycle in $aug_{val}(D)$. The result then will follow from Lemmas 10 and 11.

Call the edges of length $-val$ and val added to D when constructing $aug_{val}(D)$ *new* edges. Let some cycle q in D have $k \geq 0$ new edges of length $-val$. It follows from Lemma 3 that, if q is a forward cut-cycle, $k \geq 1$ and q must have exactly $k - 1$ new edges of length *val*. Otherwise, q has at least k such edges. The result follows immediately for any *val* in this latter case. In the case where q is a forward cut-cycle, we know that $q - \{\text{new edges}\}$ must have length at least $val_{\max}$, the value of a maximum flow in N, and that at least one forward cut-cycle realizes this value exactly. Thus, since $k \geq 1$, $\sigma(w)$ is defined everywhere exactly when $val \leq val_{\max}$. □

6. *Computing a Maximum Flow in an Undirected Network*

Given a planar undirected flow network N, we construct a dual D as before, except the edges in D are undirected and, consequently, no back edges of length 0 are introduced. Let μ be a path from a vertex of D adjoining s to a vertex of D adjoining t that is of minimum length relative to the edge-length function l. Using μ defined in this way, we construct D^0 as in Section 3. In D^0 there are no edges incident on the left for any μ-vertex F_i^0 and no edges incident on the right for any μ-vertex F_i^1. This property serves to force the images of paths between F_i^0 and F_i^1 to have a wrap of 1. Such paths map to cut-cycles in N.

A lemma due to Reif justifies this construction.

LEMMA 12 [16]. *For any μ-vertex F_i in D, there exists a cut-cycle containing F_i of minimum length that has exactly one subpath in common with μ.*

PROOF. As a contrary assumption, let q be of minimum length among cut-cycles passing through F_i, let q have two or more nonconsecutive subpaths in common with μ, and let q minimize the number of such subpaths over all cut-cycles containing F_i. Thus there are two subpaths $(F_w, \ldots, F_x)$ and $(F_y, \ldots, F_z)$ where $w \leq x < y \leq z$ and $F_l \notin q$ for all $x < l < y$. Let $p_\mu = (F_x, \ldots, F_y)$ be a subpath of μ. The subgraph $q \cup p_\mu$ divides the plane into three regions, one of which contains s and not t. Let the boundary of this region be q'. But $l(q') \leq l(q)$, and therefore q' is an F_j-minimum cut-cycle with fewer maximal subpaths in common with μ, a contradiction of the assumed minimality of q. □

COROLLARY. *Let p be a minimum length simple path between F_i^0 and F_i^1 in D^0. Then the image of p is a cut-cycle in D of minimum length among all cut-cycles that contain F_i.*

One shortest path between all pairs of vertices of D^0 gives a set of $O(n)$ minimum cut-cycle lengths, one for each μ-vertex F_i, $i = 1, \ldots, d$. Minimizing over this set gives a globally minimum cut-cycle length.

THEOREM 3. *The capacity of a minimum cut in a planar undirected network can be found in $O((\log n)^2)$ time using $O(n^3)$ processors.*

The same construction as in Section 5 can be used to find a flow function itself.

ACKNOWLEDGMENTS. I am grateful to Shankar Venkatesan for discussions and coauthorship of an early, conference version of this paper. By his own request, he is not an author of this final version. Thanks also are due to the referees for presentational suggestions which improved the paper.

REFERENCES

1. BONDY, J. A., AND MURTY, U. S. R. *Graph Theory with Applications.* Elsevier North-Holland, New York, 1977.
2. FORD, L. R., AND FULKERSON, D. R. Maximal flow through a network. *Can. J. Math. 8*, 3 (1956), 399–404.
3. FREDERICKSON, G. N. Shortest paths problems in planar graphs. In *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science.* IEEE, New York, 1983, pp. 242–247.
4. GALIL, Z. A new algorithm for the maximal flow problem. In *Proceedings of the 19th IEEE Symposium on Foundations of Computer Science.* IEEE, New York, 1978, pp. 231–245.
5. GOLDBERG, A. V., AND TARJAN, R. E. A new approach to the maximum flow problem. In *Proceedings of the 18th ACM Symposium on Theory of Computing* (Berkeley, Calif., May 28–30). ACM, New York, 1986, pp. 136–146.
6. GOLDSCHLAGER, L. M. A unified approach to models of synchronous parallel machines. In *Proceedings of the 10th ACM Symposium on Theory of Computing* (San Diego, Calif., May 1–3). ACM, New York, 1978, pp. 89–94.
7. GOLDSCHLAGER, L., SHAW, R., AND STAPLES, J. The maximum flow problem is log space complete for *P*. *Theor. Comput. Sci. 21*, 1 (Oct. 1982), 105–111.
8. HARARY, F. *Graph Theory.* Addison-Wesley, Reading, Mass., 1969.
9. HASSIN, R. Maximum flow in (s, t)-planar networks. *Inf. Process. Lett. 13*, 3 (Dec. 1981), 107.
10. HASSIN, R., AND JOHNSON, D. B. An $O(n \log^2 n)$ algorithm for maximum flow in undirected planar networks. *SIAM J. Comput. 14*, 3 (Aug. 1985), 612–624.
11. ITAI, A., AND SHILOACH, Y. Maximum flow in planar networks. *SIAM J. Comput. 8*, 2 (May 1979), 135–150.
12. JANIGA, L., AND KOUBEK, V. A note on finding minimum cuts in directed planar networks by parallel computation. *Inf. Process. Lett. 21* (1985), 75–78.
13. JOHNSON, D. B., AND VENKATESAN, S. Using divide and conquer to find flows in directed planar networks on $O(n^{3/2} \log n)$ time. In *Proceedings of the 20th Annual Allerton Conference on Communication, Control, and Computing.* (Oct.) Univ. of Illinois, Urbana-Champaign, Ill., 1982, pp. 898–905.
14. KLEIN, P. N., AND REIF, J. H. An efficient parallel algorithm for planarity. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science.* IEEE, New York, 1986, pp. 465–477.
15. LAWLER, E. *Combinatorial Optimization, Networks and Matroids.* Holt, Rinehart, and Winston, New York, 1976.
16. REIF, J. Minimum $(s - t)$-cut of a planar undirected network in $O(n \log^2 n)$ time. *SIAM J. Comput. 12*, 1 (1983), 71–81.

17. SHILOACH, Y., AND VISHKIN, U. An $O(n^2 \log n)$ parallel MAX-FLOW algorithm. *J. Algorithms 3*, 2 (June 1982), 128–146.
18. SLEATOR, D. An $O(nm \log n)$ algorithm for maximum network flow. Ph.D. dissertation, Computer Science Dept., Stanford Univ., Stanford, Calif., 1980.
19. SLEATOR, D., AND TARJAN, R. A data structure for dynamic trees. In *Proceedings of the 13th ACM Symposium on Theory of Computing* (Milwaukee, Wis., May 11–13). ACM, New York, 1981, pp. 114–122.

RECEIVED MAY 1984; REVISED AUGUST 1986, MARCH 1987; ACCEPTED APRIL 1987