JOURNAL OF ALGORITHMS **23,** 121–138 (1997)
ARTICLE NO. AL960831

Edge-Disjoint (s, t)-Paths in Undirected Planar Graphs in Linear Time*

Karsten Weihe[†]

Universität Konstanz, Fakultät für Mathematik und Informatik, Postfach 5560 D 188, 78434 Konstanz, Germany

Received December 6, 1994

We consider the following problem. Let $G = (V, E)$ be an undirected planar graph and let $s, t \in V$, $s \neq t$. The problem is to find a set of pairwise edge-disjoint paths in G, each connecting s with t, of maximum cardinality. In other words, the problem is to find a maximum unit flow from s to t. The fastest algorithm in the literature has running time $\mathcal{O}(|V| \log |V|)$. We give a linear time algorithm in this paper. © 1997 Academic Press

1. INTRODUCTION

Let $G = (V, E)$ be an undirected, planar graph embedded in the plane without crossings of edges. Moreover, let $s, t \in V$, $s \neq t$. The problem is to find a maximum number of simple paths connecting s with t such that no two paths have an edge of G in common (*edge-disjoint Menger problem*). This problem is of some interest in connection with VLSI design and in the layout of traffic and communication networks, since there the underlying networks are often undirected and planar. See [RLWW95] for a survey.

Note that this problem can be viewed as a special case of the undirected *maximum flow problem* when the underlying graph is planar and all capacities are equal to one. See [AMO89] for an introduction to network flows and [AMO93] for an extensive survey. In particular, Chap. 8.4 of [AMO93] introduces flows in planar networks.

* This work was supported by the Technische Universität Berlin under Grant FIP 1/3 and by the Deutsche Forschungsgemeinschaft under Grant Wa 654/10-2. A preliminary version has been published under the same title in "Proceedings of the Second European Symposium on Algorithms (ESA '94)," Springer Lecture Notes in Computer Science, Vol. 855, pp. 130–140.

[†] E-mail: karsten.weihe@uni-konstanz.de.

In [Rei81], Reif gives an algorithm that finds a minimum (s, t)-cut in undirected, planar graphs with respect to *arbitrary* positive capacities. This algorithm consists of a reduction to $\mathcal{O}(\log n)$ shortest path computations in the dual graph and, hence, requires $\mathcal{O}(n \log^2 n)$ time, where $n = |V|$. (Note that $|E| \in \mathcal{O}(n)$ for planar graphs.) The algorithm relies on ideas developed by Itai and Shiloach [IS79] and by Hassin [Has81] for the (s, t)-planar case, that is, s and t are incident to the outer face. Hassin and Johnson showed how to make use of the ideas in [Rei81] to solve, within the
same asymptotic running time, the maximum flow problem itself in undirected, planar graphs with arbitrary positive capacities [HJ85]. Based on Frederickson's decomposition technique, both algorithms can be implemented so as to run in $\mathcal{O}(n \log n)$ time [Fre87]. Frederickson's technique is applied to solve a single shortest path problem in linear time even for arbitrary capacities (with $\mathcal{O}(n \log n)$ preprocessing time in addition). Because of this preprocessing, we cannot do better by replacing Dijkstra's algorithm with a simple breadth-first search in the unit case, which is the special case we consider here.

In this paper, another approach is introduced, which yields an $\mathcal{O}(n)$ algorithm. In the case of (s, t)-planar graphs, this algorithm reduces to a special case of the algorithm in [IS79].

Very recently, the *vertex-disjoint* version of this problem could be solved in linear time as well [RLWW93]. That algorithm is based on similar geometric insights, but runs fairly differently. The correctness proof for the vertex-disjoint case is *much* more complicated. In view of the result presented here, it seems that the general approach behind these algorithms conforms to the edge-disjoint case much better than to the vertex-disjoint case.

One essential idea of the edge-disjoint case, which does not work in the vertex-disjoint case, is the reduction to an analogous flow problem in the residual graph of a circulation. The particular circulation c that we use in the following is in a sense dual to the shortest distances of all faces from the outer face (when the immediate distance of two neighbored faces is defined equal to the capacity of the separating edge). This has been shown by Hassin [Has81]. Khuller *et al.* [KNK93] define a certain lattice on all circulations and show that c is the zero element of this lattice. In [Wei94], such a reduction is used to solve the maximum (s, t)-flow problem in directed planar graphs in $\mathcal{O}(n \log n)$ time.

The paper is organized as follows. In Section 2, the algorithm is introduced. It will immediately turn out that the overall running time is dominated by the time required to perform a certain sequence of *union-find* operations [Tar86]. Using a technique developed by Gabow and Tarjan [GT85], these operations can, too, be implemented so as to run in linear time. Finally, in Section 3, correctness of the algorithm is proved.

2. THE ALGORITHM

We are now going to give an informal explanation of the algorithm. A *formal* description is given in Table 1, and an example, in Figs. 1–13. However, we will not solve the problem in G itself. Rather, we define an auxiliary *directed* graph $G^{\rightarrow} = (V, E^{\rightarrow})$ and from this, in a second step, another auxiliary directed graph $G_c^{\rightarrow} = (V, E_c^{\rightarrow})$. Then we determine a *maximum unit flow* from s to t in $G_c^{\rightarrow}$ and transform the solution back to

TABLE 1
Formal Description of the Algorithm

```
1. Construct $G^{\rightarrow} = (V, E^{\rightarrow})$ from $G$;
2. construct the circulation $c$ and the residual graph $G_c^{\rightarrow} = (V, E_c^{\rightarrow})$;
3. initialize the union-find structures of all vertices: each inclusion-maximal interval of arcs
   pointing to $v \in V$ in $G_c^{\rightarrow}$ forms one initial set in the union-find structure of $v$;
4. FOR $a \in E_c^{\rightarrow}$ DO $f_a := 0$;
5. let $a_1, \dots, a_k \in E_c^{\rightarrow}$ be the arcs of $G_c^{\rightarrow}$ leaving $s$;
6. FOR $i := 1$ TO $k$ DO right_first_search ($i$);
7. FOR $(v, w) \in E^{\rightarrow}$ DO
       IF $c_{(v,w)} = 0$ THEN $F_{(v,w)} := f_{(v,w)}$
                          ELSE $F_{(v,w)} := 1 - f_{(w,v)}$;
8. construct from $F$ a collection of edge-disjoint $(s, t)$-paths in $G$;
PROCEDURE right_first_search ($i$: Integer);
1. $f_{a_i} := 1$;
2. $a := a_i$;
3. WHILE head ($a$) $\notin \{s, t\}$ DO go_forward ($a$);
PROCEDURE go_forward (VAR $a \in E_c^{\rightarrow}$);
1. $v :=$ head($a$);
2. $S :=$ find($a$);
3. $a := S$.next;
4. $f_a := 1$;
5. $a' :=$ the counterclockwise next arc after $a$ in the adjacency list of $v$;
6. IF $a'$ leaves $v$
   THEN $S$.next $:= a'$
   ELSE $T :=$ find($a'$);
        IF $S \neq T$
        THEN $U :=$ union($S, T$);
             $U$.next $:= T$.next;
```

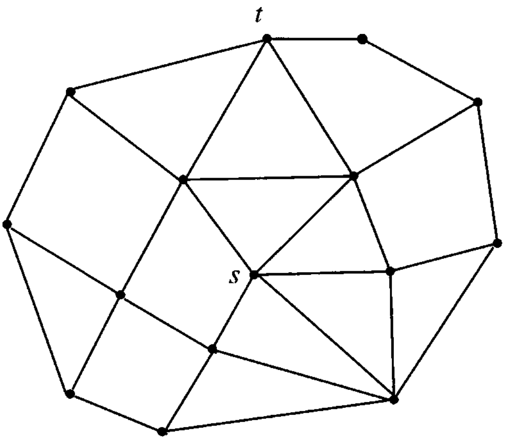

FIG. 1. An input instance $G = (V, E)$.

a maximum unit flow from s to t in $G^{\rightarrow}$. Finally, the latter flow is transformed to a maximum number of edge-disjoint (s, t)-paths in G.

From G to $G^{\rightarrow}$. The graph $G^{\rightarrow} = (V, E^{\rightarrow})$ is the directed, symmetric graph that arises from G when we replace each undirected edge $\{v, w\} \in E$ by two directed arcs, (v, w) and (w, v). See Figs. 1 and 2. From now on, we will restrict our attention to the problem of finding a *maximum unit flow F*

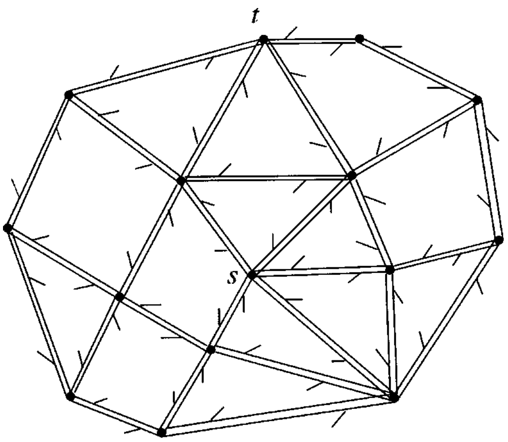

FIG. 2. The directed, symmetric graph $G^{\rightarrow} = (V, E^{\rightarrow})$. The direction of an arc is indicated by a "half-arrow."

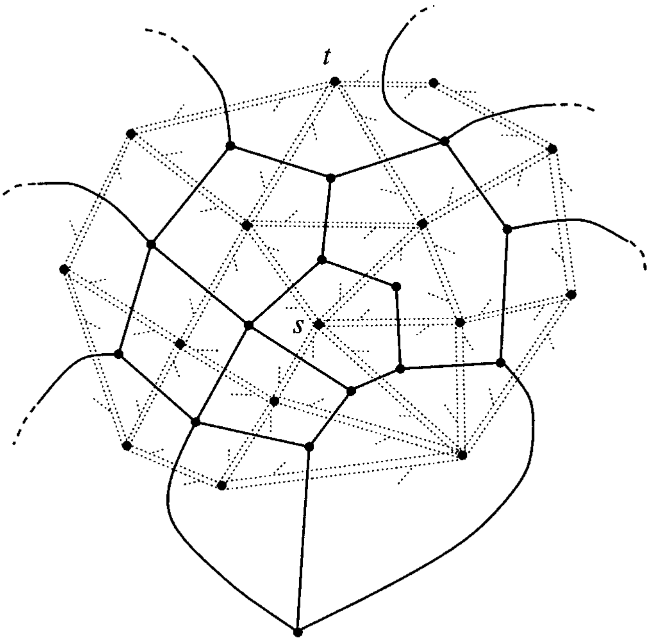

FIG. 3. The dual graph of $G^{\rightarrow}$ (solid). The dashed edges indicate continuations up to the dual vertex in the outer face below.

from s to t in $G^{\rightarrow}$. This is a 0/1-weighting of all arcs of $G^{\rightarrow}$ such that

$$\sum_{w:(v,w)\in E^{\rightarrow}} F_{(v,w)} = \sum_{w:(w,v)\in E^{\rightarrow}} F_{(w,v)}$$

for $v \in V \setminus \{s,t\}$ (*flow conservation conditions*) and the *total flow value*

$$\sum_{w:(s,w)\in E^{\rightarrow}} F_{(s,w)} - \sum_{w:(w,s)\in E^{\rightarrow}} F_{(w,s)}$$

is maximum. See again [AMO93].

To justify our focus on flows in $G^{\rightarrow}$, we note that a maximum collection of edge-disjoint simple (s,t)-paths in G can be constructed from any maximum unit flow from s to t in $G^{\rightarrow}$ in linear time by eliminating all cycles in the flow. This can be done by a repeated depth-first search on all arcs in $E^{\rightarrow}$ with positive flow value. Whenever such a search runs into a cycle, the flow values of all arcs on this cycle are set to zero, the cycle is removed from the depth-first stack, and the search is continued. When a search is completed and not all arcs have been seen so far, another search is started with the tail of such an arc as the new root. Since we need not consider any arc more than once, this transformation is linear.

From $G^{\rightarrow}$ to $G_c^{\rightarrow}$. A *circulation* c is an (s,t)-flow with total flow value 0. In other words, *all* vertices of V satisfy the flow conservation conditions with respect to c. If a circulation c is a unit flow, this means that c decomposes into edge-disjoint cycles. Henceforth, let c be a unit circulation in $G^{\rightarrow}$ (see Fig. 4).

The *residual graph* with respect to unit circulation c, denoted by $G_c^{\rightarrow} = (V, E_c^{\rightarrow})$, is defined as follows. For each arc $(v,w) \in E^{\rightarrow}$ with $c_{(v,w)} = 0$, we have $(v,w) \in E_c^{\rightarrow}$. On the other hand, for each arc $(v,w) \in E^{\rightarrow}$ with $c_{(v,w)} = 1$, we have $(w,v) \in E_c^{\rightarrow}$. (This is not the usual definition of residual networks, but in the case of unit capacities, it is completely equivalent.) Hence $G_c^{\rightarrow}$ contains pairs of parallel arcs in general. Whenever we refer to a double arc (v,w), we mean exactly either of the copies. If $(v,w) \in E_c^{\rightarrow}$ appears twice, one copy corresponds to $(v,w) \in E^{\rightarrow}$, and the other one to $(w,v) \in E^{\rightarrow}$. Note that for $v \in V$, the number of arcs of $E_c^{\rightarrow}$ pointing to v equals the number of arcs of $E_c^{\rightarrow}$ leaving v.

There is a natural one-to-one correspondence between unit (s,t)-flows in $G^{\rightarrow}$ and in $G_c^{\rightarrow}$, namely, a flow F in $G^{\rightarrow}$ and a flow f in $G_c^{\rightarrow}$ correspond to each other if and only if we have $F_{(v,w)} = f_{(v,w)}$ for $c_{(v,w)} = 0$ and $F_{(v,w)} = 1 - f_{(w,v)}$ for $c_{(v,w)} = 1$. In other words, the flow value is switched if and only if the arc itself is switched. Corresponding flows have the same total flow value. Therefore, it suffices to find a maximum flow in $G_c^{\rightarrow}$, and this is exactly the core of the algorithm.

Specific choice of circulation c. We construct a unit circulation c in $G^{\rightarrow}$ such that $G_c^{\rightarrow}$ contains no clockwise cycles. See Figs. 4 and 5. We need

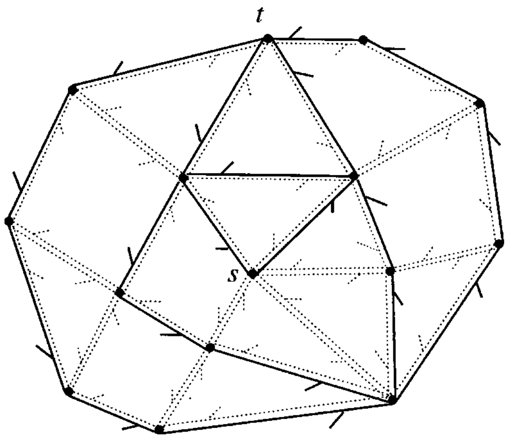

FIG. 4. The unit circulation c. Arcs with positive flow value are solid. The circulation c here consists of three nested counterclockwise cycles, and the ith cycle from outside separates the faces with distance at least i from the other faces.

some terminology to characterize the residual graph $G_c^{\rightarrow}$ of c (and, hence, c itself). The *dual graph* of $G^{\rightarrow}$ is the solid (undirected) graph shown in Fig. 3. That is, one vertex for each face of $G^{\rightarrow}$, and any two dual vertices that are separated by exactly one primal pair of arcs are connected by a dual edge. The *distance* of a dual vertex $\mathscr{F}$ is the length of a shortest path (i.e., number of edges) in the dual graph from the dual vertex in the outer face to $\mathscr{F}$.

Now consider a pair $(v,w),(w,v) \in E^{\rightarrow}$. Let $\mathscr{F}_1$ and $\mathscr{F}_2$ denote the dual vertices neighbored to this pair. More precisely, when we pass over from $\mathscr{F}_1$ to $\mathscr{F}_2$, then we cross (v,w) from right to left and (w,v) from left to right. If the distances of $\mathscr{F}_1$ and $\mathscr{F}_2$ are equal, we have $(v,w),(w,v) \in E_c^{\rightarrow}$. Otherwise, if $\mathscr{F}_1$ is closer to the outer face than $\mathscr{F}_2$, we have two copies of (v,w) in $E_c^{\rightarrow}$. Otherwise, we have two copies of (w,v) in $E_c^{\rightarrow}$. This definition of $G_c^{\rightarrow}$ determines the circulation c itself. The circulation c may be viewed as the union of the clockwise "contour lines" between the different distance levels of dual vertices. See Fig. 4. Obviously, the distances of all faces from the outer face can be computed in linear time by applying a breadth-first search to the dual graph, with the outer face vertex being the root. From that, the circulation c and the corresponding residual graph $G_c^{\rightarrow}$ are easily computed in linear time as well. This justifies our focus on the problem of finding a maximum unit (s,t)-flow in $G_c^{\rightarrow}$.

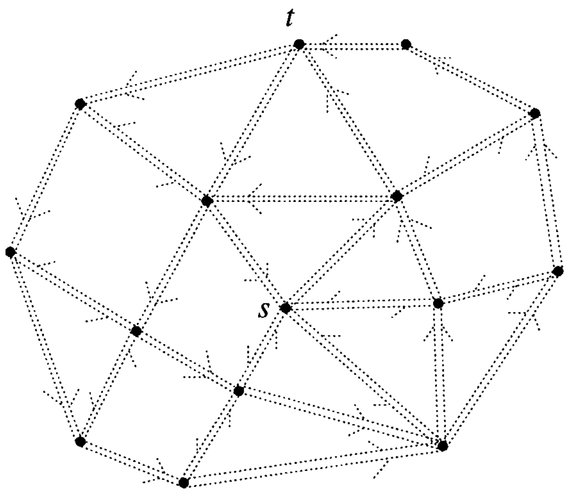

FIG. 5. The residual graph $G_c^{\rightarrow} = (V, E_c^{\rightarrow})$. Two parallel arcs have the same orientation if and only if the face on the right side is closer to the outer face than the face on the left side. Note that $G_c^{\rightarrow}$ has no clockwise cycles.

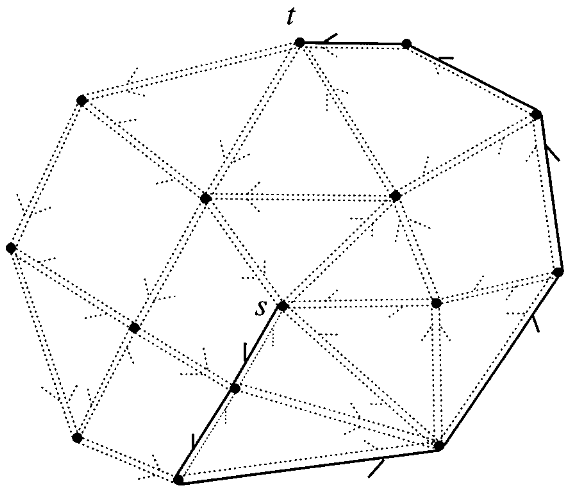

FIG. 6. After the first call to the procedure *right_first_search*. Remember that the ordering of all edges $a_1, a_2, \ldots, a_k$ is arbitrary.

As mentioned in the Introduction, this circulation c is the zero element of the lattice defined in [KNK93]. This circulation can be computed by solving a single shortest-path problem in the dual graph [Has81], which can be done in linear time even for *general* capacities in planar networks [KRRHS93]. The procedure described above may be viewed as a simplified version, which works only for unit capacities.

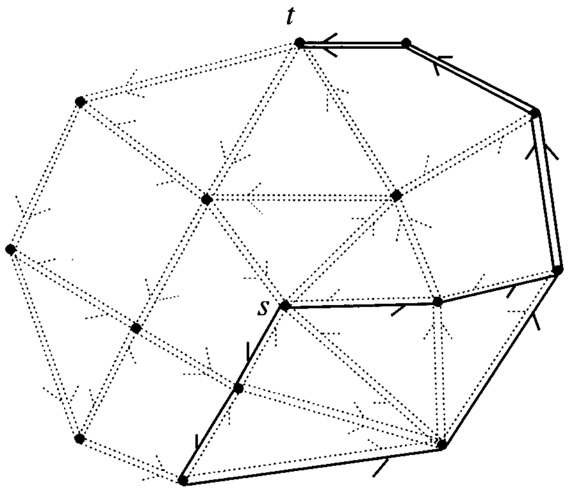

FIG. 7. After the second call.

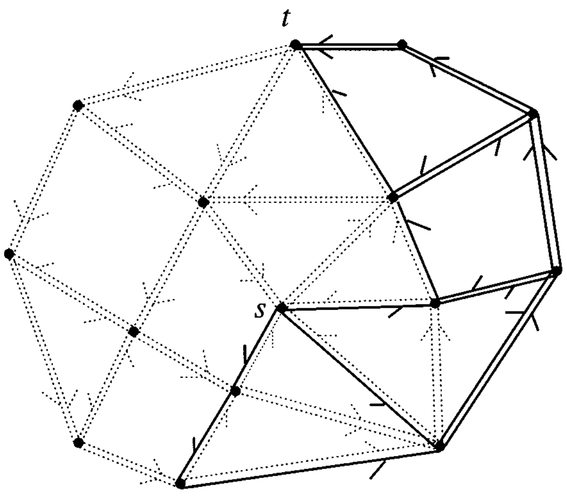

FIG. 8. After the third call.

Core procedure. The core procedure works on $G_c^{\to}$. The *adjacency list* of $v \in V$ consists of all arcs in $E_c^{\to}$ that are incident to v, that is, leaving v or pointing to v. We assume that the adjacency lists of all vertices are sorted according to a fixed, crossing-free embedding of $G_c^{\to}$ in the plane. Such an ordering of all adjacency lists is usually called a *combinatorial embedding* and can be obtained in linear time, see [HT74], for example. In particular, this defines a fixed ordering of two parallel arcs in both adjacency lists. Moreover, we assume that any two *anti*parallel arcs are

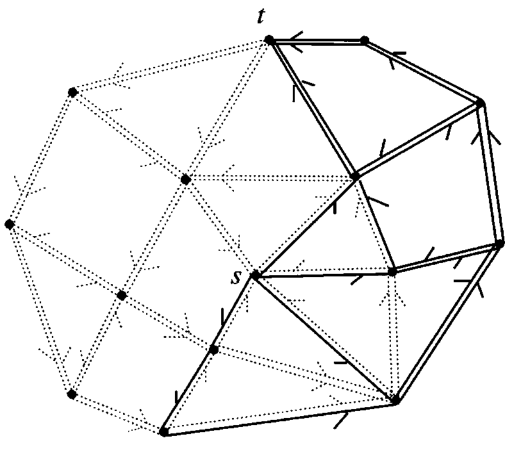

FIG. 9. After the fourth call.

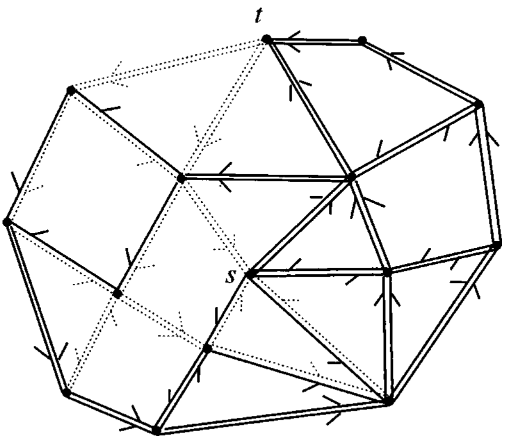

FIG. 10. The final flow f after the fifth, and last, call to *right_first_search*. The path found in this iteration ends with s and is highly pathological.

embedded such that they form a *counter*clockwise cycle of length 2. W.l.o.g., t is incident to the outer face of $G_e^{\rightarrow}$, as shown in Figs. 1–13. (Since the algorithm requires only a combinatorial embedding, this only means the outer face is chosen to be incident to t.)

In the beginning of the core procedure, all arcs $a \in E_c^{\rightarrow}$ carry zero flow, $f_a = 0$. Let $a_1, \dots, a_k$ be the arcs of $G_c^{\rightarrow}$ leaving s, in arbitrary order. The procedure to determine a maximum flow f is essentially a loop over

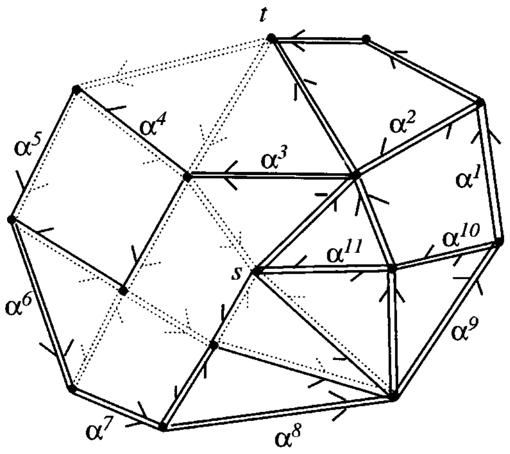

FIG. 11. The sequence of arcs constructed in Section 4 to prove maximality of the final flow f. We have $l = 11$ and $\mu = 9$. In particular, $\mathscr{C} = (\alpha^1 - \alpha^2 - \cdots - \alpha^9 - \alpha^1)$.

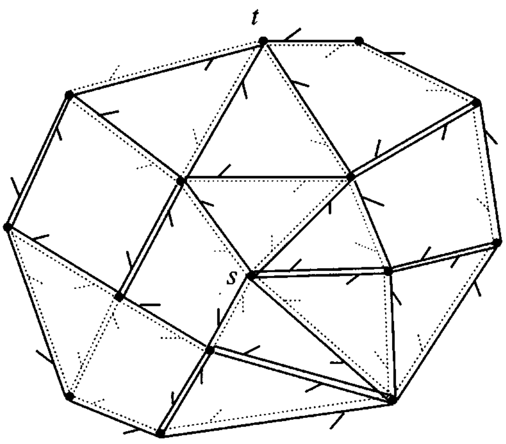

FIG. 12. The flow F in $G^{\rightarrow}$ corresponding to f.

$1,\ldots,k$. In the ith iteration of this loop, we determine a directed path in $G_c^{\rightarrow}$ and switch, on-line, the flow values of each arc on this path from 0 to 1, when we pass the arc. The path determined in the ith iteration starts with a_i, ends either at s or at t, and, clearly, contains only arcs which have been carrying zero flow so far. In particular, after each iteration, f is a unit (s,t)-flow with nonnegative total flow value. We never need to correct the flow value of any arc, once we have changed it from 0 to 1.

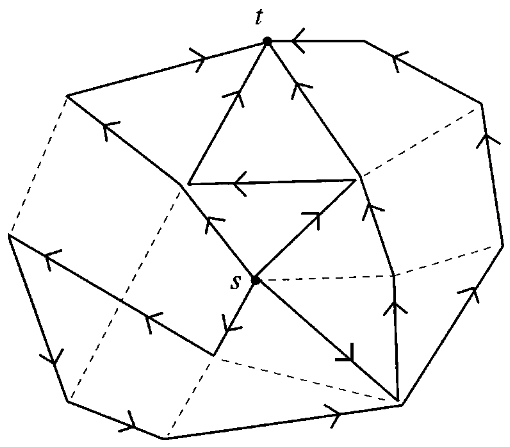

FIG. 13. One possible collection of (s,t)-paths in G resulting from F.

To determine the path in the ith iteration, say, we start a *right-first search* with a_i. This is a depth-first search, where in each step the rightmost possibility of going forward is chosen. More precisely, if v and a are the leading vertex and the leading arc of the current search path, respectively, then the next arc to be added is the counterclockwise next arc after a in the adjacency list of v that leaves v and carries zero flow. (Recall that all adjacency lists are sorted according to a fixed embedding in the plane.) As antiparallel arcs form a counterclockwise cycle, the reverse arc of a, if any, is taken only if no other arc leaves v and carries zero flow.

Since the number of arcs in $G_c^{\rightarrow}$ pointing to $v \in V$ equals the number of arcs in $G_c^{\rightarrow}$ leaving v, there always remains a possibility of going forward from v, as long as we need one to proceed with. Hence, this right-first search must finally reach s or t without performing a backtrack step in this iteration. (We allow the path to be nonsimple, that is, the path may hit the same vertex more than once.) This implies that the total number of forward steps, taken over *all* iterations of the overall loop, is linear, because in each single forward step the flow value of one arc is set from 0 to 1, and no arc with flow value 1 is changed again. Therefore, for the linear worst-case bound it suffices to show that each forward step can be done in (amortized) constant time.

Single forward step. Consider a single step of the ith right-first search, where $v \in V \setminus \{s, t\}$ is the leading vertex and $(u, v) \in E_c^{\rightarrow}$ is the leading arc of the search path. Then we have to determine the counterclockwise next arc after (u, v) in the adjacency list of v which leaves v and still carries zero flow. In order to determine this arc efficiently in each step, we in turn maintain a (dynamically changing) arc set $S_{(v,w)}$ for each arc (v, w) with $f_{(v,w)} = 0$. The set $S_{(v,w)}$ consists of all arcs $(u, v) \in E_c^{\rightarrow}$ such that (v,w) is the counterclockwise next arc after (u, v) in the adjacency list of v which leaves v and still carries zero flow. Thus, for fixed v, the sets $S_{(v,\cdot)}$ partition the cyclic adjacency list of v, restricted to in-going arcs, into disjoint intervals. More precisely, at each stage this is the finest interval partition of the arcs pointing to v such that any two subsequent intervals are separated by an arc that leaves v and carries zero flow.

Each set $S_{(v,w)}$ keeps the corresponding arc (v, w) "in mind." Hence, finding the next arc for going forward amounts to finding the set to which the leading arc of the search path belongs. So let $(u, v) \in S_{(v,w)}$ be the leading arc of the search path at some stage. Let a be the counterclockwise next arc after (v, w) in the adjacency list of v. If a leaves v, then a must still carry zero flow, because anyway, (v, w) must have been considered before a for going forward from v. Then $S_{(v,w)}$ must from now on keep a "in mind" instead of (v, w). On the other hand, if a points to v, $S_{(v,w)}$ must from now on store the arc (v, x) with $a \in S_{(v,x)}$. In other words, the arc sets $S_{(v,w)}$ and $S_{(v,x)}$ have to be united, unless $S_{(v,w)} = S_{(v,x)}$ already. Therefore, maintaining and using these arc sets amounts to one *union-find* problem for each vertex.

Union-find ([Tar86], Chap. 2). This union-find problem is of a very restricted type. In fact, the sets to be maintained for vertex v are disjoint intervals in the cyclic list of all arcs pointing to v. Moreover, only intervals which are neighbored in this cycle list are subject to union operations. Gabow and Tarjan have given a linear time technique for union-find problems like this, but only for the case that the underlying list is not *cyclic* but *linear* [GT85]. (In general, this technique applies to *tree* structured problems, not only to linear ones.) In [Wei93, WW95] it is, respectively, shown how to extend this technique to cyclic lists: Roughly speaking, in the beginning we break the cyclic list between two initial partition sets, which makes the list linear. All union operations except at most one are done according to Gabow and Tarjan's technique. The only possible exception is the case that the two sets around the breakpoint have to be united. In this case, we do *not* apply the union operation provided by Gabow and Tarjan's data structure, but simply keep "in mind" that, henceforth, these two sets are merely two halves of the same set.

Formal description. A formal description is given in Table 1. For each arc set S, $S.next$ is the arc kept "in mind" by S. The arc set to which arc a belongs is returned by the function call $find(a)$. Two arc sets S and T are united by the operation $union(S, T)$, provided S and T are neighbored intervals in the adjacency list of the same vertex. (For convenience, this includes the irregular "unions" which cannot be handled by the data structure of Gabow and Tarjan.) In particular, we require $S \neq T$, that is, at least two arc sets in that adjacency list have still "survived" all previous union operations. The name of the resulting, united, arc set is returned by function *union*. Finally, the function call $head(a)$ returns the vertex that arc $a \in E_c^{\rightarrow}$ points to. We refer to Table 1 for all further details. There the algorithm is more or less formulated in a *top-down* fashion; that is, each subroutine is defined only after its application is given. See Figs. 1−13 for an example.

The overall result of this section is summarized in the following theorem.

THEOREM 2.1. *The algorithm in Table* 1 *constructs a unit (s, t)-flow f in $G_c^{\rightarrow}$ with nonnegative total flow value. In particular, it constructs a unit (s, t)-flow F in $G^{\rightarrow}$ with nonnegative total flow value and, hence, a (possibly empty) set of pairwise edge-disjoint (s, t)-paths in G. All this is done in linear time (if the technique of Gabow and Tarjan is applied to construct f).*

Informal insights. To end this section, we give an informal description of the geometric insights behind the algorithm and its correctness proof. A rigorous proof is then given in the next section.

First, consider the (s,t)-planar case, that is, s and t have a face of G in common, which is w.l.o.g. the outer face. In this case, it is obvious that a repeated right-first search yields a maximum solution for both undirected and directed instances. In fact, in this case we need not even eliminate clockwise cycles. The reason is that, in the (s,t)-planar case, there is a boundary to "lean on," namely the segment of the boundary of G counterclockwise after s and before t. Thus, repeated right-first search means that we "pack" all paths as tightly as possible along this segment, and maximality is obvious, since we waste no space by this. In contrast, if G is *not* (s,t)-planar, there is no such boundary. As a consequence, it may not be correct to draw each path as far right as possible, because then there is no boundary to stop the paths when they are "driven" *too* far right by right-first search. In fact, a search may return to s very soon (along a clockwise cycle), although it could reach t on a way "less right."

However, things are different when there are no clockwise cycles in the graph. In this particular case, the search may return to s only along a *perfectly counterclockwise cycle*. However, before the path describes a complete counterclockwise cycle, it will first see all possibilities to run out of the cycle and to enter its right side, which contains t, since t is on the boundary of the graph. Because of right-first search, the algorithm would prefer any such "exit" from a counterclockwise cycle and is, hence, forced into this cycle only if there is no such exit at all. Since such an exit is exactly what we need for reaching t from arc a_i, say, we return to s in the ith iteration if and only if a_i cannot help augment the current partial solution. (At this point we leave a gap in the informal argumentation, because there are two different kinds of potential exits: forward arcs with zero flow and backward arcs with positive flow, whose flow values could be reset to zero. But the algorithm considers only the first kind in order to touch each arc at most once.)

3. CORRECTNESS

Because of Theorem 2.1, it remains to show that the final flow f has maximum total flow value among all unit (s,t)-flows in $G_c^{\rightarrow}$.

The Labeling Theorem of Ford and Fulkerson [FF62, AMO93] says that a unit (s,t)-flow f in $G_c^{\rightarrow}$ is optimal if and only if there is no (simple) augmenting (s,t)-path in $G_c^{\rightarrow}$ with respect to f. That is, a path from s to t that may contain an arc a of $G_c^{\rightarrow}$ in forward *or* backward direction, but in forward direction only if $f_a = 0$, and in backward direction only if $f_a = 1$.

Since all arcs $a_1, \dots, a_k$ carry unit flow, such an augmenting (s,t)-path must start with an arc that points to s and carries unit flow. We will show that this is not possible. So suppose for a contradiction that there is an arc (w,s) where a simple augmenting (s,t)-path starts.

We will consider a certain sequence of arcs, $\alpha^1 = (v_0, v_1)$, $\alpha^2 = (v_1, v_2)$, $\alpha^3 = (v_2, v_3)$, up to $\alpha^l = (v_{l-1}, v_l)$, with $f_{\alpha^j} = 1$ for all $j = 1, \dots, l$. See Fig. 11. The arcs $\alpha^1, \dots, \alpha^l$ are defined recursively "the other way round." More precisely, we have $v_l = s$ and $v_{l-1} = w$ and hence $\alpha^l = (w,s)$. For $j < l$, α^j is the clockwise next arc after $\alpha^{j+1} = (v_j, v_{j+1})$ in the adjacency list of v_j that points to v_j and carries positive flow. The arc α^1 is the first arc in this sequence such that either $v_0 = s$ or the arc to be chosen next already belongs to the sequence. Clearly, for $v_j \neq s$, we always find an arc with positive flow that points to v_j, since we reach v_j via an arc with positive flow that leaves v_j. Hence, this sequence of arcs is well defined (and finite, of course).

The idea is to show that there is $\mu \in \{1, \dots, l\}$ such that the arcs $\alpha^1, \dots, \alpha^\mu$ form a cycle $\mathscr{C}$ which excludes t but does not exclude s or w, and no augmenting path can leave $\mathscr{C}$ and enter its exterior except, possibly, at s, if s belongs to $\mathscr{C}$. As a consequence, no *simple* augmenting path starting with (w,s) can reach t, because $\mathscr{C}$ blocks all these paths (see Fig. 11).

Now, if the sequence terminates with α^1 because of $v_0 = s$, the cycle $\mathscr{C}$ is defined as $\mathscr{C} = (\alpha^1 - \alpha^2 - \cdots - \alpha^{l-1} - \alpha^l - \alpha^1)$, which means $\mu = l$. Otherwise, $\mathscr{C}$ is defined as $\mathscr{C} = (\alpha^1 - \alpha^2 - \cdots - \alpha^{\mu-1} - \alpha^\mu - \alpha^1)$, where α^μ had been added next to the sequence, if we had not deliberately terminated the sequence with α^1. In other words, α^μ is the clockwise next arc after α^1 in the adjacency list of $v_0 = v_\mu$ which points to v_μ and carries positive flow. For convenience, we henceforth identify v_i with $v_{i \bmod \mu}$.

Notice that $\mathscr{C}$ may not be a simple cycle. In fact, in general we may have $v_i = v_j$ for $i \neq j$ and, thus, $\mathscr{C}$ may contain subcycles. But at least one can prove the following fact.

LEMMA 3.1. *The cycle $\mathscr{C}$ does not cross itself; that is, there are no $i, j \in \{1, \dots, \mu\}$, $i \neq j$, such that $v_i = v_j$ and the counterclockwise ordering of α^i, α^{i+1}, α^j, and α^{j+1} around v_i is $\alpha^i \prec \alpha^j \prec \alpha^{i+1} \prec \alpha^{j+1}$.*

Proof. In this case, α^j had been chosen as the next arc after α^{i+1} rather than α^i. ∎

COROLLARY 3.2. *The interior and the exterior, the right side and the left side of $\mathscr{C}$ are well defined. Since $G_c^{\rightarrow}$ contains no clockwise cycle, $\mathscr{C}$ is a counterclockwise cycle. In other words, the exterior of $\mathscr{C}$ is its right side.*

LEMMA 3.3. *Any arc that points to a vertex on $\mathscr{C}$ from the exterior of $\mathscr{C}$ carries zero flow, except, possibly, in one situation: s belongs to $\mathscr{C}$ and s is the head of the arc.*

Proof. Suppose a points to a vertex $v_j \neq s$ on $\mathscr{C}$, is outside $\mathscr{C}$, and carries positive flow. There may be several arcs of $\mathscr{C}$ pointing to v_j, but w.l.o.g. (v_{j-1}, v_j) is the clockwise next such arc after a in the adjacency list of v_j. As a is outside $\mathscr{C}$, a points to $\mathscr{C}$ from the right side. Hence, the clockwise order of (v_j, v_{j+1}), a, and (v_{j-1}, v_j) around v_j is $(v_j, v_{j+1}) \prec a \prec (v_{j-1}, v_j)$. Therefore, a had become the next member (i.e., α^j) of the sequence after $\alpha^{j+1} = (v_j, v_{j+1})$ rather than (v_{j-1}, v_j). ▮

LEMMA 3.4. *Any arc that points from a vertex of $\mathscr{C}$ to the exterior of $\mathscr{C}$ carries positive flow.*

Proof. Suppose there is an arc a that points from v_j to the exterior of $\mathscr{C}$ and carries zero flow. Then we have $v_j \neq s$, because $f_{a_1} = f_{a_2} = \cdots = f_{a_k} = 1$.

Consider the counterclockwise next arc of $\mathscr{C}$ after a in the adjacency list of v_j. As the exterior of $\mathscr{C}$ is its right side, this arc leaves v_j, say (v_j, v_{j+1}). Now let (x, v_j) be the arc that was assigned flow value 1 immediately before (v_j, v_{j+1}) during the algorithm. Then the clockwise ordering of a, (v_{j-1}, v_j), (x, v_j), and (v_j, v_{j+1}) around v_j is $a \prec (v_{j-1}, v_j) \preccurlyeq (x, v_j) \prec (v_j, v_{j+1})$, where the placement of (x, v_j) follows from Lemma 3.3. But then, because of our right-first strategy, a had been preferred to (v_j, v_{j+1}) as the next arc after (x, v_j) to be assigned a positive flow value during the algorithm. ▮

LEMMA 3.5. *The exterior of $\mathscr{C}$ contains t, but neither s nor w.*

Proof. First note that t cannot lie on $\mathscr{C}$, because no arc leaving t is given positive flow value during the algorithm. Recall that t is incident to the outer face of G. Therefore, t cannot belong to the interior of $\mathscr{C}$ either. Hence, t belongs to the exterior of $\mathscr{C}$.

Now consider s and w. We make a case distinction. If s belongs to $\mathscr{C}$, then we have $v_0 = v_\mu = s$, and all arcs α^j belong to $\mathscr{C}$. In particular, $(w, s) = \alpha^l$ does, and w belongs to $\mathscr{C}$, too. On the other hand, assume that s does not belong to $\mathscr{C}$. As the arcs of $\mathscr{C}$ have positive flow value, there must be a directed path p from s to $\mathscr{C}$ such that all arcs on p have positive flow value. If s belongs to the exterior of $\mathscr{C}$, then the last arc of p points to $\mathscr{C}$ from outside, which contradicts Lemma 3.3. Therefore, s belongs to the interior of $\mathscr{C}$, and since w is incident to s, w must belong either to $\mathscr{C}$ itself or to its interior. ▮

Now we are in a position to prove correctness of the algorithm.

THEOREM 3.6. *For $(w, s) \in E_c^{\rightarrow}$, no simple augmenting (s, t)-path with respect to the final flow f may start with (w, s). In particular, the final flow f is maximum.*

Proof. Suppose there is a simple augmenting (s,t)-path p starting with (w,s). By Lemma 3.5, p must leave $\mathscr{C}$ and enter its exterior at some vertex different from s. There are two possibilities for the first arc of p outside $\mathscr{C}$: Either it points to $\mathscr{C}$ and carries positive flow, or it leaves $\mathscr{C}$ and carries zero flow. But this contradicts either Lemma 3.3 or Lemma 3.4. ▮

REFERENCES

[AMO89] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin, Network flows, *in* ‘‘Handbooks in Operations Research and Management Science,’’ (George L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, eds.), Vol. 1, North-Holland, Amsterdam, 1989.

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin, ‘‘Network flows,’’ Prentice-Hall, 1993.

[FF62] L. R. Ford and D. R. Fulkerson, ‘‘Flows in networks,’’ Princeton Univ. Press, Princeton, 1962.

[Fre87] Greg N. Frederickson, Fast algorithms for shortest paths in planar graphs with applications, *SIAM J. Comput.* **16** (1987), 1004–1022.

[GT85] Harold N. Gabow and Robert E. Tarjan, A linear-time algorithm for a special case of disjoint set union, *J. Comput. Syst. Sci.* **30** (1985), 209–221.

[Has81] Refael Hassin, Maximum flows in (s,t) planar networks, *Inform. Process. Lett.* **13** (1981), 107.

[HJ85] Refael Hassin and David S. Johnson, An $\mathscr{O}(n \log^2 n)$ algorithm for maximum flow in undirected planar networks, *SIAM J. Comput.* **14** (1985), 612–624.

[HT74] John E. Hopcroft and Robert E. Tarjan, Efficient planarity testing, *J. Assoc. Comput. Mach.* **21** (1974), 549–568.

[IS79] Alon Itai and Yossi Shiloach, Maximum flows in planar networks, *SIAM J. Comput.* **8** (1979), 135–150.

[KNK93] Samir Khuller, J. Naor, and Philip Klein, The lattice structure of flow in planar graphs, *SIAM J. Discrete Math.* **30** (1993), 477–490.

[KRRHS93] Philip Klein, Satish B. Rao, Monika Rauch-Henzinger, and S. Subramanian, Faster shortest-path algorithms for planar graphs, *in* ‘‘Proceedings of the 26th Annual ACM Symposium on Theory of Computing, STOC’94,’’ 1994.

[Rei81] John H. Reif, Minimum *s-t*-cut of a planar undirected network in $\mathscr{O}(n \log^2 n)$ time, *SIAM J. Comput.* **12** (1981), 71–81.

[RLWW93] Heike Ripphausen-Lipa, Dorothea Wagner, and Karsten Weihe, The vertex-disjoint Menger-problem in planar graphs, *in* ‘‘Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA’93,’’ 1993, pp. 112–119.

[RLWW95] Heike Ripphausen-Lipa, Dorothea Wagner, and Karsten Weihe, Efficient algorithms for disjoint paths in planar graphs, *in* ‘‘DIMACS Series in Discr. Math. and Comp. Sci.,’’ (William Cook, Laszlo Lovász, and Paul Seymour, eds.), Vol. 20, pp. 295–354, American Mathematical Society, 1995.

[Tar86] Robert E. Tarjan, Data structures and network algorithms, *SIAM*, 1986.

[Wei93] Karsten Weihe, Multicommodity flows in even, planar networks, *in* ‘‘Algorithms and Computation, 4th International Symposium, ISAAC ’93,’’ (K. W. Ng, Prabhaker Raghavan, N. V. Balasubramanian, and F. Y. L. Chin, eds.), pp. 333–342, Springer-Verlag, Lecture Notes in Computer Science, Vol. 762, 1993.

[Wei94] Karsten Weihe, Maximum (s, t)-flows in planar network in $O(n \log n)$ time, *in* ''Proceedings of the 35th Annual Symposium on Foundations of Computer Science, FOCS'94,'' 1994, pp. 178–189.

[WW95] Dorothea Wagner and Karsten Weihe, A linear time algorithm for edge-disjoint paths in planar graphs, *Combinatorica* **15** (1995), 135–150.