KYBERNETIKA — VOLUME 28 (1992), NUMBER 1, PAGES 37 - 49

MINIMUM CUT IN DIRECTED PLANAR NETWORKS

LADISLAV JANIGA AND VÁCLAV KOUBEK

An algorithm which for any planar directed network with n nodes finds its minimum cut in time $O(n\log^2(n)/\log(\log(n)))$ is presented. For the case of $s-t$-network this time is reduced by the factor of $\log(n)$, i.e. to $O(n\log(n)/\log(\log(n)))$.

1. INTRODUCTION

We present an algorithm which for any planar directed network with n nodes finds its minimum cut in time $O(n\log^2(n)/\log(\log(n)))$. Such an algorithm running in $O(n\log^2(n))$ time is already known for undirected networks [14]. The case of directed networks seems to be somewhat more difficult, however we use similar techniques to those in [9], [14]. We use two tricks: The first one replaces the set of all cuts by an essentially smaller one and the latter consists of an application of the divide-and-conquer principle. Both of these tricks are based on the planarity.

The first idea reduces our problem of finding a minimum cut to the problem of finding shortest paths in the dual multigraph. This idea appears in [4], and we use the most effective implementation of the shortest path algorithm due to Dijkstra [1]. The second idea is an application of the divide-and-conquer principle which was used by Reif [14] for undirected networks. We had to modify Reif's approach substantially because he used it for undirected networks.

The running time of our algorithm is $O(n\log^2(n)/\log(\log(n)))$, in general, and for special cases $O(n\log(n)\log(\log(n)))$ (when the capacity values are polynomially bounded nonnegative integers) and $O(n\log(n))$ (when the capacities are in $0,1$). For the case of $s-t$-networks all of the running times are reduced by the factor of $\log(n)$.

The plan of the paper is the following: The first section introduces basic definitions. In the second one we prove that it suffices to restrict to special cuts, called cut-cycles. The results of the third section enable us to apply the paradigm "divide and conquer" for finding a minimum cut-cycle. The last section is devoted to a description of our algorithm.

The computational model used in this paper is a RAM with a unit cost and length of words $O(m)$. It means that all arithmetic operations on numbers of length $< cm$ for some constant $c > 0$ are performed in one step. The bound m, used here, is the minimal

length enabling to write all input numbers in binary code (i.e. $m = \max\{\log_2(n)\} \cup \{\log_2(c(e)); e \text{ is an arc}\}$ where n is a number of nodes of an input graph). The other technical machinery used here is a standard one, the reader will find all the basic notions, definitions as well as results together with motivations in the monograph [4].

2. BASIC NOTIONS AND FACTS

A *network* N is a quadruple $N = (G, s, t, c)$ where:

(1) $G = (V, E)$ is a symmetric directed graph (i.e. $(a, b) \in E$ implies $(b, a) \in E$);
(2) s and t are two distinguished vertices of G, called the source and the sink of the network respectively;
(3) c is a capacity function from E into the set $\mathbb{R}_+$ of all nonnegative reals such that for every $(a, b) \in E$ we have either $c(a, b) > 0$ or $c(b, a) > 0$.

From now on we shall denote an undirected edge of the form $\{a, b\}$ by $a - b$ and a directed arrow from a to b by $a \to b$.

The graph G will be given by lists V, vE, $v \in V$, where $vE = \{w \in V; (v, w) \in E\}$ is the list of all successors of the vertex v. Moreover, the structure of any vertex $w \in vE$ contains the number $c(v, w)$ and the pointer to the vertex $v \in wE$.

Below we shall often study different subgraphs $G' = (V, E')$ of G. Any subgraph G' of G will be determined by a mapping $\varphi_{G'} : E \longrightarrow \{0, 1\}$ such that $\varphi_{G'}$ is the characteristic function of E'. Note that a subgraph G' need not be symmetric.

A *flow* f in a network N is any function from E into $\mathbb{R}_+$ such that:

(1) $f(x \to y) \leq c(x \to y)$ for every $x \to y \in E$,
(2) for any vertex v except for s and t we have:

$$\sum_{u \to v \in E} f(u \to v) = \sum_{v \to w \in E} f(v \to w).$$

The *value of the flow* f is the following number

$$\sum_{s \to u \in E} f(s \to u) - \sum_{w \to s \in E} f(w \to s) = \sum_{v \to t \in E} f(v \to t) - \sum_{t \to v \in E} f(t \to v)$$

and it is denoted by $|f|$. A flow f is said to be *maximum* if $|f| \geq |g|$ for any flow g in N. The value of maximum flows in N is denoted by $|N|$.

For any set S of arrows we denote $|S| = \sum_{x \to y \in S} c(x \to y)$. We say that a set A of arrows of G *crosses a directed path* $P = \{x_0 \to x_1 \to \ldots \to x_n\}$ if $x_i \to x_{i+1} \in A$ for some $i \in \{0, 1, \ldots, n-1\}$. A set C of arrows of G is called a *cut* in N if any directed path from s to t in G is crossed by C. A cut C is said to be *minimum* in N if for any cut D in N we have $|C| \leq |D|$.

Let $G' = (V, E')$ be any subgraph of G. We say that C is a *minimum G'-cut* if C is a cut, $C \subseteq E'$, and for every cut $C' \subseteq E'$ we have $|C| \leq |C'|$.

Our definitions of a network and a cut are slightly modified in comparison to the classical ones, where only paths containing arrows of positive capacity are considered. We start with symmetric graphs which enable us to use the dual graph of G. For this sake it is convenient to suppose that G is symmetric (in which case any pair of arrows $a \to b$ and $b \to a$ can be represented by $a - b$, and no difficulties with dual multigraphs arise). Since the capacity function can be zero for some arrows, this assumption is not restrictive. This modification is immaterial with respect to the classical results, but the fact that a cut can contain arrows with zero capacity will be very useful.

The following theorem is classical:

Ford-Fulkerson Theorem 2.1 [4]. The value of a maximum flow in any network N is equal to the value of a minimum cut in N.

A network $N = (G, s, t, c)$ is said to be a *P-network* if:

(1) G is a planar connected graph without loops;

(2) N contains a positive path from s to t, i.e., there is a directed path from s to t in N containing arrows of positive capacities only.

If N is a P-network then G is said to be a *P-graph.*

The planarity of G is an essential restriction, the other condition is not of basic importance. Note that if there is no positive path from s to t then $|N| = 0$. Using the depth-first search technique [1] on the arrows with positive capacities, it is easy to decide in $O(|V| + |E|)$-time whether a positive path from s to t exists. If there is a positive path from s to t, then $|N|$ depends on the component containing s and t only, and since a construction of connected components of a graph requires $O(|V| + |E|)$-time (see [1]), we can restrict ourselves to P-networks.

We will assume that a P-graph G together with a fixed embedding of G into the plane is given. This can be assumed without loss of generality, since such an embedding can be found for any P-graph in $O(n)$ time [8], [12].

We recall the notion of a *dual multigraph* $D(G) = (D(V), D(E))$ of a planar graph G:

$D(V)$ is the set of all faces of G;

$D(E) = \{D(e); e \in E\}$ where $D(e)$ is an edge connects two faces $F_1, F_2 \in D(V)$ such that e is bordering both F_1 and F_2.

For a face $F \in D(V)$ denote by $B(F)$ the set of all arrows $a \to b$ bordering F. For an arrow $a \to b$ define $D(a \to b) = D(a - b)$, for a set of arrows A put $D(A) = \{D(a \to b); a \to b \in A\}$.

For any planar graph G its dual $D(G)$ is clearly planar again, and an embedding of G into the plane defines naturally an embedding of $D(G)$ such that any face of G is represented by some of its inner points, and $D(e)$ is the unique edge connecting corresponding faces meeting e in an inner point. For this reason, we will assume a common embedding of G and $D(G)$. To distinguish between objects in G and $D(G)$ we call edges, paths, cycles etc. in $D(G)$ D-edges, D-paths, D-cycles etc. We say that a D-path (or a D-cycle) P is contained in a subgraph $G' = (V, E')$ of G if for every edge

$e \in P$ there exists an arrow $a \to b \in E'$ with $D(a \to b) = e$. Then we write $P \subseteq G'$. Analogously, for a path P we write $P \subseteq G'$ if $P \subseteq E'$.

For a set of D-edges C and a directed path $P : (x_0 \to x_1 \to ... \to x_n)$, we say that P *crosses* C *k-times* if $k = |\{i; D(x_i \to x_{i+1}) \in C\}|$. If P crosses C k-times for some $k > 0$ then P *crosses* C.

A vertex-simple D-cycle C (forming a closed and non self-intersecting curve in the plane) *separates* two vertices x and y if every directed path from x to y crosses C.

Two arrows $a \to b$ and $c \to d$ bordering a face F are said to be *confluent* if one of them belongs to the clockwise orientation and the other to the counter-clockwise orientation of the border of F.

A sequence of arrows $e_1, ..., e_n$ is *confluent* if e_i and e_{i+1} are confluent for every $i \in \{1, ..., n-1\}$. A confluent sequence $e_1, ..., e_n$, $n > 1$ is said to be *closed* if e_n and e_1 are also confluent. If, moreover, $D(e_1), ..., D(e_n)$ is a D-cycle then $e_1, ..., e_n$ is said to be a *confluent cycle*. A confluent cycle whose members form a cut is called a *cut-cycle*.

A set of arrows $\{A_1, ..., A_n\}$ is called *closed confluent* if it can be arranged into a closed confluent sequence.

Note that the sets $\{(s, x) \in E;\ x \in V\}, \{(x, t) \in E;\ x \in V\}$ are closed confluent sets such that the arranged confluent sequences are cut-cycles. We denote C_s, C_t the corresponding cut-cycles.

For a set A of arrows and a vertex v we denote

$Acc(x, A) = \{y \in V;$ there exists a directed path from x to y which does not cross $A\}$,
$Acc(A, x) = \{y \in V;$ there exists a directed path from y to x which does not cross $A\}$.

Let $F = \{x_0 - x_1 - ... - x_n - x_0\}$ be a face and let $a \to b$, $c \to d \in B(F)$ be confluent arrows. Without loss of generality we can assume that $a = x_0$, $b = x_n$. Then $c = x_i$, $d = x_{i+1}$ for some $i \in \{0, 1, ..., n-1\}$. Denote by $Out(a \to b,\ c \to d) = x_0 \to x_1 \to ... \to x_i, In(a \to b,\ c \to d) = x_n \to x_{n-1} \to ... \to x_{i+1}$. If $C = \{a_i \to b_i;\ i \in \{0, 1, ..., m\}\}$ is a closed confluent sequence then a concatenation of $In(a_0 \to b_0, a_1 \to b_1), In(a_1 \to b_1, a_2 \to b_2), ..., In(a_n \to b_n, a_0 \to b_0)$ is denoted by $In(C)$ and analogously $Out(C)$ is a concatenation of $Out(a_0 \to b_0, a_1 \to b_1)$, $Out(a_1 \to b_1, a_2 \to b_2), ...,$ $Out(a_n \to b_n, a_0 \to b_0)$. Clearly, the following holds:

Statement 2.2. For a closed confluent sequence $C, In(C)$ and $Out(C)$ are cycles such that for every $a \to b \in C$ every vertex of $Out(C)$ belongs to $Acc(a, C)$ and every vertex of $In(C)$ belongs to $Acc(C, b)$.

Statement 2.3. Every closed confluent sequence induces a closed edge-simple D-path; every such D-path is induced by exactly two closed confluent sequences where one is obtained from the other by reversing the arrows.

Note that the Dijkstra algorithm for finding a short path from a single source, see [12], p. 40 can be modified to the algorithm *Cycle1* for a finding minimal confluent path or a minimal confluent cycle with a suitable property, e. g., there exists an algorithm which

for a given subgraph G' of G and a given arrow $a \to b$ finds a confluent cycle $C \subseteq G'$ containing $a \to b$ with the smallest $|C|$. Dijkstra algorithm uses a priority queue Q and the number of operations with Q is proportional to the number m of arrows in the input graph G'. If the operations with Q require $O(f(m))$ time for a function f, then the Dijkstra algorithm uses $O(m + f(m))$ time. The same facts hold for *Cycle1*.

3. CUT–CYCLES

The following lemmas generalize the ideas of Reif [14] for directed networks. Applying them we reduce the set of cuts to be searched through to those cuts that are confluent cycles. Finally, we give a characterization of a confluent cycle which is a cut.

Lemma 3.1. Every cut contains a closed confluent sequence which is a cut.

Proof. Let R be an arbitrary cut. Let $R' = \{a \to b \in R; a \in Acc(s, R), b \notin Acc(s, R)\}$. Clearly, $R' \subseteq R$ and R' is a cut. Choose $a_0 \to b_0 \in R'$ and let F be a face with $a_0 \to b_0 \in B(F)$. Since $a_0 \in Acc(s, R), b_0 \notin Acc(s, R)$ there exists $a_1 \to b_1 \in R' \cap B(F)$ such that:

(1) $a_0 \to b_0$ and $a_1 \to b_1$ are confluent with respect to F;
(2) one of two paths connecting a_0 and a_1 bordering F lies in $Acc(s, R)$.

Note that $a_1 \to b_1 \in R'$ is uniquely determined by F and the conditions (1) and (2). Denote $a_1 \to b_1 = \gamma(a_0 \to b_0, F)$, then $a_0 \to b_0 = \gamma(a_1 \to b_1, F)$. Define S by induction: $a_0 \to b_0 \in S$. If $a_i \to b_i \in S$ and $D(a_i \to b_i) = F_0 - F_1$ then $\{a_{i-1} \to b_{i-1}, a_{i+1} \to b_{i+1}\} = \{\gamma(a_i \to b_i, F_0), \gamma(a_i \to b_i, F_1)\}$. Since G is finite there exists the smallest j with $a_{j+1} \to b_{j+1} = a_i \to b_i$ for $i < j$. By the properties of γ we obtain that $i = 0$, and hence, S is a closed confluent sequence.

Finally, we prove that either S or $R'' = R' - S$ is a cut. Assume that neither is. Let $P = \{s = u_0 \to u_1 \to ... \to u_{k-1} \to u_k = t\}$ be a directed path which does not cross R''. Since S is not a cut there exists the greatest i with $u_l \in Acc(s, S)$ for every $l \leq i$ and the smallest j with $u_l \in Acc(s, S)$ for every $l \geq j$. Let P' be the concatenation of $u_0 \to u_1 \to ... \to u_i$, a part of $Out(S)$ containing u_i and u_j, and $u_j \to u_{j+1} \to ... \to u_k$. Obviously, P' is a directed path from s to t. By Statement 21.2, P' does not cross S. By the definition of S, $Out(S) \subseteq Acc(s, R')$, and thus, P' does not cross R' – a contradiction, because R' is a cut.

The proof is now completed by induction. If S is not a cut, we take $R'' = R' - S$ in place of R' and repeat the whole construction. After a finite number of steps we necessarily obtain a confluent closed sequence of arrows in R which is a cut. □

Lemma 3.2. Every cut contains a cycle cut.

Proof. Let R be a cut. According to Lemma 3.1 there exists a closed confluent sequence $S = (A_1, ..., A_n) \subseteq R$ which is a cut. If S is a confluent cycle, we are done, otherwise the dual path $D(S)$ induced by S can be split to two parts, a vertex simple D-cycle C' and $D(S) - C'$. Then there exists a confluent cycle $C \subseteq S$ with $D(C) =$

C'. Analogously as in Lemma 3.1 we prove that either C or $S - C$ is a cut and this construction can be applied recursively until C is the desired cut-cycle. □

Corollary 3.3. In any P-network N there is a minimum cut which is a confluent cycle.

Corollary 3.3 has been used in [10] for a proposal design of a parallel algorithm constructing a minimum cut in any P-network. Note that Corollary 3.3 reduces the set of cuts which have to be searched for finding a minimum cut. The following lemma and corollary characterize a confluent cycle which is a cut; this characterization has been also used in [10].

For a subgraph G' of G for a directed path P and a face F a cut-cycle C is called a *minimum* (F, P, G')*−cut-cycle* if $C \subseteq G'$, $P \cap C \cap B(F) \neq \emptyset$, and $|C|$ has the smallest possible value.

Lemma 3.4. Let C be a D-cycle. Then for every pair x, y of vertices of G the following conditions are equivalent:

(1) C separates x and y;

(2) there is a directed path from x to y crossing C an odd number of times;

(3) every directed path from x to y crosses C an odd number of times.

Moreover, if the vertices x and y are not separated by C, then there exists a directed path from x to y that does not cross C.

Proof. Let $P = \{x = u_0 \to u_1 \to \ldots \to u_k = y\}$ be a directed path. Put $C' = \{a \to b \in E; D(a \to b) \in C\}$. If P does not cross C, then C does not separate x and y. If C crosses P exactly once, then by Jordan's theorem there exists i with $u_i \in Acc(x, C'), u_{i+1} \notin Acc(x, C')$. Then $y \in Acc(C', u_{i+1})$, and hence, $y \notin Acc(x, C')$. Thus, C separates x and y. If C crosses P k-times for $k > 1$, then there exist the greatest i with $u_l \in Acc(x, C')$ for every $l \leq i$ and the smallest $j > i$ with $u_j \in Acc(x, C')$. By Statement 2.3 there exists a closed confluent sequence C'' with $C'' \subseteq C'$ and $C = D(C'')$. By Statement 2.2 we can replace the part of P between u_i and u_j by the part of $In(C'')$ or $Out(C'')$, and we obtain a directed path P' from x to y such that C crosses P' $(k-2)$-times. By induction we obtain that (2) ⇒ (1) ⇒ (3). Since (3) ⇒ (2) is obvious, the equivalence of (1), (2), and (3) is proved. The last statement follows immediately. □

Corollary 3.5. Let C be a confluent cycle. Then the following are equivalent:

(1) C is a cut;

(2) there is a directed path $P = \{s = x_0 \to x_1 \to \ldots \to x_m = t\}$ crossing C an odd number of times, and for the smallest j with $D(x_j \to x_{j+1}) \in D(C)$ we have that $x_j \to x_{j+1} \in C$;

(3) every directed path $P = \{s = x_0 \to x_1 \to \ldots \to x_m = t\}$ crosses $D(C)$ an odd number of times, and for the smallest j with $D(x_j \to x_{j+1}) \in D(C)$ we have that $x_j \to x_{j+1} \in C$.

Proof. Since C is a confluent cycle we have that $D(C)$ is a D-cycle.

(1) $\Rightarrow$ (3). If C is a cut, then $D(C)$ separates s and t. Now let $P = \{s = x_0 \rightarrow x_1 \rightarrow ... \rightarrow x_m = t\}$ be a directed vertex-simple path from s to t. Let j be the smallest number with $D(x_j \rightarrow x_{j+1}) \in D(C)$. Assume that $x_{j+1} \rightarrow x_j \in C$. Let k be the smallest number with $x_l \in Acc(D(C), t)$ for every $l \geq k$. Then the concatenation of $x_0 \rightarrow x_1 ... \rightarrow x_{j+1}$, the part of $Out(C)$ connecting x_{j+1} and x_k, and $x_k \rightarrow x_{k+1} \rightarrow ... \rightarrow x_m$ is a directed path P' which does not cross C – a contradiction.

(3) $\Rightarrow$ (2). Trivial.

(2) $\Rightarrow$ (1). Let P be the path in question. Since it crosses $D(C)$ an odd number of times, $D(C)$ separates s and t. Clearly, $V = Acc(s, D(C)) \cup Acc(D(C), t)$. According to the properties of P, there exists $a \rightarrow b \in C$ with $a \in Acc(s, D(C)), b \in Acc(D(C), t)$. Hence, by Statement 2.2, $Out(C) \subseteq Acc(s, D(C))$, therefore $c \rightarrow d \in C$ if and only if $c \in Acc(s, D(C)), d \in Acc(D(C), t)$, and $c \rightarrow d \in E$. Thus C is a cut. □

Corollary 3.5 enables us to modify Algorithm *Cycle1* to the algorithm *Cycle2* constructing a minimum (P, F, G') cut-cycle where $P = (s = x_0 \rightarrow x_1 \rightarrow ... \rightarrow x_p = t)$ is a vertex-simple directed path in G, F is a face, $G' = (V, E')$ is a subgraph of G with $P \cap B(F) \cap E' \neq \emptyset$. By Corollary 3.5 it suffices to construct a confluent cycle $C = \{a_k \rightarrow b_k \in E; k \in \{0, 1, ..., q-1\}\}$ such that

(1) $C \subseteq E'$ and $a_0 \rightarrow b_0 \in B(F) \cap P$;

(2) P crosses $D(C)$ an odd number times;

(3) for the smallest k with $D(x_k \rightarrow x_{k+1}) \in D(C)$ we have $x_k \rightarrow x_{k+1} \in C$;

(4) $|C|$ has the smallest possible value.

For this reason we define, for a confluent sequence Q

$Cross(Q) = 1$ if P crosses $D(Q)$ by an odd numbers of times;

$Cross(Q) = 0$ if P crosses $D(Q)$ by an even number of times.

$First(Q) = \min\{l; D(x_l \rightarrow x_{l+1}) \in D(Q)\}$ if the set is non-empty;

$First(Q) = \infty$ if $D(P) \cap D(Q) = \emptyset$.

$Okay(Q) = 1$ if $First(Q) = k \neq \infty$ and $x_k \rightarrow x_{k+1} \in Q$;

$Okay(Q) = 0$ otherwise.

We have to find a confluent cycle C with the smallest $|C|$ such that

(1) $C \subseteq E'$;

(2) the first arrow of C belongs to $P \cap B(F)$;

(3) $Cross(C) = 1, Okay(C) = 1$.

We modify the Dijkstra algorithm in such a way that it constructs confluent sequences Q together with $|Q|$, $|Cross(Q)|$, $|First(Q)|$, $|Okay(Q)|$ satisfying (1) and (2). It stops if it finds a confluent cycle C with $Cross(C) = Okay(C) = 1$. If the number of arrows of G' is m, then *Cycle2* requires $O(m + f(m))$ time where $f(m)$ is the time needed for the operations with priority queue.

Corollaries 3.3 and 3.5 yield the algorithm used in [10]. It selects a directed path P from s to t, for every arrow $e \in P$ it finds a cut-cycle C containing e with the smallest $|C|$, and finally it computes a minimum over all arrows in P. In this paper we present a finer algorithm.

4. APPLICATION OF DIVIDE AND CONQUER

Let C and D be two cut-cycles. We say that they are *collision-free* if $Out(D) \subseteq Acc(s, D(C))$ or $Out(C) \subseteq Acc(s, D(D))$ and for every $a \to b \in D$ we have $b \to a \notin C$.

Let C and D be two collision-free cut-cycles with $Out(C) \subseteq Acc(s, D)$. Then the subgraph $Str(C, D) = (V, E')$ (not necessarily a full one) is said to be a *strip* with the source boundary D and the sink boundary C where

$$E' = \{x \to y \in E; x, y \in Acc(s, D) \cap Acc(C, t)\} \cup C \cup D.$$

A strip $Str(C, D)$ is *degenerated* if $C \cap D \neq \emptyset$.

A subgraph (V, E') of G is said to be a *strip* if there are two collision-free cut-cycles C and D such that $(V, E') = Str(C, D)$.

Lemma 4.1. Let $(V, E') = Str(D_1, D_2)$ for two collision-free cut-cycles D_1 and D_2. If C is a minimum $Str(D_1, D_2)$ cut-cycle, then for every $a \to b \in C$ we have $b \to a \notin D_1 \cup D_2$.

Proof. Let $a \to b \in C$ and $b \to a \in D_2$. Using Lemma 3.4 we can construct a path P_1 from s to a which does not cross C and another path P_2 from a to t with $P_2 \subseteq Acc(D_2, t)$. But the concatenation of those paths is a directed path from s to t having an empty intersection with C – a contradiction. The case when $b \to a \in D_1$ is analogous. □

Lemma 4.2. Let $(V, E') = Str(D_1, D_2)$ be a degenerated strip and let P be a directed path from s to t with $P \cap D_1 = P \cap D_2 = \{a \to b\}$. If $C \subseteq Str(D_1, D_2)$ is a confluent cycle containing $a \to b$, then C is a cut-cycle. Let F be a face with $a \to b \in B(F)$. If C is a minimum $(P, F, Str(D_1, D_2)$ cut-cycle, then it is a minimum $Str(D_1, D_2)$ cut.

Proof. Since $P \cap E' = \{a \to b\}$, we conclude, by Corollary 3.5 that every confluent cycle containing $a \to b$ is a cut and that every minimum $Str(D_1, D_2)$ cut-cycle must contain $a \to b$. □

By Lemma 4.2, Algorithm *Cycle1* with parameters $Str(D_1, D_2)$ and $a \to b$ for degenerated D_1 and D_2 computes a minimum $Str(D_1, D_2)$ cut-cycle and requires $O(m + f(m))$ time where m is the number of arrows in $Str(D_1, D_2)$.

Lemma 4.3. Let P be a directed path from s to t. Assume that D_1, D_2 are two collision free cut-cycles and put $(V, E') = Str(D_1, D_2)$. Let F be a face with $B(F) \cap P \cap E' \neq \emptyset$ and let C be a minimum $(P, F, Str(D_1, D_2))$ cut-cycle. Then D_1, C and C, D_2 are collision-free cut-cycles and there exists a minimum $Str(D_1, D_2)$ cut-cycle D with either $D \subseteq Str(D_1, C)$ or $D \subseteq Str(C, D_2)$ or $|D| = |C|$.

Proof. By Lemma 4.1 D_1, C and C, D_2 are collision-free cut-cycles. Let C' be a minimum $Str(D_1, D_2)$ cut-cycle, with $|C'| < |C|$. Define

$$B_1 = \{x \to y \in E; x \in Acc(s, D(C \cup C')), y \notin Acc(s, D(C \cup C'))\},$$

$$B_2 = \{x \to y \in E; y \in Acc(D(C \cup C'), t), x \notin Acc(D(C \cup C'); t)\}.$$

Analogously, as in the proof of Lemma 3.1, we obtain that both B_1 and B_2 are closed confluent sequences which are cuts contained in $C' \cup C$ because C and C' are cut-cycles. Choose $x \to y \in C \cap P \cap B(F)$, then we have that $x \to y \notin B_1 \cap B_2$ for otherwise $x \to y$ would belong to C'.

Hence either $x \to y \notin B_1$ or $x \to y \notin B_2$. Suppose that $x \to y \notin B_1$ (the second case is fully analogous) and assume that $B_1 = (u_1 \to v_1, ..., u_p \to v_p)$. Thus there is some $a \to b \in B_1 \cap C'$. Assume that there exists some $a' \to b' \notin B_1 - C'$ otherwise $B_1 = C'$ holds. Let i be an index such that $u_i \to v_i \in C', u_{i+1} \to v_{i+1} \notin C'$ and let j be the smallest index greater than i such that $u_j \to v_j \in C'$ (j taken modulo p). Then the sequence $(u_{i+1} \to v_{i+1}, ..., u_{j-1} \to v_{j-1})$ is contained in C, and there is a confluent sequence $(w_l \to z_l, ..., w_q \to z_q)$ contained in C' and connecting $u_i \to v_i$ and $u_j \to v_j$ ($u_{i+1} \to v_{i+1}$ and $w_l \to z_l$ border the face and are not confluent). Let C_1, or C_2 be confluent sequences obtained by the following substitution into C', or C:

$$w_l \to z_l, ..., w_q \to z_q \text{ by } u_{i+1} \to v_{i+1}, ..., u_{j-1} \to v_{j-1} \text{ or}$$

$$u_{i+1} \to v_{i+1}, ..., u_{j-1} \to v_{j-1} \text{ by } w_l \to z_l, ..., w_q \to z_q.$$

Clearly, C_1 and C_2 are closed confluent sequences.

Since $u_i \to v_i \in C \cap C'$, we obtain $u_i \in Acc(s, C \cup C')$, $v_i \in Acc(C \cup C', t)$, and hence, there exist directed paths P_1 from s to u_i and P_2 from v_i to t that do not cross $C \cup C'$. Let P be a directed path which is a concatenation of $P_1, u_i \to v_i$, and P_2. Since P crosses C_1 and C_2 only once, by Corollary 3.5 C_1 and C_2 are cuts. Since $x \to y \in C_2$, we conclude that $|C_2| \geq |C|, |C'| \leq |C_1|$ and so

$$\sum_{r=i+1}^{j-1} c(u_r \to v_r) = \sum_{r=l}^{q} c(w_r \to z_r).$$

Now by induction we obtain that $|C'| = |B_1|$ and $B_1 \subseteq Str(D_1, C)$. □

Lemma 4.4. Let $(V, E') = Str(D_1, D_2)$ and let $P = (s = x_0 \to ... \to x_p = t)$ be a directed path. If $\{x_i \to x_{i+1}\} \in D_1 \cap P$ and $\{x_j \to x_{j+1}\} \in D_2 \cap P$ with $i < j$, then for any cut-cycle $C \subseteq (V', E')$ there exists $k \in \{i, ..., j\}$ with $x_k \to x_{k+1} \in C \cap P$.

Proof. Clearly, there exist directed paths P_1 from s to x_i, P_2 from x_{j+1} to t that cross neither D_1 nor D_2. Let P' be the concatenation of $P_1, x_i \to x_{i+1} \to ... \to x_{j+1}$ and P_2. Then for $u \to v \in C \cap P'$ we have $u \to v = x_k \to x_{k+1}$ for some $k \in \{i, ..., j\}$. □

5. THE ALGORITHM

Let $P = (x_0 \to x_1 \to ... \to x_q)$ be a directed path in a graph then the *face length* of P is the smallest number k such that there exists a sequence $F_0, F_1, ..., F_{k-1}$ of faces of G with $P \subseteq \cup_{i=0}^{k-1} B(F_i)$. A sequence $\{F_i; i \in k\}$ is called a *face decomposition* of P. A *face diameter* of a network N is the smallest face length of directed paths from s to t.

We describe an auxiliary procedures *Path* which finds a directed path P from s to t with the smallest face length and a face decomposition of P. We use again the Dijkstra algorithm on G with a single source s. The algorithm constructs a directed path P from s with a face decomposition. The path P is labeled by the face length of P. Hence the algorithm finds a face diameter of N with a directed path P from s to t with the smallest face length. The face decomposition is constructed in a standard way. The algorithm *Path* requires $O(n + f(n))$ time where n is the number of vertices in V.

Finally, we describe auxiliary functions applied in the main algorithm.

Function *Minimum* – its inputs are three cut-cycles C_0, C_1, C_2 together with their values $|C_0|, |C_1|, |C_2|$. *Minimum* computes a cut-cycle $C \in \{C_0, C_1, C_2\}$ with the minimum value. Clearly, *Minimum* requires a constant time.

Function *Degener* – its inputs are two collision-free cut-cycles C, D. *Degener* decide whether $C \cap D \neq \emptyset$, and if $C \cap D \neq \emptyset$, then it finds $a \to b \in C \cap D$. To compute it, first *Degener* marks all vertices $x \in V$ with $x \to y \in C$ for some $y \in V$. Secondly, it decides whether there exists $x \in V$ with $x \to y \in C, x \to z \in D$ for some $y, z \in V$, and finally, it decides whether $y = z$. Since C and D are given as link lists, we see that *Degener* uses $O(|C| + |D|)$ time.

Function *Strip* – its inputs are a strip $Str(C, D)$, cut-cycle $C_1 \subseteq Str(C, D)$ such that C, C_1 and C_1, D are collision-free, and $i \in \{-1, 1\}$. *Strip* computes arrows in $Str(C, C_1)$ if $i = 1$, or arrows in $Str(C_1, D)$ if $i = -1$. By the depth-first-search it marks all vertices in $Acc(s, C_1) \cap Acc(C, t)$ or in $Acc(s, D) \cap Acc(C_1, t)$. Then by a systematic search it finds all arrows in $Str(C, C_1)$ or in $Str(C_1, D)$. Since G is planar, it requires $O(m)$ time where m is a number of edges in $Str(C, D)$.

Finally we describe a main auxiliary procedure *MinCyc*. We assume that a directed path P from s to t with a face decomposition $\{F_i; i \in \{0, 1, ..., q-1\}\}$ is given. An input is a strip $Str(D_1, D_2)$ and two numbers j, k with $0 \leq j \leq k \leq q, D_1 \cap P \cap B(F_j) \neq \emptyset \neq D_2 \cap P \cap B(F_k)$. *MinCyc* computes a minimum $Str(D_1, D_2)$ cut-cycle. We describe this procedure:

Apply *Degener* for D_1, D_2;
if D_1, D_2 are degenerated and $a \to b \in D_1 \cap D_2$ *then*
C_1 is a cut-cycle constructed by *Cycle1* for $a \to b, Str(D_1, D_2)$
else if $j = k$ *then*
C is a cut-cycle constructed by *Cycle2* for $P, F_j, Str(D_1, D_2)$
else if $k = j + 1$ *then*
C_1 is a cut-cycle constructed by *Cycle2* for $P, F_j, Str(D_1, D_2)$
C_2 is a cut-cycle constructed by *Cycle2* for $P, F_k, Str(D_1, D_2)$
C is a cut-cycle determined by *Minimum* from C_1, C_2, D_1
else r is an integer part of $(j + k)/2$
C_1 is a cut-cycle constructed by *Cycle2* for $P, F_r, Str(D_1, D_2)$
create $Str(D_1, C_1)$ by *Strip*
find the smallest $l > j$ with $C_1 \cap P \cap B(F_l) \neq \emptyset$
C_2 is a cut-cycle constructed by *MinCyc* for $P, Str(D_1, C_1), j, l$
create $Str(C_1, D_2)$ by *Strip*
find the biggest $l' < k$ with $C_1 \cap P \cap B(F_{l'}) \neq \emptyset$
C_3 is a cut-cycle constructed by *MinCyc* for $P, Str(C_1, D_2), l', k$
C is a cut-cycle determined by *Minimum* from C_1, C_2, C_3
endif endif endif
output: C is a minimum $Str(D_1, D_2)$ cut-cycle.

The correctness of the above procedure follows from Lemmas 4.2, 4.3 and 4.4.

Lemma 5.1. Suppose that the priority queue used in the Dijkstra algorithm requires $O(f(m))$ time for m operations, where $f(a+b) \geq f(a) + f(b)$ for every positive integers a, b. Then *MinCyc* requires $O((m + f(m)) \max\{1, \log(k - j)\})$ time where m is the number of arrows in $Str(D_1, D_2)$.

Proof. The statement will be proved by induction on $k - j$. If $k = j$, then *MinCyc* requires $O(m) + O(m + f(m)) = O(m + f(m))$ time, according to the time estimates for *Degener*, *Cycle1*. If $k = j+1$, then we analogously obtain that *MinCyc* requires $O(m) + O(m + f(m)) + O(m + f(m)) + O(1) = O(m + f(m))$ time. Assume that the statement holds for every $k - j < n$ where $n > 1$. From the time estimates for *Cycle1*, *Cycle2*, *Strip*, *Degener*, and *Minimum* and by induction assumption the procedure *MinCyc* requires $O(m) + O(m + f(m)) + O(1) + O(m + f(m)) + O(m) + O(m) + O((m_1 + f(m_1)) \log(l - j)) + O(m) + O(m) + O((m_2 + f(m_2)) \log(k - l')) + O(1) = O(m + f(m)) + O((m_1 + f(m_1)) \log(l - j)) + O((m_2 + f(m_2)) \log(k - l'))$, where m_1 is the number of arrows in $Str(D_1, C_1)$, m_2 is the number of arrows in $Str(C_1, D_2)$, since a finding of l and l' needs only $O(m)$ time. Since $l - j, k - l' \leq (k - j)/2$ and since $m_1 + m_2 = m$ implies $O(m + f(m)) = O(m_1 + f(m_1)) + O(m_2 + f(m_2))$ we conclude that *MinCyc* uses $O(m + f(m)) + O((m_1 + f(m_1)) \log(l - j)) + O((m_2 + f(m_2)) \log(k - l')) = O(m + f(m)) + O((m_1 + f(m_1))(\log(k - j) - 1)) + O((m_2 + f(m_2))(\log(k - j) - 1)) = O(m + f(m)) + O((m + f(m))(\log(k - j) - 1)) = O((m + f(m)) \log(k - j))$ time. □

Finally we describe the main algorithm *MinCut*

Apply *Path* – we obtain a path P with a face length p.
Apply *MinCyc* for $P, G = Str(C_s, C_t), 0, p-1$.

The correctness follows from the correctness of the algorithms *Path* and *MinCyc*.

From Lemma 5.1 and the time estimate for *Path* we obtain

Theorem 5.2. Assume that the face diameter of N is p. Let the priority queue used in the Dijkstra algorithm require $O(f(m))$ time for m operations, where $f(a+b) \geq f(a)+f(b)$ for every positive integers a, b. Then *MinCut* constructs a minimum cut-cycle and requires $O((n+f(n))\max\{1, \log(p)\})$ time, where n is number of arrows in G.

Corollary 5.3. Let N be a planar network with n vertices and a face diameter p. The algorithm *MinCut* constructs a minimum cut-cycle in N and requires:

(1) $O((n\log(n)/\log(\log(n)))\max\{1, \log(p)\}) = O(n\log^2(n)/\log(\log(n)))$ time; if, moreover, N is a $s-t-$network then *MinCut* requires $O(n\log(n)/\log(\log(n)))$ time;

(2) if the values of the capacity function c are nonnegative integers less than n^k, for a fixed k, then *MinCut* requires $O(n\log(\log(n))\max\{1, \log(p)\})$ time. If, moreover, N is a $s-t$-network then *MinCut* requires $O(n\log(\log(n)))$ time;

(3) if the values of the capacity function c are negative integers less than k, for a fixed k, then *MinCut* requires $O(n\max\{1, \log(p)\})$ time. If, moreover, N is a $s-t$-network then *MinCut* requires $O(n)$ time.

Proof. If we represent the priority queue by an AF-queue, see [6], then the amortization cost of operations INSERT and DECREASE is constant and the amortization cost of operation DELETE and MINDELETE is $O(\log(n)/\log(\log(n)))$, thus (1) is proved. In case (2) we represent a priority queue by a data structure suggested in [3]. Finally, in case (3), a priority queue is represented by a list, then an operation takes $O(1)$ time since the length of a list is bounded by k, and (3) is proved. □

Remark. Itai and Shiloach [9] worked in a more restricted computational model that uses only the arithmetical operations addition and subtraction. The AF-queue and the priority queue described in [3] does not work in such a model. Therefore (1) and (2) of Corollary 5.3 does not hold in this model. We must represent a priority queue in this model either as a (2,4)-tree or a heap, and then an implementation of the algorithm *MinCut* runs in $O(n\log(n)\max\{1, \log(p)\}) = O(n\log^2(n))$ time (for an $s-t$-network *MinCut* runs in $O(n\log(n))$ time).

(Received July 16, 1990.)

REFERENCES

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

[2] E. W. Dijkstra: A note on two problems in connections with graphs. Numer. Math. *1* (1959), 269 – 271.

[3] P. van Emde Boas, R. Kaas and E. Zijlstra: Design and implementation of an efficient priority queue. Math. Systems Theory *10* (1977), 99 – 127.

[4] L. R. Ford and D. R. Fulkerson: Flows in Networks. Princeton University Press, Princeton, N. J. 1962.

[5] L. M. Fredman and R. E. Tarjan: Fibonacci heaps and their uses in improved network optimization algorithms. J. Assoc. Comput. Mach. *34* (1987), 596 – 615.

[6] L. M. Fredman and D. E. Willard: Trans-dichotomous algorithms for minimum spanning trees and shortest paths. In: Proc. 31st FOCS, 1990, pp. 719 – 725.

[7] R. Hassin and D. B. Johnson: An $O(n \log^2 n))$ algorithm for maximum flow in undirected planar network. SIAM J. Comput. *14* (1985), 612 – 624.

[8] J. E. Hopcroft and R. E. Tarjan: Efficient planarity testing. J. Assoc. Comput. Mach. *21* (1974), 549 – 568.

[9] A. Itai and Y. Shiloach: Maximum flow in planar networks. SIAM J. Comput. *8* (1979), 135 – 150.

[10] L. Janiga and V. Koubek: A note on finding minimum cuts in directed planar network by parallel computations. Inform. Process. Lett. *21* (1985), 75 – 78.

[11] D. B. Johnson and S. Venkatesan: Using divide and conquer to find flows in directed planar networks in $O(n^{1.5} \log(n))$ time. In: Proc. 20th Annual Allerton Conf. of Communication, Control and Computing, 1982, pp. 898 – 905.

[12] K. Mehlhorn: Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness. EATCS Monographs on Theoretical Computer Science, Springer-Verlag, Berlin – Heidelberg – New York – Tokio 1984.

[13] O. Ore: Theory of Graphs. Amer. Math. Soc. Coll. Publ., Vol. XXXVIII, Providence, R. I. 1962.

[14] J. H. Reif: Minimum S-T cut of a planar undirected network on $O(n \log^2(n))$ time. In: Automata, Languages and Programming (S. Even, D. Kariv, eds., Lecture Notes in Computer Science 115), Springer-Verlag, Berlin – Heidelberg – New York – Tokio 1981, pp. 56 – 67.

RNDr. Ladislav Janiga, CSc., Moravanů 68, 169 00 Praha 6. Czechoslovakia.

RNDr. Václav Koubek, CSc., matematicko-fyzikální fakulta University Karlovy (Faculty of Mathematics and Physics – Charles University), Malostranské nám. 25, 118 00 Praha 1. Czechoslovakia.