アルゴリズム 54－5
（1996. 10. 17）

# 最小カット問題の簡潔かつ構成的な証明

永持 仁，石井 利昌，茨木 俊秀

京都大学 工学部 数理工学教室
606-01 京都市左京区吉田本町
e-mail: naga,ishii,ibaraki@kuamp.kyoto-u.ac.jp

**あらまし** [H. Nagamochi and T. Ibaraki, Computing edge-connectivity of multigraphs and capacitated graphs, SIAM J. Discrete Mathematics, 5, 1992, pp. 54-66] により提案された，最小カットを求めるアルゴリズムの正当性について，近年，いくつかの簡潔な証明が紹介されている．本研究では，これらとはまた別の簡潔な証明を示す．またこの証明は構成的であり，ある特別な 2 点対 $s$, $t$ 間の最大フローを $O(m \log n)$ 時間で求めるアルゴリズムを構築する ($n$, $m$ はそれぞれグラフの点数，辺数である)．

**和文キーワード：** 最小カット，最大フロー，辺連結度，合法的順序付け，動的木構造．

# A Simple and Constructive Proof of a Minimum Cut Algorithm

Hiroshi NAGAMOCHI, Toshimasa ISHII and Toshihide IBARAKI

Department of Applied Mathematics and Physics,
Graduate School of Engineering,
Kyoto University
Kyoto 606-01, Japan.

**Abstract** For the correctness of the minimum cut algorithm proposed in [H. Nagamochi and T. Ibaraki, Computing edge-connectivity of multigraphs and capacitated graphs, SIAM J. Discrete Mathematics, 5, 1992, pp. 54-66], several simple proofs have been presented recently. This paper gives yet another simple proof. As a byproduct, it can provide an $O(m \log n)$ time algorithm that outputs a maximum flow between a pair of vertices $s$ and $t$, which are selected by the algorithm, where $n$ and $m$ are numbers of the vertices and edges, respectively.

**英文 key words:** minimum cut, maximum flow, edge-connectivity, maximum adjacency ordering, dynamic tree structure.

# 1 Introduction

Let $G = (V, E, c_G)$ stand for an edge-weighted undirected graph with a set $V$ of *vertices* and a set $E$ of *edges* weighted by $c_G : E \to R^+$, where $R^+$ is the set of non-negative reals. An undirected edge $e$ with end vertices $u$ and $v$ is also denoted by $\{u, v\}$, and its weight $c_G(e)$ by $c_G(u, v)$ $(= c_G(v, u))$. For two graphs $G = (V, E, c_G)$ amd $G' = (V, E, c_{G'})$ in the same vertex set $V$ and edge set $E$, we write $G' \subseteq G$ if $c_{G'}(e) \leq c_G(e)$, $e \in E$. We denote $n = |V|$ and $m = |E|$. A singleton set $\{x\}$ may be written as $x$. For a nonempty subset $Z \subseteq V$, the subgraph induced from $G$ by $Z$ is denoted by $G[Z]$. For two nonempty disjoint subsets $X, Y \subset V$, define $E_G(X, Y) = \{\{u, v\} \in E \mid u \in X,\ v \in Y\}$, and $c_G(X, Y) = \sum_{e \in E_G(X,Y)} c_G(e)$. Let $E^+(G)$ denote the set of edges with positive weights. We say that two vertices $u$ and $v$ are connected if there is a path $P \subseteq E^+(G)$ between $u$ and $v$. A subset $Z \subseteq V$ is called a *component* of $G$ if any two vertices in $Z$ are connected and no vertex in $V - Z$ is connected to a vertex in $Z$.

We define a flow in an undirected graph $G = (V, E, c_G)$. For this, we regard $G$ as a directed graph $\vec{G} = (V, \vec{E})$, where $\vec{E}$ is a set of arcs (= directed edges) obtained by giving orientation either $(u, v)$ or $(v, u)$ (arbitrarily) to each edge $\{u, v\} \in E$. An arc with tail $u$ and head $v$ is denoted by $(u, v)$. For two specified vertices $s$ and $t$, a function $f : \vec{E} \to R$ is called a $(s,t)$-flow if $f$ satisfies $\sum_{(v,w) \in \vec{E}} f(v, w) = \sum_{(w,v) \in \vec{E}} f(w, v)$ for each $v \in V - \{s, t\}$ and $|f(v, w)| \leq c(v, w)$ for each edge $(v, w) \in E$, where $f(v, w) \geq 0$ (resp. $f(v, w) < 0$), $(v, w) \in \vec{E}$ means a flow from $v$ to $w$ (resp. from $w$ to $v$). The flow value of $f$ is defined by $\sum_{(s,w) \in \vec{E}} f(s, w) - \sum_{(w,s) \in \vec{E}} f(w, s)$ $(= -\sum_{(t,w) \in \vec{E}} f(t, w) + \sum_{(w,t) \in \vec{E}} f(w, t))$. An $(s,t)$-flow with the maximum value among all $(s,t)$-flows is called a *maximum* $(s,t)$*-flow*.

A *cut* is defined as a subset $X$ of $V$ with $\emptyset \neq X \neq V$, and the *size* of cut $X$ is defined by $c_G(X, V - X)$, which may also be written as $c_G(X)$. A cut with the minimum size is called a *(global) minimum cut*, and its size is called the *edge-connectivity* of $G$. The *local edge-connectivity* $\lambda_G(x, y)$ for two vertices $x, y \in V$ is defined to be the minimum size of a cut in $G$ that separates $x$ and $y$, or equivalently the value of a maximum $(x, y)$-flow [1].

The minimum cut algorithm by Nagamochi and Ibaraki [7] repeats finding a pair of vertices $v$ and $w$ such that

$$\lambda_G(v, w) = c_G(w), \tag{1}$$

and contracting those vertices into a single vertex, until the resulting graph has only one vertex. Clearly, the edge-connectivity of $G$ is equal to the minimum among all $c_G(w)$ that appeared during this process, i.e.,

$$\min\{c_{G_i}(w_i) \mid i = 1, 2, \ldots, n-1\}, \tag{2}$$

where $G_i$ is the graph in the $i$-th iteration and $(v_i, w_i)$ is the pair in $G_i$ satisfying (1). A global minimum cut in $G$ is given by the set of vertices that have been contracted into the vertex $w_{i^*}$ attaining the minimum in (2). Reference [7] proves that the pair of vertices $v$ and $w$ satisfying (1) in each iteration can be found in $O(m + n \log n)$ time, and the entire running time of the minimum cut algorithm is $O(n(m + n \log n))$, which is currently one of the best among existing deterministic algorithms.

Given $G = (V, E, c_G)$ (not necessarily connected), an ordering $v_1, v_2, \ldots, v_n$ of all vertices in $V$ is called a *maximum adjacency (MA) ordering* (also called *legal* in [3]) in $G$ if it satisfies $c_G(\{v_1, v_2, \ldots, v_i\}, v_{i+1}) = \max[c_G(\{v_1, v_2, \ldots, v_i\}, u) \mid u \in \{v_{i+1}, \ldots, v_n\}]$, $1 \leq i \leq n-1$. Such an ordering can be found in $O(m + n \log n)$ time [7].

**Lemma 1** [2, 4, 6, 7, 14] *For $G = (V, E, c_G)$, let $v_1, v_2, \ldots, v_n$ be an MA-ordering of all vertices in $G$. Then the last two vertices $v_{n-1}$ and $v_n$ satisfy*

$$\lambda_G(v_{n-1}, v_n) = c_G(v_n). \tag{3}$$

□

The original paper [7] handles only edges with positive weights, and shows in Lemma 5.1(2) that $\lambda_G(v_h, v_n) = c_G(v_n)$ holds for the vertex $v_h$ with the largest index $h$ such that $E_G(v_h, v_n) \neq \emptyset$, which seems weaker than (3). However, it can be easily extended to imply (3) by introducing edges with zero weight in its proof, because allowing edges $e = \{x, y\}$ with $c_G(x, y) = 0$ in algorithm CAPFOREST [7] does not affect the correctness of the lemma. However, the proof of the lemma was rather technical and complicated, since it first proved the case of rational-valued weights, and then extended the argument to real-valued weights. Since an MA-ordering has some other useful applications [3, 6, 10], many researchers have studied properties of an MA-ordering, and discovered some simpler proofs of Lemma 1 [2, 4, 14]. Also [12] generalizes the lemma to a symmetric submodular function $c_G : 2^V \to R^+$.

# 2 A New Proof of Lemma 1

We now present a new simple proof of Lemma 1. Our proof not only shows (3), but also provides an

efficient construction of a maximum $(v_{n-1}, v_n)$-flow (none of the previous proofs can do this).

We start with the following observation.

**Claim 1** *Let $v_1, \ldots, v_n (n \geq 2)$ be an MA-ordering in $G$. Then each component of $G$ consists of vertices $v_j, v_{j+1}, \ldots, v_h$ with consecutive indices. In particular, $v_{n-1}$ and $v_n$ belong to the same component if $c_G(v_n) > 0$.*

**Proof:** Assume that a component consists of vertices whose indices are not consecutive. Let $Z$ be the component that contains a vertex with the smallest index among such components. Then we can choose three vertices $v_i \in Z$, $v_{i+1} \notin Z$ and $v_j \in Z$ for some $1 \leq i < i+1 < j \leq n$. Let $Z'$ be the component containing $v_{i+1}$. By the choice of $Z$, $Z'$ contains no vertex $v_k$ with index $k \leq i$, and hence $c_G(\{v_1, \ldots, v_i\}, v_{i+1}) = 0$. On the other hand, $c_G(\{v_1, \ldots, v_i\}, v_j) > 0$ holds since $v_i$ and $v_j$ are in the same component. However, $c_G(\{v_1, \ldots, v_i\}, v_{i+1}) < c_G(\{v_1, \ldots, v_i\}, v_j)$ contradicts the definition of an MA-ordering. This proves the first statement in the claim, from which the second statement is immediate. □

Given an MA-ordering $v_1, v_2, \ldots, v_n$ in $G$, we write $u \leq u'$ if $u = v_i$ and $u' = v_{i'}$ for $i \leq i'$, and arrange the edges in $E_G(\{v_1, \ldots, v_{i-1}\}, v_i)$ $(i = 2, 3, \ldots, n)$ in the order of

$$e_{i,1} = \{u_{i,1}, v_i\}, \cdots, \ e_{i,r_i} = \{u_{i,r_i}, v_i\}, \qquad (4)$$

where $r_i = |E_G(\{v_1, \ldots, v_{i-1}\}, v_i)|$, so that $u_{i,1} \leq u_{i,2} \leq \cdots \leq u_{i,r_i}$ holds. Now given a real $\delta \geq 0$, we define the weight functions $c_{G_\delta}$ and $c_{\overline{G}_\delta}$ as follows: $c_{G_\delta}(u_{i,j}, v_i) = c_G(u_{i,j}, v_i)$ if $j \leq p_i - 1$, $c_{G_\delta}(u_{i,j}, v_i) = \delta - \sum_{1 \leq j \leq p_i - 1} c_G(u_{i,j}, v_i)$ if $j = p_i$, $c_{G_\delta}(u_{i,j}, v_i) = 0$ if $j > p_i$, and $c_{\overline{G}_\delta}(e) = c_G(e) - c_{G_\delta}(e)$ for all edges $e \in E$, where $p_i$ denotes the smallest index such that $\sum_{1 \leq j \leq p_i} c_G(u_{i,j}, v_i) \geq \delta$ (we interpret $p_i = r_i + 1$ if $\sum_{1 \leq j \leq r_i} c_G(u_{i,j}, v_i) < \delta$). We call the resulting graphs $G_\delta = (V, E, c_{G_\delta})$ and $\overline{G}_\delta = (V, E, c_{\overline{G}_\delta})$ *$\delta$-skeleton* and *$\delta$-skin* of $G$ (with respect to ordering $v_1, \ldots, v_n$), respectively. As already noted in [10] for unweighted graphs, we see the following property.

**Claim 2** *Let $v_1, \ldots, v_n (n \geq 2)$ be an MA-ordering in $G$, and $G_\delta = (V, E, c_{G_\delta})$ and $\overline{G}_\delta = (V, E, c_{\overline{G}_\delta})$ be the $\delta$-skeleton and the $\delta$-skin of $G$, where $\delta$ is real satisfying $0 \leq \delta \leq c_G(v_n)$. Then the same ordering $v_1, \ldots, v_n$ remains an MA-ordering in $G_\delta$ and $\overline{G}_\delta$.*

**Proof:** By the definition of $G_\delta$ and $\overline{G}_\delta$, $c_{G_\delta}(\{v_1, \ldots, v_{i-1}\}, u) = \min[\delta, c_G(\{v_1, \ldots, v_{i-1}\}, u)]$, $c_{\overline{G}_\delta}(\{v_1, \ldots, v_{i-1}\}, u) = \max[c_G(\{v_1, \ldots, v_{i-1}\}, u) - \delta, 0]$ for each $i$ $i = 2, \cdots, n$ and $u \in \{v_i, v_{i+1}, \ldots, v_n\}$. Then, for each $i = 2, \ldots, n$, we have

$c_{G_\delta}(\{v_1, ..., v_{i-1}\}, v_i) = \min[\delta, c_G(\{v_1, ..., v_{i-1}\}, v_i)]$
$= \min[\delta, \max\{c_G(\{v_1, ..., v_{i-1}\}, u) | u \in \{v_i, ..., v_n\}\}]$
$= \max\{\min[\delta, c_G(\{v_1, ..., v_{i-1}\}, u) | u \in \{v_i, ..., v_n\}]\}$
$= \max\{c_{G_\delta}(\{v_1, \cdots, v_{i-1}\}, u) \ | u \in \{v_i, \cdots, v_n\}\}$,
implying that ordering $v_1, \ldots, v_n$ is an MA-ordering in $G_\delta$. Similarly for $\overline{G}_\delta$. □

To prove Lemma 1, we assume without loss of generality that $c_G(v_n) > 0$ holds in a given MA-ordering $v_1, \ldots, v_n$ ($c_G(v_n) = 0$ trivially implies (3)). We consider the directed graph $\vec{G} = (V, \vec{E})$ obtained from $G$ by regarding each edge $\{v_j, v_i\} \in E$ $j < i$ as an arc $(v_j, v_i)$. For each $i = 2, \ldots, n$, let $First[E_G(\{v_1, \cdots, v_{i-1}\}, v_i)]$ denote the edge $\{u_{i'}, v_i\}$ with the smallest index $i'$ among the edges in $E_G(\{v_1, \ldots, v_{i-1}\}, v_i) \cap E^+(G)$ if $E_G(\{v_1, \cdots, v_{i-1}\}, v_i) \cap E^+(G) \neq \emptyset$ and let $First[E_G(\{v_1, \cdots, v_{i-1}\}, v_i)] = \emptyset$ otherwise (i.e., if $E_G(\{v_1, \cdots, v_{i-1}\}, v_i) \cap E^+(G) = \emptyset$). Let $Forest(G)$ be the forest $(V, E')$ with $E' = \{First[E_G(\{v_1, \cdots, v_{i-1}\}, v_i)] \mid i = 2, \cdots, n\}$.

By construction of $Forest(G)$ we see that there is no cycle $C$ with $C \subseteq Forest(G)$ in $Forest(G)$; i.e., $Forest(G)$ is a collection of trees.

Clearly, $c_{Forest(G)}(v_n) > 0$. Let $\delta > 0$ a real number which is smaller than the weight of any edge in $Forest(G)$. Then the $\delta$-skeleton of $G$ satisfies $G_\delta \subseteq Forest(G)$. Since Claims 1 and 2 imply that $v_{n-1}$ and $v_n$ are connected in $G_\delta$, $v_{n-1}$ and $v_n$ are connected by a path $P \subseteq E(Forest(G))$. Let $T^*$ be the tree in $Forest(G)$ that contains $v_n$ and $v_{n-1}$, and let $v_c$ be the least common ancestor of $v_n$ and $v_{n-1}$ in $T^*$. Clearly, $P \subseteq E(T^*)$ and $v_c$ is in $V(P)$. Let $\varepsilon > 0$ be the minimum value of the edge weights on $P$. Then a $(v_{n-1}, v_n)$-flow $f$ with value $\varepsilon$ in $Forest(G)$ is obtained as follows:

$$f(e) = \begin{cases} \varepsilon & \text{for } e \in \vec{E} \text{ on } P_1 \\ -\varepsilon & \text{for } e \in \vec{E} \text{ on } P_2 \\ 0 & \text{for } e \in \vec{E} \text{ that is not on } P_1 \cup P_2, \end{cases} \qquad (5)$$

where $P_1$ is the path between $v_n$ and $v_c$ and $P_2$ is the path between $v_{n-1}$ and $v_c$. For this $\varepsilon$, we consider the $\varepsilon$-skeleton $G_\varepsilon$ of $G$. Let $f_1 := f$, which is a $(v_{n-1}, v_n)$-flow, and let $\varepsilon_1 := \varepsilon$. Clearly,

$$c_{\overline{G}_{\varepsilon_1}}(v_n) = c_G(v_n) - \varepsilon_1$$

holds in the $\varepsilon_1$-skin $\overline{G}_{\varepsilon_1}$ of $G$, and, by Claim 2, the same ordering $v_1, \ldots, v_n$ remains an MA-ordering in $\overline{G}_{\varepsilon_1}$. Therefore, if $c_{\overline{G}_{\varepsilon_1}}(v_n) > 0$ still holds, then we can find another $(v_{n-1}, v_n)$-flow $f_2$ by applying the above procedure to a new $G' := \overline{G}_{\varepsilon_1}$.

Repeating this procedure until $c_{G'}(v_n) = 0$ holds in the resulting graph $G'$, we find a sequence of

$(v_{n-1},v_n)$-flows $f_1, f_2 \ldots, f_k$. The number of repetition of constructing $\varepsilon_i$-skeleton and $\varepsilon_i$-skin is at most $m$, since no edge weight increases during this process, and at least one edge with positive weight becomes zero weighted when an $(\varepsilon_1+\cdots+\varepsilon_i)$-skin of $G$ is constructed. By construction, we easily see that the resulting flow $f_1 + f_2 + \ldots + f_k$ is a $(v_{n-1}, v_n)$-flow of $G$, and the weight of this flow is equal to $c_G(v_n)$, proving $\lambda_G(v_{n-1}, v_n) = \sum_{j=1}^{k} \varepsilon_j = c_G(v_n)$.

# 3 Algorithm

The proof of Lemma 1 is constructive, from which is a polynomial time algorithm for computing a maximum $(v_{n-1}, v_n)$-flow is constructed. In this section, we present an $O(m \log n)$ time algorithm for finding a maximum $(v_{n-1}, v_n)$-flow from an MA-ordering $v_1, \cdots, v_n$ in an undirected graph $G$.

At first, we give an $O(mn)$ time algorithm called MAX-FLOW, which is a naive implementation of the proof of Lemma 1.

For this, we induce some notations. After the $i$-th iteration of the procedure in the proof of Lemma 1, we assume that a $(v_{n-1}, v_n)$-flow of $G$ with value $\delta = \varepsilon_1 + \varepsilon_2 + \cdots + \varepsilon_i$ has been obtained. For the current flow value $\delta$, consider a forest $T = Forest(\overline{G}_\delta)$ of the $\delta$-skin of $G$, and we denote as follows:

$cur(v_i) :=$
$\sum_{j=1}^{j'} c_G(u_{i,j}, v_i)$ $i = 2, \cdots, n$, where $\{u_{i,j}, v_i\}$ is defined as (4) and edge $\{u_{i,j'}, v_i\} \in E(T)$. ($cur(v_i)$ implies the current position of edge $\{u_{i,j'}, v_i\}$ in the list $E_G(\{v_1, \cdots, v_{i-1}\}, v_i)$.)

$Cap(e)$ : the current weight of edge $e \in E(T)$ (i.e., $Cap(e) = cur(v_i) - \delta$).

Let $T$ denote the current forest $Forest(\overline{G}_\delta)$ in the $\delta$-skin of $G$. In the $(i+1)$-th iteration of the procedure, we find a $(v_{n-1}, v_n)$-flow as follows. We find the path $P$ between $v_{n-1}$ and $v_n$ in the tree containing $v_{n-1}$ and $v_n$ in the $T$, and the minimum weight $\varepsilon_{i+1}$ on $P$. According to (5), we obtain a $(v_{n-1},v_n)$-flow with value $\delta + \varepsilon_{i+1}$. We then update the current $T$ to obtain $Forest(\overline{G}_{\delta+\varepsilon_{i+1}})$ as follows. We decrease the weight of edges on $P$ by $\varepsilon_{i+1}$, and for each edge $e = \{v_j, v_l\}$ $j < l$ whose value becomes zero, delete $e$ from the current $T$, and add the next element of the edge $e$ in the list $E_G(\{v_1, \cdots, v_{l-1}, v_l\})$ (if any). For each $e = \{v_j, v_l\}$ $j < l$ in the current $T$ where $e$ is not on $P$ and the current $cur(v_l) > \delta + \varepsilon_{i+1}$, we only decrease the weight of the edge by $\varepsilon_{i+1}$, i.e., $Cap(e) := cur(v_l) - (\delta + \varepsilon_{i+1})$. For each $e = \{v_j, v_l\}$ $j < l$ in the current $T$ where $e$ is not on $P$ and the current $cur(v_l) \leq \delta + \varepsilon_{i+1}$, we scan the edge list $E_G(\{v_1, \cdots, v_{l-1})$ in the order of (4) from the current edge $e$ to find edge $e' = \{v_{l,j'}, v_l\}$ where $\sum_{h=1}^{j'-1} c_G(v_{l,h}, v_l) \leq \delta + \varepsilon_{i+1}$ and $\sum_{h=1}^{j'} c_G(v_{l,h}, v_l) > \delta + \varepsilon_{i+1}$ and let the weight of $e'$ $Cap(e') := \sum_{h=1}^{j'} c_G(v_{l,h}, v_l) - (\delta + \varepsilon_{i+1})$. It is clearly followed from the definitions of $\delta$-skeleton and $\delta$-skin that edge $e'$ is in $Forest(\overline{G}_{\delta+\varepsilon_{i+1}})$ and its weight $Cap(e')$ is equal to the weight of the edge in $Forest(\overline{G}_{\delta+\varepsilon_{i+1}})$ that corresponds to $e'$.

**Procedure MAX-FLOW**

**Input:** an edge-weighted undirected graph $G = (V, E, c_G)$, an MA-ordering $v_1, \cdots, v_n$ of all vertices in $G$, the linked lists $E_G(\{v_1, \cdots, v_{i-1}\}, v_i)$ for all $i = 2, \cdots, n$, and a directed graph $\vec{G} = (V, \vec{E})$ defined in the proof of Lemma 1.

**Output:** a maximum flow $f(e)$ $e \in \vec{E}$ from $v_{n-1}$ to $v_n$.

```
1  begin
2    δ := 0; f(e):= 0 for each arc e ∈ Ē;
3    T := Forest(G), where E(Forest(G)) =
     {First[E_G({v_1,...,v_{i-1}}, v_i)] | i = 2,···,n};
4    Cap(e) := c_G(e) for each e ∈ E(T);
5    For i = 2,···,n, cur(v_i) := c_G(v_j, v_i) for edge
     {v_i, v_j} ∈ E(T) i > j (if any),
     cur(v_i) = 0 otherwise;
6    while δ < c_G(v_n) do
7      Find the least common ancestor v_c of v_n
       and v_{n-1} in T, and let P_1 (resp. P_2) be the
       path between v_c and v_n (resp. between v_c
       and v_{n-1}) in T;                              (a)
8      ε := min{Cap(e) | e ∈ P_1 ∪ P_2 };              (b)
9      f(e) := f(e) + ε for each arc e ∈ P_1;          (c)
10     f(e) := f(e) − ε for each arc e ∈ P_2;          (d)
11     Cap(e) := Cap(e) − ε for each e ∈ P_1 ∪ P_2;    (e)
12     For each e = {v_j, v_i} j < i where Cap(e)
       becomes zero in operation (e),                  (f)
13       Output f(v_j, v_i);
14       Delete e from T;
15       If the list E_G({v_1, ···, v_{i-1}}, v_i) has the
         next element e' = {v_{j'}, v_i} of {v_j, v_i} then
16         Insert the edge e' into T;
17         cur(v_i) := cur(v_i) + c_G(v_{j'}, v_i);
18         Cap(e') := c_G(e');
19       end {if}
20     δ := δ + ε;                                     (g)
21     For each edge e = {v_j, v_i} j < i that is not
       on P,
22       If cur(v_i) > δ then
23         Cap(e) := cur(v_i) − δ;                     (h1)
24       else then
25       while cur(v_i) ≤ δ do                         (h2)
26         Let {v_j, v_i} be the edge in E(T)
           with j < i;
27         Output f(v_j, v_i);
28         Delete edge {v_j, v_i} from T;
```

```
29          If the list E_G({v_1, ..., v_{i-1}}, v_i) has the
            next element e' = {v_{j'}, v_i} of {v_j, v_i}
              then
30            Insert e' in T;
31            cur(v_i) := cur(v_i) + c_G(v_{j'}, v_i);
32            If cur(v_i) > δ then
33              Cap(v_i) := cur(v_i) − δ;                (i)
34            end {if}
35          end {if}
36        end; {while}
37    end; {while}
38    each e ∈ E⃗(T), output f(e);
39 end. {MAX-FLOW}                                        □
```

The correctness of MAX-FLOW follows from the fact that the algorithm computes flow $f$ in the same way of the proof Lemma 1. Let us analyze the running time.

We can construct $Forest(G)$ in $O(n)$ time by picking up the first element in each list $E_G(\{v_1, \ldots, v_{i-1}\}, v_i)$, $i = 2, \cdots, n$ which stores the edges in the order of (4). The least common ancestor $v_c$ of $v_n$ and $v_{n-1}$ in $Forest(G)$ and the minimum weight in the path $P$ between $v_{n-1}$ and $v_n$ can be computed in $O(n)$ time. We will never scan any edge whose weight becomes zero once in the procedure. We say that such edge is saturated. Since one iteration of the outer while loop saturates at least one edge, the total number of iterations of the outer while loop is at most $m$. Hence (a)-(g) operations take $O(mn)$ time in the whole procedure. Before operations (h1)(h2), we can scan all edges and edge weights in the current $T$ in $O(n)$ time in one iteration of the outer while loop. In operations (h1)(i), updating operation can be computed in $O(1)$ time per one edge. The number of edges whose weights are updated in one iteration of the outer while loop is $O(n)$. Since the total number of iterations of the outer while loop is at most $m$, these operations take $O(mn)$ time in the whole procedure. In operation (h2), deletion operation takes $O(1)$ time per one edge. When we have deleted edge $\{v_j, v_i\}$ $j < i$ from $T$, we can scan the next element of the edge list $E_G(\{v_1, \cdots, v_{i-1}\}, v_i)$ in $O(1)$ time. Since we will never scan any edge that is deleted once, this operation takes $O(m)$ time in the whole procedure. From above, MAX-FLOW computes a maximum $(v_{n-1}, v_n)$-flow in $O(mn)$ time.

Now let us reduce this time complexity to $O(m \log n)$. For this, we consider an efficient implementation of MAX-FLOW. If we look up all edges in a path between $v_n$ and $v_{n-1}$ to find a $(v_{n-1}, v_n)$-flow in $Forest(G)$ in the while loop, each iteration would take $\Omega(n)$ time. To save this computation, we represent $Forest(G)$ in the dynamic tree structure due to Sleator and Tarjan[13], by which we can increase all weights of edges in a path in a tree by the same amount in $O(\log n)$ time. Based on this, each of operations (a)-(g) can be carried out in $O(\log n)$ time. Since edges that are deleted from $T$ once in the procedure will never be looked up again, we output a flow $f(e)$ for each $e \in \vec{E}$ when $e$ is deleted. When we update the current $T$ after we have obtained a $(v_{n-1}, v_n)$-flow with value $\delta$ in $T$, MAX-FLOW may possibly scan one edge $m$ times to update its weight by operations (h1)(i), from which updating the current forest may take $O(m^2)$ time in the whole procedure (in fact, since at most $n-1$ edges are updated in one iteration of the outer while loop, this updating operation takes $O(mn)$ time). For this, we improve MAX-FLOW as follows. We do not compute operations (h1)(i) in MAX-FLOW, and, for each edge $e = \{v_j, v_i\}$ $j < i$ in the current forest with the current $cur(v_i) > \delta$, we neither scan the edge $e$ nor update its edge weight except operation (e). That is, the number of scanning edges except operations (a)-(g) is at most one per one edge, since we scan edges only by deletion operation. Then the edge set of the current forest after each iteration of the outer while loop in this improved algorithm is equal to the edge set of that in MAX-FLOW, since operation (h1) implies that each edge $e = \{v_j, v_i\}$ $j < i$, $i = 2, \cdots, n$ always satisfies the current $cur(v_i) >$ the current $\delta$ and $cur(v_i) - c_G(v_j, v_i) \leq \delta$ in the improved algorithm. Since the weight of each edge in the current forest in this improved algorithm is at least the weight of that in MAX-FLOW, the following argument holds:

> suppose that we have obtained a $(v_{n-1}, v_n)$-flow with value $\delta$ after iterations of the outer while loop,
> the current forest $\supseteq Forest(\overline{G}_\delta)$.

That is, there is always a $(v_{n-1}, v_n)$-flow in the current forest after each iteration of the outer while loop until the degree of $v_n$ becomes zero in the improved algorithm, which implies that the correctness of the improved algorithm has been proved.

Let us review the dynamic trees. Given a set of rooted trees that are edge-disjoint each other and a vertex weight, this structure supports the following operations on a vertex $v$:

**AddPath($w, x$):** increase by $x$ each of the weights of all vertices (containing vertex $w$) on the path from vertex $w$ to the root such that $T$ contains vertex $w$.

**MinPath($w$):** return the minimum weight of the vertex that is on the path $P$ from vertex $w$ to the root of the tree $T$ which contains $w$, and return the vertex closest to the root of $T$ among those achieving this minimum in $P$.

**Cut($v, w$):** divide the tree containing vertices edge $\{v, w\}$ into two trees by deleting the edge $\{v, w\}$

from the tree.

**Link**$(v, w)$: link two trees which contain vertices $v$ and $w$, respectively by adding edge $\{v, w\}$ and make vertex $w$ the parent of vertex $v$.

**Common**$(v, w)$: return the least common ancestor of vertex $v$ and vertex $w$ in the tree that contains $v$ and $w$.

It is shown that any $m$ of the above operations can be carried out in $O(m \log n)$ time in the dynamic trees with $n$ vertices [13].

We describe algorithm MAX-FLOW by using the operations on dynamic trees. In preliminaries, we introduce some notations. In the procedure we always look up a set of trees denoted by the current $T$. Suppose that we have obtained a $(v_{n-1}, v_n)$-flow with value $\delta$ after some iterations of the outer while loop. The current $T$ is the same as $Forest(\overline{G}_\delta)$ except edge weights. Define $p(v)$ $(v \in V)$ the parent of $v$ in the current $T$. We use $cur(v_i)$ $i = 2, \cdots, n$ as defined in MAX-FLOW. For each $i = 2, \cdots, n$, $cap(v_i)$ corresponds to $Cap(v_j, v_i)$ in MAX-FLOW, where edge $\{v_j, v_i\}$ $j < i$ is the edge of $T$. For each $i = 2, \cdots, n$, $F(v_i)$ denotes a flow from $v_j$ to $v_i$ on edge $\{v_j, v_i\} \in E(T)$ $j < i$. In the following procedure, we consider a set of rooted trees and two types of vertex weights ($cap()$, $f()$) on $T$. For each type of weight, we prepare a distinct dynamic tree data structure. Since the trees $T$ are common to these two types of weights, we simply denote each operation on a type of weight without specifying the data structure which supports operations for the weight. For example, we denote AddPath($v_n$;$cap$) if we compute AddPath($v_n$) with respect to $cap()$.

The following algorithm MAX-FLOW on Dynamic Trees is outlined as follows. After $i$-th iteration of the procedure, we assume that a $(v_{n-1}, v_n)$-flow of $G$ with value $\delta = \varepsilon_1 + \varepsilon_2 + \cdots + \varepsilon_i$ has been obtained. Let $T$ denote the current forest $\subseteq Forest(\overline{G}_\delta)$. In the $(i+1)$-th iteration of the procedure, we find a $(v_{n-1}, v_n)$-flow as follows. We find the path $P$ between $v_{n-1}$ and $v_n$ in the tree containing $v_{n-1}$ and $v_n$ in the $T$, and the minimum weight $\varepsilon_{i+1}$ on $P$. According to (5), we obtain a $(v_{n-1},v_n)$-flow with value $\delta + \varepsilon_{i+1}$. We then update the current $T$ to obtain the forest $\subseteq Forest(\overline{G}_{\delta+\varepsilon_{i+1}})$ as follows. We decrease the weight of edges on $P$ by $\varepsilon_{i+1}$, and for each edge $e = \{v_j, v_l\}$ $j < l$ whose value becomes zero, delete $e$ from the current $T$, and add the next element of the edge $e$ in the list $E_G(\{v_1, \cdots, v_{l-1}, v_l\})$ (if any). We do not scan any edge $e = \{v_j, v_l\}$ $j < l$ in the current $T$ where $e$ is not on $P$ and the current $cur(v_l) > \delta + \varepsilon_{i+1}$. For each $e = \{v_j, v_l\}$ $j < l$ in the current $T$ where $e$ is not on $P$ and the current $cur(v_l) \leq \delta + \varepsilon_{i+1}$, we scan the edge list $E_G(\{v_1, \cdots, v_{l-1})$ in the order of (4) from the current edge $e$ to find edge $e' = \{v_{l,j'}, v_l\}$ where $\sum_{h=1}^{j'-1} c_G(v_{l,h}, v_l) \leq \delta + \varepsilon_{i+1}$ and $\sum_{h=1}^{j'} c_G(v_{l,h}, v_l) > \delta + \varepsilon_{i+1}$ and let the weight of $e'$ $c_G(v_{l,j'}, v_l)$. It is clearly followed from the definitions of $\delta$-skeleton and $\delta$-skin that edge $e'$ is in $Forest(\overline{G}_{\delta+\varepsilon_{i+1}})$ and its weight is at least the weight of the edge that corresponds to $e'$ in $Forest(\overline{G}_{\delta+\varepsilon_{i+1}})$.

### Procedure MAX-FLOW on Dynamic Trees

**Input:** an edge-weighted undirected graph $G = (V, E, c_G)$, an MA-ordering $v_1, \cdots, v_n$ of all vertices in $G$, the linked lists $E_G(\{v_1, \cdots, v_{i-1}\}, v_i)$ for all $i = 2, \cdots, n$ and a directed graph $\vec{G} = (V, \vec{E})$ defined as the proof of Lemma 1.

**Output:** a maximum flow $f(e)$, $e \in \vec{E}$ from $v_{n-1}$ to $v_n$.

```
1  begin
2    δ := 0; f(e) := 0 for each arc e ∈ Ē;
3    Make one-vertex tree {v_i} for each v_i i = 1,
     ···, n;
4    Make a dynamic tree T to use the first
     element of the list E_G({v_1, ···, v_{i-1}}, v_i);
     (That is, for each i  i = 2, ···, n, if
     First[E_G({v_1, ..., v_{i-1}}, v_i)] ≠ ∅, Link(v_i, u_i)
     for {u_i, v_i} = First[E_G({v_1, ..., v_{i-1}}, v_i)]. )
5    cur(v_i) := c_G(v_i, p(v_i)) for each i = 2, ···, n;
6    cap(v_i) := c_G(v_i, p(v_i)) for each i = 2, ···, n;
     cap(v_1) := ∞;
7    F(v_i) := 0 for each i = 2, ···, n;
8    while δ < c_G(v_n) do
9      v_c := Common(v_{n-1}, v_n);                    (a)
10     If p(v_c) exists then
11       v' := p(v_c) and Cut(v_c, p(v_c));
12     end {if}
13     cap(v_c) := ∞;
14     ε := min{MinPath(v_n;cap), MinPath(v_{n-1};
       cap)};                                          (b)
15     AddPath(v_n, ε;F);
16     F(v_c) := F(v_c) − ε;                           (c)
17     AddPath(v_{n-1}, −ε;F); F(v_c) := F(v_c) + ε;   (d)
18     AddPath(v_n, −ε;cap),
       AddPath(v_{n-1}, −ε;cap);                       (e)
19     If v' exists in (a) then
20       Link(v_c, v') and cap(v_c) := c_G(v_c, v');
21     end {if}
22     while MinPath(v_n; cap) = 0 do                  (f1)
23       Let v_j be a vertex that is found by
         MinPath(v_n, cap);
24       f(p(v_j), v_j) := F(v_j);
25       Cut(v_j, p(v_j));
26     end; {while}
27     while MinPath(v_{n-1}; cap) = 0 do              (f2)
28       Let v_j be a vertex that is found
         by MinPath(v_{n-1}, cap);
```

```
29      f(p(v_j), v_j) := F(v_j);
30      Cut(v_j, p(v_j));
31    end; {while}
32    For each v_j that is found from these two
      while loops,                                   (f3)
      If the list E_G({v_1, ..., v_{j-1}}, v_j) has the next
      element {u_j, v_j} then
33      Link(v_j,u_j);
34      cur(v_j) := cur(v_j) + c_G(u_j, v_j);
35      cap(v_j) := c_G(u_j, v_j);
36    else then cap(v_j) := ∞;
37    δ := δ + ε;                                    (g)
38    while min{cur(v_i) | i = 2, ..., n} ≤ δ do     (h)
39      Let v_h be a vertex with cur(v_h) =
        min{cur(v_i) | i = 2, ..., n} ≤ δ;
40      f(p(v_h), v_h) := F(v_h);
41      Cut(v_h, p(v_h));
42      If the list E_G({v_1, ..., v_{h-1}}, v_h) has the
        next element {u_h, v_h} then
43        Link(v_h, u_h);
44        cur(v_h) := cur(v_h) + c_G(u_h, v_h);
45        cap(v_h) := c_G(u_h, v_h);
46      else then cap(v_h) := ∞;
47    end; {while}
48  end; {while}
49  f(e) := F(v_j) for each e = (v_i, v_j) ∈ E⃗(T) i < j;
50 end. {MAX-FLOW on Dynamic Trees}                  □
```

**Lemma 2** *Algorithm MAX-FLOW on Dynamic Trees computes a maximum flow between $v_{n-1}$ and $v_n$ in $O(m \log n)$ time.*

**Proof:** Since the correctness of Algorithm MAX-FLOW on Dynamic Trees has been described, we consider the time complexity of this algorithm. All lists $E_G(\{v_1, \cdots, v_{i-1}\}, v_i)$, $i = 1, \cdots, n-1$ can be obtained in $O(m + n \log n)$ time.

Making the dynamic tree for an initial $T$ takes $O(n \log n)$ time, since we can make $T$ to call $n-1$ times Link(,) by looking up the first element of the list $E_G(\{v_1, \cdots, v_{i-1}\}, v_i)$ for each $i$ $(i = 1, \cdots, n-1)$.

We next consider the complexity in the outer while loop. Note that once the current $cap(v_j)$ becomes zero by AddPath(, ; $cap$) operation, or when "the current $cur(v_j) \leq$ the current $\delta$" holds, we do not have to look up the edge $(v_j, p(v_j))$ later. Since one iteration of the outer while loop clearly saturates at least one edge, the number of repetition is at most $m$.

Each of operations (a)-(e) and (g) takes $O(\log n)$ time since any of these operation can be carried out by performing at most nine tree operations. Therefore these operations takes $O(m \log n)$ time in the entire algorithm.

Now consider operations (f1), (f2), (f3), and (h), we will show that each operation takes $O(\log n)$ time. We store $cur(v_j)$ $(j = 2, \cdots, n)$ in the heap, by which finding a vertex for $\min\{cur(v_i) \mid i = 2, \cdots, n\} \leq \delta$ in operation (h) takes $O(1)$ time, and updating a $cur(v_j)$ can be done in $O(\log n)$ time. Then each of operations (f1)-(f3) and (h) consists of at most one tree operation and the update of $cur(\cdot)$, but both one tree operation and updating $cur(\cdot)$ takes $O(\log n)$ time. Since any edge that is saturated once will never be looked up later, these operations are computed $O(m)$ times. Therefore these operations takes $O(m \log n)$ time in the total iterations.

From above, the entire algorithm takes $O(m \log n)$ time. □

**Theorem 1** *For the vertices $v_{n-1}$ and $v_n$ in Lemma 1, a maximum $(v_{n-1}, v_n)$-flow can be computed in $O(m \log n)$ time.* □

It is known [11] that there is a *directed acyclic graph* $DAG_{s,t}$ that represents for all minimum cuts (not necessarily global minimum cuts) separating $s$ and $t$ can be obtained in $O(m+n)$ time from a given maximum $(s,t)$-flow. Hence, based on a maximum $(v_{n-1}, v)$ flow, we can find all other minimum cuts separating $v_{n-1}$ and $v_n$ in $O(m \log n)$ time.

Furthermore, it is known that there is a *cactus* $\Gamma(G)$ that represents all global minimum cuts in $G$, and that such a cactus representation $\Gamma(G)$ can be found in $O(nm + n \cdot TDAG_{s,t})$ time [8], where $TDAG_{s,t}$ denotes the time required to compute the $DAG_{s,t}$ for the minimum cuts separating some two vertices $s$ and $t$ ($s$ and $t$ are not necessarily specified in advance). Therefore, since the $DAG_{s,t}$ for some $s$ and $t$ can be computed in $O(m \log n)$ time, we can compute a cactus representation $\Gamma(G)$ in $O(nm \log n)$ time, although a slightly faster $O(nm \log(n^2/m))$ time algorithm based on the maximum flow algorithm is known for constructing a cactus representation [5].

From Lemma 1 and the property that the original MA-ordering remains an MA-ordering in $k$-skeleton and $k$-skin, we also observe the next.

**Remark 1** For a graph $G = (V, E, c_G)$, an MA-ordering $v_1, v_2, \ldots, v_n$ in $G$ and a real $k$ with $0 \leq k \leq c_G(v_n)$, a pair of $k$-skeleton $G_k$ and $k$-skin $\overline{G}_k$ of $G$ satisfy $\lambda_{G_k}(v_{n-1}, v_n) = k$ and $\lambda_{\overline{G}_k}(v_{n-1}, v_n) = c_G(v_n) - k$. □

Remark 1 plays a crucial role in designing an efficient algorithm for computing the *edge-connectivity function* $\Lambda_G(k)$ of a graph $G$ [9], where $\Lambda_G(k)$ is defined to be the smallest total amount of weights added to make $G$ $k$-edge-connected.

## References


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

[2] A. Frank, *On the edge-connectivity algorithm of Nagamochi and Ibaraki*, Manuscript, Laboratoire Artemis, IMAG, Université J. Fourier, Grenoble, March 1994.

[3] A. Frank, T. Ibaraki and H. Nagamochi, *On sparse subgraphs preserving connectivity properties*, J. Graph Theory, 17, 1993, 275–281.

[4] S. Fujishige, *A note on Nagamochi and Ibaraki's min-cut algorithm and its simple proofs by Stoer, Wagner and Frank*, Manuscript, Forschungsinstitut für Diskrete Mathmatik, Universität Bonn, June 1994.

[5] H. N. Gabow, *A representation for crossing set families with applications to submodular flow problems*, Proc. 4th SODA, 1993, 202–211.

[6] H. Nagamochi and T. Ibaraki, *A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph*, Algorithmica, 7, 1992, 583–596.

[7] H. Nagamochi and T. Ibaraki, *Computing edge-connectivity of multigraphs and capacitated graphs*, SIAM J. Disc. Math., 5, 1992, 54–66.

[8] H. Nagamochi and T. Kameda, *Constructing cactus representation for all minimum cuts in an undirected network*, Operations Research Society of Japan, 39, 1996, 135–158.

[9] H. Nagamochi, T. Shiraki and T. Ibaraki, *Computing edge-connectivity augmentation function in $\tilde{O}(nm)$ time*, Technical report, Information Proceeding Society of Japan, AL-53-7, 1996.

[10] T. Nishizeki and S. Poljak, *k-connectivity and decomposition of graphs into forests*, Disc. Appl. Math., 55, 1994, 295–301.

[11] J.C. Picard and M. Queyranne, *On the structure of all minimum cuts in a network and applications*, Math. Prog. Study, 13, 1980, 8–16.

[12] M. Queyranne, *A combinatorial algorithm for minimizing symmetric submodular functions*, Proc. 6th SODA, 1995, 98-101.

[13] D. D. Sleator and R. E. Tarjan, *A data structure for dynamic trees*, J. Comput. System Sci., 26, 1983, 362–391.

[14] M. Stoer and F. Wagner, *A simple min cut algorithm*, Lecture Notes in Computer Science 855, Springer-Verlag, 2nd ESA, 1994, 141–147.