Algorithmica
DOI 10.1007/s00453-009-9284-5

Efficient Algorithms for the Problems of Enumerating Cuts by Non-decreasing Weights

Li-Pu Yeh · Biing-Feng Wang · Hsin-Hao Su

Received: 8 August 2008 / Accepted: 27 January 2009

Abstract In this paper, we study the problems of enumerating cuts of a graph by non-decreasing weights. There are four problems, depending on whether the graph is directed or undirected, and on whether we consider all cuts of the graph or only s-t cuts for a given pair of vertices s, t. Efficient algorithms for these problems with $\tilde{O}(n^2m)$ delay between two successive outputs have been known since 1992, due to Vazirani and Yannakakis. In this paper, improved algorithms are presented. The delays of the presented algorithms are $O(nm\log(n^2/m))$. Vazirani and Yannakakis's algorithms have been used as basic subroutines in the solutions of many problems. Therefore, our improvement immediately reduces the running time of these solutions. For example, for the minimum k-cut problem, the upper bound is immediately reduced by a factor of $\tilde{O}(n)$ for $k = 3, 4, 5, 6$.

Keywords Algorithms · Graphs · Minimum cuts · Maximum flows · Suboptimal cuts · Enumeration

1 Introduction

Let $G = (V, E)$ be an edge-weighted, directed or undirected graph, where V is the vertex set and E is the edge set. Let $n = |V|$ and $m = |E|$. A *cut* is a partition of

This research is supported by the National Science Council of the Republic of China under grants NSC-97-2221-E-007-118. A preliminary version of this paper was presented at the *14th Annual International Computing and Combinatorics Conference*.

L.-P. Yeh · B.-F. Wang (✉) · H.-H. Su
Department of Computer Science, National Tsing Hua University, Hsinchu, 30043 Taiwan, ROC
e-mail: bfwang@cs.nthu.edu.tw

L.-P. Yeh
e-mail: dr928304@cs.nthu.edu.tw

H.-H. Su
e-mail: u941591@oz.nthu.edu.tw

Published online: 14 February 2009

the vertex set V into two non-empty disjoint subsets. The *weight* of a cut (X, Y) is the total weight of the edges that go from X to Y. Let $s, t \in V$ be two vertices. An *s-t cut* is a cut (X, Y) such that $s \in X$ and $t \in Y$. The *minimum cut problem* is to find a cut of minimum weight and the *minimum s-t cut problem* is to find an s-t cut of minimum weight. Efficient algorithms for these two problems have numerous real-world applications such as generating traveling salesperson cutting planes, parallel computing, clustering, VLSI design, and network reliability [1, 3, 8, 9]. The most fundamental tool for solving the minimum cut and the minimum s-t cut problems is the maximum flow computation. For the computation, Goldberg and Tarjan [6] had an $O(nm \log(n^2/m))$-time algorithm and King, Rao, and Tarjan [18] had an $O(nm \log_{m/n \log n} n)$-time algorithm. As a consequence of the well-known maximum-flow minimum-cut theorem [1], the minimum s-t cut problem can be solved in $\tilde{O}(nm)$ time, where $\tilde{O}(f)$ denotes $O(f \log^c f)$ for some constant c. For the minimum cut problem, Hao and Orlin [9] had an $O(nm \log(n^2/m))$-time algorithm. For undirected graphs, better results for the minimum cut problem are known. Nagamochi and Ibaraki [20] gave an $O(nm + n^2 \log n)$-time algorithm and Karger [15] gave an $O(m \log^3 n)$-time randomized algorithm.

In many important applications, such as the all terminal network reliability problem, the vertex packing problem, and the maximum closure problem, finding all minimum cuts or nearly minimum cuts might be more useful than finding one minimum cut [4, 5, 17, 26]. For both directed and undirected graphs, the number of minimum s-t cuts can be exponential [26]. Picard and Queyranne [26] showed that after a maximum flow computation, all minimum s-t cuts of a directed or undirected graph can be enumerated with $O(n)$ delay between two successive outputs. For directed graphs, the number of minimum cuts can also be exponential [28]. It is easy to extend Picard and Queyranne's algorithm in [26] to enumerate all minimum cuts of a directed graph. Dinits, Karzanov, and Lomonosov [4] showed that the number of minimum cuts of an undirected graph is $O(n^2)$. For the problem of finding all minimum cuts of an undirected graph, Nagamochi, Nakamura, and Ishii [25] had an $O(nm + n^2 \log n)$ time algorithm, and Karger [15] had an $O(n^2 \log n)$-time randomized algorithm. An *α-minimum cut* is a cut of weight at most α times the minimum, where $\alpha \geq 1$ is a constant. Karger [15] showed that the number of α-minimal cuts of an undirected graph is $O(n^{\lfloor 2\alpha \rfloor})$. For the problem of finding all α-minimal cuts of an undirected graph, Nagamochi, Nishimura, and Ibaraki [23] had an $O(nm + n^{\lfloor 2\alpha \rfloor} m)$-time algorithm, and Karger and Stein [16] had an $O(n^{2\alpha} \log^2 n)$-time randomized algorithm.

Vazirani and Yannakakis [28] introduced the problems of enumerating cuts of a graph by non-decreasing weights. There are four problems, depending on whether the graph is directed or undirected, and on whether we consider all cuts of the graph or only s-t cuts for a given pair of vertices s, t. These enumeration problems were motivated by an application in studying the reliability and connectivity of networks [28]. For each of the problems, Vazirani and Yannakakis gave an efficient algorithm that requires at most $n - 1$ maximum flow computations between two successive outputs. Since a maximum flow computation can be done in $\tilde{O}(nm)$ time, the delays of their algorithms are $\tilde{O}(n^2 m)$. In this paper, for each of the enumeration problems, an improved algorithm is presented. The delays of the presented algorithms

are $O(nm\log(n^2/m))$. Our algorithms have the same schema as Vazirani and Yannakakis's algorithms. Our improvement is based on a delicate application of Hao and Orlin's minimum cut algorithm to their framework. Vazirani and Yannakakis's algorithms have been used as basic subroutines in the solutions of many problems. Therefore, our improvement reduces the running time of these solutions. For example, for the minimum k-cut problem, the upper bound is immediately reduced by a factor of $\tilde{O}(n)$ for $k = 3, 4, 5, 6$.

The rest of this paper is organized as follows. Notation and preliminaries are given in the next section. In Sect. 3, we review Vazirani and Yannakakis's algorithm for the problem of enumerating all cuts of a directed graph. In Sect. 4, an improved algorithm is presented for the same problem. In Sect. 5, we show how to modify the algorithm in Sect. 4 so as to solve the other three problems. In Sect. 6, we describe existing algorithms whose running time can be immediately reduced by our enumeration algorithms. Finally, in Sect. 7, we conclude this paper.

2 Preliminaries

Let $G = (V, E)$ be a directed or undirected graph, where V is the vertex set and E is the edge set. Let $n = |V|$ and $m = |E|$. Each edge $(u, v) \in E$ has a nonnegative real weight $w(u, v)$. A *cut* of G is a partition of the vertices into two non-empty subsets X and Y. For any cut (X, Y), we say that the vertices in X are on the *source side* and the vertices in Y are on the *sink side*. Let $C(G)$ be the set of cuts of G. If G is directed, the *weight* of a cut (X, Y) is the total weight of the edges going from X to Y; otherwise, it is the total weight of the edges having one end vertex in X and the other in Y. A *minimum cut* of G is a cut of minimum weight. As an illustrative example, consider the undirected graph in Fig. 1. In this example, $(\{v_1, v_2, v_4, v_5\}, \{v_3, v_6, v_7\})$ is a minimum cut and its weight is 9. For convenience, in this paper, we usually omit set braces around singletons, writing, for example, v instead of $\{v\}$.

Let S, T be two disjoint subsets of V. An *S-T cut* is a cut (X, Y) such that $S \subseteq X$ and $T \subseteq Y$. A *minimum S-T cut* is an S-T cut of minimum weight. The *partially specified cut set* with respect to (S, T) is defined as $P(S, T) = \{(X, Y) | (X, Y)$ is an S-T cut of $G\}$. Let $m(S, T)$ be a minimum cut in $P(S, T)$. By definition, $m(S, T)$ is just a minimum S-T cut of G. If S and T are non-empty, a minimum S-T cut can be found by a maximum flow computation. Thus, we have the following.

Lemma 1 [1] *Let S, T be two disjoint subsets of V. If S, T are non-empty, a minimum S-T cut can be found by a maximum flow computation.*

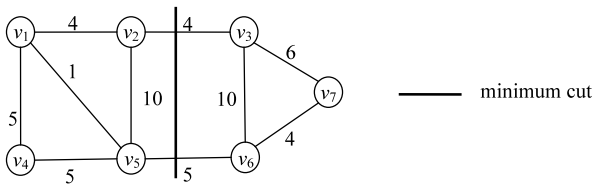


Fig. 1 A minimum cut of an undirected graph G

3 Vazirani and Yannakakis's Algorithm for Enumerating All Cuts of a Directed Graph

In this section, we review Vazirani and Yannakakis's algorithm for enumerating all cuts of a directed graph $G = (V, E)$.

The vertices in V are numbered from 1 to n. Each cut (X, Y) of G is represented by an n-bit binary string $b_1 b_2 \dots b_n$ as follows: $b_i = 0$ if and only if vertex $i \in X$. Consider a complete binary tree of height n. (See Fig. 2 for an example of $n = 4$.) Its leaves are named in the standard way by binary strings of length n, internal nodes at depth $k \geq 1$ are named by binary strings of length k, and root is named by the empty string ε. The root represents the cut set $C(G)$; and, each node $v = b_1 b_2 \dots b_k$ represents the partially specified cut set $P(S, T)$, where $S = \{i \mid b_i = 0, 1 \leq i \leq k\}$ and $T = \{i \mid b_i = 1, 1 \leq i \leq k\}$. For each internal node v, let $P(v)$ denote the partially specified cut set represented by v and let $m(v)$ denote the minimum cut in $P(v)$. Let $v = b_1 b_2 \dots b_i$ be a node and $l = b_1 b_2 \dots b_i b_{i+1} \dots b_n$ be a leaf in the subtree rooted at v. For $i < k \leq n$, we call $b_1 b_2 \dots b_{k-1} \overline{b}_k$ an *immediate child* of the path from v to l. An illustration is given in Fig. 2, in which $n = 4$, $v = 1$, $l = 1011$, and the immediate children of the path from v to l are 11, 100, and 1010. Clearly, the partially specified cut sets represented by the immediate children of the path from v to l form a partition of $P(v) - \{l\}$.

Vazirani and Yannakakis's algorithm is as follows. A heap Π is used to store a set of partially specified cut sets $P(v)$, using $m(v)$ as keys. In the course of the algorithm, all $P(v)$ in Π form a partition of the cuts which have not been output. Initially, Π contains only $P(\varepsilon)$, which is just the cut set $C(G)$. At each step, we extract a partially specified cut set, say $P(v)$, from Π, and output $m(v)$. Then, for each immediate child u on the path from v to $m(v)$, we compute $m(u)$ and then insert $P(u)$ into Π. The partially specified cut sets represented by the immediate children of the path from v to $m(v)$ form a partition of $P(v) - \{m(v)\}$. Therefore, $m(v)$ is excluded from further consideration.

Let x^k be the sequence of x repeated k times, where x is 0 or 1. For example, $1^5 = 11111$. Vazirani and Yannakakis's algorithm is formally described as follows.

Algorithm 1 (Enumeration_Vazirani_Yannakakis)

Input: a directed graph $G = (V, E)$
Output: all cuts of G in the order of non-decreasing weights

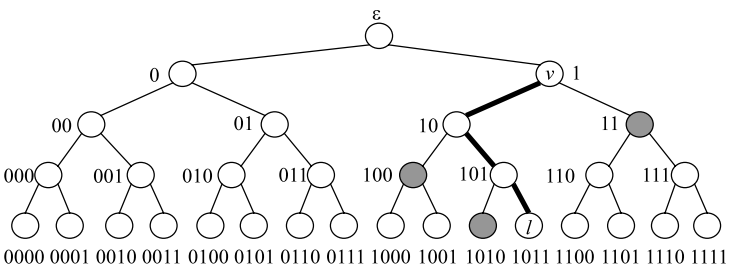


Fig. 2 Immediate children

begin

1. **for** $k \leftarrow 1$ **to** $n-1$ **do** compute $m(0^k)$ and $m(1^k)$
2. $\Pi \leftarrow \{(P(\varepsilon), m(\varepsilon))\}$ $\quad /* P(\varepsilon) = C(G), m(\varepsilon)$ is the minimum cut of G
3. **while** $\Pi \neq \emptyset$ **do**
4. **begin**
5. $\quad (P(v), m(v)) \leftarrow$ the element in Π with minimum $m(v)$
6. $\quad \Pi^* \leftarrow \{(P(u), m(u)) | u$ is an immediate child on the path from v to $m(v)\}$
7. $\quad \Pi \leftarrow \Pi - (P(v), m(v)) \cup \Pi^*$ $\quad /*$ delete $P(v)$ and insert a partition of $P(v) - m(v)$
8. $\quad$ **output** $(m(v))$
9. **end**

end

Lines 1 and 2 are initialization steps. The computation of $m(0^k)$ in line 1 is done as follows. First, by using $n-1$ maximum flow computations, we compute c_i as a minimum $\{1, 2, \ldots, i\} - \{i+1\}$ cut for $1 \leq i < n$. Then, we compute each $m(0^k)$ as the minimum cut in $\{c_k, c_{k+1}, \ldots, c_{n-1}\}$. The computation of $m(1^k)$ in line 1 is done similarly. Since $m(\varepsilon)$ can be computed as the smaller one in $\{m(0), m(1)\}$, line 2 requires $O(1)$ time. Therefore, the initialization steps require $2n-2$ maximum flow computations. The delay between two successive outputs is analyzed as follows. The computation of all $m(u)$ in line 6 is the bottleneck. For each u, if $u = 0^k$ or $u = 1^k$ for some integer k, $m(u)$ was found in line 1; otherwise, according to Lemma 1, it can be computed by using a maximum flow computation. Since $|\Pi^*| \leq n-1$, line 6 requires at most $n-1$ maximum flow computations. Each maximum flow computation can be done in $\tilde{O}(nm)$ time [6, 18]. Thus, we have the following.

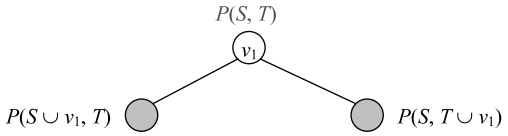
Theorem 1 [28] *The cuts of a directed graph can be enumerated in the order of non-decreasing weights with $\tilde{O}(n^2 m)$ time delay between two successive outputs.*

4 An Improved Algorithm for Enumerating All Cuts of a Directed Graph

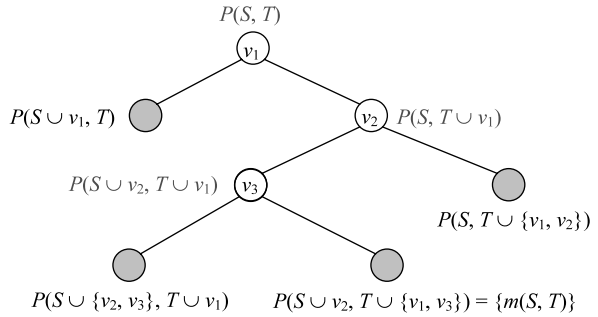
In this section, an improved algorithm is proposed for enumerating all cuts of a directed graph. The delay between two successive outputs is $O(nm \log(n^2/m))$.

4.1 The Algorithm

Let $S, T \subset V$ be two disjoint subsets. By using a vertex v in $V - (S \cup T)$, we can partition the partially specified cut set $P(S, T)$ into two disjoint subsets $P(S \cup v, T)$ and $P(S, T \cup v)$. Let $U = (v_1, v_2, \ldots, v_{n-|S|-|T|})$ be a sequence of the vertices in $V - (S \cup T)$. The *extract-min partition* of $P(S, T)$ induced by U is a partition obtained as follows: First, partition $P(S, T)$ into two subsets by using the vertex v_1; then, recursively, partition the subset containing $m(S, T)$ by the sequence $(v_2, v_3, \ldots, v_{n-|S|-|T|})$. An illustration is given in Fig. 3. Note that in an extract-min partition of $P(S, T)$, the subset containing $m(S, T)$ is a singleton.



(a) partition $P(S, T)$ into two subsets by v_1



(b) recursively partition $P(S, T \cup v_1)$ into three subsets by (v_2, v_3)

Fig. 3 Extract-min partition, where $U = (v_1, v_2, v_3)$ and $m(S, T) = (S \cup v_2, T \cup \{v_1, v_3\})$

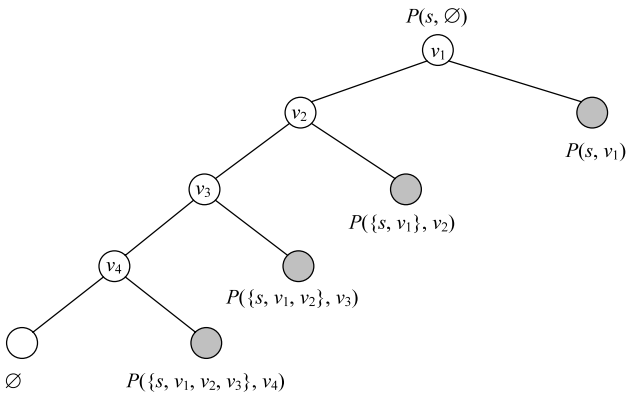


Fig. 4 A basic partition of $P(s, \emptyset)$, where $U = \{v_1, v_2, v_3, v_4\}$

Select an arbitrary vertex s. We partition $C(G)$ into two subsets $P(s, \emptyset)$ and $P(\emptyset, s)$. Let $U = (v_1, v_2, \ldots, v_{n-1})$ be a sequence of the vertices in $V - s$. The *basic partition* of $P(s, \emptyset)$ induced by U is $\{P(\{s, v_1, \ldots, v_{i-1}\}, v_i) | 1 \leq i \leq n-1\}$. An illustration is given in Fig. 4. Note that every subset in a basic partition of $P(s, \emptyset)$ contains non-empty source and sink sides. Similarly, define the *basic partition* of $P(\emptyset, s)$ induced by U as $\{P(v_i, \{s, v_1, \ldots, v_{i-1}\}) \mid 1 \leq i \leq n-1\}$.

We are ready to present a high-level description of our enumeration algorithm. It is as follows.

Algorithm 2 (Enumeration_Directed_Graph)

Input: a directed graph $G = (V, E)$
Output: all cuts of G in the order of non-decreasing weights
begin

1. $\Pi \leftarrow$ Basic_Partition /* Π stores a partition of $C(G)$ and the minimum cuts of its subsets
2. **while** $\Pi \neq \emptyset$ **do**
3. **begin**
4. $(P(S,T), m(S,T)) \leftarrow$ the element in Π with minimum $m(S,T)$
5. $\Pi^* \leftarrow$ Extract_Min_Partition $(P(S,T), m(S,T))$
6. $\Pi \leftarrow \Pi - (P(S,T), m(S,T)) \cup \Pi^*$
7. **output** $(m(S,T))$
8. **end**

end

Procedure (Basic_Partition)
begin

1. $B_0 \leftarrow$ a basic partition of $P(s, \emptyset)$
2. $B_1 \leftarrow$ a basic partition of $P(\emptyset, s)$
3. **for** each $P(S,T) \in B_0 \cup B_1$ **do** compute $m(S,T)$
4. **return** $(\{(P(S,T), m(S,T)) | P(S,T) \in B_0 \cup B_1\})$

end

Procedure (Extract_Min_Partition $(P(S,T), m(S,T))$)
begin

1. $R \leftarrow$ an extract-min partition of $P(S,T)$
2. **for** each $P(S',T') \in R - \{m(S,T)\}$ **do** compute $m(S',T')$
3. **return** $(\{(P(S',T'), m(S',T')) | P(S',T') \in R - \{m(S,T)\})$

end

Algorithm 2 uses the same schema as Algorithm 1. There are two differences. First, to avoid handling partially specified cut sets with empty source or sink sides, in Algorithm 2, the partially specified cut sets initially stored in Π are elements of basic partitions of $P(s, \emptyset)$ and $P(\emptyset, s)$. Such an initialization is done by Basic_Partition. The second difference is as follows. Let $P(S,T)$ be the partially specified cut set in Π with minimum $m(S,T)$ at some iteration of Algorithms 1 or 2. Algorithm 1 numbers the vertices from 1 to n at the beginning. To extract the minimum cut $m(S,T)$ from $P(S,T)$, Algorithm 1 partitions $P(S,T)$ by using the vertices in $V - (S \cup T)$ increasingly. That is, the sequence of vertices used to partition $P(S,T)$ is predetermined. Therefore, in Algorithm 1, it always holds that $S \cup T = \{1, 2, \ldots, |S| + |T|\}$. Algorithm 2 does not predetermine the sequence. Instead, it allows the flexibility of

partitioning $P(S, T)$ by using any sequence of the vertices in $V - (S \cup T)$, which is done in line 1 of Extract_Min_Partition. Our improvement is based on such flexibility.

The detailed implementations of Basic_Partition and Extract_Min_Partition are described, respectively, in Sects. 4.2 and 4.3.

4.2 Basic Partition

We only describe the computation of B_0 and the minimum cuts in its subsets. The computation of B_1 and the minimum cuts in its subsets is done similarly. A simple implementation is as follows: Select an arbitrary sequence U of the vertices in $V - s$, compute B_0 as the basic partition of $P(s, \emptyset)$ induced by U, and then compute the minimum cut of each subset by a maximum flow computation. Such an implementation needs $n - 1$ maximum flow computations and thus requires $\tilde{O}(n^2 m)$ time. In the following, an $O(nm \log(n^2/m))$-time implementation is presented.

The trick here is to select a specific sequence U. Hao and Orlin [9] had an efficient algorithm for computing a minimum cut of a directed graph. We determine the sequence U by making use of their algorithm. Given a directed graph $G = (V, E)$, Hao and Orlin's algorithm finds a minimum cut as follows. First, select an arbitrary vertex $s \in V$. Then, compute a minimum cut C_1 subject to the condition that s is on the source side. And then, compute a minimum cut C_2 subject to the condition that s is on the sink side. Clearly, the smaller one of C_1 and C_2 is a minimum cut. The cut C_2 is computed by firstly reversing each edge of G and then applying the same computation of C_1. The computation of C_1 is described below. First, set $S = \{s\}$ and $T = V - s$. Then, repeatedly, select a sink vertex $t \in T$, compute a minimum S-t cut, and then transfer t from T to S until T is empty. In total, $n - 1$ cuts are computed. Hao and Orlin showed that the $n - 1$ cuts can be computed in $O(nm \log(n^2/m))$ time if we select the sink vertex t in a careful way at each time. Finally, C_1 is computed as the smallest one of the $n - 1$ cuts. Let $(v_1, v_2, \ldots, v_{n-1})$ be the sequence of vertices in the order of their selection as sinks during the above computation of C_1. Then, the ith cut being computed is a minimum $\{s, v_1, \ldots, v_{i-1}\}$-$v_i$ cut of G, $1 \le i \le n - 1$. Therefore, we have the following.

Lemma 2 [9] *Given a directed graph $G = (V, E)$ and a vertex $s \in V$, we can determine a sequence $(v_1, v_2, \ldots, v_{n-1})$ of the vertices in $V - s$ and compute the cuts $m(\{s, v_1, \ldots, v_{i-1}\}, v_i)$, $i = 1, 2, \ldots, n - 1$, in $O(nm \log(n^2/m))$ time.*

According to Lemma 2, we implement the computation of B_0 as follows. First, by using Hao and Orlin's algorithm, we determine a sequence $U = (v_1, v_2, \ldots, v_{n-1})$ of the vertices in $V - s$ and compute the cuts $m(\{s, v_1, \ldots, v_{i-1}\}, v_i)$, $i = 1, 2, \ldots, n-1$. Then, we compute B_0 as the basic partition of $P(s, \emptyset)$ induced by U. The overall time complexity is $O(nm \log(n^2/m))$. Therefore, we have the following.

Lemma 3 *Basic_Partition can be implemented in $O(nm \log(n^2/m))$ time.*

4.3 Extract-min Partition

It is easy to implement Extract_Min_Partition in $O(n^2 m \log(n^2/m))$ time by using maximum flow computations. In this section, we show that Extract_Min_Partition can be efficiently implemented in $O(nm \log(n^2/m))$ time. We need some more notation and definitions. A *flow network* $G = (V, E)$ is a directed graph in which each edge $(u, v) \in E$ has a nonnegative real weight $w(u, v)$. For convenience, if $(u, v) \notin E$, we assume that $w(u, v) = 0$. Let $S, T \subset V$ be two disjoint non-empty subsets. An *S-T flow* in G is a real-valued function $f : V \times V \to R$ that satisfies the following three properties [1]:

- *Capacity constraint*: $f(u, v) \leq w(u, v)$ for all $u, v \in V$.
- *Antisymmetry constraint*: $f(u, v) = -f(v, u)$ for all $u, v \in V$.
- *Flow conservation constraint*: $\sum_{v \in V} f(u, v) = 0$ for all $u \in V - (S \cup T)$.

The quantity $f(u, v)$ is called the *flow* from vertex u to vertex v. The *value* of a flow f is the total flow out from the vertices in S. Given a flow f and a pair of vertices $u, v \in V$, the *residual weight* of (u, v) is given by $r_f(u, v) = w(u, v) - f(u, v)$. Given a flow network G and a flow f, the *residual network* of G induced by f is $G_f = (V, E_f)$, where E_f is the set of edges (u, v) in E with $r_f(u, v) > 0$. The weight of each $(u, v) \in E_f$ is $r_f(u, v)$. According to the well-known maximum-flow minimum-cut theorem, we have the following two lemmas.

Lemma 4 *Let f be a maximum S-T flow and (X, Y) be a minimum S-T cut. Then, $r_f(u, v) = 0$ for any $u \in X$ and $v \in Y$.*

Lemma 5 *Let (X, Y) be an S-T cut. If there is an S-T flow f such that $r_f(u, v) = 0$ for any $u \in X$ and $v \in Y$, then (X, Y) is a minimum S-T cut.*

From Lemmas 4 and 5, it is easy to conclude that following:

Lemma 6 *Let f be a maximum S-T flow and (X, Y) be a minimum S-T cut. Then, for any two subsets $S', T' \subset V$ such that $S \subseteq S' \subseteq X$ and $T \subseteq T' \subseteq Y$, f is a maximum S'-T' flow and (X, Y) is a minimum S'-T' cut.*

Our implementation of Extract_Min_Partition consists of two phases. Let $P(S, T)$ be the given partially specified cut set. Let $m(S, T) = (S^*, T^*)$, $q = |S^* - S|$, and $r = |T^* - T|$. Phase 1 determines a sequence $(s_1, s_2, \ldots, s_q)$ of the vertices in $S^* - S$, partitions $P(S, T)$ into $q + 1$ subsets $P(S \cup \{s_1, s_2, \ldots, s_{i-1}\}, T \cup s_i)$, $i = 1, 2, \ldots, q + 1$, where $s_{q+1} = \emptyset$, and computes the minimum cut in each subset. After Phase 1, the minimum cut (S^*, T^*) is contained in the subset $P(S \cup \{s_1, s_2, \ldots, s_q\}, T) = P(S^*, T)$. Then, Phase 2 determines a sequence $(t_1, t_2, \ldots, t_r)$ of the vertices in $T^* - T$, further partitions the subset $P(S^*, T)$ into $r + 1$ subsets $P(S^* \cup t_i, T \cup \{t_1, t_2, \ldots, t_{i-1}\})$, $i = 1, 2, \ldots, r + 1$, where $t_{r+1} = \emptyset$, and computes the minimum cut in each subset. An illustration is given in Fig. 5.

We proceed to present the detailed implementation of Phase 1. For convenience, we assume that S contains only a single vertex s and T contains only a single vertex t. In case this is not true, we simply contract S and T, respectively, to create two

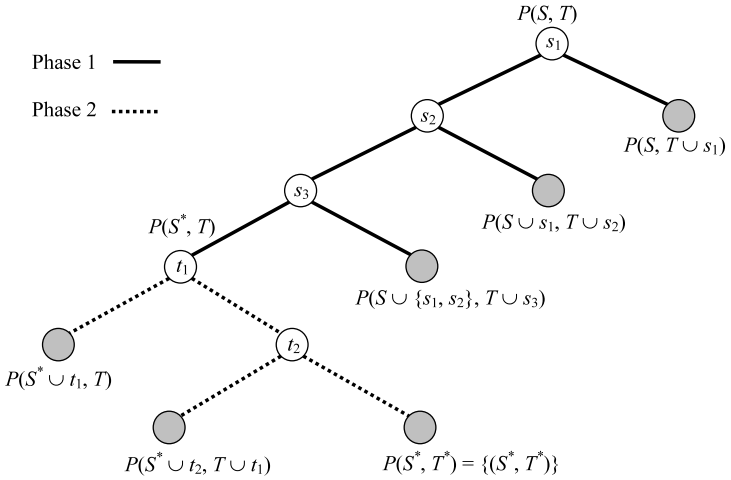


Fig. 5 Extract_Min_Partition, where $(S^*, T^*) = (S \cup \{s_1, s_2, s_3\}, T \cup \{t_1, t_2\})$

new vertices. Our problem is the following: Given G, s, t, and $m(s,t) = (S^*, T^*)$, determine a sequence $(s_1, s_2, \ldots, s_q)$ of the vertices in $S^* - s$ and compute the minimum cuts $m(S_i, \{t, s_i\}), i = 1, 2, \ldots, q$, where $S_i = \{s, s_1, \ldots, s_{i-1}\}$. Note that the computation of $m(S_{q+1}, t)$ is unnecessary, since $m(S_{q+1}, t) = m(S^*, t) = (S^*, T^*)$. We solve the above problem as follows. First, compute f as a maximum s-t flow in G. Next, obtain a graph G' by removing T^* from the residual network G_f. The vertex set of G' is S^*. Then, by using Hao and Orlin's algorithm, determine a sequence $(s_1, s_2, \ldots, s_q)$ of the vertices in $S^* - s$ and compute a minimum S_i-s_i cut, denoted by (α_i, β_i), of G' for $1 \le i \le q$, where $S_i = \{s, s_1, \ldots, s_{i-1}\}$. Finally, compute $m(S_i, \{t, s_i\}) = (\alpha_i, T^* \cup \beta_i)$ for $1 \le i \le q$. The overall time complexity is $O(nm \log(n^2/m))$. The correctness is ensured by the following lemma.

Lemma 7 *Let $s, t \in V$ be two vertices, (S^*, T^*) be a minimum s-t cut of G, and f be a maximum s-t flow in G. Let G' be the graph obtained by removing T^* from the residual network G_f. Let $S_1, S_2 \subset S^*$ be two disjoint non-empty subsets such that $s \in S_1$, and let (α, β) be a minimum S_1-S_2 cut of G'. Then, $(\alpha, T^* \cup \beta)$ is a minimum S_1-$(t \cup S_2)$ cut of G.*

Proof According to Lemma 5, we prove this lemma by showing that there is an S_1-$(t \cup S_2)$ flow f^* in G such that $w(u,v) - f^*(u,v) = 0$ for any $u \in \alpha$ and $v \in T^* \cup \beta$. For any $u, v \in S^*$, let $w'(u,v)$ be the weight of the edge (u,v) in G', which by definition is $w(u,v) - f(u,v)$. Let f' be a maximum S_1-S_2 flow in G'. Let f^* be the *flow sum* of f and f', which is defined by

$$f^*(u,v) = \begin{cases} f(u,v) & \text{if } u \in T^* \text{ or } v \in T^*, \\ f(u,v) + f'(u,v) & \text{otherwise} \end{cases}$$

for all $u, v \in V$. Since f is an s-t flow in G, f ' is an S_1-S_2 flow in G', $s \in S_1$, and $w'(u,v) = w(u,v) - f(u,v)$ for all $u, v \in S^*$, it is easy to conclude that f^*

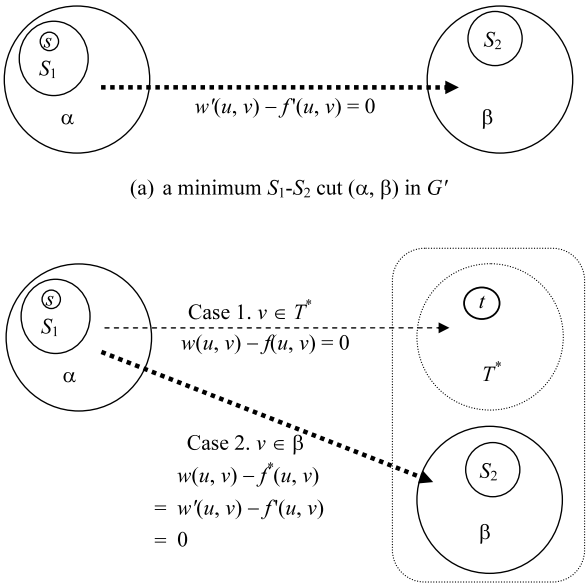


Fig. 6 An illustration for Lemma 6

satisfies the three properties of an S_1-$(t \cup S_2)$ flow. In the following, we show that $w(u, v) - f^*(u, v) = 0$ for any $u \in \alpha$ and $v \in T^* \cup \beta$.

Consider a fixed pair of $u \in \alpha$ and $v \in T^* \cup \beta$. (See Fig. 6.) Note that (α, β) is a partition of S^* and thus $u \in S^*$. Since (S^*, T^*) is a minimum s-t cut of G and f is a maximum s-t flow in G, by Lemma 4, $w(x, y) - f(x, y) = 0$ for any $x \in S^*$ and $y \in T^*$. Thus, if $v \in T^*$, $w(u, v) - f^*(u, v) = w(u, v) - f(u, v) = 0$. (See Case 1 of Fig. 6(b).) Assume that $v \in \beta$. (See Case 2 of Fig. 6(b).) Since (α, β) is a minimum S_1-S_2 cut of G' and f' is a maximum S_1-S_2 flow in G', $w'(u, v) - f'(u, v) = 0$. Thus, $w(u, v) - f^*(u, v) = w(u, v) - (f(u, v) + f'(u, v)) = w'(u, v) - f'(u, v) = 0$, which completes the proof of this lemma. □

Next, consider the implementation of Phase 2. Our problem is the following: Given G, S^*, T, and $m(S^*, T) = (S^*, T^*)$, determine a sequence $(t_1, t_2, \ldots, t_r)$ of the vertices in $T^* - T$ and compute the minimum cuts $m(S^* \cup t_i, T_i)$, $i = 1, 2, \ldots, r$, where $T_i = T \cup \{t_1, \ldots, t_{i-1}\}$. Let H be the *transpose* of G, which is obtained from G by reversing each edge. For any two non-empty disjoint subsets A, B of V, let $m_H(A, B)$ denote the minimum A-B cut in H. Clearly, for any cut (X, Y) in G, its weight is the same as the weight of (Y, X) in H. Thus, our problem can be restated as follows: Given H, T, S^*, and $m_H(T, S^*) = (T^*, S^*)$, determine a sequence $(t_1, t_2, \ldots, t_r)$ of the vertices in $T^* - T$ and compute the minimum cuts $m_H(T_i, S^* \cup t_i)$, $i = 1, 2, \ldots, r$, where $T_i = T \cup \{t_1, \ldots, t_{i-1}\}$. This is a special case of the problem in Phase 1, in which the sink side of the given $m_H(T, S^*)$ is the same as the parameter S^*. Therefore, Phase 2 can be implemented in $O(nm \log(n^2/m))$ time as follows. First, obtain a directed graph H' from the residual network G_f by removing S^* and then reversing each edge. Recall that f is the S-T maximum flow

computed in Phase 1. Let g be the flow defined by $g(u, v) = f(v, u)$ for any $u, v \in V$. Clearly, H' is the graph obtained by removing S^* from the residual network H_g. Note that since H is the transpose of G and by Lemma 6 f is a maximum S^*-T flow in G, it is easy to conclude that g is a maximum T-S^* flow in H. Next, by using Hao and Orlin's algorithm, determine a sequence $(t_1, t_2, \ldots, t_r)$ of the vertices in $T^* - T$ and compute a minimum T_i-t_i cut, denoted by (β_i, α_i), of H' for $1 \leq i \leq r$, where $T_i = T \cup \{t_1, \ldots, t_{i-1}\}$. Finally, compute $m(S^* \cup t_i, T_i) = (S^* \cup \alpha_i, \beta_i)$ for $1 \leq i \leq r$. We obtain the following:

Lemma 8 *Extract_Min_Partition can be implemented in $O(nm \log(n^2/m))$ time.*

Consequently, we have the following:

Theorem 2 *The cuts of a directed graph can be enumerated in the order of non-decreasing weights with $O(nm \log(n^2/m))$ time delay between two successive outputs.*

5 Enumerating All Cuts of an Undirected Graph and All s-t Cuts of a Graph

We solve the problem of enumerating all cuts of an undirected graph G as follows. Since G is undirected, two cuts (X, Y) and (Y, X) are the same. To avoid encountering a cut twice, we firstly select an arbitrary vertex s and assume that s is always on the source side. That is, we only consider the cuts in $P(s, \emptyset)$. Next, G is transformed into a directed graph G' by replacing each undirected edge (u, v) with two directed edges (u, v) and (v, u), each having the same weight as the original edge. Clearly, for any cut (X, Y), its weights in G and in G' are the same. Then, we enumerate the cuts in $P(s, \emptyset)$ by applying Algorithm 2 to G' with the following slight modification: Basic_Partition only returns a basic partition of $P(s, \emptyset)$ and the minimum cuts in its subsets.

Theorem 3 *The cuts of an undirected graph can be enumerated in the order of non-decreasing weights with $O(nm \log(n^2/m))$ time delay between two successive outputs.*

Next, consider the problem of enumerating all s-t cuts of a directed graph for a given pair of vertices $s, t \in V$. We do the enumeration by applying Algorithm 2 with the following simple modification: Basic_Partition only returns $(P(s, t), m(s, t))$. We obtain the following:

Theorem 4 *The s-t cuts of a directed graph can be enumerated in the order of non-decreasing weights with $O(nm \log(n^2/m))$ time delay between two successive outputs.*

As indicated in [28], the problem of enumerating all s-t cuts of an undirected graph can be treated as a special case of enumerating all s-t cuts of a directed graph

by replacing each edge by two arcs with opposite directions. Therefore, we have the following:

Theorem 5 *The s-t cuts of an undirected graph can be enumerated in the order of non-decreasing weights with $O(nm \log(n^2/m))$ time delay between two successive outputs.*

6 Applications

In this section, we describe existing algorithms whose running time can be immediately reduced by our enumeration algorithms.

Let G be an undirected graph and $k \geq 2$ be an integer. A *k-cut* of G is a partition of the vertex set V into k non-empty disjoint subsets. The definition of a k-cut is a generalization of the definition of a cut. More specifically, a "cut" and a "2-cut" refer to the same thing. The *minimum k-cut problem* is to find a k-cut that minimizes the total weight of the edges whose endpoints are in different subsets. Goldschmidt and Hochbaum [7] showed that the minimum k-cut problem is NP-hard if k is part of the input and presented an $O(n^{k^2/2-3k/2+5} m \log(n^2/m))$-time algorithm. Kamidoi, Yoshida, and Nagamochi [12] had an $O(n^{(4+o(1))k})$-time algorithm for the minimum k-cut problem. Very recently, Thorup [27] improved this upper bound to $\tilde{O}(n^{2k})$. In several special cases, better results are known. For $k = 3$, Kapoor [13] and Kamidoi, Wakabayashi, and Yoshida [11] showed that the problem can be solved in $O(n^4 m \log(n^2/m))$ time, Nagamochi and Ibaraki [21] had an $O(n^3 m \log(n^2/m))$-time algorithm, and Burlet and Goldschmidt [2] had an $O(n^3 m + n^4 \log n)$-time algorithm. For $k = 4$, Kamidoi, Wakabayashi, and Yoshida [11] had an $O(n^5 m \log(n^2/m))$-time algorithm, and Nagamochi and Ibaraki [21] had an $O(n^4 m \log(n^2/m))$-time algorithm. For $k = 5$ and 6, Nagamochi, Katayama, and Ibaraki [24] showed that the problem can be solved in $O(n^k m \log(n^2/m))$ time. For $k = 3, 4, 5$, and 6, Levine [19] gave $O(n^{k-2} m \log^3 n)$-time randomized algorithms.

Consider the minimum k-cut problem with $k \geq 3$. If we can identify a component X of a minimum k-cut, then the other $k-1$ components can be computed by solving the minimum $(k-1)$-cut problem on the subgraph induced by $V - X$. A set of cuts is called a *k-candidate set* if it contains a cut (X, Y) such that either X or Y is a component of a minimum k-cut. For any $j \geq 1$, let $M(j)$ be the time required for computing the smallest j cuts. For $k = 3$ and 4, Nagamochi and Ibaraki [21] gave the following important result: a k-cut π and a set D of $O(n)$ 2-cuts can be determined in $M(2n-2)$ time such that either π is a minimum k-cut or D is a k-candidate set. Later, as an extension of this work, Nagamochi, Katayama, and Ibaraki [24] further showed that such k-cut π and set D can be determined in $M(15n-60)$ time for $k = 5$ and 6. Given a k-candidate set D, a minimum k-cut can be found by simply applying a minimum $(k-1)$-cut algorithm $2|D|$ times. Let T_k be the time required for solving the minimum k-cut problem. By using Vazirani and Yannakakis's enumeration algorithm, $O(M(2n-2)) = O(M(15n-60)) = \tilde{O}(n^3 m)$. Therefore, for $k = 3, 4, 5, 6$, the minimum k-cut problem can be solved in

$$T_k = \tilde{O}(n^3 m) + O(n \times T_{k-1}) = \tilde{O}(n^k m + n^{k-2} \times T_2) = \tilde{O}(n^k m)$$

time [21, 24]. By using our enumeration algorithm, $O(M(2n-2)) = O(M(15n-60)) = O(n^2 m \log(n^2/m))$ and thus the following result is obtained.

Theorem 6 *The minimum k-cut problem can be solved in $O(n^{k-1} m \log(n^2/m))$ time for $k = 3, 4, 5, 6$.*

Zhao, Nagamochi, and Ibaraki [29] had a simple approximation algorithm for the minimum k-cut problem. The performance ratio is $2 - (3/k)$ for an odd k and is $2 - (3k-4)/(k^2-k)$ for an even k. The algorithm first divides V into two components by using a minimum 2-cut algorithm. Then, it repeatedly divides the components into smaller components by applying a minimum 3-cut algorithm until there are k components. At each iteration, the number of components increases by at least 2. Thus, at most $k/2$ minimum 3-cut computations are required. By using the minimum 3-cut algorithm Zhao, Nagamochi, and Ibaraki [21], implemented their approximation algorithm in $O(kn^3 m \log(n^2/m))$ time. By using our result on the minimum 3-cut problem, the following is obtained.

Theorem 7 *The approximation algorithm in* [29] *can be implemented in $O(kn^2 m \times \log(n^2/m))$ time.*

An *ideal cut* of a directed acyclic graph is a cut (X, Y) such that there is no edges directed from Y to X. Vazirani and Yannakakis [28] showed that the problem of enumerating the ideal cuts of a directed acyclic graph by their weights can be reduced in $O(m)$ time to the problem of enumerating the s-t cuts of a directed graph. Therefore, we have the following:

Theorem 8 *The ideal cuts of a directed acyclic graph can be enumerated in the order of non-decreasing weights with $O(nm \log(n^2/m))$ time delay between two successive outputs.*

Given a network of n vertices, each of whose m links is assumed to fail independently with some probability, the *all-terminal network reliability problem* is to determine the probability that the network becomes disconnected due to edge failures. This problem is NP-complete [14]. Given an approximation ratio $\varepsilon > 1$, Karger [14] had an approximation scheme that initially computes the smallest $O(n^{2\alpha})$ cuts, where $\alpha = O(1 - \log \varepsilon / \log n)$, and then determines a solution based on the cuts. By using Vazirani and Yannakakis's algorithm, the running time is $\tilde{O}(mn^{2+2\alpha} + (n/\varepsilon)^{2^{O(-\log_n \varepsilon)}})$. By using our result, the following is obtained.

Theorem 9 *The approximation scheme in* [14] *can be implemented in $O(mn^{1+2\alpha} \times \log(n^2/m) + (n/\varepsilon)^{2^{O(-\log_n \varepsilon)}})$ time.*

7 Concluding Remarks

In this paper, improved algorithms were proposed for the problems of enumerating the cuts of a graph by their weights. The presented algorithms use the same schema

as Vazirani and Yannakakis's enumeration algorithms. To enumerate a cut, their algorithms require at most $n-1$ maximum flow computations, and ours require one maximum flow computation and two invocations of Hao and Orlin's minimum cut algorithm. Note that our two invocations of Hao and Orlin's algorithm are applied, respectively, to two disjoint parts of the input graph.

Theoretically, our algorithms reduced the asymptotic upper bound for enumerating a cut by a factor of $\tilde{O}(n)$. In the following, the practical performance of our algorithms is discussed. The practical running time of minimum cut algorithms have been extensively studied in the literature [3, 10, 22]. In order to perform meaningful comparisons, Nagamochi, Ono, and Ibaraki [22] had developed a problem generator and six families of graph instances for evaluating and comparing the performance of minimum cut codes. Nagamochi and Ibaraki's [20] had an $O(nm + n^2 \log n)$-time minimum cut algorithm. Nagamochi, Ono, and Ibaraki's experimental results in [22] indicated that the running time of Nagamochi and Ibaraki's algorithm is comparable to 2–3 executions of a maximum flow algorithm. Based on the same test families, Jünger and Rinaldi's [10] had a comparison between Nagamochi and Ibaraki's algorithm and Hao and Orlin's algorithm. According to their experimental results, on average, the running time of Hao and Orlin's algorithm is less than twice the running time of Nagamochi and Ibaraki's algorithm. Thus, on the test families developed in [22], the practical running time of Hao and Orlin's algorithm is about 4–6 times the running time of a maximum flow algorithm. And therefore, the improvement of our enumeration algorithms is significant from both the theoretical and practical points of view.

References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications, 1st edn. Prentice-Hall, Englewood Cliffs (1993)
2. Burlet, M., Goldschmidt, O.: A new and improved algorithm for the 3-cut problem. Oper. Res. Lett. **21**, 225–227 (1997)
3. Chekuri, C.S., Goldberg, A.V., Karger, D.R., Levine, M.S., Stein, C.: Experimental study of minimum cut algorithms. In: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithm, pp. 324–333 (1997)
4. Dinits, E.A., Karzanov, A.V., Lomonosov, M.V.: On the structure of a family of minimal weighted cuts in a graph. In: Fridman, A.A. (ed.) Studies in Discrete Optimization, pp. 290–306. Nauka, Moscow (1976) (Original article in Russian. Translation available from NTC-National Translations Center, Library of Congress, Cataloging Distribution Service, Washington DC 20541, USA (NTC 89-20265))
5. Fleischer, L.: Building chain and cactus representations of all minimum cuts from Hao-Orlin in the same asymptotic run time. J. Algorithms **33**, 51–72 (1999)
6. Goldberg, A.V., Tarjan, R.E.: A new approach to the maximum flow problem. J. ACM **35**, 921–940 (1988)
7. Goldschmidt, O., Hochbaum, D.S.: Polynomial algorithm for the k-cut problem for fixed k. Math. Oper. Res. **19**, 24–37 (1994)
8. Gomory, R.E., Hu, T.C.: Multi-terminal network flows. J. Soc. Ind. Appl. Math. **9**, 551–570 (1961)
9. Hao, J., Orlin, J.B.: A faster algorithm for finding the minimum cut in a directed graph. J. Algorithms **17**, 424–446 (1994)
10. Jünger, M., Rinaldi, G.: Practical performance of efficient minimum cut algorithms. Algorithmica **26**(1), 172–195 (2000)
11. Kamidoi, Y., Wakabayashi, S., Yoshida, N.: A divide-and-conquer approach to the minimum k-way cut problem. Algorithmica **32**, 262–276 (2002)

12. Kamidoi, Y., Yoshida, N., Nagamochi, H.: A deterministic algorithm for finding all minimum k-way cuts. SIAM J. Comput. **36**, 1315–1327 (2006)
13. Kapoor, S.: On minimum 3-cuts and approximating k-cuts using cut trees. In: Proceedings of the 5th Integer Programming and Combinatorial Optimization Conference, pp. 132–146 (1996)
14. Karger, D.R.: A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem. SIAM J. Comput. **29**, 492–514 (1999)
15. Karger, D.R.: Minimum cuts in near-linear time. J. ACM **47**, 46–76 (2000)
16. Karger, D.R., Stein, C.: A new approach to the minimum cut problem. J. ACM **43**, 601–640 (1996)
17. Karzanov, A.V., Timofeev, E.A.: Efficient algorithms for finding all minimal edge cuts of a nonoriented graph. Cybernetics **22**, 156–162 (1986) (Translated from Kibernetika **2**, 8–12 (1986))
18. King, V., Rao, S., Tarjan, R.E.: A faster deterministic maximum flow algorithm. J. Algorithms **17**, 447–474 (1994)
19. Levine, M.S.: Faster randomized algorithms for computing minimum {3, 4, 5, 6}-way cuts. In: Proceedings of the Eleventh ACM–SIAM Symposium on Discrete Algorithms, pp. 735–742 (2000)
20. Nagamochi, H., Ibaraki, T.: Computing the edge-connectivity of multigraphs and capacitated graphs. SIAM J. Discrete Math. **5**, 54–66 (1992)
21. Nagamochi, H., Ibaraki, T.: A fast algorithm for computing minimum 3-way and 4-way cuts. Math. Program. **88**, 507–520 (2000)
22. Nagamochi, H., Ono, T., Ibaraki, T.: Implementing an efficient minimum capacity cut algorithm. Math. Program. **67**, 325–341 (1994)
23. Nagamochi, H., Nishimura, K., Ibaraki, T.: Computing all small cuts in undirected networks. SIAM J. Discrete Math. **10**, 469–481 (1997)
24. Nagamochi, H., Katayama, S., Ibaraki, T.: A faster algorithm for computing minimum 5-way and 6-way cuts in graphs. J. Comb. Optim. **4**, 151–169 (2000)
25. Nagamochi, H., Nakamura, S., Ishii, T.: Constructing a cactus for minimum cuts of a graph in $O(mn + n^2 \log n)$ time and $O(m)$ space. IEICE Trans. Inf. Syst. **E86-D**, 179–185 (2003)
26. Picard, J.C., Queyranne, M.: On the structure of all minimum cuts in a network and applications. Math. Program. Study **13**, 8–16 (1980)
27. Thorup, M.: Minimum k-way cuts via deterministic greedy tree packing. In: Proceedings of the 40th Annual ACM Symposium on Theory of Computing, pp. 159–165 (2008)
28. Vazirani, V., Yannakakis, M.: Suboptimal cuts: their enumeration, weight, and number. In: Proceedings of the 19th International Colloquium on Automata, Languages and Programming, pp. 366–377 (1992)
29. Zhao, L., Nagamochi, H., Ibaraki, T.: Approximating the minimum k-way cut in a graph via minimum 3-way cuts. J. Comb. Optim. **5**, 397–410 (2001)