Theory Comput. Systems **41**, 369–380 (2007)
DOI: 10.1007/s00224-007-2010-2

**Theory of Computing Systems**

# A Simple and Fast Min-Cut Algorithm

Michael Brinkmeier

Institute for Theoretical and Technical Computer Science,
Technical University of Ilmenau, 98693 Ilmenau, Germany
mbrinkme@tu-ilmenau.de

**Abstract.** We present an algorithm which calculates a minimum cut and its weight in an undirected graph with nonnegative real edge weights, $n$ vertices and $m$ edges, in time $O(\max(\log n, \min(m/n, \delta_G/\varepsilon))n^2)$, where $\varepsilon$ is the minimal edge weight, and $\delta_G$ is the minimal weighted degree. For integer edge weights this time is further improved to $O(\delta_G n^2)$ and $O(\lambda_G n^2)$. In both cases these bounds are improvements of the previously known best bounds of deterministic algorithms. These were $O(nm + n^2 \log n)$ for real edge weights and $O(nM + n^2)$ and $O(M + \lambda_G n^2)$ for integer weights, where $M$ is the sum of all edge weights.

## 1. Introduction

The problem of finding a minimum cut of a graph appears in many applications, for example, in network reliability, clustering, information retrieval and chip design. More detailed, a minimum cut of an undirected graph with edge weights, is a set of edges with minimum sum of weights, such that its removal would cause the graph to become unconnected. The total weight of edges in a minimum cut of $G$ is denoted by $\lambda_G$ and is called the (*edge-*)*connectivity* of $G$.

In the literature many algorithms can be found applying various methods. One group of algorithms is based on the well-known result of Ford and Fulkerson [FF] regarding the duality of maximum $s$–$t$-flows and minimum $s$–$t$-cuts for arbitrary vertices $s$ and $t$. In [GH] Gomory and Hu presented an algorithm which calculated $n-1$ maximum $s$–$t$-flows from a given source $s$ to all other vertices $t$. Hao and Orlin adapted the maximum flow algorithm of Goldberg and Tarjan [GT] and were able to construct a minimum cut of a directed graph with nonnegative real numbers as edge weights in time $O(nm \log(n^2/m))$.

Nagamochi, Ibaraki and others [NI1], [NOI], [NII] described an algorithm without using maximum flows. Instead they constructed spanning forests and iteratively contracted edges with high weights. This lead to an asymptotic runtime of $O(nm + n^2 \log n)$ on undirected graphs with nonnegative real edge weights. On undirected, unweighted multigraphs they obtained a runtime of $O(n(n + m))$. Translated to integer weighted graphs without parallel edges, this corresponds to a runtime of $O(n(n + M))$ where $M$ is the sum of all edge weights. Using a "searching" technique, they improved this to $O(M + \lambda_G n^2)$.

Nagamochi and Ibaraki's approach was refined in [SW] by Stoer and Wagner. They replaced the construction of spanning forests with the construction of *Maximum Adjacency Orders* but the asymptotic runtime stayed unchanged.

Karger and Stein [KS] used the contraction technique for a randomized algorithm calculating the minimum cut of undirected graphs in $O(n^2 \log^3 n)$ expected time. Later Karger [K1], [K2] presented two related algorithms using expected time $O(m \log^3 n)$ and $O(n^2 \log n)$.

The last two algorithms are related to an approach of Gabow [G] based on matroids. He presented an algorithm for the minimum cut of an directed, unweighted graph requiring $O(\lambda m \log(n^2/m))$ time and based on this an algorithm for undirected unweighted graphs with $O(m + \lambda^2 n \log(n/\lambda))$, where $\lambda$ is the weight of a minimum cut.

In this paper we propose two changes of the algorithm of Stoer and Wagner [SW], which calculates a *maximum adjacency order* on the vertices of an undirected graph and then contracts the last two vertices in this order. Repeating this $n - 1$ times allows the construction of a minimum cut. Our first change leads to a reduced average runtime by contracting more than one pair of vertices if possible, reducing the asymptotic worst-case runtime for real weighted edges to $O(\max(\log n, \min(m/n, \delta_G/\varepsilon))n^2)$, where $\varepsilon$ is the minimal edge weight. In fact the same idea was already used in later versions of the algorithms of Nagamochi and Ibaraki [NOI], but for some reason did not find its way into the MA-order-based algorithms (even those described by Nagamochi and Ibaraki) [SW], [NI3] and the analysis of the worst-case runtime.

For integer weighted edges our algorithm allows an additional relaxation of the applied maximum adjacency orders, which in turn allows usage of an alternative data structure for construction of the order. These *priority queues with threshold* reduce the asymptotic runtime for undirected graphs with nonnegative integer weights to $O(\delta_G n^2)$, instead of $O(Mn + n^2)$ as obtained by Nagamochi and Ibaraki.[1] Applying the same "search" as Nagamochi and Ibaraki, we obtain a time of $O(\lambda_G n^2)$ instead of $O(M + \lambda_G n^2)$.

Independent of the types of edge weights, the algorithm presented in this paper requires at most the same time as the algorithm of Stoer and Wagner [SW].

## 2. The Problem and Notations

In the following let $G = (V, E)$ be an undirected graph without multiple edges and self-loops. Let $n = |V|$ be the number of vertices and let $m = |E|$ be the number of edges. The latter ones are weighted by positive real numbers or integers, given by a map $w: V \times V \to \mathbb{R}^+$ with $w(u, v) = w(v, u)$ and $w(u, v) = 0$ if and only if

[1] Remember that $M$ is the sum of all edge weights and $m$ is the number of edges.

$(u, v) \notin E$. The *degree* $\deg(v)$ of a vertex is the sum of weights of incident edges, i.e. $\deg(v) = \sum_{u \in V} w(u, v)$. The *minimal degree* over all vertices of $G$ is denoted by $\delta_G$. The sum of all edge weights is $M = \frac{1}{2} \sum_{v \in V} \deg(v)$.

A *cut* of $G$ is an (unordered) partition $(C, V \setminus C)$ of the vertex set into two parts. For shorter notation the cut will often be written as $C$. The *weight* $w(C)$ of the cut $C$ is the total weight of cut edges, i.e.

$$w(C) = \sum_{\substack{(u,v) \in E \\ u \in C, v \notin C}} w(u, v).$$

For two vertices $u$ and $v$ a *$u$–$v$-cut* is a cut $C$ such that $u \in C$ and $v \notin C$ or vice versa, i.e. *$C$ separates $u$ and $v$*. A *minimum $u$–$v$-cut* is a $u$–$v$-cut, whose weight is minimum amongst all $u$–$v$-cuts of $G$. The weight of a minimum $u$–$v$-cut is denoted by $\lambda_G(u, v)$. A *minimum cut* is a cut $C$ with minimal weight among all cuts of $G$. Its weight $\lambda_G$ is called the *edge-connectivity* of $G$. Obviously we have

$$\lambda_G = \min\{\lambda_G(u, v) \mid u, v \in V\}.$$

For a subset $U \subseteq V$ of vertices $G[U]$ denotes the induced subgraph of $G$, i.e. the graph consisting of the vertices in $U$ and all edges of $G$ between them. If $U = \{v_!, \ldots, v_m\}$ we also write $G[v_1, \ldots, v_m]$.

Like the ones of Nagamochi and Ibaraki and Stoer and Wagner our algorithm will be a *contraction algorithm*. Basically this means that it identifies one or more pairs of "critical" vertices and contracts (or merges) them, obtaining a graph with less nodes. Let $u$ and $v$ be two vertices of $G = (V, E)$. The graph $G/u \sim v$ is obtained from $G$ by identifying $u$ and $v$, this means that $u$ and $v$ are replaced by a new vertex $[u] = [v]$ and that the weights of the edges $(x, u)$ and $(x, v)$ are added, i.e. $w(x, [u]) = w(x, u) + w(x, v)$.

Contraction algorithms rely on the following well-known theorem, relating the edge-connectivity of a graph with the one of a quotient graph.

**Theorem 1** (Theorem 2.1 of [SW]). *Let $u$ and $v$ be two vertices of an undirected, weighted graph $G = (V, E)$. Then the edge-connectivity of $G$ is the minimum of the weights of a minimum $v$–$u$-cut and the edge-connectivity of the graph $G/u \sim v$, obtained by the identification of $u$ and $v$, i.e.*

$$\lambda_G = \min(\lambda_G(u, v), \lambda_{G/u \sim v}).$$

*Proof.* We have to differentiate two cases. Either each minimum cut of $G$ separates $u$ and $v$, then $\lambda_G = \lambda_G(u, v)$ and $\lambda_{G/u \sim v} > \lambda_G$, or there exists at least one minimum cut not separating $u$ and $v$ and hence induces a minimum cut of $G/u \sim v$, leading to $\lambda_G = \lambda_{G/u \sim v} \leq \lambda_G(u, v)$. □

## 3. Maximum Adjacency Orders

The key for the algorithms of Nagamochi and Ibaraki and Stoer and Wagner are *maximum adjacency orders* on the vertices of the graph. The vertices $v_1, \ldots, v_n$ are arranged in a maximum adjacency order, if for each $v_i$, $i > 1$, the sum of weights from $v_i$ to all preceding vertices $v_1, \dots, v_{i-1}$ is maximal among all vertices $v_k$ with $k \geq i$.

**Definition 1** (Maximum Adjacency Order). Let $G = (V, E)$ be an undirected, weighted graph. An order $v_1, v_2, \dots, v_n$ on the vertices of $G$ is a *maximum adjacency order* or *MA-order* if

$$w(v_1, \dots, v_{i-1}; v_i) := \sum_{j=1}^{i-1} w(v_i, v_j) \geq \sum_{j=1}^{i-1} w(v_k, v_j) =: w(v_1, \dots, v_{i-1}; v_k)$$

for all $k \geq i$. The values $w(v_1, \dots, v_{i-1}; v_i)$ are called *adjacencies*.

The foundation of the algorithms of Nagamochi and Ibaraki and Stoer and Wagner is the observation that the degree of $v_n$ in an MA-order is equal to the weight of a minimum $v_n$–$v_{n-1}$-cut in $G$.

**Lemma 1** (Lemma 3.1 of [SW]). *For each MA-order $v_1, \dots, v_n$ of the undirected, weighted Graph $G = (V, E)$, the cut $(\{v_1, \dots, v_{n-1}\}, \{v_n\})$ is a minimum $v_n$–$v_{n-1}$-cut.*

Our algorithm is additionally based on the following simple observation.

**Lemma 2.** *Let $v_1, \dots, v_n$ be an MA-order of $G = (V, E)$. Then $v_1, \dots, v_l$ is an MA-order of $G[v_1, \dots, v_l]$.*

*Proof.* Since

$$\sum_{j=1}^{i-1} w(v_i, v_j) \geq \sum_{j=1}^{i-1} w(v_k, v_j)$$

for all $k \geq i$ for all $2 \leq i \leq l$, the same obviously holds for $i \leq k \leq l$. □

**Corollary 1.** *For each MA-order $v_1, \dots, v_n$ of $G = (V, E)$ we have*

$$\lambda_G(v_i, v_{i-1}) \geq \lambda_{G[v_1,\dots,v_i]}(v_i, v_{i-1}) = w(v_1, \dots, v_{i-1}; v_i).$$

*Proof.* The equality of $w(v_1, \dots, v_{i-1}; v_i)$ and $\lambda_{G[v_1,\dots,v_i]}(v_i, v_{i-1})$ is a direct consequence of Lemma 2 and 1. Since the weight of a minimal $v_i - v_{i-1}$-cut increases, if vertices and edges are added to the graph, the inequality is immediately clear. □

For $\tau > 0$, an MA-order $v_1, \dots, v_n$ of the vertices of $G = (V, E)$, and $1 \leq i \leq n$, let the graphs $G_i^\tau$ be defined by $G_1^\tau := G$ and $G_{i+1}^\tau := G_i^\tau$ if $w(v_1, \dots, v_i; v_{i+1}) < \tau$, and

$$G_{i+1}^\tau \in \{G_i^\tau, G_i^\tau/[v_i] \sim v_{i+1}\}$$

if $w(v_1, \ldots, v_i; v_{i+1}) \geq \tau$ where $[v_i]$ is the vertex in $G_i^\tau$ representing the class of vertices in $G$ contracted so far to $v_i$. In other words, the graphs $G_i^\tau$ are obtained from $G$ by iteratively contracting pairs of vertices $v_i$ and $v_{i+1}$, such that the adjacency of $v_{i+1}$ is greater than or equal to $\tau$. The resulting vertices represent sets, denoted by $[v_i]$, of vertices in the original graph $G$. The last graph $G_n^\tau$ of this sequence is denoted by $G^\tau$.

**Theorem 2.** *For an MA-order $v_1, \ldots, v_n$ of $G = (V, E)$ and $\tau > 0$ the following equation holds for $1 \leq i \leq n$*:

$$\min(\lambda_G, \tau) = \min(\delta_G, \lambda_{G_i^\tau}, \tau).$$

*Proof.* We prove the theorem by induction over $i$. For $i = 1$ the statement is trivial, since $\delta_G \geq \lambda_G = \lambda_{G_1^\tau}$. Now assume that $\min(\lambda_G, \tau) = \min(\delta_G, \lambda_{G_{i-1}^\tau}, \tau)$ for $i > 1$. If $G_i^\tau = G_{i-1}^\tau$, then the statement obviously holds. Now assume $G_i^\tau = G_{i-1}^\tau/[v_{i-1}] \sim v_i$. Then Lemma 1 induces $\lambda_{G_{i-1}^\tau} = \min(\lambda_{G_{i-1}^\tau}([v_{i-1}], v_i), \lambda_{G_i^\tau})$ and hence

$$\min(\lambda_G, \tau) = \min(\delta_G, \lambda_{G_{i-1}^\tau}, \tau) = \min(\delta_G, \lambda_{G_{i-1}^\tau}([v_{i-1}], v_i), \lambda_{G_i^\tau}, \tau).$$

Since $[v_{i-1}]$ represents a set of vertices in $G$, each $[v_{i-1}]$–$v_i$-cut in $G_{i-1}^\tau$ induces a $v_{i-1}$–$v_i$-cut in $G$ and hence, by definition of $G_i^\tau$ and Corollary 1,

$$\lambda_{G_{i-1}^\tau([v_{i-1}], v_i)} \geq \lambda_G(v_{i-1}, v_i) \geq w(v_1, \ldots, v_{i-1}; v_i) \geq \tau,$$

thus

$$\min(\lambda_G, \tau) = \min(\delta_G, \lambda_{G_i^\tau}, \tau). \qquad \square$$

Even though it is not clear how many vertices $G^\tau$ contains, we can limit the total weight of all edges in $G^\tau$.

**Lemma 3.** *For an MA-order $v_1, \ldots, v_n$ on $G = (V, E)$ and $\tau > 0$ the total weight of all edges of $G^\tau$ is less than $(n - 1)\tau$.*

*Proof.* The lemma is proved by induction over the number $n$ of vertices in $G$. It is obviously true for $n = 1$.

Since $v_1, \ldots, v_{n-1}$ is an MA-order of $G[v_1, \ldots, v_{n-1}]$, we know that the total edge weight of the induced subgraph of $G^\tau$ is less than $(n-2)\tau$. If the adjacency of $v_n$ is less than $\tau$, then $G^\tau$ obviously has total edge weight less than $(n - 1)\tau$.

Now assume that the adjacency of $v_n$ is at least $\tau$. Let $v_i$ be the first vertex in the MA-order, such that $w(v_1, \ldots, v_i; v_n) \geq \tau$, implying $w(v_1, \ldots, v_j; v_n) \geq \tau$ for all indices $i \leq j \leq n-1$. Hence, the adjacencies of $v_i, \ldots, v_{n-1}$ are at least $\tau$ and the complete sequence is contracted. Therefore, at most the edges between $v_n$ and $v_1, \ldots, v_{i-1}$ "survive" in $G^\tau$ and the total edge weight of $G^\tau$ is less than $(n - 1)\tau$. $\square$

## 4. Lax Adjacency Orders

In this section we use the threshold $\tau$ for the contraction in each round to relax the restrictions of the maximum adjacency order. The basic observation leading to this improvement is the fact that the exact order inside a sequence of vertices with adjacency $\geq \tau$ does not matter. Hence, during the construction of the adjacency order, we may choose any vertex, as long as its adjacency is above the threshold (if possible). We only have to choose a vertex of maximum adjacency if it is below $\tau$.

**Definition 2** (Lax Adjacency Order). Let $G = (V, E)$ be an undirected, weighted graph and $\tau \geq 0$. An order $v_1, v_2, \ldots, v_n$ on the vertices of $G$ is a *lax adjacency order* or *LA-order for threshold* $\tau$, if

$$\min(\tau, w(v_1, \ldots, v_{i-1}; v_i)) \geq \min(\tau, w(v_1, \ldots, v_{i-1}; v_k))$$

for all $k \geq i$.

Now we have to prove that a LA-order of threshold $\tau$ is an allowed replacement for an MA-order in the construction of $G^\tau$.

**Lemma 4** (LA- and MA-Orders). *Let $u_1, \ldots, u_n$ be an LA-order of $G = (V, E)$ and let $i_1 < \ldots < i_k$ be the indices of all vertices with $w(u_1, \ldots, u_{i_l - 1}; u_{i_l}) < \tau$ for $1 \leq l \leq k$. Then there exists an MA-order $v_1, \ldots, v_n$ with*

1. *$v_{i_l} = u_{i_l}$ for $1 \leq l \leq k$ and*
2. *if $u_j = v_{\bar{j}}$ and $i_l < j < i_{l+1}$ for $1 \leq l < k$, then $i_l < \bar{j} < i_{l+1}$ and*
3. *if $u_j = v_{\bar{j}}$ and $i_k < j$, then $i_k < \bar{j}$.*

In other words, for each LA-order there exists an MA-order, such that the vertices with adjacencies below the threshold are at the same positions, which we call “fix points”. Furthermore, vertices with adjacencies equal to or greater than $\tau$ may be permuted, as long as they stay between the same “fix point”. The sequences between two subsequent fix points are called “high adjacency sequences” and their members “vertices of high adjacency”.

*Proof of Lemma* 4. We are going to construct an MA-order $v_1, \ldots, v_n$ for each fix point $i_l$, such that the two conditions of the lemma hold for all indeces up to position $i_l$.

Obviously $i_1 = 1$ and hence, each MA-order beginning with $u_1$ satisfies the conditions up to position $i_1$.

Now assume that we have an MA-order $v_1, \ldots, v_n$ with $v_{i_k} = u_{i_k}$ for $1 \leq i \leq l$, and the high adjacency vertices up to position $i_l$ are only permuted in their specific sequences.

If $i_l = i_k$, i.e. if the last fixed point is reached, we are done.

Otherwise, observe that $w(u_1, \ldots, u_{i_{l+1}-1}; u_k) < \tau$ for each $i_{l+1} \leq k$, since

$$\tau > w(u_1, \ldots, u_{i_{l+1}-1}; u_{i_{l+1}}) \geq w(u_1, \ldots, u_{i_{l+1}-1}; u_k).$$

If $i_{l+1} = i_l + 1$, then we obviously may extend $v_1, \dots, v_{i_l}$ by $u_{i_{l+1}}$ and obtain an MA-order, which respects the fixed points and high adjacency sequences up to position $i_{l+1}$.

If $i_{l+1} > i_l + 1$, then there obviously exists a vertex (e.g. $u_{i_l+1}$) with an adjacency equal to or greater than $\tau$. Since all vertices $u_k$ with $k \geq i_{l+1}$ have low adjacency, the next vertex in each MA-order extending $v_1, \dots, v_{i_l}$, has to be in the high adjacency sequence between $i_l$ and $i_{l+1}$.

As long as vertices of the high adjacency sequence between $i_l$ and $i_{l+1}$ are not integrated into the MA-order, it does not matter which vertex is chosen. Simply choose the first unused vertex of the sequence. Since all vertices preceding it in the LA-order are already used in the MA-order, its adjacency is at least $\tau$, requiring the next vertex to be one member of the sequence (their adjacency is below $\tau$ as observed above). $\square$

Since the high adjacency sequences of an LA- and a corresponding MA-order are contracted to a single vertex, the resulting graph $G^\tau$ is the same.

**Theorem 3.** *For an LA-order $v_1, \dots, v_n$ with threshold $\tau > 0$ on $G = (V, E)$ the following equation holds for $1 \leq i \leq n$:*

$$\min(\lambda_G, \tau) = \min(\delta_G, \lambda_{G^\tau}, \tau).$$

## 5. The Algorithm

Theorem 3 provides us with a simple algorithm for the determination of a minimum cut. Let $\tau > 0$ be given.

```
1. While $G$ contains two or more vertices, repeat
   (a) Determine a vertex $[v]$ of $G$ with minimal degree.
   (b) If $\delta_G < \tau$, set cut $= [v]$.
   (c) $\tau = \min(\tau, \delta_G)$.
   (d) $G = G^\tau$ for some LA-order with threshold $\tau$.
2. The cut is given by the vertices in $[v]$ and has weight $\tau$.
```

First, the algorithm terminates, since the adjacency of the last vertex $v_n$ is its degree and hence equal to or greater than $\delta_G$ and $\tau$. Therefore $G^\tau$ has less vertices than $G$ and the algorithm reduces the number of nodes in each round.

**Lemma 5.** *Let $\lambda(G, \tau)$ denote the resulting value $\tau$ of the algorithm with inputs G and $\tau$. Then $\lambda(G, \tau) = \min(\lambda_G, \tau)$.*

*Proof.* If $G$ has exactly two nodes, then the algorithm returns $\min(\tau, \delta_G)$ and one vertex as cut. If $G$ has more than two nodes then the algorithm returns the same result as if it was started with $G^\tau$ and $\tau = \min(\delta_G, \tau)$, i.e.

$$\lambda(G, \tau) = \lambda(G^{\min(\tau, \delta_G)}, \min(\tau, \delta_G)).$$

By induction over the number of nodes and Theorem 3 this leads to

$$\lambda(G,\tau) = \min(\lambda_{G^{\min(\tau,\delta_G)}}, \tau, \delta_G) = \min(\lambda_G, \tau). \qquad \square$$

For the calculation of $G^\tau$ we have to construct an LA-order and then iteratively contract subsequent vertices $v_{i-1}$ and $v_i$ with $w(v_1, \dots, w_{i-1}; v_i) \geq \delta_G$, where $\delta_G$ is the minimum degree. One way to implement one of these rounds would be to order these nodes and then contract some of them. However, in fact ordering and contraction may be done at the same time. Assume that $v_1, \dots, v_n$ is an arbitrary order of the vertices. Then

$$w(v_1, \dots, v_{i-1}; v_i) = w(v_1, \dots, v_j \sim v_{j+1}, \dots, v_{i-1}; v_i),$$

i.e. contraction of preceeding vertices does not affect the adjacency of a vertex. Hence we may construct the LA-order and, as soon as the adjacency is greater than or equal to $\delta_G$, we may contract the newly added and the last vertex.

The LA-order may be build using a maximum priority queue. Assume that we have $n'$ vertices and $m'$ edges. Each vertex has to be inserted and extracted at most once, leading to $n'$ inserts and extractions. The priorities are updated at most once for each edge, leading to $m'$ operations, resulting in a runtime $O(n'T_{\text{insert}} + n'T_{\text{extractMax}} + m'T_{\text{increaseKey}})$ for queue operations per round.

For the first round we have $n' = n$ and $m' = m$. For the subsequent rounds, the total edge weight is bounded by $\delta_G n$ (Lemma 3). If $\varepsilon > 0$ is the minimal edge weight, this bounds the number of edges by $\delta_G n/\varepsilon$. Furthermore, we have at most $m$ edges, implying a bound of $\min(m, \delta_G n/\varepsilon)$. This leads to a worst case runtime $O(nT_{\text{insert}} + nT_{\text{extractMax}} + \min(m, (\delta_G/\varepsilon)n)T_{\text{increaseKey}})$ in all subsequent rounds.

Together with the time of $O(n^2)$, required for the $n-1$ contractions of vertex pairs, the sum over at most $n-1$ rounds is

$$O\left(n^2 + n^2 T_{\text{insert}} + n^2 T_{\text{extractMax}} + \left(m + \min\left(\frac{m}{n}, \frac{\delta_G}{\varepsilon}\right) n^2\right) T_{\text{increaseKey}}\right).$$

Using Fibonacci Heaps, our algorithm has an amortized runtime of

$$\begin{aligned} O\left(n^2 + n^2 + n^2 \log n + m + \frac{\delta_G}{\varepsilon} n^2\right) &= O\left(m + n^2\left(\log n + \min\left(\frac{m}{n}, \frac{\delta_G}{\varepsilon}\right)\right)\right) \\ &= O\left(\max\left(\log n, \min\left(\frac{m}{n}, \frac{\delta_G}{\varepsilon}\right)\right) n^2\right). \end{aligned}$$

**Theorem 4.** *Let $G = (V, E)$ be an undirected graph with $n$ vertices and $m$ edges.*

1. *If its edges are weighted by nonnegative real numbers, Algorithm* 1 *calculates a minimum cut in $O(\max(\log n, \min(m/n, \delta_G/\varepsilon))n^2)$ time, where $\varepsilon$ is the minimal weight of an edge.*
2. *If its edges are weighted by nonnegative integers, Algorithm* 1 *calculates a minimum cut in $O(\max(\log n, \min(m/n, \delta_G))n^2)$ time.*

**Algorithm 1**: The Minimum Cut Algorithm

```
Input: An undirected graph $G = (V, E)$ with weights $w$
Output: A set cutof vertices of $G$ forming a minimal cut of weight $\tau$.

cut ← $\emptyset$, $\tau \leftarrow \infty$

while $|V| >= 2$ do
    forall $v \in V$ do
        Q.insert($v$, 0)
        if deg($v$) $< \tau$ then $\tau \leftarrow$ deg($v$), cut ← $[v]$
    end
    while Q is not empty do
        adj ← Q.maxKey()
        $v \leftarrow$ Q.extractMax()
        forall $u \in V$ with $(v, u) \in E$ do
            if $u \in$ Q then Q.increaseKey($u$, Q.key($u$) $+ w(v, u)$)
        end
        if $\tau \leq$ adj then
            contract $v$ into $u$
            $[u] \leftarrow [u] \cup [v]$
        end
        $u \leftarrow v$
    end
end
```

As the following examples show, our estimation of $n - 1$ required rounds is strict (Example 2), but in many cases the algorithm needs less rounds (Examples 1 and 3), leading to a lower runtime in many cases.

**Example 1.** If $G = (V, E)$ has vertices $v_1, \ldots, v_n$ and edges $(v_i, v_{i+1})$ for $1 \leq i < n$ each with weight 1, i.e. it is an unweighted straight line, then $v_1, \ldots, v_n$ is a LA-order and $v_1$ is a vertex of minimum degree 1. Hence the algorithm contracts all vertices of $G$ in the first and only round, requiring $O(n + m + \log n)$ time.

**Example 2.** If $G$ is an unweighted circle with $n$ vertices $v_1, \ldots, v_n$, i.e. it has edges $(v_i, v_{i+1})$ for $1 \leq i < n$ and $(v_n, v_1)$, each with weight 1. Then $v_1, \ldots, v_n$ is a LA-order and each vertex has minimum degree. Since the adjacencies of the $v_i$, except for $v_n$, are all 1, only $v_{n-1}$ and $v_n$ are contracted, leading to a circle with one vertex less. Hence the algorithm requires $(n - 1)$ rounds, as does the original algorithm of Stoer and Wagner, leading to the same runtime.

**Example 3.** On random graphs with $n$ vertices and $m$ randomly chosen edges, the number of rounds required by the algorithm varies depending on $n$ and $m$. For graphs with $n$ between 1000 and 5000 and $m$ between 1000 and $200n$, Figure 1 shows the number of rounds over the ratio $m/n$. The experiments showed that the minimal number of required rounds increases with the edge–vertex ratio, i.e. the average degree in the random graph.

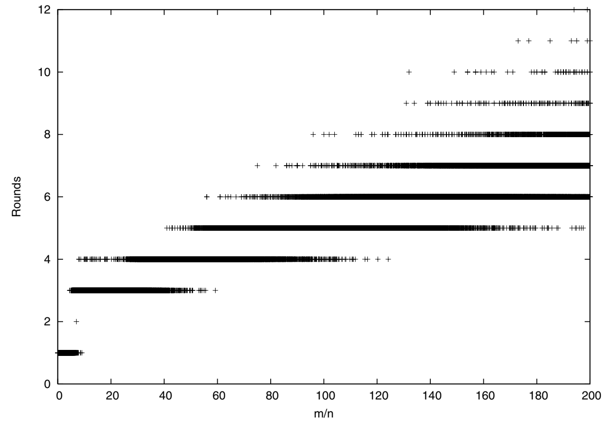

**Fig. 1.** The number of rounds required for random graphs.

Obviously an LA-order for threshold $\tau$ is an LA-order for a lower threshold $\tau' < \tau$. Hence we may decrease the weight during a round, possibly increasing the number of contractions. This observation may be used in two ways. First, the degree of the newly contracted node may be compared with the current threshold. If it is smaller, then the new node describes a cut of lower weight. Secondly, the set of all scanned vertices is a cut of the graph. Its weight can be updated easily, since for each cut $C$ and each vertex $v \notin C$, the following holds:

$$w(C \cup \{v\}) = w(C) + \deg(v) - 2 \sum_{\substack{(v,u) \\ u \in C}} w(v, u).$$

Translated to our situation, the weight of the cut increases by the degree of the added node minus twice its adjacency. If the resulting weight is lower than the currently known bound for the minimum cut, we may continue using the better, i.e. lower, value instead. These facts may be used to improve Algorithm 1, i.e. to lower the threshold during the course of one round.

## 5.1. *Priority Queues with Threshold*

Up to this point we did not use the advantages of the threshold $\tau$ of lax adjacency orders. Since we do not have to differentiate between vertices with priorities above $\tau$, we may use an alternative data structure, leading to a decreased worst case runtime $O(\delta_G n^2)$.

We describe a *priority queue with threshold* $\tau$, which allows the same operations as a priority queue, namely `insert`, `increaseKey` and `extractMax`. The two first operations behave as usual, but the third is changed slightly. `extractMax` returns an entry of maximal priority, as a regular priority queue does, if the maximal priority is lower than the threshold $\tau$. Otherwise, it returns an entry with a not necessarily maximal priority $\geq \tau$.

A priority queue with threshold $\tau$ consists of $\tau + 1$ buckets $B_0, \ldots, B_\tau$. For $0 \leq j \leq \tau - 1$ the bucket $B_j$ contains all entries $(v, j)$ with value $v$ and priority $j$. The last bucket,

$B_\tau$ contains all entries $(v, p)$ with $p \geq \tau$ in arbitrary order. In addition we maintain the maximal index of a nonempty bucket in $j_{\max}$. The operations are implemented in the following way:

– `insert`$(v, p)$: Insert $(v, p)$ into $B_{\min(\tau,p)}$ and update $j_{\max}$ if necessary. This requires $O(1)$ time.
– `increaseKey`$((v, p), q)$: If $\min(\tau, q) > p$ the remove $(v, p)$ from $B_p$ and insert $(v, q)$ into $B_{\min(\tau,q)}$ and update $j_{\max}$ if necessary. This takes $O(1)$ time.
– `extractMax`: Simply remove the first entry from bucket $B_{j_{\max}}$. If this was the last element in the bucket, search for the next nonempty bucket in decreasing order starting at $j_{\max} - 1$. In the worst case all $\tau$ buckets have to be examined, requiring time $O(\tau)$.

Applying the same estimations as for Theorem 4, we obtain the following result.

**Theorem 5.** *A minimum cut of an undirected graph $G = (V, E)$ with edges weighted by nonnegative integers, can be calculated in $O(\delta_G n^2 + m) = O(\delta_G n^2)$ time.*

*Proof.* As seen above, we require time

$$O\left(n^2 + n^2 T_{\text{insert}} + n^2 T_{\text{extractMax}} + \left(m + \min\left(\frac{m}{n}, \delta_G n^2\right)\right) T_{\text{increaseKey}}\right).$$

Using priority queues with thresholds, leads to a time of

$$O\left(n^2 + n^2 + \delta_G n^2 + \left(m + \min\left(\frac{m}{n}, \delta_G n^2\right)\right)\right) = O(\delta_G n^2). \qquad \square$$

By using a technique of Nagamochi and Ibaraki [NI2] we may reduce the asymptotic runtime to $O(\lambda_G n^2)$.

**Corollary 2.** *Let $G = (V, E)$ be an undirected, integer weighted graph.*

1. *Given $\tau > 0$ we may check in $O(\tau n^2)$ time, whether $\lambda_G < \tau$. If this is the case a minimum cut can be computed in the same time.*
2. *A minimum cut of $G$ can be calculated in $O(\lambda_G n^2)$ time.*

*Proof.* Following Theorem 2, we have $\min(\lambda_G, \tau) = \min(\delta_G, \lambda_{G^\tau}, \tau)$. Hence we may calculate $\min(\lambda_G, \tau)$ by using our algorithm with a starting threshold of $\tau$. This requires $O(\tau n^2)$ time. If $\lambda_G < \tau$, we obtain a minimum cut of $G$, and part 1 is proved.

We can use the first part for the second part. For increasing $i = 1, 2, \ldots$ check whether $\lambda_G < 2^i$ using the above algorithm. This is repeated until $2^{i-1} \leq \lambda_G < 2^i$. In this case we obtain a minimum cut. The total runtime is

$$\begin{aligned} O(2n^2) + \cdots + O(2^{i-1}n^2) + O(2^i n^2) &= O((2 + \cdots + 2^i)n^2) = O(2^{i+1}n^2) \\ &= O(\lambda_G n^2). \qquad \square \end{aligned}$$

## References

[CGK$^+$] Chandra Chekuri, Andrew V. Goldberg, David R. Karger, Matthew S. Levine, and Clifford Stein. Experimental study of minimum cut algorithms. In *Proccedings of the Symposium on Discrete Algorithms*, pages 324–333, 1997.

[FF] Lestor R. Ford and Delbert R. Fulkerson. Maximal flow through a network. *Canad. J. Math.*, 8:399–404, 1956.

[G] Harold N. Gabow. A matroid approach to finding edge connectivity and packing arborescences. *J. Comput. System Sci.*, 50(2):259–273, 1995.

[GH] Ralph E. Gomory and T. C. Hu. Multi-terminal network flows. *J. Soc. Indust. Appl. Math.*, 9:551–570, 1961.

[GT] Andrew V. Goldberg and Robert E. Tarjan. A new approach to the maximum flow problem. *J. ACM*, 35:921–940, 1988.

[K1] David R. Karger. Minimum cuts in near-linear time. In *Proc. STOC*, pages 56–63, 1996.

[K2] David R. Karger. Minimum cuts in near-linear time. *CoRR*, cs.DS/9812007, 1998.

[KS] David R. Karger and Clifford Stein. A new approach to the minimum cut problem. *J. ACM*, 43(4):601–640, 1996.

[M] David W. Matula. A linear time 2+epsilon approximation algorithm for edge connectivity. In *Proc. SODA*, pages 500–504, 1993.

[NI1] Hiroshi Nagamochi and Toshihide Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM J. Discrete Math.*, 5(1):54–66, 1992.

[NI2] Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse $k$-connected spanning subgraph of a $k$-connected graph. *Algorithmica*, 7(5&6):583–596, 1992.

[NI3] Hiroshi Nagamochi and Toshihide Ibaraki. Graph connectivity and its augmentation: applications of MA orderings. *Discrete Appl. Math.*, 123(1-3):447–472, 2002.

[NII] Hiroshi Nagamochi, Toshimasa Ishii, and Toshihide Ibaraki. A simple proof of a minimum cut algorithm and its applications. *Inst. Electron. Inform. Comm. Eng. Trans. Fundamentals*, E82-A(10):2231–2236, Oct. 1999.

[NNI] Hiroshi Nagamochi, Kazuhiro Nishimura, and Toshihide Ibaraki. Computing all small cuts in undirected networks. In Ding-Zhu Du and Xiang-Sun Zhang, editors, *ISAAC*, pages 190–198. Volume 834 of Lecture Notes in Computer Science. Springer, Berlin, 1994.

[NOI] Hiroshi Nagamochi, Tadashi Ono, and Toshihide Ibaraki. Implementing an efficient minimum capacity cut algorithm. *Math. Program.*, 67:325–341, 1994.

[PR] Manfred Padberg and Giovanni Rinaldi. An efficient algorithm for the minimum capacity cut problem. *Math. Program.*, 47:19–36, 1990.

[SW] Mechtild Stoer and Frank Wagner. A simple min-cut algorithm. *J. ACM*, 44(4):585–591, 1997.

*Received November* 8, 2005 *and in final form February* 6, 2006. *Online publication May* 11, 2007.