SERIE B — INFORMATIK

A Simple Min Cut Algorithm

Mechtild Stoer *
Frank Wagner**

B 94–12
May 1994

Abstract

We present an algorithm for finding the minimum cut of an edge-weighted graph. It is simple in every respect. It has a short and compact description, is easy to implement and has a surprisingly simple proof of correctness. Its runtime matches that of the fastest algorithm known. The runtime analysis is straightforward. In contrast to nearly all approaches so far, the algorithm uses no flow techniques. Roughly spoken the algorithm consists of about $|V|$ nearly identical phases each of which is formally similar to Prim's minimum spanning tree algorithm.

*Konrad-Zuse-Zentrum für Informationstechnik Berlin, Heilbronner Str. 10, 10711 Berlin, Germany, e-mail: stoer@zib-berlin.de.
**Institut für Informatik, Fachbereich Mathematik und Informatik, Freie Universität Berlin, Takustraße 9, 14195 Berlin-Dahlem, Germany, e-mail: wagner@math.fu-berlin.de.

1 Overview

Graph connectivity is one of the classical subjects in graph theory, and has many practical applications, e. g. in chip and circuit design, reliability of communication networks, transportation planning and cluster analysis. Finding the minimum cut of an edge-weighted graph is a fundamental algorithmical problem. Precisely, it consists in finding a nontrivial partition of the graph's vertex set V into two parts such that the sum of the weights of the edges connecting the two parts is minimum. The usual approach to solve this problem is to use its close relationship to the maximum flow problem. The famous Max-Flow-Min-Cut-Theorem by Ford and Fulkerson [FF56] showed the duality of the maximum flow and the so-called minimum s-t cut. There, s and t are two vertices which are the source and the sink in the flow problem and have to be separated by the cut, i.e., they have to lie in different parts of the partition. Until recently all cut algorithms were essentially flow algorithms using this duality. Finding a minimum cut without specified vertices to be separated can be done by finding minimum s-t cuts for all $\mathcal{O}(|V|^2)$ pairs of vertices and then selecting the lightest one. Gomory and Hu [GH61] reduced this to $\mathcal{O}(|V|)$ pairs of vertices and thus $\mathcal{O}(|V|)$ maximum flow computations.
Recently Hao and Orlin [HO92] showed how to use the maximum flow algorithm by Goldberg and Tarjan [GT88], the fastest so far, in order to solve the minimum cut problem as fast as the maximum flow problem, i.e., in time $\mathcal{O}(|V||E|\log(|V|^2/|E|))$. In the same year Nagamochi and Ibaraki [NI92a] published the first minimum cut algorithm that is not based on a flow algorithm, has the slightly better running time of $\mathcal{O}(|V||E| + |V|^2 \log |V|)$ but is still rather complicated. In the unweighted case they use a fast search technique to decompose a graph's edge set E into subsets $E_1, \ldots, E_\lambda$ such that the union of the first k E_i's is a k-edge-connected spanning subgraph of the given graph and has size at most $k|V|$. They simulate this approach in the weighted case. Their work is one of a number of papers treating questions of graph connectivity by non-flow-based methods [NP89, NI92a, M93].
In this context we present in this paper a remarkably simple minimum cut algorithm with the optimal running time established in [NI92b]. We reduce the complexity of the algorithm of Nagamochi and Ibaraki by avoiding the unnecessary simulated decomposition of the edge set. This enables us to give a comparably straightforward proof of correctness avoiding e. g. the distinction between the unweighted, integer-, rational-, and real-weighted case.

2 The Algorithm

Throughout the paper we deal with an ordinary undirected graph G with vertex set V and edge set E. Every edge e has positive real weight $w(e)$.
In order to describe the idea of the algorithm we start by reminding the reader of Prim's minimum spanning tree algorithm:

MinimumSpanningTree(G, w, a)

$A \leftarrow \{a\}$
$T \leftarrow \emptyset$
while $A \neq V$
    add to A the most loosely connected vertex
    add to T the connecting edge

A subset A of the graph's vertices grows starting with an arbitrary single vertex until A is equal to V. In each step the vertex outside of A *most loosely connected* with A is added. Formally, we add a vertex

$$z \notin A \text{ such that } w(a,z) = \min\{w(b,y) \mid b \in A, y \notin A, by \in E\}$$

where $w(b,y)$ is the weight of edge by and az is the *connecting* edge.
At the end the set of all the connecting edges then forms a minimum spanning tree.
The simple minimum cut algorithm we describe here consists of $|V|-1$ phases each of which is very similar to Prim's algorithm:

MinimumCutPhase(G, w, a)
$A \leftarrow \{a\}$
while $A \neq V$
    add to A the most tightly connected vertex
store the cut-of-the-phase and shrink G by merging the two vertices added last

A subset A of the graph's vertices grows starting with an arbitrary single vertex until A is equal to V. In each step the vertex outside of A *most tightly connected* with A is added. Formally, we add a vertex

$$z \notin A \text{ such that } w(A,z) = \max\{w(A,y) \mid y \notin A\}$$

where $w(A,y)$ is the sum of the weights of all the edges between A and y.
At the end of each such phase the two vertices added last are *merged*, i.e., the two vertices are replaced by a new vertex, and any edges from the two vertices to a remaining vertex are replaced by an edge weighted by the sum of the weights of the previous two edges. Edges joining the merged nodes are removed.
The cut of V that separates the vertex added last from the rest of the graph is called the *cut-of-the-phase*. The lightest of these cuts-of-the-phase is the result of the algorithm, the desired minimum cut.

MinimumCut(G, w, a)
while $|V| > 1$
    MinimumCutPhase(G, w, a)
    if the cut-of-the-phase is lighter than the current minimum cut
        then store the cut-of-the-phase as the current minimum cut

Notice that the starting vertex a stays the same throughout the whole algorithm.

3 An Example

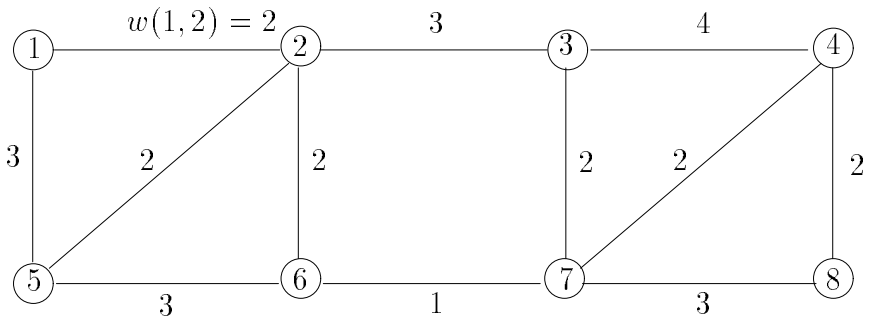

Figure 1: A graph $G = (V, E)$ with edge-weights.

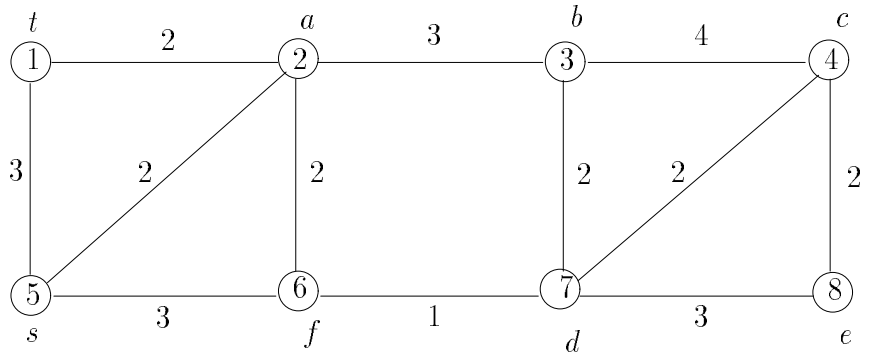

Figure 2: The graph after the first MINIMUMCUTPHASE(G, w, a), $a = 2$, and the induced ordering a, b, c, d, e, f, s, t of the vertices. The first *cut-of-the-phase* corresponds to the partition $\{1\}, \{2, 3, 4, 5, 6, 7, 8\}$ of V with weight $w = 5$.

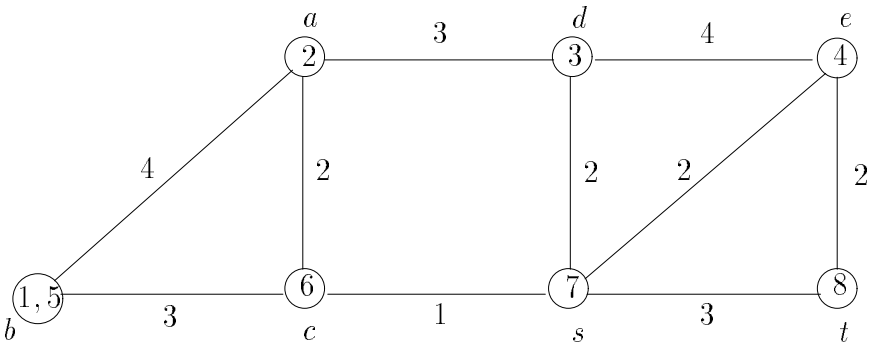

Figure 3: The graph after the second MINIMUMCUTPHASE(G, w, a), and the induced ordering a, b, c, d, e, s, t of the vertices. The second *cut-of-the-phase* corresponds to the partition $\{8\}, \{1, 2, 3, 4, 5, 6, 7\}$ of V with weight $w = 5$.

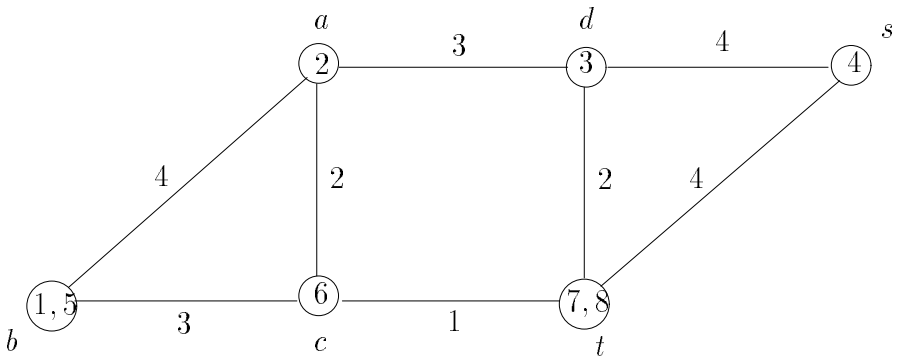

Figure 4: After the third MINIMUMCUTPHASE(G, w, a). The third *cut-of-the-phase* corresponds to the partition $\{7, 8\}, \{1, 2, 3, 4, 5, 6\}$ of V with weight $w = 7$.

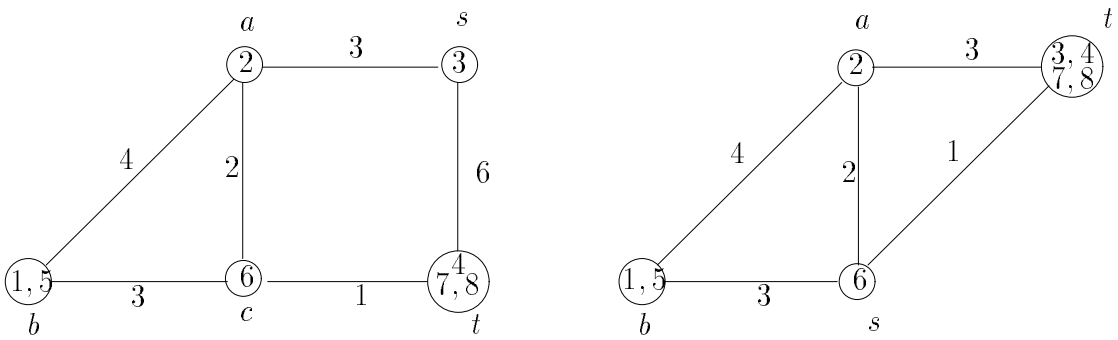

Figure 5: After the fourth and fifth MINIMUMCUTPHASE(G, w, a), respectively. The fourth *cut-of-the-phase* corresponds to the partition $\{4, 7, 8\}, \{1, 2, 3, 5, 6\}$. The fifth *cut-of-the-phase* corresponds to the partition $\{3, 4, 7, 8\}, \{1, 2, 5, 6\}$ with weight $w = 4$.

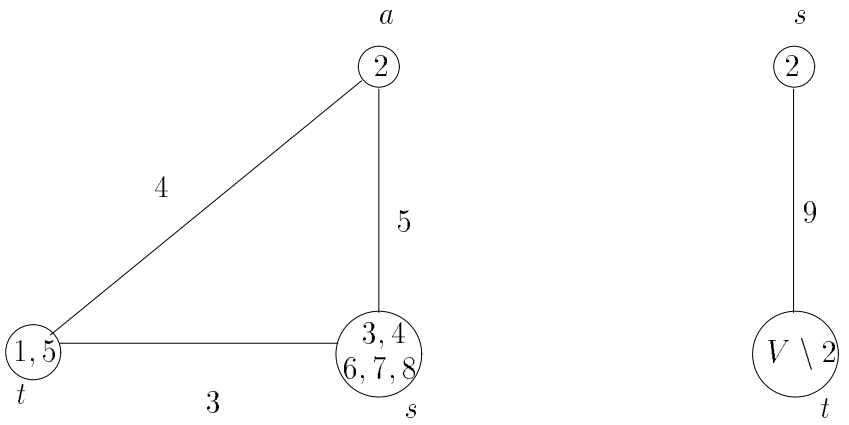

Figure 6: After the sixth and seventh MinimumCutPhase(G, w, a) respectively. The sixth *cut-of-the-phase* corresponds to the partition $\{1,5\}, \{2,3,4,6,7,8\}$ with weight $w = 7$. The last *cut-of-the-phase* corresponds to the partition $\{2\}, V \setminus \{2\}$; its weight is $w = 9$. The minimum cut of the graph G is the fifth *cut-of-the-phase* and the weight is $w = 4$.

4 Correctness

The core of the proof of correctness is the following somewhat surprising lemma.

Lemma *Each cut-of-the-phase is a minimum s-t cut in the current graph, where s and t are the two vertices added last in the phase.*

Assuming, that the lemma holds, we can show by a simple case distinction, that the smallest of these cuts-of-the-phase is indeed the minimum cut we are looking for. This is done by induction on $|V|$. The case $|V| = 2$ is trivial. If $|V| \geq 3$, look at the first phase: If G has a minimum cut, that is at the same time a minimum s-t cut, then, according to the lemma, the cut-of-the phase is already a minimum cut. If not, then G has a minimum cut with s and t on the same side. Therefore, a minimum cut of G', the input graph of phase 2, that differs from G by the merging of s and t, is a minimum cut of G. Now, by induction, the lightest of the cuts-of-the-phases 2 to $|V| - 1$ is such a minimum cut of G'. Notice that the application of phases 2 to $|V| - 1$ to G' is the same as the application of the complete algorithm to G'.

Finally, we show the claimed property of the cut-of-the-phase. The run of a MinimumCutPhase orders the vertices of the current graph linearly, starting with a and ending with s and t, according to their order of addition to A. Now we look at an arbitrary s-t cut C of the current graph and show, that it is at least as heavy as the cut-of-the-phase.

We call a vertex $v \neq a$ *active* (with respect to C) when v and the vertex added just before v are in different parts of C. Let $w(C)$ be the weight of C, A_v the set of all vertices added before v (excluding v), C_v the partition of $A_v \cup \{v\}$ induced by C, and $w(C_v)$ the weight of the induced cut, i.e., the sum of the weights of the edges going from one part of the induced partition to the other.
We show that for every active vertex v

$$w(A_v, v) \leq w(C_v)$$

by induction on the set of active vertices:
For the first active vertex the inequality is satisfied with equality. Let the inequality be true for all active vertices added up to the active vertex v, and let w be the next active vertex that is added. Then we have

$$w(A_w, w) = w(A_v, w) + w(A_w \setminus A_v, w) =: \alpha$$

Now, $w(A_v, w) \leq w(A_v, v)$ as v was chosen as the vertex most tightly connected with A_v. By induction $w(A_v, v) \leq w(C_v)$. All edges between $A_w \setminus A_v$ and w connect the different parts of C. Thus they contribute to $w(C_w$ but not to $w(C_v)$. So

$$\alpha \leq w(C_v) + w(A_w \setminus A_v, w) \leq w(C_w)$$

As t is always an active vertex with respect to C we can conclude that $w(A_t, t) \leq w(C_t)$ which says exactly that the cut-of-the-phase is at most as heavy as C.

5 Running Time

As the running time of the algorithm MinimumCut is essentially equal to the added running time of the $|V| - 1$ runs of MinimumCutPhase, which is called on graphs with decreasing number of vertices and edges, it suffices to show that a single MinimumCutPhase needs at most $\mathcal{O}(|E| + |V| \log |V|)$ time yielding an overall running time of $\mathcal{O}(|V||E| + |V|^2 \log |V|)$.
The key to implementing a phase efficiently is to make it easy to select the next vertex to be added to the set A, the most tightly connected vertex. During execution of a phase, all vertices that are not in A reside in a priority queue based on a key field. The key of a vertex v is the sum of the weights of the edges connecting it to the current A, i.e., $w(A, v)$. Whenever a vertex v is added to A we have to perform an update of the queue. v has to be deleted from the queue, and the key of every vertex w not in A, connected to v has to be increased by the weight of the edge vw, if it exists. As this is done exactly once for every edge, overall we have to perform $|V|$ ExtractMax and $|E|$ IncreaseKey operations. Using Fibonacci heaps [FT87], we can perform an ExtractMax operation in $\mathcal{O}(\log |V|)$ amortized time and a IncreaseKey operation in $\mathcal{O}(1)$ amortized time.
Thus the time we need for this key step that dominates the rest of the phase, is $\mathcal{O}(|E| + |V| \log |V|)$.
Notice, that this runtime analysis is very similar to the analysis of Prim's minimum spanning tree algorithm.

Acknowledgement

The authors thank Dorothea Wagner for her helpful remarks.

References

[FT87] M. L. Fredman and R. E. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms*, Journal of the ACM **34** (1987) 596–615

[FF56] L. R. Ford, D. R. Fulkerson, *Maximal flow through a network*, Canadian Journal on Mathematics **8** (1956) 399–404

[GT88] A. V. Goldberg and R. E. Tarjan, *A new approach to the maximum flow problem*, Journal of the ACM **35** (1988) 921–940

[GH61] R. E. Gomory, *Multi-terminal network flows*, Journal of the SIAM **9** (1961) 551–570

[HO92] X. Hao and J. B. Orlin, *A faster algorithm for finding the minimum cut in a graph*, 3rd ACM-SIAM Symposium on Discrete Algorithms (1992) 165–174

[M93] D. W. Matula *A linear time* $2+\epsilon$ *approximation algorithm for edge connectivity* , Proceedings of the 4th ACM-SIAM Symposium on Discrete Mathematics (1993) 500–504

[NI92a] H. Nagamochi and T. Ibaraki, *Linear time algorithms for finding a sparse k-connected spanning subgraph of a k-connected graph*, Algorithmica **7** (1992) 583–596

[NI92b] H. Nagamochi and T. Ibaraki, *Computing edge-connectivity in multigraphs and capacitated graphs*, SIAM Journal on Discrete Mathematics **5** (1992) 54–66

[NP89] T. Nishizeki and S. Poljak, *Highly connected factors with a small number of edges*, Preprint (1989)

[P57] R. C. Prim, *Shortest connection networks and some generalizations*, Bell System Technical Journal **36** (1957) 1389–1401