Information Processing Letters 70 (1999) 251–254

Information Processing Letters

www.elsevier.com/locate/ipl

A correctness certificate for the Stoer–Wagner min-cut algorithm

Srinivasa R. Arikati [a,1], Kurt Mehlhorn [b,*]

[a] *Cadence Design Systems, San Jose, CA, USA*

[b] *Max-Planck-Institute for Informatics, Im Stadtwald, 66123 Saarbrücken, Germany*

Received 17 February 1999

Communicated by Prof. T. Lengauer

Abstract

The Stoer–Wagner algorithm computes a minimum cut in a weighted undirected graph. The algorithm works in $n-1$ phases, where n is the number of nodes of G. Each phase takes time $O(m+n\log n)$, where m is the number of edges of G, and computes a pair of vertices s and t and a minimum cut separating s and t. We show how to extend the algorithm such that each phase also computes a maximum flow from s to t. The flow is computed in $O(m)$ additional time and certifies the cut computed in the phase. © 1999 Elsevier Science B.V. All rights reserved.

We give a correctness certificate for the min-cut algorithm of Stoer and Wagner [6]. This algorithm refines the algorithm of Nagamochi and Ibaraki [5,4]; it has the same time complexity, but is simpler. The algorithm is deterministic. There are faster randomized algorithms [2,3]. No certificates are known for the randomized algorithms.

Let G be an undirected graph with n vertices and m edges. The algorithm of Stoer and Wagner works in $n-1$ phases. In each phase it identifies two vertices and a minimum cut separating them and then collapses the two vertices into one. The global min-cut is the minimum cut found in any phase. Each phase runs in time $O(m + n\log n)$. We extend the algorithm such that it also finds a flow equal to the min-cut in each phase. The additional time required to find the flow is $O(m)$ in each phase.

We concentrate on the first phase. For an edge $e = uv$ we use $w(e)$ or $w(u,v)$ to denote its weight (which is non-negative) and for a subset $A \subseteq V$ and a vertex z we use

$$w(A,z) = \sum_{x\in A} w(x,z)$$

to denote the weighted adjacency of z to A. The Stoer–Wagner algorithm numbers the vertices of G by a so-called *maximum adjacency search*. In the first step an arbitrary vertex $a \in V$ is selected and labeled 1 and A is initialized to $\{a\}$. In the ith step, $2 \leqslant i \leqslant n$, an unlabeled vertex x with maximal adjacency to A is selected, labeled i, and added to A. Let s and t be the two vertices labeled last.

Lemma 1 [6]. *$(V-\{t\},\{t\})$ is a minimum s-t-cut.*

Let $C = w(V-\{t\},\{t\})$. We will construct a flow of value C from s to t. From now on we identify nodes with their label (in particular, $t=n$ and $s=n-1$)

* Corresponding author. Email: mehlhorn@mpi-sb.mpg.de. http://www.mpi-sb.mpg.de/~mehlhorn.

[1] Email: arikati@cadence.com.

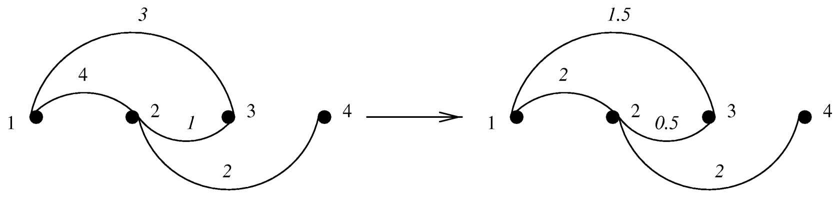

Fig. 1. A 2-reduction.

and we use A_i to denote the nodes 1 to i. Our main technical tool is the concept of an r-reduction where r is a non-negative real. An r-reduction changes the weight function w to a weight function w' such that
(1) $w'(e) \leqslant w(e)$ for all e,
(2) $w'(A_{i-1}, i) = \min(r, w(A_{i-1}, i))$ for all i.
Fig. 1 shows an example. We want to stress that an r-reduction does not uniquely determine the new weight function. The only requirements are that weights are only decreased and that the adjacency of any vertex is reduced to the minimum of r and its old adjacency.

Lemma 2. *Let w' be obtained by an r-reduction from w. Then a maximum adjacency search on (G, w') can reproduce the node labeling induced by w.*

Proof. We use induction on i. Clearly, node 1 can still be labeled 1. So, assume that nodes 1 to $i-1$ have been labeled identically and consider nodes i and j with $i < j$. Then

$$w(A_{i-1}, i) \geqslant w(A_{i-1}, j)$$

since i was labeled before j with respect to weight function w,

$$w(A_{i-1}, j) \geqslant w'(A_{i-1}, j)$$

since weights are only decreased,

$$w'(A_{i-1}, j) \leqslant r$$

since w' is obtained from w by an r-reduction, and

$$w'(A_{i-1}, i) = \min\bigl(r, w(A_{i-1}, i)\bigr).$$

Thus, $w'(A_{i-1}, i) \geqslant w'(A_{i-1}, j)$ and i can still be labeled before j. □

We define two paths, one starting from t and one starting from s. The path $p_t = v_0, v_1, \ldots$ starting from t is defined as follows. We set $v_0 = t$ and for $l \geqslant 1$ define v_l as the highest numbered node less than v_{l-1} such that $w(v_l, v_{l-1}) > 0$. The path p_s is defined analogously. In the graphs of Fig. 1 the path starting from t is 4, 2, 1 and the path starting from s is 3, 2, 1. We define the s-t-join as the highest numbered vertex that belongs to p_t and p_s.

Lemma 3.
(1) *For all i: if no node j, $j \geqslant i$, lies on p_s and p_t then $w(A_{i-1}, i) \geqslant C := w(A_{n-1}, n)$.*
(2) *If $C > 0$, the s-t-join exists and $w(A_{i-1}, i) \geqslant C$ for any i larger than the s-t-join.*

Proof. (1) We use induction on i. The claim is clearly true for $i = n$. So, assume $i < n$ and that no j, $j \geqslant i$, lies on p_s and p_t. Assume without loss of generality that i does not lie on p_t. Let $j > i$ be minimal such that j lies on p_t. Then $w(A_{i-1}, j) = w(A_{j-1}, j)$ since $w(l, j) = 0$ for all l, $i \leqslant l < j$, by definition of the path p_t. Also, $w(A_{j-1}, j) \geqslant C$ by induction hypothesis and $w(A_{i-1}, i) \geqslant w(A_{i-1}, j)$ by the definition of maximum adjacency search.

(2) We only need to show the existence of the s-t-join. Assume that the s-t-join does not exist. Then one of the paths p_s or p_t cannot extend all the way to node 1, say p_s. Let $i > 1$ be the last node on p_s. Then $w(A_{i-1}, i) = 0$. On the other hand, $w(A_{i-1}, i) \geqslant C > 0$ by part (1), a contradiction. □

Let k be the s-t-join and consider the path starting in s, following p_s to k, and then following (the reversal of) p_t to t. Call this path p and let c be the smallest weight of any edge on p. We can send c amount of flow from s to t along p.

Consider the following $(C-c)$-reduction. For any node v reduce the weight of the incoming edges, i.e., the edges connecting v to smaller numbered nodes, to $\min(C-c, w(A_v, v))$. It is completely arbitrary which

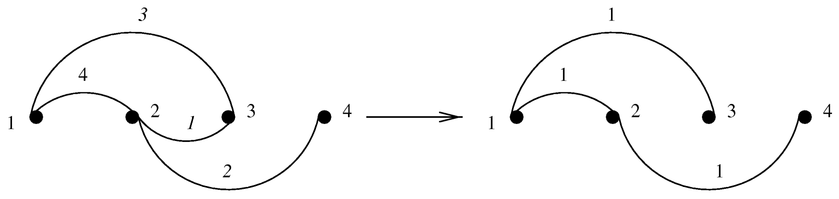

Fig. 2. We have $C = 2$, $k = 2$, $p = (4, 2, 3)$, and $c = 1$. We perform a 1-reduction. In this reduction the weight of any edge on p is reduced by one. The result of the 1-reduction is shown. In the resulting graph there is path q from s to t of weight one. The two paths p and q form a flow of value 2 from s to t.

weights are reduced except if v lies on p and is not the lowest numbered node on p. In the latter case the weight of the edge of p into v must be reduced by c. The reduction just described has the following properties:

- the weight of the edges on p is reduced by c, i.e., we constructed a flow of amount c through p;
- the same node labeling is still possible after the reduction by Lemma 2;
- $w'(A_{n-1}, n) = C - c$.

The complete algorithm is now as follows. We construct an an s-t-flow of value $C := w(A_{n-1}, n)$ incrementally by a sequence of reductions.

```
while $C > 0$
{ let $k$ be the $s$-$t$-join;
  define the path $p$ as above and let $c$ be the smallest
  weight of any edge on $p$.
  perform a $(C - c)$-reduction. In this reduction
  reduce the weight of all edges of $p$ by $c$.
}
```

The correctness of this algorithm follows directly from Lemmas 2 and 3. The first lemma guarantees that the reductions do not change the node numbering and the second lemma guarantees the existence of the path p and that $w'(A_{i-1}, i) = C - c$ for all nodes i with $i > k$ after the reduction. In particular, $w'(A_{n-1}, n) = C - c$ and hence the total weight of all paths constructed is C. Fig. 2 shows an example.

We still need to describe a linear time implementation. To this end we study the global effect of the algorithm on a vertex i. In each iteration the weight of zero or more edges ji with $j < i$ is decreased. If i lies on p and is larger than the s-t-join then we must reduce the weight of the edge on p; otherwise we have complete freedom in which weights to reduce. Let us agree that we reduce the weights of edges ji with j as small as possible. In other words we view the edges ji with $j < i$ ordered by increasing j and either reduce the weight of the edge on the left end of the list (if no flow is sent through i) or on the right end of the list (if flow is sent through i). Since we are only interested in the flow there is no need to make the former kind of reduction. This leads to the following implementation.

First, sort for each i the edges ji with $j < i$ in increasing order of j. This takes linear time by bucket sort. Then initialize the flow construction as follows. Saturate the edge st, if it exists and remove it. Let $C = w(A_{n-1}, n)$ (after the removal of st), saturate all edges into t, and push flow C out of s (starting with the edges leading to large j's). More precisely, set

$$f(j, t) = w(j, t) \quad \text{for all } j$$

and set

$$f(s, j) = \min\left(w(s, j), C - \sum_{l>j} f(s, l)\right) \quad \text{for all } j.$$

Also, set $i = n - 2$ and $C_i = C$. We will next iteratively decrement i down to 1. In the general step, the flow through the edges across the cut $(A_i, V - A_i)$ has already been fixed. The edges carry flow C_i from s and the same amount of flow towards t. Consider the edges ik with $k > i$. They carry a flow c_s from s and a flow c_t towards t. Let $C_{i-1} = C_i - \min(c_s, c_t)$. Assume without loss of generality that $c_s \geqslant c_t$. We forward flow $c_s - c_t$ along the edges ji with $j < i$. We do so by starting with the edges leading to large j's. By our previous arguments the capacity of the edges ji, $j < i$, suffices to forward the flow. Altogether we have established our claim that additional time $O(m)$ suffices to construct a maximum s-t-flow.

References

[1] J. Hao, J.B. Orlin, A faster algorithms for finding the minimum cut in a graph, in: Proc. 3rd Annual Symposium on Discrete Algorithms (SODA ’92), Vol. 3, ACM/SIAM, January 1992, pp. 165–174.

[2] D.R. Karger, Random sampling in cut, flow, and network design problems, in: Proc. 26th Annual ACM Symposium on Theory of Computing (STOC ’94), Montréal, Québec, May 1994, pp. 648–657.

[3] D.R. Karger, C. Stein, A new approach to the minimum cut problem, J. ACM 43 (4) (1996) 601–640.

[4] N. Nagamochi, T. Ibaraki, Computing edge-connectivity in multigraphs and capacitated graphs, SIAM J. Discrete Math. 5 (1) (1992) 54–66.

[5] N. Nagamochi, T. Ibaraki, A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph, Algorithmica 7 (1992) 583–596.

[6] M. Stoer, F. Wagner, A simple min-cut algorithm, J. ACM 44 (4) (1997) 585–591.