Lecture notes for "Analysis of Algorithms": **Global minimum cuts** (Draft)

Lecturer: *Uri Zwick* *

Abstract

We describe deterministic and randomized algorithms for finding global minimum cuts in *undirected* graphs.

1 Global and s-t cuts in undirected graphs

Let $G = (V, E, w)$ be a weighted undirected graph, where $w : E \to \mathbb{R}^+$ is a weight (or cost, or capacity) function defined on its edges. A (global) cut $\{S, T\}$ of G is a partition of V into two disjoint non-empty sets, i.e., $S, T \neq \phi$ and $S \cap T = \phi$. Edges in the set $E(S,T) = \{e \in E \mid |e \cap S| = 1\}$ are said to *cross* the cut. Let $s, t \in V$. A cut $\{S, T\}$ is said to be an s-t cut if and only if $|\{s,t\} \cap S| = 1$, i.e., $s \in S$ and $t \in T$, or $s \in T$ and $t \in S$. The weight $w(S,T)$ of a cut $\{S, T\}$ is defined to be the sum of the weights of the edges that cross the cut:

$$w(S,T) \; = \sum_{e \in E(S,T)} w(e) \; .$$

A cut $\{S, T\}$ of G is said to be a *global min-cut* if and only if the weight $w(S,T)$ of the cut is the smallest possible, i.e., for every other cut (S', T') of G we have $w(S,T) \leq w(S',T')$. An s-t *min-cut* is defined similarly.

Given $s, t \in V$, an s-t min-cut of G can be found using a network flow algorithm, relying on the celebrated max-flow min-cut algorithm. We show below that finding *global* min-cuts is easier.

If $A, B \subseteq V$ and $A \cap B = \phi$, we let $E(A,B) = \{e \in E \mid |e \cap A| = |e \cap B| = 1\}$ and $w(A,B) = \sum_{e \in E(A,B)} w(e)$. (Note that this is an extension of the definition above.) The following trivial facts would be used below: If $A = A_1 \cup A_2$ and $A_1 \cap A_2 = \phi$, then $w(A,B) = w(A_1,B) + w(A_2,B)$. If $A_1, A_2 \subseteq A$, $B_1, B_2 \subseteq B$ and $B_1 \cap B_2 = \phi$, then $w(A_1,B_1) + w(A_2,B_2) \leq w(A,B)$.

*School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. E–mail: zwick@tau.ac.il

Function `GlobalMinCut`(G)

if $V = \{a, b\}$ **then**
  return $(\{a\}, \{b\})$
else
  $\langle C_1, s, t\rangle \leftarrow$ `stMinCut`(G)
  $C_2 \leftarrow$ `GlobalMinCut`$(G/\{s,t\})$
  if $w(C_1) \le w(C_2)$ **then return** C_1 **else return** C_2

Figure 1: The global min-cut algorithm of Stoer and Wagner.

2 The Stoer-Wagner algorithm

Let $G = (V, E, w)$ be a weighted undirected graph. Let $\{S, T\}$ be a global min-cut of G. Suppose that $s, t \in V$. If $|\{s, t\} \cap S| = 1$, then $\{S, T\}$ is clearly also an s-t min-cut of G.

Stoer and Wagner [SW97] proposed the following method for finding a global min-cut of a graph G. Start by finding an s-t min-cut $\{S, T\}$ of G, for some two vertices $s, t \in V$. Then, by the above observation, either $\{S, T\}$ is also a global min-cut of G, or in any global min-cut of G vertices s and t must belong to the same side of the cut. The global min-cut in this case can be found by finding the global min-cut in the graph $G/\{s, t\}$ obtained by *merging* s and t into a new vertex st. If s and t are connected by an edge then this edge disappears. If s and t both have edges to some vertex v, then the weight of the edge from the new vertex st to v is $w(s, v) + w(t, v)$. (This is similar to the operation of contracting an edge, though in this case, we do not assume that s and t are connected by an edge.)

Two problems immediately come to mind. The first is how do we find an s-t min-cut of G? It was claimed above that finding an s-t min-cut is harder than finding a global min cut. The second problem is how do we know whether the s-t min-cut found is also a global min-cut of not?

The answer to the second question is simple. We simply compute a global min-cut in the smaller graph $G/\{s, t\}$, by a recursive call to the algorithm, and compare its weight to the weight of the s-t min-cut found. The cut with the smaller weight among these two is a global min-cut of G. The overall structure of the algorithm of Stoer and Wagner, which we call `GlobalMinCut` is given in Figure 1.

That leaves us with the more challenging problem of finding an s-t min-cut. Finding an s-t min-cut of G does indeed seem to be a harder problem than finding a global min-cut, when s and t are *specified*. Note that here we do not care who s and t are, we are just interested in finding a cut $\{S, T\}$ and two vertices s and t such that $\{S, T\}$ is an s-t min-cut. Indeed, s and t are not passed to procedure `stMinCut` called by `GlobalMinCut`. They are *returned* by it.

Finding a cut $\{S, T\}$ *and* a pair of vertices s, t such that $\{S, T\}$ is a min s-t cut is a much easier problem for which Stoer and Wagner [SW97] found a simple and elegant solution. A high level description of their algorithm, called `stMinCut` is given in Figure 2.

Function `stMinCut`(G)

$A \leftarrow \{a\}$
while $A \neq V$ **do**
  Let $v \notin A$ be such that $w(A, \{v\})$ is maximized
  $A \leftarrow A \cup \{v\}$
Let s and t be the last two vertices added to A
return $\langle (V - \{t\}, \{t\}), s, t \rangle$

Figure 2: High level description of `stMinCut`.

Lemma 2.1 *If s and t are the next to last and last vertices added to A by* `stMinCut`*, then $(V - \{t\}, \{t\})$ is an s-t min-cut of G.*

Proof: We prove the lemma by induction on the number of vertices in G. If $n = |V| = 2$, the claim is obvious. We assume, therefore, that the claim holds for all graphs with at most $n-1$ vertices and show that it also holds for all graphs with n vertices.

Let $G = (V, E)$ be a graph on n vertices. Let $v_1 = a, v_2, \ldots, v_{n-1} = s, v_n = t$ be the order in which the vertices of G are added to A. For every $1 \leq i \leq n$, let $V_i = \{v_1, v_2, \ldots, v_i\}$ and let $G_i = G[V_i]$ be the subgraph of G induced by V_i. It is easy to check that $v_1, v_2, \ldots, v_i$ is also an order in which `stMinCut` can add vertices to A when `stMinCut` is run on $G[V_i]$. [1]

Let $s = v_{n-1}$ and $t = v_n$ be the last two vertices added to A by `stMinCut`. Let $\{S, T\}$ be an arbitrary s-t min-cut. We have to show that $w(V - \{t\}, \{t\}) \leq w(S, T)$. Let $t' = v_i$ be the last vertex from $T - \{t\}$ added to A, and let $s' = v_{i+1}$. Also let $V' = V_{i+1}$, $S' = S \cap V'$ and $T' = T \cap V'$. Also let $S'' = \{v_{i+1}, \ldots, v_{n-1}\}$. By the definition of i we get that $S'' \subseteq S$. Also, $S' \subseteq S$ and $T - T' = \{v_n\}$. As $s' \in S'$ and $t' \in T'$, we get that (S', T') is an s'-t' cut in $G' = G[V']$. By the induction hypothesis, $w(V' - \{s'\}, \{s'\}) \leq w(S', T')$.

Consider now the iteration in which $s' = v_{i+1}$ was added to A by `stMinCut` when run on G. At that stage $A = V' - \{s'\} = \{v_1, v_2, \ldots, v_i\}$, and s' and t were both out of A. As s' was chosen by the algorithm, we get that $w(V' - \{s'\}, \{t\}) \leq w(V' - \{s'\}, \{s'\})$. (Recall that the algorithm chooses a vertex v that maximizes $w(A, \{v\})$.) Combined with the induction, hypothesis, we get that

$$w(\{v_1, v_2, \ldots, v_i\}, \{v_n\}) \;=\; w(V' - \{s'\}, \{t\}) \;\leq\; w(V' - \{s'\}, \{s'\}) \;\leq\; w(S', T') \;.$$

Putting everything together, we get that

$$\begin{array}{rcccc} w(V - \{t\}, \{t\}) & = & w(\{v_1, \ldots, v_i\}, \{v_n\}) & + & w(\{v_{i+1}, \ldots, v_{n-1}\}, \{v_n\}) \\ & \leq & w(S', T') & + & w(S'', T - T') \\ & \leq & & w(S, T) & \end{array}$$

$\square$

[1] We say here that this is an order in which `stMinCut` *can* add vertices to A as the order is not uniquely determined when there are ties.

```
Function stMinCut(G)
  PQ ← φ
  foreach u ∈ V do
    key[u] ← 0
    insert(PQ, u, key[u])
  s, t ← nil
  while PQ ≠ φ do
    u ← extractmax(PQ)
    s ← t ; t ← u
    foreach (u, v) ∈ E do
      if v ∈ PQ then
        key[v] ← key[v] + w(u, v)
        increasekey(PQ, v, key[v])
  return ⟨(V − {t}, {t}), s, t⟩
```

Figure 3: Complete description of `stMinCut`.

An efficient implementation of `stMinCut` using a priority queue is given in Figure 3. Note the similarity of the algorithm to the algorithms of Dijkstra and Prim. The following two theorems are now immediate.

Theorem 2.2 *The running time of* `stMinCut`*, when implemented using Fibonacci heaps, is* $O(m + n \log n)$.

Theorem 2.3 *The running time of* `GlobalMinCut` *is* $O(mn + n^2 \log n)$.

3 The Karger-Stein random contraction algorithm

The Stoer-Wagner algorithm performs n iterations in each of which a pair of vertices s and t is selected and then merged. Selecting each such pair of vertices takes $O(m + n \log n)$ time. Is there a faster way of selecting pairs of vertices to be merged?

Karger and Stein [KS96] suggest the following intriguing way of selecting the next pair of vertices to be merged: Choose a *random* edge and contract it. The probability of choosing each edge is proportional to its weight. In other words, an edge $e \in E$ is chosen with probability $w(e)/W$, where $W = \sum_{e \in E} w(e)$.

Contracting an edge may create self-loops and parallel edges. Self-loops are discarded. Parallel edges are either kept, or replaced by a single edge whose weight if the sum of the weights of the edges it replaces. The contraction of edges creates *super-vertices* which correspond to subset of vertices of the original graph. Edges are contracted until there are only two super-vertices left.

Function `RandMinCut`(G)

while $\lvert V\rvert > 2$ **do**
    Pick a random edge and contract it
    (The probability of choosing an edge is proportional to its weight.)
return the remaining two super-vertices

Figure 4: The random contraction algorithm of Karger.

These two remaining super-vertices are returned by the algorithm. A pseudo-code of the algorithm is given in Figure 4. What is the probability that the cut returned by `RandMinCut` is a global min-cut of G?

If $G = (V, E)$ is a graph and $e \in E$, we let G/e be the graph obtained from G by contracting e. If $e = \{s, t\}$, then the two vertices s, t in G are replaced by the super-vertex st. Similarly, if $S \subseteq V$, we let S/e be $S \cup \{st\} - \{s, t\}$, if $s, t \in S$, and $S/e = S$, otherwise. We claim:

We let $d_w(u) = \sum_{\{u,v\}\in E} w(u, v)$ be the *weighted degree* of $u \in V$. Note that if the graph is unweighted, i.e., $w(e) = 1$, for every $e \in E$, then $d_w(u)$ is just the *degree* of u, i.e., the number of edged incident to u.

Lemma 3.1 *Let $G = (V, E, w)$ be a weighted undirected graph. Let $\{S, T\}$ be a global min-cut of G. If $e \notin E(S, T)$, then $(S/e, T/e)$ is a global min-cut of G/e and $w(S/e, T/e) = w(S, T)$.*

Proof: Any cut of G/e corresponds to a cut with the same weight of G. Thus, the weight of any cut of G/e is at least $w(S, T)$. On the other hand, if $e \notin E(S, T)$, i.e., $e \subseteq S$ or $e \subseteq T$, then $w(S/e, T/e) = w(S, T)$, and hence $(S/e, T/e)$ is a global min cut of G. □

Lemma 3.2 *Let $G = (V, E)$ be an weighted undirected Let $\{S, T\}$ be a fixed global min-cut of G. Let $e \in E$ be a random edge selected with probability proportional to its weight. Then,* $\Pr[e \in E(S, T)] \leq \frac{2}{n}$*, where $n = |V|$ is the number of vertices of G.*

Proof: As each edge $e \in E$ is chosen with probability $w(e)/W$, where $W = \sum_{e\in E} w(e)$, the probability of choosing an edge from $E(S, T)$ is exactly $w(S, T)/W$.

For every $v \in V$, the cut $\{\{v\}, V - \{v\}\}$ is a cut of weight $d_w(v)$. As $\{S, T\}$ is a minimum cut, we get that $d_w(v) \geq w(S, T)$. Thus, $W = \sum_{e\in E} w(e) = \frac{1}{2}\sum_{v\in V} d_w(v) \geq \frac{n}{2} w(S, T)$.

As a consequence,

$$\Pr[e \in E(S, T)] \;=\; \frac{w(S, T)}{W} \;\leq\; \frac{w(S, T)}{\frac{n}{2}\, w(S, T)} \;=\; \frac{2}{n}\ .$$

□

Theorem 3.3 *Let $G = (V, E)$ be an weighted undirected graph. Let $\{S, T\}$ be a fixed global min-cut of G. Then, the probability that algorithm* `RandMinCut` *returns the cut $\{S, T\}$ is at least $\frac{1}{\binom{n}{2}}$.*

Function `FastRandMinCut`$(G = (V, E))$

if $|V| \le 6$ **then**
  return `GlobalMinCut`(G)
$t \leftarrow \lceil n/\sqrt{2} + 1 \rceil$
for $i \leftarrow 1$ **to** 2 **do**
  $G_i \leftarrow$ `RandContract`(G, t)
  $(S_i, T_i) \leftarrow$ `FastRandMinCut`(G_i)
if $w(S_1, T_1) \le w(S_2, T_2)$ **then**
  return (S_1, T_1)
else
  return (S_2, T_2)

Figure 5: The recursive random contraction algorithm of Karger and Stein.

Proof: The algorithm returns the cut $\{S, T\}$ if and only if no edge selected by the algorithms belongs to this cut. By Lemma 3.1 as long as this condition holds, the cuts corresponding to $\{S, T\}$ in the contracted graphs are all global min-cuts. By Lemma 3.2, we get that when the number of (super-)vertices in the graph is i, the probability that the next edge selected by the algorithm belongs to the cut $\{S, T\}$ is at most $\frac{2}{i}$. Thus, the probability that the cut $\{S, T\}$ is the cut returned by the algorithm is at least

$$\left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\left(1 - \frac{2}{n-2}\right) \cdots \frac{2}{4} \cdot \frac{1}{3} \;=\; \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdots \frac{2}{4} \cdot \frac{1}{3} \;=\; \frac{2}{n(n-1)} \,.$$

□

Corollary 3.4 *An undirected graph on n vertices has at most $\binom{n}{2}$ global min-cuts.*

4 The Karger-Stein recursive contraction algorithm

A randomized algorithm for computing global min-cuts using random edge contractions whose success probability is much larger than that of algorithm `RandMinCut` is given in Figure 5.

Let $T(n)$ be the running time of algorithm `FastRandMinCut` on an n-vertex graph and let $P(n)$ be the probability that it returns a specific global min-cut. We have

$$T(n) \;=\; 2T(\lceil \frac{n}{\sqrt{2}} + 1 \rceil) + O(n^2) \,.$$

It is not difficult to see that $T(n) = O(n^2 \log n)$. We also have

$$P(n) \;=\; 1 - \left(1 - \frac{1}{2} P(\lceil \frac{n}{\sqrt{2}} + 1 \rceil)\right)^2 \;=\; P(\lceil \frac{n}{\sqrt{2}} + 1 \rceil) - \frac{1}{4} P(\lceil \frac{n}{\sqrt{2}} + 1 \rceil)^2 \,.$$

Define a sequence p_k by $p_1 = 1$ and

$$p_{k+1} \;=\; p_k - \frac{1}{4}p_k^2 \quad , \quad \text{for } k > 1 \;.$$

It is easy to see that $P(n) = p_k$, where $k = 2\log_2 n + O(1)$ is the number of recursive calls made by algorithm.

We prove by induction that $p_k \geq \frac{1}{k}$. For $k = 1$, the claim clearly holds. For $k > 1$ we get $p_{k+1} \geq \frac{1}{k} - \frac{1}{4k^2} \geq \frac{1}{k+1}$, as required. (The fact that $p_{k+1} \geq \frac{1}{k} - \frac{1}{4k^2}$ follows from the induction hypothesis and for the fact that the function $f(x) = x - \frac{x^2}{4}$ is an increasing function for $0 < x \leq 2$.)

Putting everything together, we get that $P(n) = \Omega(\frac{1}{\log n})$.

Thus to obtain a global min-cut with a probability of, say, $1 - \frac{1}{n}$, we need to run `FastRandMinCut` only $\Theta(\log^2 n)$ times, giving us a total running time of $O(n^2 \log^3 n)$.

References

[KS96] D.R. Karger and C. Stein. A new approach to the minimum cut problem. *Journal of the ACM*, 43:601–640, 1996.

[SW97] M. Stoer and F. Wagner. A simple min-cut algorithm. *Journal of the ACM*, 44(4):585–591, 1997.