Karger 2000 ppt

2021년 3월 8일 월요일 오후 9:20

Journal of the ACM ~~2020~~ 2000

Minimum Cuts in Near-Linear Time

by D. R. Karger

Presented by TAMREF (Changki Yun, **I am worried**)

2021.03.08 project-tcs

0. Introduction

The Problem

We solve the *minimum cut* problem (Min-Cut) in undirected, (weighted) graph.

- In undirected graph G, a *cut* on $\emptyset \neq S \subsetneq V(G)$ is set of edges $uv \in E(G)$, such that $u \in S$ and $v \notin S$. Set of such edges are called *cutset*.
  - There are at most $2^n - 2$ different cutsets if n-vertex graph.
- In weighted graph, *weight of a cut* means sum of weights belongs to the cutset.
- Cut with minimum weight (or cutset, weight itself) is called the *minimum cut* or *edge connectivity*.

Clarifications

- We are solving Global Min-Cut.
  - (s, t) min-cut requires minimum edge-weights to disconnect specific pair of vertices s and t.
- Weights are considered to be **positive integers**.
  - This allows us to perceive them as *multi-edges*.
  - Min-Cut with negative weights is NP-hard, equivalent to MAXCUT.

History

Flow approach

Min-Cut equals the minimum among (s, t) min-cuts. By Maxflow-Mincut theorem, it is equal to (s, t)-maxflow.

Hence $\binom{n}{2}$ max-flow gives the answer.

- Actually, $n - 1$ flow computations is enough. (Gomory-Hu 1961)

However, even $\tilde{O}(E)$ flow computation requires about $\tilde{O}(nm)$ time. It's far from near-linearity ($\tilde{O}(m)$), our goal.

Contraction approach

If $e \in G$ does not belong to Min-Cut, Min-Cut on G equals to the Min-Cut on G/e.

- Karger's algorithm randomly contract edges. This Monte-Carlo algorithm guesses Min-Cut *with high probability* in $O(n^2 \log^3 n)$ time.
- Stoer-Wagner algorithm runs in $O(nm + n^2 \log n)$ time.

Nice approach, but still far from the goal.

Tree-packing approach

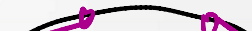
Theorem (NW). If Min-Cut is c, there are at least $\frac{c}{2}$ edge-disjoint spanning trees **(tree-packing)**.

Number the trees in maximum tree packing, by $T_1, \cdots, T_k$ by $2k \geq c$.

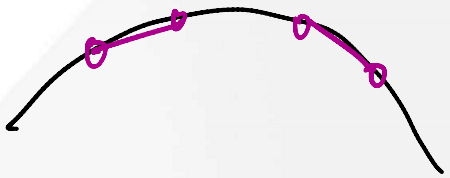
For any Min-Cut, they must exhaust the cutset.

- If e in cutset C does not belong to any T_i, $C - \{e\}$ is still a cutset, violating the minimality. Note that $G - T_1 - \cdots - T_k$ is already disconnected.

Hence, pigeon-hole principle guarantees that **for any min-cut C,**
There is a spanning tree T_α in the packing such that $|E(T_\alpha) \cap C| \leq 2$.
The situation will be named as **"C 2-respects T"** or **"T 2-constrains C"**.



Tree-packing approach : Question 1



Suppose we barely found such tree. Then what?

Note that for any spanning tree T, fetching $E(T_\alpha) \cap C$ instead of whole C is enough.

- For any $u, v \in V(G)$, u and v are separated by C **iff** the unique shortest path in T between u, v crosses with odd number of cut edges.

Hence, enough to line-up all edge subset of $E(T)$ with size ≤ 2, and examine the cut weights.

Naive approach costs about $O(mn^2)$. But Karger(2000) reduces this to $O(m \log^2 n)$

.

Tree-packing approach : Question 2

OK. Finding T that 2-constrains an arbitrary Min-Cut is good. But there are $c/2$ indistinguishable candidates of them in Maximum Tree Packing!

It's the main difficulty of the problem. We apply *random sampling technique* to reduce the number of candidates. Detailed answer is bit involved.

Tree-packing approach : Question 3

Do we know anything fast about finding the maximum tree packing?

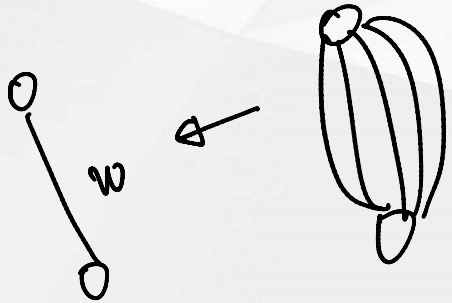
- The most naive algorithm by Edmonds is prohibitively slow.
- Gabow(1992) developed an algorithm $c/2$ edge-disjoint spanning trees in $O(mc\log n)$ time, which would be hired for us.
  - More background stays behind, but we don't discuss about it.
- Although Gabow's algorithm may not return the *maximum* tree packing, but we will guarantee that **for any min-cut C, many trees in the selected packing 2-respects C**, which is crucial for us.

Combined with random sampling technique, we will also drop the cost related to c to $O(\log n)$.

Tree-packing approach : Question 4

Nash-Williams does not holds for weighted graph! Is it Ok for 2-respectation argument?

Using multi-edge argument, NW theorem stands for the weighted analogy. We will clarify the application about weighted case, but most of the proofs are simpler in unweighted case.



Tree-packing approach : Result

Theorem. Let $f(m,n)$ be the time to compute the minimum-weight among the cuts 2-respecting a fixed tree $\mathcal{T}$.
Then, one can obtain Min-Cut in $O(f(m,n)+m+n\log^3 n)$ time with constant probabilty and $O(f(m,n)\log n+m+n\log^3 n)$ with high probability.

- $O(\frac{f(m,n)\log n}{\log\log n}+n\log^6 n)$ speedup is also suggested, but unwanted today.

Recent SOTA algorithms on Min-Cut are optimizations on $f(m,n)$.

- $f(m,n)=O(m\log^2 n)$. (Karger 2000)
- $f(m,n)=O(m\log n+n\log^4 n)$. (Mukhopadhyay, Nanongkai 2020)
- $f(m,n)=O(m\log n)$. (GMW19, state-of-the art)

1. Random Sampling Technique

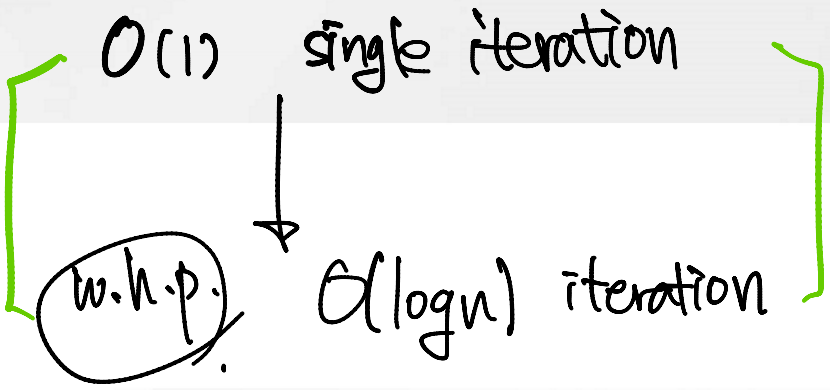
Combinatorics

Clarifications

Here we build some combinatorial arguments crucial to justify the random sampling technique.

Some terms:

- By writing *with high probability (w.h.p)*, we mean the probability to fail decays with $O(n^{-c})$ scale for some $c > 0$.

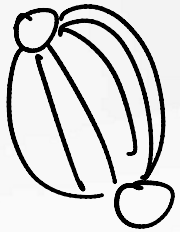


(w.h.p.) $O(\log n)$ iteration

Weighted Tree Packing

Weighted Tree Packings are set of trees $T_1, \cdots, T_k$ each assigned with some weight, such that for all $e \in E(G)$,

$$\sum_{e \in T_i} w(T_i) \leq w(e).$$



Value of the packing is sum of the weights on the tree.

NW theorem says that, there is a weighted tree packing with value $\geq c/2$, where c denotes the Min-Cut.

Lemma. Let C is a cutset with weight αc,
and P be a weighted tree packing with value βc.
Then $\frac{1}{2}(3 - \frac{\alpha}{\beta})$ fraction of trees (by weight) in P *2-constrains* C.

Corollary. Let C be any min-cut and P be any max-weight tree packing. Half of the trees in P 2-constrains C.

- This comes from $\alpha = 1, \beta \geq 1/2$.

Proof comes from simple Markov's inequality. Give a look if interested.

- @Karger2000, Lem2.3 and Cor2.4

So, **IF** we could find a packing P with value βc for some constant $\beta > \frac{1}{3} + \varepsilon$, random selection on P has constant probability to 2-constrain a min-cut.

But still, finding a tree packing with value k takes $O(mn)$ or $O(mk \log n)$ time, which is prohibitively large for our purpose.

Thus we need a modification on G with our sampling.

Counting α-min-cuts

Definition. A cut in G is *α-minimum* if its weight is smaller than α times of the min-cut.

- Note that it must be cut itself.

For a cycle C_n with $n \gg \alpha$, any even-sized edge subset is a cut. Thus the number of α-min cuts are

$$\binom{n}{2} + \binom{n}{4} + \cdots + \binom{n}{\lfloor 2\alpha \rfloor} = \Theta(n^{\lfloor 2\alpha \rfloor})$$

for $\lfloor 2\alpha \rfloor$ even.

Karger (1993) showed that number of α-min cut is $O(n^{2\alpha})$.
In this paper, he improved this to $O(n^{\lfloor 2\alpha \rfloor})$.

Counting α-min-cuts

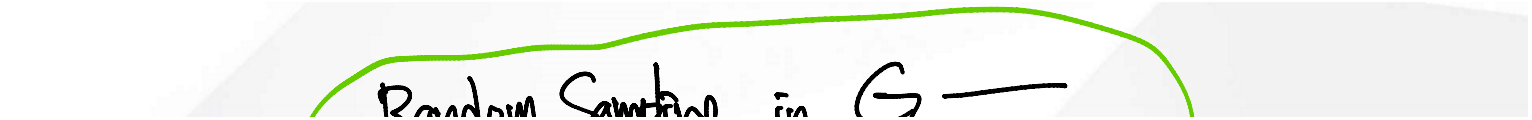
Lemma. There are $O(n^{\lfloor 2\alpha \rfloor})$ α-min-cuts. (Karger 2000, Lem 3.2)

Theorem. The number of α-min-cuts is

$$\frac{1}{\lfloor 2\alpha \rfloor + 1 - 2\alpha} \binom{n}{\lfloor 2\alpha \rfloor} \left(1 + O(\frac{1}{n})\right)$$

Proof of Lemma 3.2 is simple and interesting. Take a look if you want.

These discussions extends to weighted graphs as well.



Sampling

Random Sampling in G —
for cuts, flows, N.D.P.

Karger (1999) developed a decent result about cuts, flows when we randomly sample some edges in a graph.

G : unweighted,

The main result would be served first.

Theorem. Given a parameter $0 < \varepsilon < 1$, there is a well-selected probability p_ε : Selecting edges with probability p_ε to obtain the graph H, any cuts in G has the value in H varies in $[1-\varepsilon, 1+\varepsilon]$, compared to its expected value by ratio *w.h.p.*

Here, 'cuts' are regarded as vertex partitions.

Corollary. In the same preset, any min-cuts in G are $(1+\varepsilon)$-min-cut in H *w.h.p.*

Skeletons

Definition. An **unweighted** graph G with each edge e attached with 'selection probability' p_e, a *skeleton* of G is a random object, where presence of each edge is considered as a Bernoulli event.

For any cut C with value c, expected value of the cut in skeleton is $\hat{c} := \sum_{e \in C} p_e c$.

Theorem. If $\varepsilon = \sqrt{\frac{3(d+2)\ln n}{c}}$ and $\varepsilon < 1$, every cut in the skeleton has value between $[1-\varepsilon, 1+\varepsilon]$ times its expected value with probability $1 - O(n^{-d})$.

Skeletons (Cont'd)

Definition. If the skeleton has $p_e = p$ for all $e \in E(G)$, it's called p-skeleton or $G(p)$.

Proposition. For each cut with value c, $\hat{c} = pc$.

Theorem. Where min-cut has value c. Set $p = 3(d+2)\ln n/\varepsilon^2 c$ for given ε. Then cuts in p-skeleton has value $[1-\varepsilon, 1+\varepsilon]$ times to its expected value with probability $O(n^{-d})$.

Proof. Just note that $\varepsilon = \sqrt{\frac{3(d+2)\ln n}{pc}} \geq \sqrt{\frac{3(d+2)\ln n}{\hat{c}}}$ for any cuts.

How to guess c_0?

→ Matula (1993)

⇒ 3-approx for c_0.

Skeletons (Cont'd)

Skeletons (Cont'd)

Corollaries. In $G(p)$,

- Expected number of edges is $O(n\varepsilon^{-2}\log n)$.
  - There's additional scheme called "sparse certificate" to replace $O(m)$ might be replaced with $O(nc)$. (Nagamochi, Ibaraki 1992)
- Minimum cut of is expected to be $O(\varepsilon^{-2}\log n)$.

Discussions on (integer) weighted graphs is available by sampling from *binomial distribution*.

- Number of distinct sampled edge is still expected to be $O(n\varepsilon^{-2}\log n)$, to note.

Though it's demanding to discuss, the same result holds and construction of skeleton (on weighted graph as well) is still $\tilde{O}(m)$.

Back to original problem

So using the "skeleton graph", we go through the following steps.

- For $\varepsilon = 1/6$, construct a skeleton H with $m' = O(n\varepsilon^{-2} \log n)$ edges and min-cut $c' = O(\varepsilon^{-2} \log n)$.
- Run Gabow's tree packing algorithm, to get a packing with value $\geq c'/2$. This takes $O(m'c' \log n) = O(n \log^3 n)$ time.

For any Min-Cut C of G, C is an $(1+\varepsilon)$-min-cut in H *w.h.p.* Therefore, $\frac{1}{2}(3 - (1+\varepsilon)/(0.5)) = \frac{1}{3}$ of them 2-constrains C.

- Try all $O(\log n)$ trees in the packing, to get minimum cut 2-respecting the tree, using $O(m \log^2 n)$ time for each tree.

$$3 - \frac{(1+1/6)}{0.4} = 3 - \frac{7/6}{2/5}$$

$$= 3 - \frac{35}{12}$$

$$= \frac{1}{12}$$

$= \frac{1}{12}$.

Recap

- The main achievement is reducing the cost for Gabow's tree-packing algorithm to fetch a tree packing with value $\geq 0.5c'$.
- A simple-for-machine algorithm to obtain a tree packing with value $\geq 0.4c'$. Some future papers use this routine as a workaround for Gabow's.
- From now on, we fix a tree T and examine minimum values among the cuts 2-respecting T.

2. Cuts 2-respecting a tree

Entrance

This part is not the state-of-the-art now. It's replaced with Gawrychowski et al, 2019.

But as a purely readable sub-quadratic approach, chill out and enjoy!

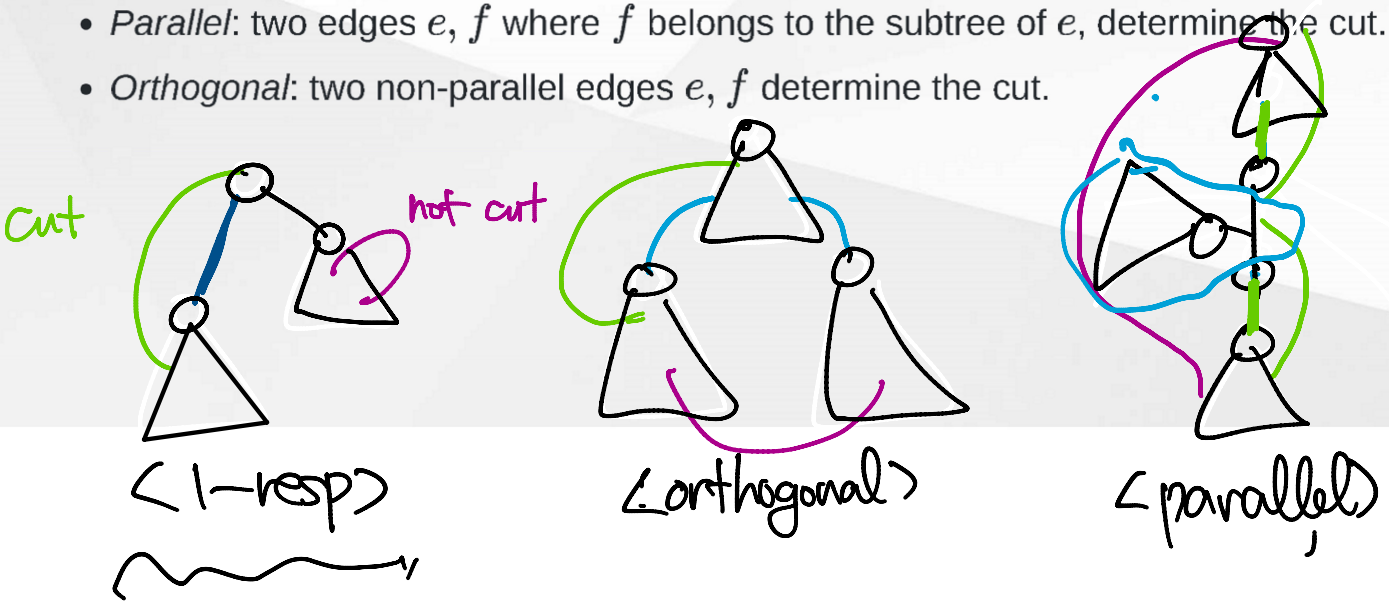
The Problem

So, there will be ≤ 2 edges in the tree T determining the cutset.

There are 3 different cases if we fix the root of T.

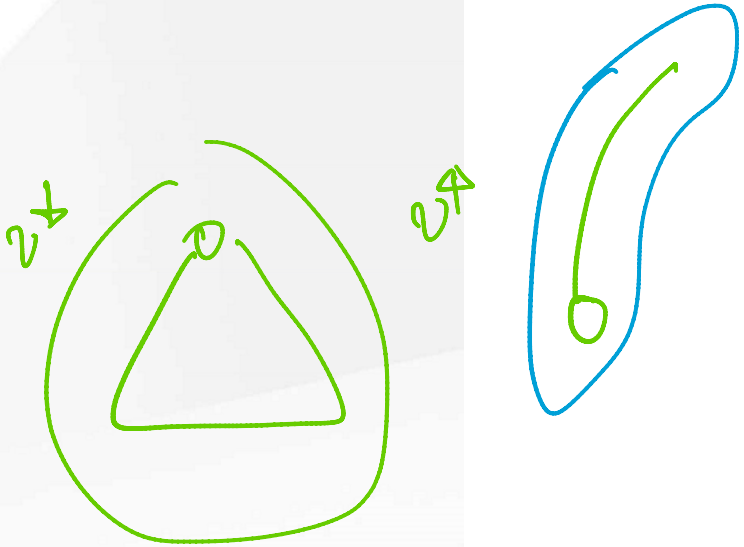
The cases

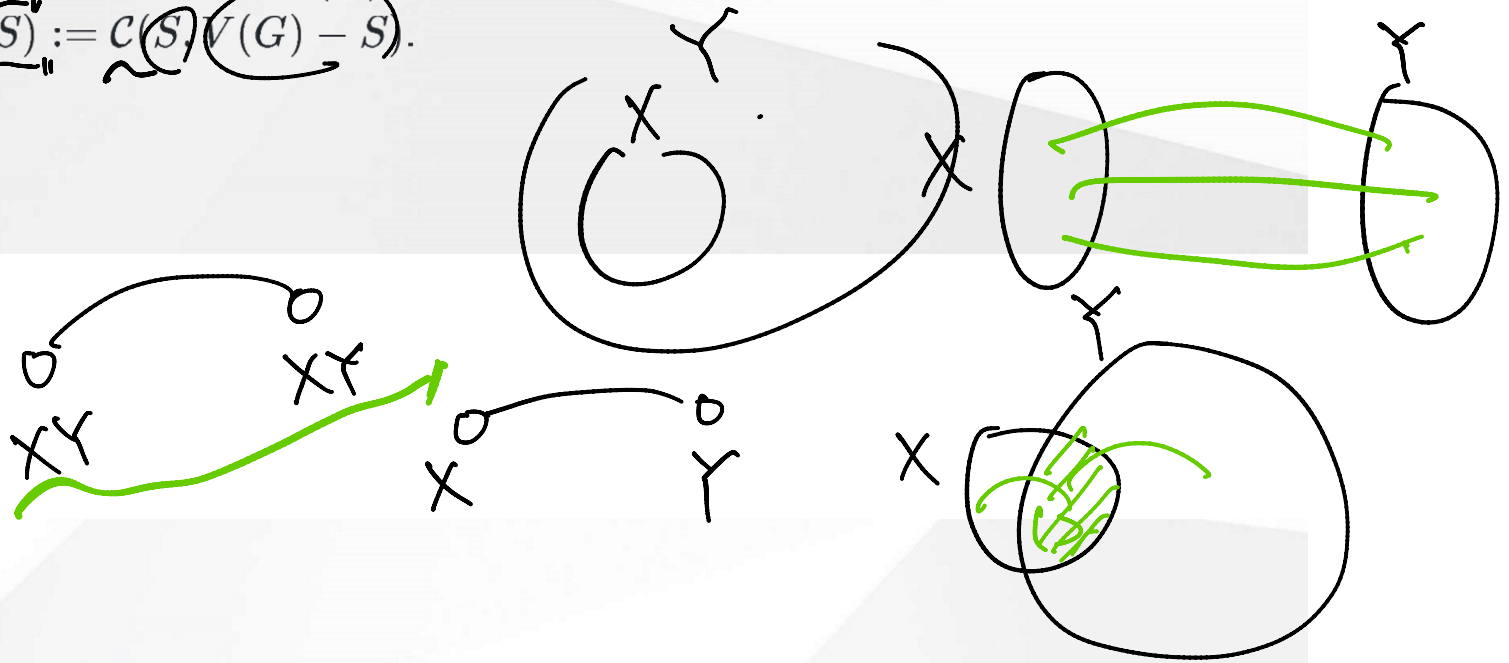
- 1-*respecting*: single edge e determines the cut.
- *Parallel*: two edges e, f where f belongs to the subtree of e, determine the cut.
- *Orthogonal*: two non-parallel edges e, f determine the cut.



Terms

- $v^{\downarrow}$: vertices in subtree of v, including itself
- $v^{\uparrow}$: vertices on the path to root from v, inncluding itself
- $f^{\downarrow}(v)$: If $f : V(G) \to \mathbb{R}$, $f^{\downarrow}(v) := \sum_{u \in v^{\downarrow}} f(u)$.
  - e.g. If $f(v) = 1$, $f^{\downarrow}(v)$ is the size of its subtree.
  - Karger calls this "treefix-sum"
- $\mathcal{C}(X, Y)$: For $X, Y \subset V(G)$, sum of weights of edges crossing them.
  - $\mathcal{C}(S) := \mathcal{C}(S, V(G) - S)$.





Case 1 : 1-respecting

Case 1 : 1-respecting

It suffices to calculate $\mathcal{C}(v^{\downarrow})$ for all v. Simple DP will do.

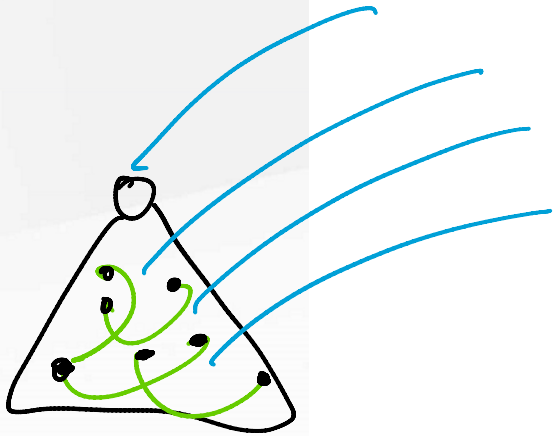
$$\mathcal{C}(v^{\downarrow}) = \delta^{\downarrow}(v) - 2\rho^{\downarrow}(v).$$

- $\delta(v) := \sum_{u \in \mathcal{N}_G(v)} \mathrm{w}(uv)$. "weighted degree"
- $\rho(v) := \sum_{LCA(x,y)=v} \mathrm{w}(xy)$.

Note that weights are counted on G.

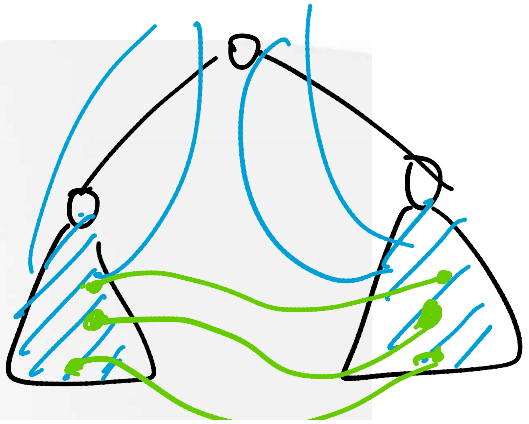
δ, ρ could be calculated in $O(m)$ time easily. We're done with it.

$O(n+m)$.



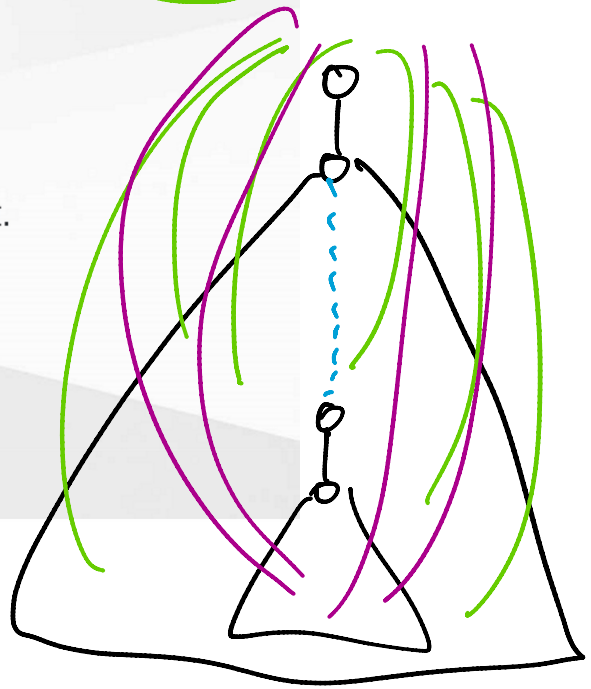
Case 2 : Orthogonal

So, we call v, w are orthogonal iff $v \notin w^{\downarrow}$ and $w \notin v^{\downarrow}$.



So, we call v, w are orthogonal iff $v \notin w^{\downarrow}$ and $w \notin v^{\downarrow}$.

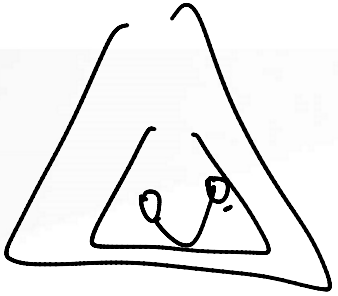
We should compute $\mathcal{C}(v^{\downarrow}) + \mathcal{C}(w^{\downarrow}) - 2\mathcal{C}(v^{\downarrow}, w^{\downarrow})$ for each v, w.



Case 3 : Parallel

If $v \in w^{\downarrow}$, we need to compute $\mathcal{C}(w^{\downarrow} - v^{\downarrow})$ - the only isolated component.

Which turns out to equal with $\mathcal{C}(w^{\downarrow}) - \mathcal{C}(v^{\downarrow}) + 2\mathcal{C}(v^{\downarrow}, w^{\downarrow} - v^{\downarrow})$.



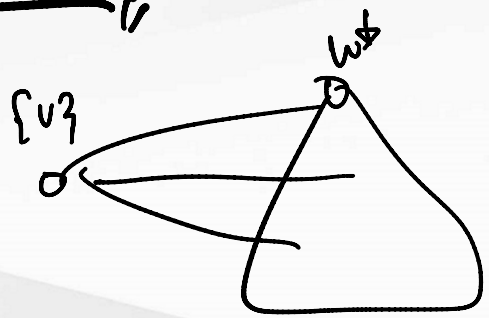
$O(n^2)$ solution for both cases

$\mathcal{C}(v^{\downarrow}, w^{\downarrow} - v^{\downarrow}) = \mathcal{C}(v^{\downarrow}, w^{\downarrow}) - 2p^{\downarrow}(v)$. Hence, computing $\mathcal{C}(v^{\downarrow}, w^{\downarrow})$ for arbitrary v, w is enough.

v, w is enough.

Let $f_v(w) := \mathrm{w}(vw)$. Then $f_v^{\downarrow}(w) = \mathcal{C}(v, w^{\downarrow})$.

Putting $g_w(v) := f_v^{\downarrow}(w)$, $g_w^{\downarrow}(v) = \mathcal{C}(v^{\downarrow}, w^{\downarrow})$. Done!



As a side-product, we may obtain an oracle-structure to answer the following question for $\alpha < 3/2$:

"Given a cut specified the vertex partition, is it α-minimum? if so, what is it's value?"

Just storing a $O(\log n)$ sized tree packing, and check whether the cut 2-respects one of them. Store all α-minimum cuts (as cut edges) 2-respecting one of them. All α-minimum cuts are highly to captured in the lookup table, and there are $O(n^2)$ distinct such cuts.

Karger says this improves $\alpha < 7/6$ past upper bound, $O(n^2)$-space representation of all α-minimum cuts.

$O(m \log^2 n)$ 2-respecting cuts

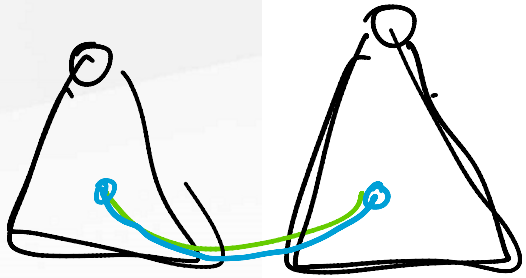
The orthogonal case

We need to find the minimizer for

$$\mathcal{C}(v^{\downarrow}) + \mathcal{C}(w^{\downarrow}) - 2\mathcal{C}(v^{\downarrow}, w^{\downarrow}).$$

Define *v-precut*

$$\mathcal{C}_v(w) := \begin{cases} \mathcal{C}(w^{\downarrow}) - 2\mathcal{C}(v^{\downarrow}, w^{\downarrow}) & v \perp w \\ \infty & \text{otherwise} \end{cases}$$



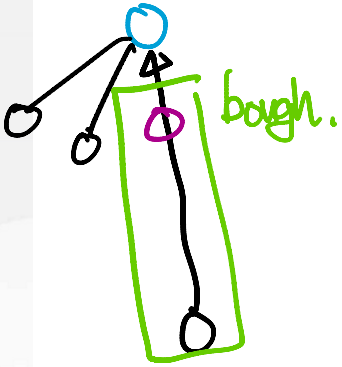
Moreover, define *minimum v-precut* $\mathcal{C}_v$ as

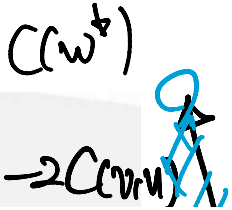
$$\mathcal{C}_v := \min\{\mathcal{C}_v(w) : \exists v', w' \text{ s.t. } v' \in v^{\downarrow}, w' \in w^{\downarrow}\}.$$

Then, $\min_v(\mathcal{C}(v^{\downarrow}) + \mathcal{C}_v)$ works. As $v^{\downarrow}, w^{\downarrow}$ are on the same side of the cut, Hence there is an at least one edge between them to be useful.

Bough decomposition

- Bough (another name for branch) is a path from a leaf to reach the first vertex (or the root) having ≥ 2 children. (the vertex does not counted into the bough)
- We may construct a bough decomposition by contracting a bough into the vertex. Contraction of all boughs halves the number of leaves, indicating that $O(\log n)$ contraction would be enough.
- In this paper, we will calculate $\mathcal{C}_v$ for a leaf and a bough, and examine how to contract.





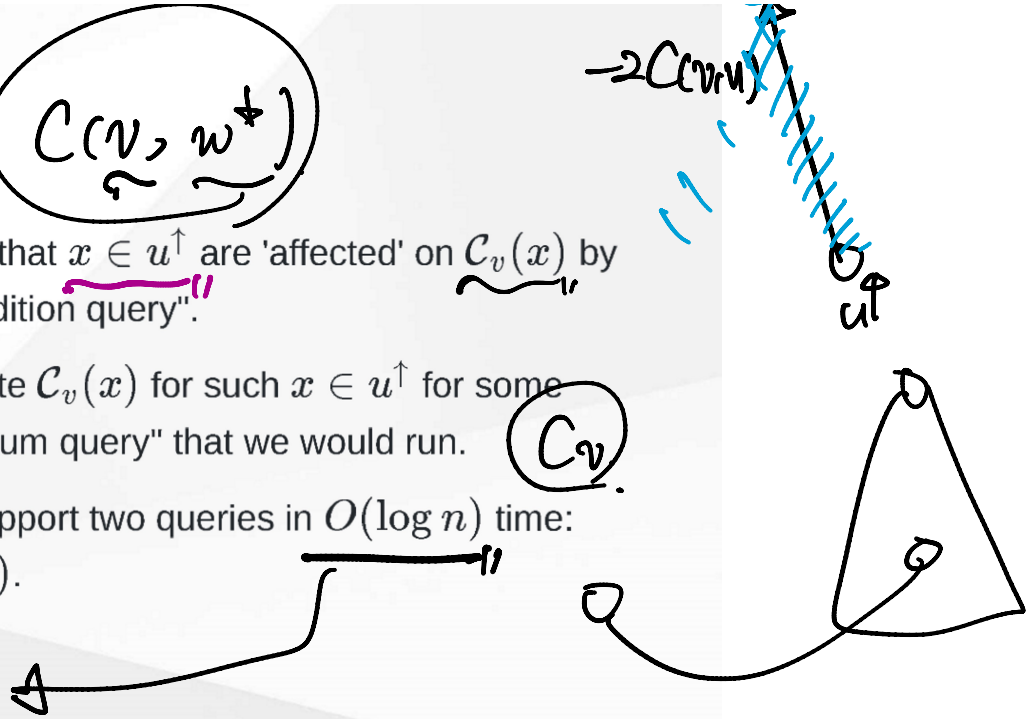
Step 1: A leaf

To get $\mathcal{C}(v, w^{\downarrow})$, for each $u \in \mathcal{N}_G(v)$ we note that $x \in u^{\uparrow}$ are 'affected' on $\mathcal{C}_v(x)$ by $-2\mathcal{C}(v,u)$. Thus we translate this into "path addition query".

Also, for the condition on $\mathcal{C}_v$ it suffices to calculate $\mathcal{C}_v(x)$ for such $x \in u^{\uparrow}$ for some $u \in \mathcal{N}_G(v)$; this limits the range of "path minimum query" that we would run.

Application of a `Link-Cut Tree` is enough to support two queries in $O(\log n)$ time: Maintain array called `val[]` initiated with $C(x^{\downarrow})$.

- `AddPath(v, x)` : `val[i] += x` for $i \in v^{\uparrow}$.
- `MinPath(v)` : get $\min_{i \in v^{\uparrow}} val[i]$.



Step 1: A leaf

Thus, we completely handle the logic for leaves:

Assume initiation with `val[x] =` $\mathcal{C}(x^{\downarrow})$.

```
def LocalUpdate(v):
    AddPath(v, infinity) # to guarantee orthogonality
    for u in neighbor_G[v]:
      AddPath(u, -2*C(v, u))
    ret = infinity
    for u in neighbor_G[v]:
      ret = min(ret, MinPath(u))
    return ret
```

The procedure uses $O(\deg_G(v) \log n)$ time.

Step 2: A Bough

For a vertex v in a bough, v is a leaf or has a unique child u. There's a handy lemma about $\mathcal{C}_v$:

Lemma. $\mathcal{C}_v = \mathcal{C}_u$, or $\mathcal{C}_v = \mathcal{C}_v(w)$ for $w \in x^{\uparrow}$, and $x \in \mathcal{N}_G(v)$.

Thus the same `LocalUpdate` procedure is applicable:

```
def MinPreCut(v):
  if v is a leaf:
    return LocalUpdate(v)
  u = child(v)
  return min(MinPreCut(u), LocalUpdate(v))
```

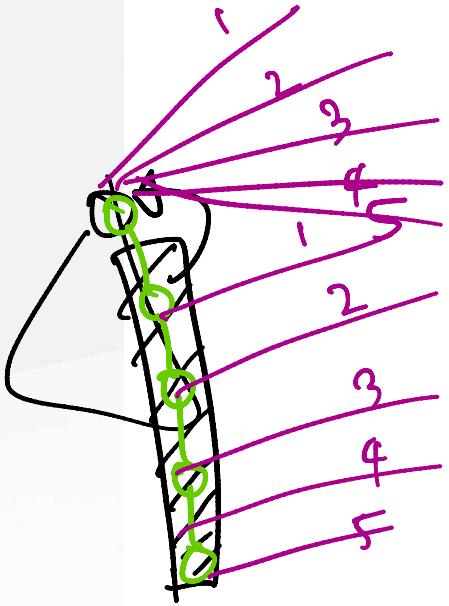
Step 3 : Contraction

To handle multiple boughs, reverting queries on LCT is easily done by changing sign.

If a vertex v has a bough $B \in v^{\downarrow}$, converting all edges xy for $x \in B$ into vy (i.e. contracting B into v) reduces the problem size.

The conversion costs $O(m)$ for each step. As bough decomposition runs $O(\log n)$ loops, worst-case behaviour is $O(m \log n)$.

Number of queries on LCT is also $O(m)$, as each edges are incident to at most 2 different boughs. Thus total cost is $O(m \log^2 n)$.



The parallel case

Expanding the formula:

$$\mathcal{C}(w^{\downarrow} - v^{\downarrow}) = C(w^{\downarrow}) + 2C(w^{\downarrow}, v^{\downarrow}) - (\mathcal{C}(v^{\downarrow}) + 4\rho^{\downarrow}(v))$$

Hence we optimize the quantity

$$\mathcal{C}_v(w) := \min_{w \in v^{\uparrow}} \mathcal{C}(w^{\downarrow}) + 2\mathcal{C}(w^{\downarrow}, v^{\downarrow})$$

Simply initiating `val[x] =` $C(x^{\downarrow})$ and using LCT to update $2\mathcal{C}(v, w^{\downarrow})$ leads to the same $O(m \log^2 n)$ complexity. Done!

$2\mathcal{C}(v, x)$

Summary

- $f(m, n) = O(m \log^2 n)$, from current analysis. (not SOTA)
- In $O(f(m, n) \log n + m + n \log^3 n)$ time, Min-Cut is computable *w.h.p.*
  - $O(\log \log n)$-factor speedup comes from time unbalance for ~~1-respecting cuts and 2-respecting cuts.~~
  - The algorithm involving *skeleton* is the only randomization (Monte-Carlo) step. Deranodmizing this step is crucial. One of current improvements is known to involve another randomizations.
- Acquiring a proper tree-packing on the skeleton takes most of the implementation cost. Lightening this leads to $\sim 11kB$ dedicated code.

$O\left(f(m,n) \cdot \frac{\log n}{\log \log n} + n \log^2 n\right)$

$\downarrow$

$< 10kB$.

References

- Karger, D. R. (2000). Minimum cuts in near-linear time. Journal of the ACM (JACM), 47(1), 46-76.
- Karger, D. R. (1999). Random sampling in cut, flow, and network design problems. Mathematics of Operations Research, 24(2), 383-413.
- Bhardwaj, N., Lovett, A. M., & Sandlund, B. (2019). A simple algorithm for minimum cuts in near-linear time. arXiv preprint arXiv:1908.11829.
- Gawrychowski, P., Mozes, S., & Weimann, O. (2019). Minimum Cut in $O(m \log^2 n)$ Time. arXiv preprint arXiv:1911.01145.
- Mukhopadhyay, S., & Nanongkai, D. (2020, June). Weighted min-cut: sequential, cut-query, and streaming algorithms. In Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (pp. 496-509).