16 Minimum Cuts$^{\beta}$

Let G be an arbitrary graph, and let s and t be two vertices of G. An (s,t)-*cut* (or more formally, an (s,t)-edge-cut) is a subset X of the edges of G that intersects every path from s to t in G. A *minimum* (s,t)-*cut* is an (s,t)-cut of minimum size, or of minimum total weight if the edges of G are weighted. (In this context, edge weights are normally called *capacities*.)

The fastest method to compute minimum (s,t)-cuts in *arbitrary* graphs is to compute a maximum (s,t)-flow and then exploit the classical proof of the maxflow-mincut theorem. In undirected *planar* graphs, however, this dependency is reversed; the fastest method to compute maximum flows actually computes minimum cuts first.

16.1 Duality: Shortest essential cycle

Let Σ be an undirected planar *map*, each of whose edges e has a non-negative capacity $c(e)$, and let s and t be vertices of Σ. The first step in our fast minimum-cut algorithm is to view the problem through the lens of duality. It is helpful to think of the source vertex s and the target vertex t as *punctures* or *obstacles* — points that are missing from the plane — and similarly to think of the corresponding faces s^* and t^* as *holes* in the dual map Σ^*. In other words, we should think of the dual map Σ^* as a decomposition of the *annulus* $A = \mathbb{R}^2 \setminus (s^* \cup t^*)$ rather than a map on the plane or a disk. Without loss of generality, assume that t^* is the outer face of Σ^*.

A simple cycle γ in Σ^* is *essential* if it satisfies any of the following equivalent conditions:

- γ separates s^* from t^*.
- γ has winding number ± 1 around s^*.
- γ is homotopic in A to the boundary of s^*.
- γ is homotopic in A to the boundary of t^*.

Each edge e^* in the dual map Σ^* has a *cost* or *length* $c^*(e^*)$ equal to the capacity of the corresponding primal edge e. Whitney's duality between simple cycles in Σ and and minimal cuts (bonds) in Σ^* immediately implies the following:

Lemma: *A subset X of edges is a minimum (s,t)-cut in Σ if and only if the corresponding set X^* of dual edges is a minimum-cost essential cycle in Σ^*.*

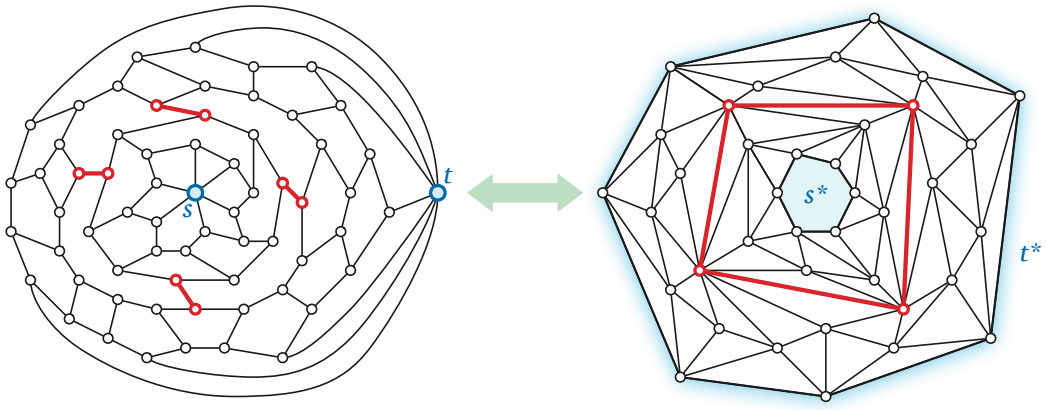

Figure 1: A minimum (s,t)-cut in a planar graph is dual to a shortest essential cycle in the annular dual map.

16.2 Crossing at most once

Now let π be a shortest path in Σ^* from any vertex of s^* to any vertex of t^*. We can measure the winding number of any directed cycle γ in Σ^* by counting the number of times γ crosses π in either direction. We have to define "crossing" carefully here, because γ and π could share edges.

Suppose $\pi = p_0 \to p_1 \to \cdots \to p_k$, where p_0 lies on s^* and p_k lies on t^*. To simplify the following definition, we add two "ghost" darts $p_{-1} \to p_0$ and $p_k \to p_{k+1}$, where p_{-1} lies inside s^* and p_{k+1} lies inside t^*. We say that a dart $q \to p_i$ *enters* π *from the left* (resp. *from the right*) if the darts $p_{i-1} \to p_i$, $q \to p_i$, and $p_{i+1} \to p_i$ are ordered clockwise (resp. counterclockwise) around p_i. Symmetrically, we say that a dart *leaves* π *to the left* (resp. *to the right*) if its reversal enters π from the left (resp. from the right). The same dart can leave π to the right and enter π to the left, or vice versa.

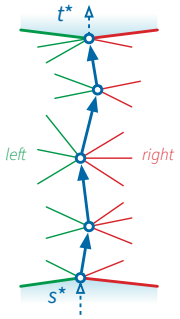

Figure 2: Edges incident to both sides of π.

A *positive crossing* between π and γ is a subpath of γ that starts with a dart entering π from the right and ends with a dart leaving π to the left, and a *negative crossing* is defined similarly. Intuitively, for purposes of defining crossings, we are shifting the path π very slightly to the left, so that it intersects the edges of Σ^* transversely. It follows that the winding number $\mathsf{wind}(\gamma, s^*)$ is the number of darts in γ that leave π to the left, minus the number of darts in γ that enter π from the left.

Crossing Lemma [Itai and Shiloach (1979)]: *The shortest essential cycle in Σ^* crosses π exactly once.*

Proof: We follow the same intuition that we used for shortest homotopic paths in the plane. Let γ be any essential cycle in Σ^*, directed so that $\mathsf{wind}(\gamma, s^*) = 1$, that crosses π more than once. Then somewhere γ must have be a negative crossing followed immediately by a positive crossing. It follows that γ has a subpath $p_i \to q \to \cdots \to r \to p_j$, where $p_i \to q$ leaves π to the right, $r \to p_j$ enters π from the right. Let γ' be the cycle obtained from γ by replacing this subpath with the subpath of π from p_i to p_j. Because π is a shortest path, γ' must be shorter than γ. We conclude that γ' is not the *shortest* essential cycle in Σ^*. □

16.3 Slicing along a path

Now let $\Delta := \Sigma^* \backslash\backslash \pi$ denote the planar map obtained by *slicing* the annular map Σ^* along path π. The slicing operation replaces π with two copies π^+ and π^-. Then for every vertex p_i of π, all edges incident to p_i on the left are redirected to p_i^+, and all edges incident to p_i on the left are redirected to p_i^-. The channel between two two paths π^+ and π^- joins s^* and t^* into a single outer face. Thus, we should think of Δ as being embedded on a disk. Every face of Σ^* except s^* and t^* appears as a face in Δ.

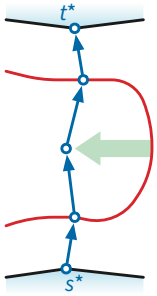

Figure 3: Any essential cycle that crosses π more than once can be shortened.

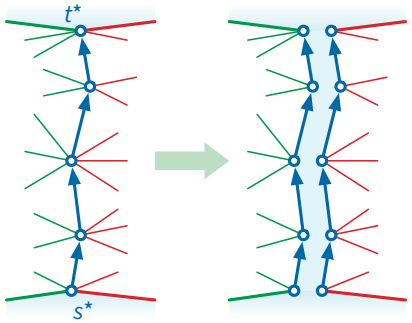

Figure 4: Slicing along π.

For any index i, let σ_i denote the shortest path in Δ from p_i^+ to p_i^-. The shortest essential cycle γ in Σ^* appears in Δ as one of the shortest paths σ_i. Thus, to compute the minimum (s,t)-cut in our original planar map Σ, it suffices to compute the length of *every* shortest path σ_i in Δ.

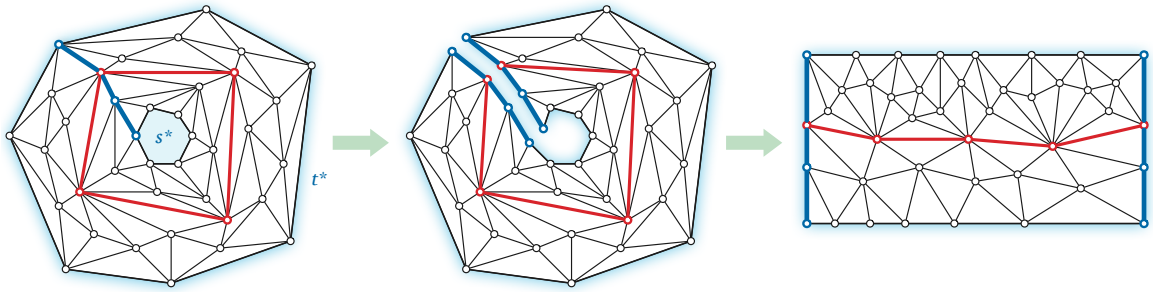

Figure 5: Slicing along π transforms the shortest essential cycle into a shortest path between clones of some vertex of π.

16.4 Algorithms

The simplest way to compute these k shortest-path distances is to run Dijkstra's algorithm at each vertex p_i^+. Assuming π has k edges, so there are $k+1$ terminal pairs $p_i^\pm$, this algorithm runs in $O(kn \log n)$ time, which is $O(n^2 \log n)$ time in the worst case. We can reduce the running time to $O(kn \log\log n)$ using the faster shortest-path algorithm we described in the previous lecture note, or even to $O(kn)$ using a linear-time shortest-path algorithm.

Alternatively, we can compute all k of these shortest paths using a multiple-source shortest-paths algorithm. The parametric MSSP algorithms of Klein and Cabello et al. both require $O(n \log n)$

time, which is faster in the worst case than $O(kn)$, but slower when k is small. In particular, even when $k = 2$, that algorithm could require $\Omega(n)$ pivots. The more recent recursive contraction algorithm runs in $O(n \log n \log k)$ time if we use Dijkstra's algorithm to compute shortest paths, or in $O(n \log k)$ time if we use a linear-time shortest-path algorithm.

An older and simpler divide-and-conquer algorithm, proposed by John Reif in 1983, exactly matches the recursive contraction MSSP algorithm. Reif's algorithm computes the median shortest path σ_m, where $m = \lfloor k/2 \rfloor$, and then recurses in each component of the sliced map $\Delta \setminus\setminus \sigma_m$. One of these components contains the terminal pairs $p_0^{\pm}, p_1^{\pm}, \ldots, p_m^{\pm}$; the other contains the terminal pairs $p_m^{\pm}, p_{m+1}^{\pm}, \ldots, p_k^{\pm}$.

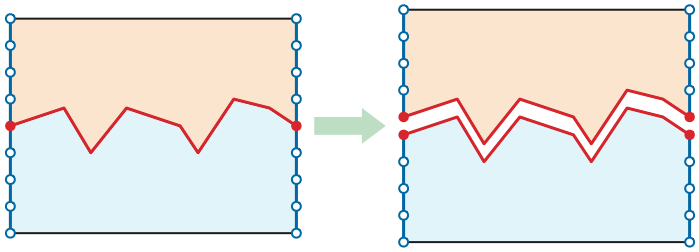

Figure 6: Reif's algorithm: Slice along the median shortest path and recurse.

Reif's algorithm falls back on Dijkstra's algorithm in two base cases. First, if $k \leq 2$, we can obviously compute each of the k shortest paths directly. A more subtle base case happens when the "floor" and "ceiling" paths collide. Let α denote the boundary path from p_0^+ to p_0^-, and let β denote the boundary path from p_k^+ to p_k^-. If α and β share a vertex x, then for every index i we have $dist(p_i^+, p_i^-) = dist(p_i^+, x) + dist(x, p_i^-)$; thus, instead of recursing, we can compute all k shortest-path distances by computing a single shortest-path tree rooted at x. (This second base case is not necessary for the correctness of Reif's algorithm, but it is necessary for efficiency.)

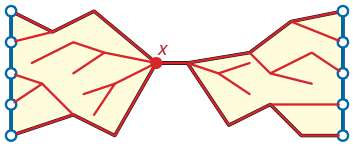

Figure 7: Base case of Reif's algorithm.

Let $T(n, k)$ denote the running time of Reif's algorithm, where $k+1$ is the number of terminal pairs and n is the total number of vertices in the map Δ. This function obeys the recurrence

$$T(n, k) = T(n_1, \lfloor k/2 \rfloor) + T(n_1, \lceil k/2 \rceil) + O(n \log n).$$

where n_1 and n_2 are the number of vertices in the two components of $\Delta \setminus\setminus \sigma_m$. The second base case ensures that each vertex and edge of Δ appears in at most $O(1)$ subproblems at any level of the recursion tree.[1] Thus, the total work at any level of recursion is $O(n \log n)$. The recursion tree has depth at most $O(\log k)$, so the overall algorithm runs in $O(n \log n \log k)$ time.

If we use a linear-time shortest-path algorithm instead of Dijkstra, the running time improves to $O(n \log k)$. This improvement was first described by Greg Frederickson in 1987, as one of the earliest applications of r-divisions.

[1] Without the second base case, it is possible for a constant fraction of the vertices to appear in a constant fraction of the recursive subproblems, leading to a running time of $O(kn \log n)$.

16.5 Faster Shortest Paths with Negative Lengths

[[[This is still really sketchy!]]]

In the previous lecture note, we described a 2009 algorithm by Klein, Mozes, and Weimann to compute shortest paths in directed planar maps, possibly with negative dart lengths, in $O(n \log^2 n)$ time. In 2010, Shay Mozes and Christian Wulff-Nilsen improved this algorithm even further by using a good r-division at each level of recursion (with $r \approx n/\log n$) instead of just one separator cycle; their improved algorithm runs in $O(n \log^2 n/\log\log n)$.

Recall the high-level description of the Klein-Mozes-Weimann algorithm:

0. Split the map Σ into two smaller maps A and B along a single cycle separator S.
1. Recursively compute $\mathsf{dist}_A(r, A)$ and $\mathsf{dist}_B(r, B)$
2. Compute $\mathsf{dist}_A(S, S)$ and $\mathsf{dist}_B(S, S)$ using MSSP in $O(n \log n)$ time.
3. Compute $\mathsf{dist}_\Sigma(r, S)$ using FR-Bellman-Ford in $O(n\alpha(n))$ time.
4. Compute $\mathsf{dist}_\Sigma(r, \Sigma)$ using reweighting and r-divisions in $O(n \log\log n)$ time.
5. Compute $\mathsf{dist}_\Sigma(s, \Sigma)$ using reweighting and r-divisions in $O(n \log\log n)$ time.

Steps 1, 4, and 5 of Mozes and Wulff-Nilsen's faster algorithm are identical. In step 2, the only minor different is that we need to run MSSP around the boundary of each hole of each piece, instead of just once around the sole cycle separator. The total time for each piece is $O(r \log r)$, so the total time for this step across the entire r-division is $O(n/r) \cdot O(r \log r) = O(n \log r)$.

The only major change to the algorithm is in step 3; we need to modify FR-Bellman-Ford to deal with holes in each piece of the r-division.

In step 2, we compute all boundary-to-boundary distances within each piece of the r-division using MSSP. Specifically, within each piece P, we run MSSP around each of the $O(1)$ boundary cycles of P. The total time for each piece is $O(r \log r)$, so the total time for this step across the entire r-division is $O(n/r) \cdot O(r \log r) = O(n \log r)$.

Modifying step 3 is more interesting. Recall that a good r-division $\mathcal{R}$ breaks Σ into $O(n/r)$ pieces, each with $O(r)$ vertices, $O(\sqrt{r})$ boundary vertices, and $O(1)$ holes; thus, overall, $\mathcal{R}$ has $O(n/\sqrt{r})$ boundary vertices, organized into $O(n/r)$ cycles. We construct a complete directed multigraph $\hat{R}$ over the boundary vertices of the r-division, with an edge $u \to v$ with weight $\mathsf{dist}_P(u, v)$ for every piece P containing both u and v.

Bellman-Ford computes shortest-path distances in $\hat{R}$ in $O(n/\sqrt{r})$ phases; in each phase, we find and relax all tense edges. Our modification of FR-Bellman-Ford breaks each relaxation phase into subphases:

```
for each piece P:
   for each boundary cycle S of P:
       for each boundary cycle T of P:
           relax every edge within P from S to T
```

We already described how to relax every edge in P from a boundary cycle *to itself* in $O(k\alpha(k))$ time (or $O(k)$ expected time). It remains to show how to relax all edges from one boundary cycle S to a different boundary cycle T.

Without loss of generality, let's assume that S and T are the only two boundary cycles in P; we'll treat any other boundary cycles as faces of P. Thus, P is a map of an *annulus*. Let $s_1, s_2, \ldots, s_k$

and $t_1, t_2, \ldots, t_l$ denote the sequences of vertices of S and T, respectively.

Let π be the shortest path in P from s_1 to t_1, and let $\Delta = P \setminus\setminus \pi$ be the map obtained by slicing Σ along the path π. As before, the slicing operation duplicates the path π and merges the faces S and T into a single outer face.

Let Δ' be a duplicate of Δ, and let Δ^2 denote the map obtained by gluing the "right" copy of π in Δ to the "left" copy of π in Δ'.

Lemma: *For all indices i and j, we have*

$$dist_P(s_i, t_j) = \min\left\{dist_{\Delta^2}(s_i, t_j),\ dist_{\Delta^2}(s_i', t_j),\ dist_{\Delta^2}(a_i, b_j')\right\}$$

.

[[[This is already claimed, but without proof, in the next lecture note.]]]

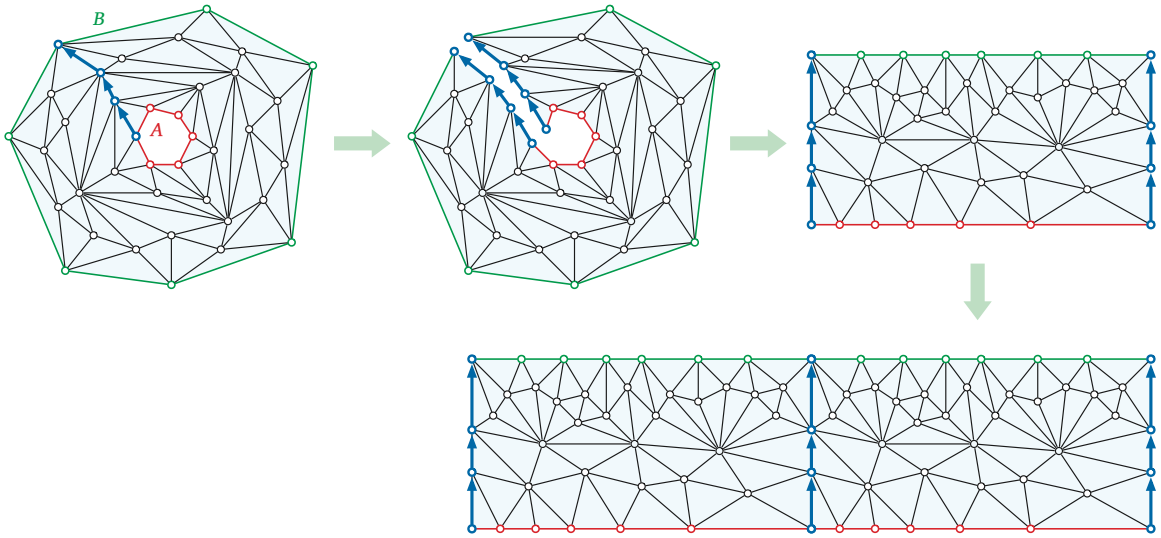

Figure 8: Reducing the boundary-to-boundary distance array of an annulus to a Monge array; compare with Figure 5.

The matrix of distances from the "top edge" of Δ^2 to the "bottom edge" of Δ^2 is Monge! So we can relax all the edges from A to B in $O(k+l)$ time with one invocation of SMAWK.

Suppose P has $h = O(1)$ holes, with $k_1, k_2, \ldots, k_h$ boundary vertices; recall that $k = \sum_i k_i = O(\sqrt{r})$. Then the total time to relax all boundary-to-boundary edges in P is

$$\sum_{i=1}^{h} O(k_i \alpha(k_i)) + \sum_{i=1}^{h}\sum_{j=1}^{h} O(k_i + k_j) \;=\; O(k\alpha(k) + kh) \;=\; O(\sqrt{r}\,\alpha(r)).$$

It follows that the time to relax *all* edges of $\mathcal{R}$ is $O((n/\sqrt{r})\alpha(r)) = o(n)$

16.6 Faster Minimum Cuts: FR-Dijkstra

Frederickson held the record for fastest planar minimum-cut algorithm for almost two and a half decades; the record was finally broken in 2011 by two independent pairs of researchers, who ultimately published their result jointly: Giuseppe Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Their $O(n \log\log n)$-time algorithm relies on an improvement to

Dijkstra's algorithm in dense distance graphs, proposed by Jittat Fakcharoenphol and Satish Rao in 2001, and now usually called *FR-Dijkstra*.

Recall from the previous lecture on shortest paths that we can compute a dense distance graph for a nice r-division in $O(n \log r)$ time. The dense distance graph has $O(n/\sqrt{r})$ vertices—the boundary vertices of the pieces of the r-division—and $O(n)$ edges. So Dijkstra's algorithm with a Fibonacci heap runs in $O(E + V \log V) = O(n + (n/\sqrt{r}) \log n)$ time.

FR-Dijkstra removes the $O(n)$ term from this running time. Specifically, within each piece of the r-division, the algorithm exploits the Monge structure in the boundary-to-boundary distances to avoid looking at every pair of boundary vertices. This is the same high-level strategy that we previously used with FR-Bellman-Ford, but with one significant difference: We do not know the relevant Monge arrays in advance. Instead, portions of each Monge array are revealed each time the Dijkstra wavefront touches the corresponding piece of the r-division.

I'll discuss FR-Dijkstra in detail, along with the faster planar minimum-cut algorithm, in the next lecture.

16.7 References

1. Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs with applications. *SIAM J. Comput.* 16(8):1004–1004, 1987.

2. Alon Itai and Yossi Shiloach. Maximum flow in planar networks. *SIAM J. Comput.* 8:135–150, 1979.

3. Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. *Proc. 43rd Ann. ACM Symp. Theory Comput.*, 313–322, 2011.

4. Shay Mozes and Christian Wulff-Nilsen. Shortest paths in planar graphs with real lengths in $O(n \log^2 n/\log\log n)$ time. *Proc. 18th Ann. Europ. Symp. Algorithms*, 206–217, 2010. Lecture Notes Comput. Sci. 6347, Springer-Verlag. arXiv:0911.4963.

5. John Reif. Minimum s-t cut of a planar undirected network in $O(n \log^2 n)$ time. *SIAM J. Comput.* 12(1):71–81, 1983.

6. Hassler Whitney. Non-separable and planar graphs. *Trans. Amer. Math. Soc.* 34:339–362, 1932.

7. Hassler Whitney. Planar graphs. *Fund. Math.* 21:73–84, 1933.

16.8 Aptly Named Sir Not

- Maximum cuts (or minimum cuts with negative capacities) [Hadlock 1975]
- Deriving maximum flows from minimum cuts [Hassin Johnson 1985]
- $O(n)$ time for unweighted graphs [Weihe 1997, Eisenstat Klein 2013, Balzotti Franciosa 2022]
- Global minimum cuts (dual to shortest weighted cycle) [Łącki Sankowski 2011]
- Minimum cuts in directed planar graphs, via double cover [Janiga Koubek 1992 but fixed, Erickson 2011, Fox 2013]
- Faster directed planar minimum cuts [Mozes Nikolaev Nussbaum Weimann SODA 2018]