Main topics are #maximum_flow in #planar_graphs.

Maximum Flow

- I checked the calendar, and it looks like there’s time for one more topic before we move on to surface embedded graphs: we’re going to consider the maximum flow problem in planar graphs.
- First, some definitions. These may seem a bit different from what you’re used to, but it makes the math work out so beautifully in the end. I’m going to show you an “antisymmetric flow” formulation.
- Take an abstract graph $G = (V, D)$.
- A flow is a function $\phi : D \to \mathbb{R}$ from darts to the reals. FLOWS CAN BE NEGATIVE. In fact, they will be for about half the darts, because we need $\phi(d) = -\phi(\text{rev}(d))$ for any flow.
- Let $s$ and $t$ be two vertices. Function $\phi$ is an $s,t$-flow if $\sum_u \phi(u \to v) = 0$ for all $v \neq s, t$. These equations are often called the conservation constraints.
- Let $c : D \to \mathbb{R}^+$ be the capacities of the darts. The $(s,t)$-flow is feasible if $\phi(d) \leq c(d)$ for all darts $d$. These inequalities are often called the capacity constraints.
- The value of an $s,t$-flow is $\sum_u \phi(u \to t)$. Our goal is to quickly find an $s,t$-flow of maximum value given a graph with non-negative capacities.
- Given an $s,t$-cut $(S, T)$, its capacity is $\sum_{u \to v : u \in S, v \in T} c(u \to v)$. Ford and Fulkerson proved the value of any $s,t$-flow is at most the capacity of any $s,t$-cut, and this is actually an equality for the maximum value flow and minimum capacity cut.
- Orlin’s ['13] algorithm can find both in $O(n^2 / \log n)$ time in an arbitrary planar graph, but (surprise!) we’re going to do better today and Monday.

$s,-$planar Graphs

- Let’s start with an easier case, where $s$ and $t$ lie on the outer face $o$. The following algorithm is by Hassin ['81].
- The minimum $s,t$-cut is a (directed) bond with two darts on the outer face, so its dual is a directed cycle through $o^\ast$. In this figure, it goes clockwise.
- $s^*$ and $t^*$ are both incident to $o^*$. Imagine slicing through $o^*$ to create two dual vertices $s'$ and $t'$. That shortest cycle through $o^*$ is now a shortest path from $s'$ to $t'$. So run Dijkstra's algorithm once to find it.
- We now know the minimum cut and its capacity, but can we find the maximum flow function itself?
- For each dual vertex $f^*$, let $\text{dist}(f^*)$ denote the distance from $s'$.
- For each dart $u \leftrightarrow v$, define $\phi(u \leftrightarrow v)$
  - $:= \text{dist}([u \leftrightarrow v]) - \text{dist}([v \leftrightarrow u])$
  - $= \text{dist}(y) - \text{dist}(x) \quad [\text{where } (u \leftrightarrow v)^* = x \leftrightarrow y]$
  - $= c(x \leftrightarrow y) - \text{slack}(x \leftrightarrow y) \quad [\text{by definition of slack}]$
  - $\leq c(x \leftrightarrow y)$
- We satisfied capacity constraints.
- If you go around any face of $G^*$ other than $s^*$ and $t^*$ summing flows, the $+ \text{dist}(y)s$ and $-\text{dist}(u)s$ cancel, leading to 0 net flow. In $G$, this means we satisfied conservation constraints.
- By running Henzinger et al. shortest paths in the dual graph, we can compute $\phi$ in only $O(n)$ time.

**Maximum Flows and Dual Shortest Paths**

- So what if $s$ and $t$ are on different faces? We'll assume $t$ lies on the outer face $o^*$.
- Venkatesan ['87] had an interesting observation. Fix a value $\lambda > 0$. Can we find a feasible $s,t$-flow of value $\lambda$ if one exists?
- Let $P$ be an arbitrary path from $s$ to $t$. Let $p(d) =$
  - $1$ if $d \in P$
  - $-1$ if $d \in \text{rev}(P)$
  - $0$ otherwise
- Imagine sending lambda units of flow from $s$ to $t$ along $P$, ignoring the capacities. Now
define the residual capacity of a dart as $c_\lambda(u \to v) = c(u \to v) - \lambda \pi(u \to v)$.

- Intuitively, the residual capacities are telling you how much more (or less!) flow can go through each dart to have a feasible flow. RESIDUAL CAPACITIES CAN BE NEGATIVE! It just means you need to reduce the flow on the dart to make the flow feasible.

- Now imagine the residual capacities as edge lengths in the dual $G^*$. We refer to the combination of $G^*$ and $c_\lambda$ as the dual residual network $G^* \lambda$. Let $\text{dist}_\lambda(f^*)$ be the distance from $o$ to $f^*$ in $G^*$ with regard to $c_\lambda$.

(The figure on the left should have $t$ and $s$ swapped.)

- These distances are well-defined if and only if there are no negative cycles wrt $c_\lambda$.

Suppose there is a negative cycle $C$ in $G^*$.

- $c_\lambda(C) = \sum_{d \in C} (c(d) - \lambda \pi(d)) < 0$
- But $\sum_{d \in C} c(d) \geq 0$ and $\lambda \geq 0$, implying $\pi(C) \geq 0$.
- $C$ goes around $s^*$ exactly once, so $\pi(C) = 1$.

(These orientations seem to be backwards.)

- Meaning $C$ goes clockwise around $s^*$. It’s dual to an $s,t$-cut!
- Also, $\sum_{d \in C^*} < \lambda$, implying $\lambda > \text{mincut} = \text{maxflow}$.

But what if $\text{dist}_\lambda$ is well-defined?

- Define $\text{slack}_\lambda(p \to q) := \text{dist}_\lambda(p) + c_\lambda(p \to q) - \text{dist}_\lambda(q) \geq 0$
- Define $\phi_\lambda(p \to q)$
  - $:= \text{dist}_\lambda(q) - \text{dist}_\lambda(p) + \lambda \pi(p \to q)$
  - $= c(p \to q) - \text{slack}_\lambda(p \to q)$
- So $\phi_\lambda(p \to q) \leq c(p \to q)$.
- And similar to before, $\sum_u \phi(u \to v) = \sum_u \lambda \pi(u \to v)$
\[ \begin{align*}
\lambda & \text{ if } v = t \\
-\lambda & \text{ if } v = s \\
0 & \text{ otherwise }
\end{align*} \]

So \( \phi_\lambda \) is a feasible \( s,t \)-flow of value \( \lambda \).

**Parametric Shortest Paths**

- But how do we find the maximum good value for \( \lambda \)?
- We’ll use a strategy similar to Monday’s. We’ll start with \( \lambda = 0 \) and compute the dual shortest path tree. Then we’ll continuously increase \( \lambda \) as we compute so-called **parametric shortest paths**.
- \( \text{dist}_\lambda \) and \( \phi_\lambda \) will vary continuously as we increase \( \lambda \), but shortest path tree \( T_\lambda \) will change at discrete **pivots** just like before.
- And just like before, we’ll wait until just before some \( \text{slack}_\lambda(p \leftrightarrow q) < 0 \), pivot \( p \leftrightarrow q \) into \( T_\lambda \), and pivot \( x \leftrightarrow q \) out.
- \( \text{slack'}_\lambda(p \leftrightarrow q) := \text{derivative of slack for } p \leftrightarrow q \) and \( \text{path}_\lambda(p) := \text{the shortest path in } T_\lambda \text{ to } p \)
- \( \text{slack'}_\lambda(p \leftrightarrow q) \)
  \[ = \text{dist'}_\lambda(p) + \pi(p \leftrightarrow q) - \text{dist'}_\lambda(q) \]
  \[ = -\pi(\text{path}_\lambda(p)) - \pi(p \leftrightarrow q) - \pi(\text{rev}(\text{path}_\lambda(q))) \]
  \[ = -\pi(\text{cycle}(T_\lambda, p \leftrightarrow q)) \]
  \[ \in \{-1, 0, 1\} \]
- Also, we see: \( \text{slack'}_\lambda(p \leftrightarrow q) = -\text{slack'}_\lambda(\text{rev}(p \leftrightarrow q)) \)
- We’ll call \( p \leftrightarrow q \) active if \( \text{slack'}(p \leftrightarrow q) = -1 \).
- But how do we find the active darts?
- Let \( L_\lambda = (G \setminus T_\lambda)^* \), the complementary spanning tree of \( G \).
- Assuming shortest paths are unique, \( L_\lambda \) is the set of **loose** edges, those where both darts have slack > 0.
- \( \text{LP}_\lambda := \text{unique path from } s \text{ to } t \text{ in } L_\lambda \)
- Lemma: \( d^* \) is active if and only if \( d \in \text{LP}_\lambda \)
  
  - Darts of \( T_\lambda \) have slack = 0, so \( \text{slack'} = 0 \) for them and their reversals.
  - \( d^* \) is active if and only if \( \pi(\text{cycle}(T_\lambda, d^*)) = 1 \) if and only if \( C(d) = (\text{cycle}(T_\lambda, d^*))^* \) is an \( s,t \)-cut
  
  - If \( d^* \) is active, then \( \text{LP}_\lambda \) contains at least one edge of \( C(d) \). But \( d \) is on the only loose edge of \( C(d) \), so \( d \) in \( \text{LP}_\lambda \)
  
  - If \( d \) in \( \text{LP}_\lambda \), then \( C(d) \) is an \( s,t \)-cut.
- So now the algorithm behaves similar to MSSP.
- As we increase \( \lambda \), \( \text{slack}_\lambda(d) \) decreases for all \( d \) in \( \text{LP}_\lambda \) and increases for all \( d \) in \( \text{rev}(\text{LP}_\lambda) \).
Equivalently, \( \phi_{\lambda}(d) \) increases along \( \text{LP}_{\lambda} \) and decreases along \( \text{rev}(\text{LP}_{\lambda}) \).

It’s like we’re pushing flow from \( s \) to \( t \) along \( \text{LP}_{\lambda} \).

Pivot \( d \) into \( T_{\lambda} \) when \( \text{slack}_{\lambda}(d) = 0 \). It’s like we saturated the dart.

Pivot \( \text{pred}(q) \Rightarrow q \) out at the same time. It’s like we made a new augmenting path!

Eventually, we do a pivot that creates a directed cycle of slack 0 darts.

But that means each of those darts is saturated. We found the minimum \( s,t \)-cut and the maximum \( s,t \)-flow!

**Pseudocode:**

- Create initial dual shortest path tree \( T_0 \)
- Maintain primal spanning tree \( L \)
- While \( L \) is connected
  - \( \text{LP} \): path from \( s \) to \( t \) in \( L \)
  - \( (p \Rightarrow q) \): min slack edge in \( \text{LP}^* \)
  - decrease slacks along \( \text{LP} \) by \( \text{slack}(p \Rightarrow q) \). Increase slacks along \( \text{rev}(\text{LP}) \) by the same amount.
  - delete \( (p \Rightarrow q)^* \) from \( L \)
  - insert \( (\text{pred}(q) \Rightarrow q)^* \) into \( L \)
  - \( \text{pred}(q) \Rightarrow p \)
  - \( \phi \leftarrow c - \text{slack} \)

**Analysis**

- Using dynamic forests, each step of the algorithm can be implemented in \( O(\log n) \) time. Our running time therefore depends on the number of pivots.
- Let \( \text{path}_i \) denote the shortest walk from \( o \) to \( q \) s.t. \( \text{pi} \left( \text{path}_i(q) \right) = i \).
- We see \( \text{dist}_{\lambda}(q) = \min_i (c_{\lambda}(\text{path}_i(q))) \).
- We can also observe that whenever \( \text{path}_{\lambda}(q) \) changes, \( \text{pi}(\text{path}_{\lambda}(q)) \) increases by 1, because \( \text{pi}(\text{cycle}(T, p \Rightarrow q)) = 1 \) if \( p \Rightarrow q \) is pivoting in.
- So now we want to know, for which \( i \) is \( p \Rightarrow q \) in \( \text{path}_i(q) \)?
- Imagine removing faces \( s^* \) and \( t^* \) from the plane. We now have a sphere with two boundary, also known as an **annulus**.
- We’ll define something called the **universal cover** as follows: Imagine cutting along that path \( P \) from earlier, turning our annulus into a disk. Now make a doubly infinite sequence \( ..., G^*_{-1}, G^*_0, G^*_1, G^*_2, ... \) of copies of \( G^* \) and paste them together along their respective copies of \( P \).
- Formally, its a plane graph \( G_{\star} = (V_{\star}, E_{\star}) \) where \( V_{\star} = \{p_i \mid p \in V^* \text{ and } i \in \mathbb{Z}\} \) and \( E_{\star} = \{p_i \Rightarrow q_j \mid i + \text{pi}(p \Rightarrow q) \in E^*\} \). We also have dart costs \( c(p_i \Rightarrow q_j) = c(p \Rightarrow q) \).
- Each vertex $p_i$ is a lift of $p$ to $Gbar^*$. There is a projection map $\omega bar(Gbar^* \rightarrow G^*)$ that drops subscripts so $\omega bar(p_i) = p$ and $\omega bar(p_i \cdot q_j) = p \cdot q$.
- The preimage $\omega bar^{-1}(Pi)$ for any path $Pi$ in $G^*$ is a doubly-infinite set of paths in $Gbar^*$ called the lifts of $Pi$. If $Pi$ starts at $p$ and ends at $q$, then for any $i$, there is a lift of $Pi$ from $p_i$ to $q_{(i + pi(Pi))}$.
- $s^* \text{ and } t^*$ lift to two unbounded faces $sbar^*$ and $tbar^*$, and every other face lists to an infinite sequence of faces.

- So now for the punchline. Each path $i(q)$ in $G^*$ lifts to a shortest path to $q_0$ from $o_{(i)}$.
- The $i$ for which $p \rightarrow q$ is in $\text{path}_i(q)$ are the set of $i$ for which the shortest path from $o_{(i)}$ goes through $p_{(pi(p \rightarrow q))} \rightarrow q_0$.
- But we saw on Monday that these $i$ are contiguous!
- So $p \rightarrow q$ pivots into $T_{\lambda}$ at most once, and it leaves at most once.
- Which implies $O(n)$ pivots.
- Which implies an $O(n \log n)$ running time!
- Some quick notes:
  - Erickson ['10] described this formulation of the algorithm based on parametric shortest paths.
  - His algorithm is essentially identical to one by Borradaile and Klein ['09]. However, they describe things mostly in the primal graph, sending flow along LP each iteration to always have a “leftmost flow” of each value lambda. Their analysis is much more complicated, because they focus on how often you can saturate each dart with this primal flow.