Combinatorial Optimization

Jacob Denson

October 2, 2016

Table Of Contents

Chapter 1

Matchings and Flows

1.1 Bipartite Matching

Let $G = (V, E)$ be a graph. A **matching** is $M \subset E$ such that no vertex in V is the endpoint of more than one edge in M. M is maximal if $|M| \geqslant |M'|$ for any other matching M'. The maximal matching problem asks us to find a fast algorithm to find a maximum matching in any graph.

There is a polynomial time algorithm which can find matchings on any graph, but the problem is much more simple if G is bipartite – that is, if we may partition V into the disjoint union $W \amalg U$ of two sets of vertices, such that every edge in E contains a point in W and a point in U. The bipartite matching asks us to find a maximal matching in a bipartite graph.

Suppose we are given a particular matching in a bipartite graph. Is there a reliable procedure to improve the matching? We could proceed by a guess and check method – we remove an edge in our matching, then try and add an edge using one of the vertices which has been freed up. If this edge cannot be taken because the end point is attached to an edge still in the matching, we remove that edge, freeing up more vertices. If we ever end up adding more edges than we started with (which occurs when we add an edge not attached to any points in the current matching), then we find a matching with an extra edge than before. This process is formalized by the 'augmenting paths' construction.

Given a particular matching M, construct a directed graph $G_M = (V \cup \{s, t\}, E_M)$, where s and t are new vertices. Let $w \in W$, $u \in U$. Construct the edges E_M such that

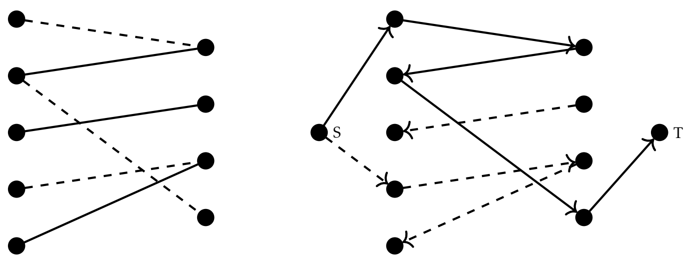

Figure 1.1: The left graph shows a matching in a graph of cardinality 3. The solid lines correspond to the edges chosen. On the right is the augmenting path graph corresponding to the matching. It contains a path from s to t, hence there is a way to improve the matching to a match of cardinality 4.

- $wu \in E_M$, if $wu \in M^c$.
- $uw \in E_M$, if $uw \in M$.
- $sw \in E_M$, if w is not the endpoint of any edge in M.
- $ut \in E_M$, if u is not the endpoint of any edge in M.

Let $(s, w_1, u_1, \dots, w_n, u_n, t)$ be a path in G_M from s to t, with $w_i \in W$, $u_i \in U$. Then $w_i u_i \in M^c$ and $u_i w_{i+1} \in M$, and w and u are both unused in M. Let M' be obtained from M by removing all edges of the form $u_i w_{i+1}$, and adding all edges of the form $w_i u_i$. It is easy to convince yourself that M' is a matching, with one more edge than M.

Conversely, suppose that there is no path from s to t in G_M. We claim M is then a maximal matching on G. Otherwise, we would have a matching M^* with $|M^*| > |M|$. Consider the multigraph $H = (V, M \amalg M^*)$. Every vertex in V has at most degree two (for it can only be the endpoint of a single edge in M, and a single endpoint in M^*), and thus V breaks into cycles and paths. For any cycle C, we find $|M \cap C| = |M^* \cap C|$, for the edges in the cycle must alternate being being in M and M^*. Since $|M^*| > |M|$, there must be a path P in H containing more edges in M^* than in M. P must therefore begin and end with a path in M^*, and we see that this path is exactly an augmenting path for M.

This argument justifies the correctness of the following matching algorithm, a variant of the 'Ford Fulkerson Algorithm'. We take any initial matching M. Then, we construct the augmenting path graph G_M, and use any of the standard algorithms to find a path from s to t. If we have $2n$ nodes, and m edges, then we can find a path in $O(m)$ time, and we can bound the number of augmentations we require to $O(n)$, so the algorithm runs in $O(nm)$ time.

1.2 Vertex Covers

A vertex cover $C \subset V$ is a set of points containing at least one endpoint of every edge in E. Clearly if M is a maximal matching in G, then $|M| \leqslant |C|$, because there is an injective function $f : M \to C$, obtained by mapping an edge to one of its endpoints which lies in C.

Theorem 1.1 (König - Egerváry). *Let $G = (W \amalg U, E)$ be bipartite, and let M be a maximal matching in G. Let $Z \subset V \cup \{s,t\}$ be the nodes reachable from s in the augmenting path graph G_M. Then $C = (W - Z) \cup (U \cap Z)$ is a cover with $|C| = |M|$.*

Proof. First we prove C is a cover. Let $wu \in E$ be arbitrary.

- Suppose that $wu \in M$. If $u \in Z$, then $w \in Z$, because uw is an edge in G_M, so that if $w \notin C$, $u \in C$.
- Suppose that $wu \notin M$. Then wu is an edge in G_M, so that if $w \in Z$, $u \in Z$, and therfore if $w \notin C$, $u \in C$.

Now we show $|U| = |M|$. First, note that C contains *exactly one* of each of the endpoints of M, so $|U| \geqslant |M|$. Conversely, let $wu \in E - M$ be arbitrary. If $u \in C$, then u is reachable from s, and hence u is the endpoint of an edge in M, for otherwise t is reachable from s through u. Conversely, if $w \in C$, then w is not reachable from s, hence w is on an edge in M (for otherwise there is an edge directly to w from s in G_M). This shows that every vertex in C is on one of the edge in M, and hence $|U| = |M|$. □

This gives us a method of finding a min cover in a graph. One equivalent form of this theorem is very useful. Given an undirected graph $G = (V,E)$, let N be the neighbour function $N(S) = \{v \in V : sv \in E, s \in S\}$.

Theorem 1.2 (Hall). *Let $G = (W \amalg U, E)$ be a bipartite graph. Then G has a matching M which covers W if and only if $|N(S)| \geqslant |S|$ for all $S \subset W$.*

Proof. Certainly if a matching M covers W, then it induces an injective function from S to $N(S)$ for each $S \subset W$. Conversely, assume no matching of M covers W. Then König's theorem tells us there is a cover C of $W \amalg U$ containing less than $|W|$ nodes. Then write $A = C \cap W$, $B = C \cap U$. We find $N(W - A) \subset B$, so $|N(W - A)| \leqslant |B| < |W| - |A|$. □

1.3 The Hopcroft-Karp Matching Algorithm

The augmenting path approach to finding a maximal matching in a Bipartite graph improves the length of a candidate matching by one for each cycle of the algorithm. If we were able to consistently find augmentations which improved matchings by more than one edge, we could find a faster algorithm for maximal matching.

If M is a matching, and N_M be the length of a shortest alternating path in G_M, we shall let a collection of vertex disjoint alternating paths $P_1, \dots, P_n$ be called **M-blocking** if $|P_i| = N_M$, and every alternating path of length N_M shares a vertex with one of the P_i. The next lemma is a simple generalization of the augmenting paths approach.

Theorem 1.3. *If $P_1, \dots, P_n$ are vertex disjoint M-alternating paths, then*

$$M' = (M - \cup P_i) \cup (\cup P_i - M)$$

is a matching with $|M'| = |M| + n$.

The Hopcroft-Karp algorithm proceeds as in Ford-Fulkerson, albeit improving matchings by taking M-blocking paths as augmentations, improving matchings in chunks. We shall discover a method to find an M-blocking in linear time, and also show the chunk method yields an asymptotic speedup in the number of cycles to find a maximal matching, so that Hopcroft-Karp is objectively faster than Ford-Fulkerson.

Lemma 1.4. *If M is a matching, and $P_1, \dots, P_n$ an M-blocking set of paths. Then $M' = (M - \cup P_i) \cup (\cup P_i - M)$, and $N_{M'} \geqslant N_M + 1$.*

Proof. Let $Q = (v_1, \dots, v_k)$ be an alternating path in $G_{M'}$ with $|Q| \leqslant |N_M|$, where v_1 and v_k are M' exposed. Then v_1 and v_k are also M exposed, because switching from M to M' only removes exposed vertices, not adds them.

Let H be the directed graph with edges taken from all the edge lying on the P_i, and the directed edges of Q, except that we remove both copies of duplicate edges. The reverse vu of any edge uv in Q that is not an edge on some P_i must be on some P_j $\square$

1.4 Network Flow

Let $G = (V, E)$ be a directed graph with two identified vertices $s \neq t$. Let $\mu : E \to \mathbf{R}^+$ be a function measuring the 'capacity' of each edge in the graph (Thinking of the edges as if they were 'pipes' which can only carry a certain throughput). Define, for $U \subset V$,

$$\delta_{\text{out}}(U) = \{w \in V - U : vw \in E\} \qquad \delta_{\text{in}}(v) = \{w \in V - U : wv \in E\}$$

A flow is a mapping $f : E \to \mathbf{R}^+$ such that $0 \leqslant f \leqslant \mu$, and for any $v \neq s, t$,

$$f(\delta_{\text{out}}(v)) = f(\delta_{\text{in}}(v)) \; ^*$$

a relation known as the flow conservation law. The aim of the maximum flow problem is to find f such that the value function

$$\text{val}(f) = f(\delta_{\text{out}}(s)) - f(\delta_{\text{in}}(s))$$

is maximized. This is essentially the amount of flow which is created at f. It is also the amount of flow which is 'absorbed' at t, because

$$\begin{aligned} f(\delta_{\text{out}}(t)) - f(\delta_{\text{in}}(t)) &= f(\delta_{\text{out}}(t)) - f(\delta_{\text{in}}(t)) + \sum_{v \neq s,t} f(\delta_{\text{out}}(v)) - f(\delta_{\text{in}}(v)) \\ &= \sum_{v \neq s} \left(\sum_{vw \in E} f(vw) - \sum_{wv \in E} f(wv) \right) \\ &= \left(\sum_{\substack{v \neq s \\ vs \in E}} f(vs) + f(ss) \right) - \left(\sum_{\substack{v \neq s \\ sv \in E}} f(sv) + f(ss) \right) \\ &= -[\delta_{\text{out}}(f, s) - \delta_{\text{in}}(f, s)] \end{aligned}$$

Thus the maximum flow problem tells us a method of maximizing the amount of flow which gets to t.

*We extend functions $g : X \to \mathbf{R}$ to $g : 2^X \to \mathbf{R}$ by defining $g(A) = \sum_{x \in A} g(x)$

There is an interesting relation between flows on graphs, and another structure on these graphs known as a cut, which we will take full advantage of in finding solutions to the max flow problem. Define a (s,t) cut on a directed graph $G = (V,E)$ with vertices s and t, to be a partition of V into two sets, one containing s, and one containing t. It is however more simple to consider a cut to be a subset C of V containing s, but not t. If we have a capacity function μ, then we define the value of the cut to be $\text{val}(C) = \mu(\delta_{\text{out}}(C))$. The min cut problem is to find a cut of smallest value.

Lemma 1.5. *If f is a flow, and C is an (s,t) cut, then $val(f) \leqslant \mu(\delta_{out}(C))$.*

Proof. Since $t \notin C$, similar manipulations to the ones above show that

$$\begin{aligned}\text{val}(f) &= \sum_{v\in C} f(\delta_{\text{out}}(v)) - f(\delta_{\text{in}}(v)) \\ &= f(\delta_{\text{out}}(C)) - f(\delta_{\text{in}}(C)) \\ &\leqslant \mu(\delta_{\text{out}}(C))\end{aligned}$$

Note that we obtain equality if $f(\delta_{\text{out}}(C)) = \mu(\delta_{\text{out}}(C))$ and $f(\delta_{\text{in}}(C)) = 0$, in which case f must be a maximal flow, and C a min cut. $\square$

As with maximal matchings, we attempt to find a maximal flow by finding ways of augmenting suboptimal flows to a maximal solution. Given a flow f, we construct the residual graph G_f, which has the same vertices as G, but whose edges are defined to be the set

$$E_f = \{uv \in E : f(uv) < \mu(uv)\} \coprod \{vu : uv \in E, f(uv) > 0\}$$

We shall denote the element vu in the second set which makes up E_f by $\overleftarrow{uv}$, since we obtained in from an edge uv in G. Also define a capacity function

$$\mu_f(uv) = \mu(uv) - f(uv) \qquad \mu_f(\overleftarrow{uv}) = f(uv)$$

Suppose we can find a simple path P from s to t in G_f, and define $\alpha = \min_{e\in P} \mu_f(e)$. Consider a new flow

$$f'(uv) = \begin{cases} f(uv) + \alpha : uv \in P \\ f(uv) - \alpha : \overleftarrow{uv} \in P \end{cases}$$

First we show that f' is a flow. By the choice of α, $0 \leqslant f' \leqslant \mu$. For each vertex $v \neq s, t$, we must show that flow conservation still holds. Let $e_1, e_1', \ldots, e_n, e_n'$ be the edges in G containing v as an endpoint obtained from P, by reversing edges of the form $\overleftarrow{uv}$, considered in the order they appear in P. We may write pair up the edges in this manner because, for every edge that enters v in P, there must be an edge that leaves v in P. Let

$$S_i = \begin{cases} f(e_i) - f'(e_i) & e_i = uv \\ f'(e_i) - f(e_i) & e_i = vu \end{cases}$$

Define S_i' similarily for e_i'. Then, since no edges are repeated,

$$\begin{aligned} f'(\delta_{\text{out}}(v)) - f'(\delta_{\text{in}}(v)) = f(\delta_{\text{out}}(v)) - f(\delta_{\text{in}}(v)) \\ + \sum_{i=1}^{n} (S_i + S_i') \end{aligned}$$

We now show $S_i + S_i' = 0$ for each i. This breaks into several cases.

- If $e_i = uv$, $e_i' = vw$, in which case $f'(e_i) = f(e_i) + \alpha$, $f'(e_i') = f(e_i') + \alpha$, and $S_i + S_i' = \alpha - \alpha = 0$.
- If $e_i = uv$, $e_i' = wv$, then e_i' was obtained from an edge of the form $\overleftarrow{vw}$ in P, in which case $f'(e_i') = f(e_i') - \alpha$, and so $S_i + S_i' = -\alpha - (-\alpha) = 0$.
- If $e_i = vu$, $e_i' = wv$, then e_i and e_i' were obtained from reversed edges in P, and so $S_i + S_i' = (-\alpha) - (-\alpha) = 0$.
- If $e_i = vu$, $e_i' = vw$, then e_i was obtained from reversing edges, and so $S_i + S_i' = (-\alpha) + \alpha = 0$.

And it follows, because flow conservation holds at v for f, that it holds at v for f' as well. Finally, we find that $\text{val}(f') = \text{val}(f) + \alpha$, which can be shown by performing an analysis, similar to v above, for s, noting that s will have an extra edge at the beginning of the path, which causes the extra α. All that remains is to show this augmentation method yields a maximum flow in all cases, after enough iterations.

Let f be a flow for which G_f has no augmenting path. Let U be the nodes reachable from s in G_f, easily verified to be a cut.

$$\text{val}(f) = \mu(\delta_{\text{out}}(U))$$

proving maximality of the flow, and minimality of the cut U. To see this, let $uv \in \delta_{\text{out}}(U)$. Then $f(uv) = \mu(uv)$, for otherwise v would be reachable from s in G_f. Similarily, we must have $f(\delta_{\text{in}}(U)) = 0$, for if $vu \in \delta_{\text{in}}(U)$, and $f(vu) > 0$, then $\overleftarrow{vu} \in G_f$, and so v is reachable from s, a contradiction. We conclude that

$$\text{val}(f) = f(\delta_{\text{out}}(U)) - f(\delta_{\text{in}}(U)) = \mu(\delta_{\text{out}}(U))$$

which shows that our augmenting paths approach works. Notice that if all edge weights are integral, then the augmenting flows are always integral, and thus there exists a maximal flow with interal weights.

The Ford Fulkerson algorithm solves the max flow problem by repeatedly augmenting an initial flow. In principle, this approach is correct, but if you do not find a residual path in a smart way, this algorithm does not terminate in polynomial time for all inputs. If all edge weights are integers $n_1, \dots, n_m$, then each augmentation increases the value of the flow by at least the greatest common factor of the n_i. Since there is a bound on the size of a maximal flow, obtained by taking the trivial cut $\{s\}$, the algorithm will eventually terminate in time proportional to

$$O\left(\frac{n_1 + \dots + n_m}{\text{lcm}(n_1, \dots, n_m)}\right)$$

which is exponential in the bit complexity of the representation. We cannot do any better than this, as the graph in the figure above provides an example. If the edge weights are rational, similar results can be obtained by multiplying out denominators.

We achieve much better estimates if we use breadth first search to find our residual path – that is, we always take the shortest path (in length, not in weight) from s to t. But this, of course, requires a careful analysis.

Theorem 1.6 (Mader). *Let G be an undirected graph, and s,t vertices not directly connected by an edge. Then the maximal number of vertex-disjoint (s,t) paths is equal to the minimum cardinality of a set $B \subset V - \{s,t\}$ which contains a vertex in any path from s to t.*

Proof. Form a directed bipartite graph W, which consists of two copies of the vertices in G. If v is a vertex, then we shall separately denote its copies by v^1 and v^2. If uv is an edge in G, attach edges u^2v^1 and v^2u^1 to W, and attach an edge v^1v^2 for each vertex v. Define a capacity function on w by

$$\mu(u^2v^1) = \mu(v^2u^1) = \infty \qquad \mu(v^1v^2) = 1$$

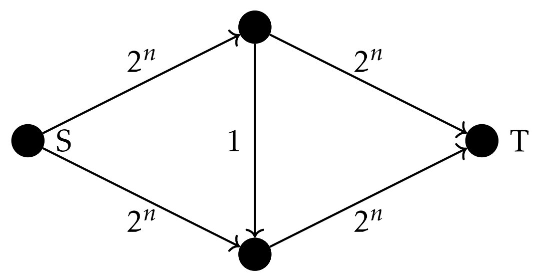

Figure 1.2: If our algorithm continuously switches between adding weights from left to right via the central edge, then we will need to compute $O(2^n)$ residual graphs before termination, even though the graph can be represented in $O(n)$ bits.

Note that any (s,t) path in G leads to an alternating path in W from s^2 to t^1, which must cross at least one 1-capacity edge because there is no edge from s to t.

Let f be an integer-valued flow in W. Then $f \leqslant 1$, because an infinite capacity edge u^2v^1 satisfies $f(u^2v^1) \leqslant f(\delta_{\text{in}}(u^2)), f(\delta_{\text{out}}(v^1))$ □

1.5 Goldberg-Tarjan Push Relabel Algorithm

Our previous algorithm found max flows in a graph by starting with a flow, then slowly augmenting it, improving the flow value until it is maximum. An alternative idea would be to take a flow-like function, which would be a maximum flow if it obeyed flow conservation, and slowly correcting the flow until we have a maximum flow. Define a **preflow** on a graph with capacity function μ to be a positive real-valued function f, defined on the edges, such that $f \leqslant \mu$, and $f(\delta_{\text{out}}(v)) - f(\delta_{\text{in}}(v)) \geqslant 0$ for all $v \neq s$. A preflow defines a residual graph G_f in the same way that a flow does. A non-negative integer valued function ψ on vertices is a **distance label** for a particular preflow f if $\psi(s) = |V|$, $\psi(t) = 0$, and $\psi(v) \leqslant \psi(w)+1$ for $vw \in G_f$. Notice that if there is a path from s to t in G_f, then a distance label cannot exist, because if it has length k, then the inequality property gives

$$\psi(s) \leqslant (k-1) + \psi(t) \leqslant (|V|-1) + \psi(t) < |V|$$

Thus if we continuously keep track of a preflow and a distance label, and keep adjusting the preflow and distance label to obtain a flow with a distance label, we have obtained a maximal flow. In order to describe a method achieving this, we call a vertex $v \neq s,t$ **active** if $f(\delta_{\text{out}}(v)) - f(\delta_{\text{in}}(v)) > 0$, and an edge $vw \in G_f$ **admissable** if $\psi(v) = \psi(w) + 1$. The goal is to find a preflow with no active vertices, in which case we have a flow.

We now describe the push-relabel algorithm for finding the maximal flow in a graph. Begin by setting

$$f(e) = \begin{cases} \mu(e) & e = sv \\ 0 & \text{otherwise} \end{cases} \qquad \psi(v) = \begin{cases} |V| & v = s \\ 0 & v \neq s \end{cases}$$

Then ψ is a distance label for the preflow f, since G_f has no edges leaving s.

Suppose there is an active vertex v relative to f. If there are no admissable edges leaving v, we may introduce an admissable edge by setting $\psi(v) = \min_{vw \in G_f} \psi(w) + 1$. This leaves the f a preflow, and ψ a distance label. If there is an admissable edge vw leaving v, we set $f(vw)$ to be

$$\min[\mu_f(vw), f(\delta_{\text{out}}(v)) - f(\delta_{\text{in}}(v))]$$

Then $f \leqslant \mu$ still holds, and $f(\delta_{\text{out}}(v)) - f(\delta_{\text{in}}(v)) \geqslant 0$, since we can only have let at most $f(\delta_{\text{out}}(v)) - f(\delta_{\text{in}}(v))$ flow out. Also $f(\delta_{\text{out}}(w)) - f(\delta_{\text{in}}(w)) \geqslant 0$, since we have only let more flow in. The distance labeling is still a distance labelling, because we introduce an edge wv in G_f, which requires that $\psi(w) \leqslant \psi(v) + 1 = \psi(w) + 2$, and an edge from vw, so that $\psi(v) = \psi(w) + 1 \leqslant \psi(w) + 1$. Thus the fact that ψ is a distance labelling of the preflow f is an invariant of the algorithm.

Lemma 1.7. *If f is a preflow and ψ a distance label for f, then if $f(\delta_{out}(v)) - f(\delta_{in}(v)) > 0$, there is a (v,s) path in G_f.*

Proof. s □

1.6 Cuts in Undirected Graphs

Let G be an undirected graph with capacities $\mu \geqslant 0$. For a subset of vertices U let $\delta(U)$ denote all edges with a single endpoint in U. A **cut** in such a graph is $\varnothing \subsetneq U \subsetneq V$, with value $\mu(\delta(U))$. The global minimum cut problem is to find the cut with $\mu(\delta(U))$ minimized, without regard to where any particular pair (s,t) resides. A naive way to calculate the global minimum cut is to calculate a minimum (s,t) cut, for all pairs (s,t). This requires n^2 calls to the maximum flow problem.

And while we discuss this problem, we briefly sketch out how to compute a max flow in an undirected graph. Given such a graph G with capacity μ, we define H to be the directed graph with the same vertices as G, and for each edge $uv \in G$, two edges uv and vu in H, with a capacity function ψ such that $\psi(uv) = \psi(vu) = \mu(uv)$. Given any flow f in the directed graph, we obtain a flow f' in the undirected graph of the same value by subtracting $\min(f(uv), f(vu))$ from each edge. Thus we can compute a max flow in $O(V^2E)$ time in an undirected graph as well. The value of min cuts in the directed graph is the same as the value of min cuts in the undirected graph, so the max flow min cut theorem holds here as well.

More smartly, we notice that if s is fixed, s lies on either side of the cut, and since $\mu(\delta(U)) = \mu(\delta(U^c))$, we need only try each (s,t) cut as t ranges over all $t \neq s$, which only requires $n-1$ calls to the max flow function. In a directed call, we need try all (s,t) cuts and (t,s) cuts, which requires $2(n-1)$ calls to max flow.

We can do much better than just computing a min-cut in $O(n)$ times. It turns out that there is a way to compactly represent all minimum cuts via $O(n)$ calls to a min-cut algorithm. The first step to the representation is to find a useful relationship between the values of the numbers $\lambda_{u,v}$, the capacities of a min (u,v) cut.

Lemma 1.8. *Let $v_1, \dots, v_k$ be vertices. Then $\lambda_{v_1,v_k} \geqslant \min(\lambda_{v_1,v_2}, \dots, \lambda_{v_{k-1},v_k})$.*

Proof. Assume that $k \geqslant 3$, since the proof is trivial for $k = 2$. Let C be a (v_1, v_k) cut with $\mu(\delta(C)) < \min(\lambda_{v_1,v_2}, \dots, \lambda_{v_{k-1},v_k})$. Inductively, we conclude that $v_2, \dots, v_k \in C$. But then by definition C cannot be a (v_1, v_k) cut. □

Let $T = (V,E)$ be a tree, and $e \in E$ an edge in the tree. Then the set of nodes in some connected components of the graph $T' = (V, E - \{e\})$ is known as the **fundamental cut**. There are only two connected components, one for each vertex on the ends of e. A **Gomory-Hu Tree** for an undirected graph G is a tree T with the same vertices as G, such that the fundamental cut for every edge $uv \in T$ is a minimal uv cut. It turns out that Gomory-Hu trees always exist, can be computed in $O(n)$ calls to a min (s,t) cut function, and compactly represent every min cut in a graph, for if $(v_1,\dots,v_n)$ is the unique simple path in T, then

$$\lambda_{v_1,v_n} = \min(\lambda_{v_1,v_2},\dots,\lambda_{v_{n-1},v_n})$$

Certainly the left side is $\geqslant$ to the right side. On the other hand, by construction of the tree, the fundamental cut of (v_i,v_{i+1}) is also a (v_1,v_n), hence the left side is $\leqslant$ to the right side.

Theorem 1.9 (Cut Submodularity)**.** *For $A,B \subset V$, we have*

$$\mu(\delta(A)) + \mu(\delta(B)) \geqslant \mu(\delta(A \cap B)) + \mu(\delta(A \cup B))$$

Proof. Fix an edge uv. We shall show that every edge counted on the right side occurs at least as many times on the left side.

- Suppose $uv \in \delta(A \cap B)$, $uv \notin \delta(A \cup B)$. Then we may assume $u \in A \cap B$, $v \notin A \cap B$, $v \in A \cup B$. Then $\mu(uv)$ is counted once on the right hand side, and once on the left hand side.

- Suppose $uv \notin \delta(A \cap B)$, $uv \in \delta(A \cup B)$. Thus we may assume $u \in A \cup B$, $v \notin A \cup B$, $u \notin A \cap B$. Then $\mu(uv)$ is counted once on the right hand side, and once on the right hand side as well.

- Suppose $uv \in \delta(A \cap B)$, $uv \in \delta(A \cup B)$. Then we may assume $u \in A \cup B$, $v \notin A \cup B$, and so $u \in A \cap B$. Then $\mu(uv)$ is counted once on the right hand side, and twice on the left hand side.

Equality holds when $\delta(A \cap B)$ is disjoint from $\delta(A \cup B)$. □

Lemma 1.10. *Let $s \neq t$ be two vertices in a graph G, and let C be a min (s,t) cut. If $u \neq v$ are also distinct vertices, then there is a min (u,v) cut D with $C \subset D$ or $D \cap C = \varnothing$.*

Proof. Note that the statement is really redundant, because $C \subset D$ occurs exactly when $D^c \cap C = \varnothing$. Let B be any minimum (u,v) cut. We will modify B to another min cut with the desired properties. Note that

$$\mu(\delta(C)) + \mu(\delta(B)) \geqslant \mu(\delta(C \cap B)) + \mu(\delta(C \cup B))$$

We may assume $s,u \in B$. Then $t \notin B$, $C \cap B$ is an (s,t) cut, hence $\mu(\delta(C \cap B)) \geqslant \mu(\delta(C))$, and so

$$\mu(\delta(B)) \geqslant \mu(\delta(C \cup B))$$

Now $u \in C \cup B$, and $v \notin C \cup B$ (WHY?), so $D = C \cup B$ satisfies the properties of the lemma. □

1.7 Potentials

Fix a directed graph G with edge costs c. A **potential** for G is a mapping $\phi : V \to \mathbf{R}$ such that

$$c_\phi(uv) = c(uv) - (\phi(v) - \phi(u)) \geqslant 0$$

for al edges uv in G. The costs c_ϕ are known as the potential costs.

Lemma 1.11. *For any $s - t$ path P in G,*

$$c_\phi(P) = c(P) - (\phi(t) - \phi(s))$$

Proof. If $P = (v_1, \dots, v_n)$, then we have an alternating sum

$$c_\phi(P) = \sum_{k=1}^{n-1} c(v_k v_{k+1}) + \phi(v_k) - \phi(v_{k+1}) = \sum_{k=1}^{n-1} c(v_k v_{k+1}) + \phi(v_1) - \phi(v_n)$$

and the right side is exactly the form considered in the statement of the theorem. □

This is a discrete version of the fact that the energy change of a particle under a conservative force is invariant of the path of the particle, but only the change of the particles position in space. A simple corollary to the lemma is that the minimal cost paths relative to the metric c are the same as the minimal cost paths relative to c_ϕ.

If a graph has negative edge weights, but possesses some potential ϕ, then we can compute minimal paths under the cost function c_ϕ (which is now has positive edge weights everywhere) using Dijkstra's algorithm. There is a simple method to finding a potential, which we now detail.

If P is a cycle in a graph G with potential ϕ, we calculate

$$c_\phi(P) = c(P) \geqslant 0$$

so that G has no negative weight cycles. Conversely, if G has no negative weight cycles, then it turns out that G has a potential, and this potential can be computed in polyomial time.

Theorem 1.12. *A potential can be computed for a graph G can be computed in $O(nm)$ time.*

Proof. Given G, compute a graph H by adding a new vertex r to G, and adding edges from r to every vertex in G. Set $c(rv) = 0$ for all vertices $v \in V$, and let $\phi(v)$ be the cost of the cheapest walk from r to v. This is well defined since G has no negative weight cycles (there is a cheapest walk with no repeated vertices, for if a path has a repeated vertex, it contains a cycle, which must have non-negative weight, and this cycle can be removed with no penalty to the length of the walk – we then need only consider the finite set of paths with no repeated vertices to obtain an answer). Note that $c(uv) + \phi(u) \geqslant \phi(v)$, so ϕ is a potential for G. The Bellman-Ford can compute ϕ in $O(nm)$ time. □

Let ω be a differential 1-form on a manifold M such that

$$\int_\gamma \omega \geqslant 0$$

for any cycle γ. Then if γ and δ are two paths from a point x to a point y, we find

$$\int_\gamma \omega - \int_\delta \omega, \int_\delta \omega - \int_\gamma \omega \geqslant 0$$

so $\int_\gamma \omega = \int_\delta \omega$, and given a fixed x, we can define

$$F(y) = \int_\gamma \omega$$

where γ is a path from x to y, and one verifies that $dF = \omega$, so that

$$\int_\gamma \omega - (F(y) - F(x)) = 0$$

a stronger fact which is analogous to the result above, although our result is weaker because in directed graphs we cannot perform the swapping trick to obtain equality of all paths.

1.8 Minimum Cost Flows

Throughout this section, we fix a directed graph with edge costs c, edge capacities μ, fixed vertices s, t, and a target flow value γ. The minimum cost flow problem tries to find a flow f with value $\text{val}(f) \geqslant \gamma$, which minimizes the cost $\text{cost}(f) = \sum c(e) f(e)$, or to determine if no such flow exists.

We can certainly compute in polynomial time if a flow exists satisfying the constraints, because we can find a maximal flow in the graph. It is not so clear that we can find such a flow with minimal cost. It shall always be the case that the minimal flow satisfies $\text{val}(f) = \gamma$, because given any flow f, the value of the flow $g = \text{val}(f)/\gamma f$ satisfies $\text{val}(g) = \gamma$, and $\text{cost}(g) = \text{val}(f)/\gamma \sum c(e) f(e)$, which will be smaller than $\text{cost}(f)$ if $\text{val}(f) \neq \gamma$.

1.9 Minimal Cost Bipartite Matching

We now take a brief aside to discuss how minimal cost flows can be used to solve the analogous problem in bipartite graphs. Given the name, the minimal cost bipartite matching problem should be fairly self explanatory. We take a particular Bipartite graph $G = (L \cup R, E)$ with edge costs c, and we must find a minimum cost perfect matching on G in polynomial time. That is, our goal is a perfect matching M which minimizes

$$\text{cost}(M) = \sum_{e \in M} c(e)$$

The similarity to the minimal cost flow problem immediately suggests a connection between the two problems.

We can formulate the perfect matching problem as a minimal cost max flow problem, by modifying slightly the standard reformulation of the maximal matching problem as a max flow problem. The original process is to construct a directed bipartite graph H, whose vertices are obtained from G by adding two new vertices s and t, orienting all edges in E so that they point from L to R, and adding new edges sv, for $v \in L$, and wt, for $w \in R$. If we define a capacity function $\mu = 1$, then there is a one-to-one correspondence between integral valued $s - t$ flows and matchings in the graph. To see this, note that an integral valued flow f gives rise to a characteristic function χ_{A_f} on the edges of G, and the flow conservation law shows that if e and e' are adjoint, then $\chi_{A_f}(e) + \chi_{A_f}(e') \leqslant 1$, so that A_f forms a matching on G. Correspondingly, the reverse process takes a matching to a characteristic function on the edges of G, which can easily be completed into a flow on H.

Now we define a cost function c' on H in a way which mirrors the cost function c on G. We let

$$c'(sv) = c'(wt) = 0 \qquad c'(vw) = c(vw)$$

Then we see that the correspondence described above preserves the cost function. That is, if a matching M corresponds to a flow f, then $\text{cost}(M) = \text{cost}(f)$, because f is obtained from M by adding edges of the form sv and wt, which add no value to the cost of the flow. Thus if we find a minimal cost maximal flow in H, we can find a minimal cost perfect matching in G, and we know we can find such a maximal flow using the successive shortest paths algorithm, and we need only compute $|L| = |R|$ successive paths, since each iteration of the algorith improves the length of the candidate path, hence now have an algorithm for the minimal cost perfect matching problem which runs in $O(n(m + n\log n))$ time.

1.10 Minimum Mean Cycle Cancelling

We already know that a flow f has minimal cost if G_f contains no negative-cost cycles. This suggests that a strategy for finding minimal cost flows is obtained from successively pruning negative-cost cycles from G_f until there are none left. As with Ford Fulkerson, we must be careful which cycles we prune in order to guarantee a good asymptotic speed to our algorithm. It turns out that fast asymptotic results are available if we prune negative cycles with 'minimum mean'.

Algorithm 1 The Minimum Mean Cycle Canceling Algorithm

```
1: $f \leftarrow$ any $(s,t)$ flow of value $\gamma$ (found in $O(n^3\sqrt{m})$ time)
2: while $G_f$ contains negative weight cycles do
3:     $C \leftarrow$ a cycle in $G$ which minimizes its mean cost $c(C)/|C|$.
4:     Augment $f$ along $C$.
5: return $f$
```

The algorithm certainly terminates, since the cost of f decreases for every iteration of the algorithm, at a rate bounded by the rational values of the edge costs. It remains to be seen that the algorithm terminates in a polynomial amount of time (and this is not immediately obvious, and is not true if we do not take cycles which minimize the mean cost).

First, we argue why we can find a minimum ratio cycle C in polynomial time.

Lemma 1.13. *Any Eulerian multigraph $H = (W, F)$ with edge costs c contains a cycle C such that*

$$\frac{c(C)}{|C|} \leqslant \frac{c(F)}{|F|}$$

Proof. Perform a cycle decomposition on H, writing $F = C_1 \amalg \cdots \amalg C_n$, we find

$$\frac{c(F)}{|F|} = \frac{c(C_1) + \cdots + c(C_n)}{|C_1| + \cdots + |C_n|}$$

If $c(C_i)/|C_i| > c(F)/|F|$ for each i, then $c(C_i)|F| > c(F)|C_i|$ also holds for each i, and so, summing up we find

$$c(F)|F| = \sum_i c(C_i)|F| > \sum_i c(F)|C_i| = |F|c(F)$$

a clear contradiction which proves the existence of a C_i satisfying the constraints of the lemma. $\square$

Given any cycle C, the corresponding multigraph (which contains duplicates of repeated edges in the cycle) is Eulerian, and can thus be broken up into disjoint cycles. The lemma above shows that if C is not a simple cycle, we can always find a strictly smaller cycle C' without increasing the mean value of the cycle, so that we need only search simple cycles to find a minimum mean.

For $k \leqslant n$, define $\delta_k(v)$ to be the cheapest closed walk from v to itself using exactly k (not necessarily distinct) edges. If we let $\delta_k(v, w)$ be the cheapest path from v to w using k edges, then we find

$$\delta_k(v, w) = \min_{u \in V} c(v, u) + \delta_{k-1}(u, w)$$

Each value of $\delta_k(v, w)$ can be computed in $O(m)$ time, and so the set of all $\delta_k(v, w)$ can be computed in $O(mn^3)$ time. Let v^* and k^* minimize $\delta_{k^*}(v^*)/k^*$, and pick k^* to be smallest to break ties. Then the walk from v^* in k^* stops is verified to be a cycle, from the above remark, and thus is a minimum ratio cycle, since it is a minimum ratio walk in general.

Theorem 1.14. *The number of iterations of the minimum mean cycle cancelling algorithm is $O(m^2 n \log n)$.*

Proof. Consider the sequence of flows

$$f_1, f_2, \dots, f_N$$

which are found in the MMCC algorithm, together with the cycles

$$C_1, \dots, C_{N-1}$$

which are augmented to produce the sequence of flows. For $i < j$, define

$$\alpha(i,j) = \{e, \overleftarrow{e} : e, \overleftarrow{e} \in C_i \cup C_j\}$$

Lemma 1.15. *If $\alpha(i,k) = \varnothing$ for $i < k < j$, then we find*

$$\frac{c(C_i)}{|C_i|} \leqslant \frac{n}{n - |\alpha(i,j)|} \frac{c(C_j)}{|C_j|}$$

Proof. Let $H = C_i \coprod C_j - \alpha(i,j)$ (essentially, H is obtained from the cycles C_i and C_j by keeping duplicate edges, and ‘cancelling out’ edges with their reverse). Then H is Eulerian, for $C_i \coprod C_j$ is Eulerian, and removing edges along with their reverse changes both the in degree and out degree of a vertex by the same amount. Furthermore, each edge in H is in E_{f_i} because $\alpha(i,k) = \varnothing$, so

$$\begin{aligned}\frac{c(C_i)}{|C_i|}(|C_i| + |C_j|) &= \frac{c(C_i)}{|C_i|}(|H| + |\alpha(i,j)|)\\ &\leqslant c(H) + |\alpha(i,j)|\frac{c(C_i)}{|C_i|}\\ &= c(C_i) + c(C_j) + |\alpha(i,j)|\frac{c(C_i)}{|C_i|}\end{aligned}$$

where we have used the fact that $c(H) = c(C_i) + c(C_j)$ since the cost of reverse edges cancels out. This can be rephrased as

$$\frac{c(C_i)}{|C_i|}\left(|C_j| - |\alpha(i,j)|\right) \leqslant \frac{c(C_j)}{|C_j|}|C_j|$$

And dividing out, we find

$$\frac{c(C_i)}{|C_i|} \leqslant \frac{|C_j|}{|C_j| - |\alpha(i,j)|} \frac{c(C_j)}{|C_j|}$$

and this inequality can be weakened to the form above, because $x/(x-y)$ is a decreasing function of x for $y > 0$, and $n/(n - |\alpha(i,j)|) \geqslant 0$. $\square$

Returning to our original proof, we see that $j = i + 1$ satisfies the theorem, and proves that the minimum mean ratios are non-decreasing. If $i < j$ is the smallest value such that $\alpha(k,i) \neq \emptyset$, for some $i \leqslant k < j$, then we find

$$\frac{c(C_i)}{|C_i|} \leqslant \frac{n}{n-2} \frac{c(C_j)}{|C_j|} = \left(1 - \frac{n}{2}\right) \frac{c(C_j)}{|C_j|}$$

and we have a bound $j \leqslant i + m$. To see this, note that at least one edge is removed in the residual graph from each of the cycles $C_i, C_{i+1}, \ldots, C_{i+m}$ during the algorithm. Thus we conclude

$$\frac{c(C_i)}{|C_i|} \leqslant \left(1 - \frac{n}{2}\right)^n \frac{c(C_{i+nm})}{|C_{i+nm}|} \leqslant e^{-1/2} \frac{c(C_{i+nm})}{|C_{i+nm}|} \leqslant 2 \frac{c(C_{i+nm})}{|C_{i+nm}|}$$

If the edge weights are integral (and all rational cost problems can be reduced to this form), this already gives us enough information to prove a polynomial time bound. Note that since the minimum mean cycles are simple, we must have

$$\frac{|c(C_i)|}{|C_i|} \geqslant \frac{1}{n}$$

and

$$\frac{|c(C_1)|}{|C_1|} \leqslant \min_{e \in E} |c(e)|$$

so if N is large enough that $2^{-N} \min_{e \in E} |c(e)| \leqslant n^{-1}$, then

$$\frac{c(C_{i+Nnm})}{|C_{i+Nnm}|} \geqslant \frac{1}{2^N} \frac{c(C_1)}{|C_1|} \geqslant -\frac{\min_{e \in E} |c(e)|}{2^N} \geqslant -\frac{1}{n}$$

which forces the mean cycle ratio to be positive, and hence the algorithm will terminate. We can choose any $\lg(n \min_{e \in E} |c(E)|) < N$, so the algorithm terminates in at most

$$nm \lceil \lg n \min_{e \in E} |c(E)| \rceil$$

iterations, which is polynomial in the bit complexity of the algorithm, since if the costs are given with l bits, then the total bit complexity is about $\Theta(n+ml)$, and $|c(E)| \leqslant 2^l$, so

$$nm\lceil \lg n \min_{e \in E} |c(E)| \leqslant nm \lg(n) + lnm$$

This finishes our initial analysis of the mean cycle canceling algorithm. $\square$

The polynomial bound for this algorithm is absolutely awful, and we can do much much better. In the next lecture we will begin the arguments that give us a much better bound on the runtime on this algorithm.