Multicommodity Flow

1 Introduction

Suppose we have a company with a factory s and a warehouse t. The quantity of goods that they can ship from the factory to the warehouse in a given time period is limited by the kind of road network connecting the two facilities. The company wants to determine the maximum quantity of goods that can be shipped through the road network. This scenario is a case of the standard max-flow problem for a single commodity network. It is well known that the max-flow is equal to the min-cut, which in this case would be the set of roads with the smallest capactiy such that removing the roads disconnects the graph. Perhaps it is a collection of bridges that cause a bottle neck when trying to cross the local river.

Now suppose instead that our company has multiple factories that each make a different product and several warehouses, each of which has a different demand for each of the different products. We define the concurrent maximum flow to be the largest fraction f such that we can fill a fraction f of each of these demands simultaneously. The min-cut is now the cut with the smallest ratio of capacity to demand that needs to cross it. In this multicommodity flow scenario it is no longer true that the max-flow is equal to the min-cut. However, Leighton and Rao proved that in a system with uniform demands, the max-flow and min-cut are separated by a factor of $O(\log n)$ where n is the number of vertices in the graph.

Multicommodity max-flow can be solved exactly via linear programming, so this result gives a way to approximate the min-cut. This approximation for min-cut leads to log factor approximations of a variety of graph partitioning problems. This in turn allows many problems that can be solved by divide and conquer algorithms that rely on graph partitioning to be approximated.

These lecture notes are based largely on Leighton and Rao's 1999 paper **Multicommodity Max-Flow Min-Cut Theorems and Their Use in Designing Approximation Algorithms**.

2 Single Commodity Flow

A directed flow network like in Figure 1 is a graph $G = (V, E)$ where:

- each edge has a capacity c_e
- a source node $s \in V$
- a sink node $t \in V$.

A flow f is a directed graph with the same vertices of G, where every edge has a value ranging from 0 to c_e, where c_e is the capacity of the edge

$$0 < f(e) < c_e$$

and follows the conservation of flow coming in and out of vertices

$$\sum_{e \text{ into v}} f(e) = \sum_{e \text{ out of v}} f(e).$$

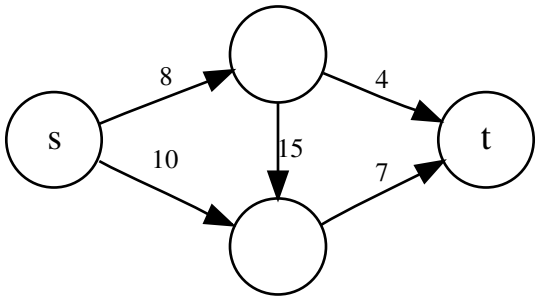

Figure 1: An example flow network

The flow value $k(f)$ of a flow f on G is the amount of flow generated at the source s

$$k(f) = \sum_{e \text{ out of } s} f(e).$$

2.1 Ford-Fulkerson Method

The Ford-Fulkerson method finds a solution S on G of maximum flow value as follows:

1. initially find a flow f
2. repeatedly find augmenting flows and add them to the flow f.

2.1.1 Finding an Initial Flow

To begin, use depth first search to find any path from s to t. This path is the initial flow f, where the minimum edge capacity of all edges along this path is the flow value

$$k(f) = \min\{c_e \text{ of edges along the path from } s \text{ to } t\},$$

as shown in red in Figure 2.

This follows from the fact that the smallest capacity edge in a path limits the total amount of flow from s to t.

2.1.2 Creating Residual Graphs

After finding a flow f, a residual graph R is constructed. R has the same set of vertices V as G. However, each directed edge in G is replaced in the residual graph R with a pair of oppositely directed edges.

In the direction of the original directed edge in G, R has an edge of capacity $c_e - k(f)$. Intuitively, this describes the flow that can still be allocated along this edge in the original flow network G.

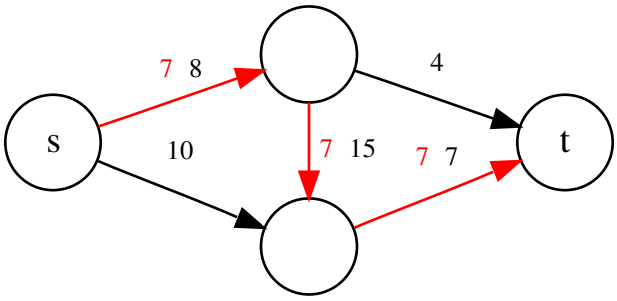

Figure 2: An initial flow

In the opposite direction of the original directed edge, R has a directed edge of capacity $k(f)$. This describes the flow that can be "pushed back" from the flow and redirected into other edges. Figure 3 describes a residual graph.

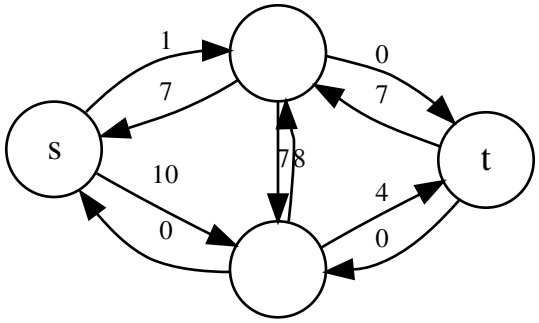

Figure 3: The Residual Graph after Initial Flow from Figure 2

2.1.3 Adding Augmenting Flows

A path from s to t in the residual graph R is an augmenting flow $f_{augment}$. The minimum edge capacity from all edges along this path is the augmenting flow value $k(f_{augment})$ of R. This is then added to f.

$$f = f + f_{augment}$$

This updates f with the new augmenting flow.

2.1.4 Iterative Augmentation

The process of generating a residual graph from f and finding augmenting flows and updating the solution f iteratively improves the max-flow solution. This suggests that each iteration finds a new path with which to push flow from s to t. Thus, the process of creating residual graphs and adding augmenting flows is repeated until no path from s to t can be found by depth first search, at which point the flow f is the max-flow of the flow network G. Such a terminating condition is guaranteed by virtue of the fact that max-flow is finite, and each iteration of augmentation increases f by an integer amount $/geq$ 1.

2.2 Max-flow Min-cut Theorem

When Ford-Fulkeron ends, a cut C is defined as the set A of vertices reachable from s, and $\bar{A}$, the set of vertices unreachable from s. Such a cut is guaranteed to be found, since a flow is finite and flow is augmented by an integer ≥ 1 (as mentioned above).

The capacity of this cut, $k(C)$, is the sum of capacities of all edges with exactly one end in A.

$$k(C) = \sum_{e:u\in A, v\in \bar{A}} C(e)$$

Because max-flow is finite, and augmenting flows strictly increase flow f by an integer amount, a terminating flow $f_{\text{terminating}}$ is guaranteed, thus also guaranteeing the cut C. From the construction of the residual graph R at Ford-Fulkerson's termination, $f_{\text{terminating}} = k(C)$.

Since any flow, $f_{\text{terminating}}$ included, is at most min-cut, and min-cut must be at least $k(C)$. Combining these, we have

$$\text{min-cut} \leq k(C) \ = f_{\text{terminating}} \leq \text{min-cut}.$$

Therefore, $f_{\text{terminating}}$ is the max-flow, $k(C)$ is the min-cut, and we have shown that

$$\text{max-flow} = \text{min-cut}.$$

3 Multicommodity Flow

A *multicommodity flow network* is a graph $G = (V, E)$ with

- pairs of vertices $s_i, t_i \in V$, each representing a source and sink for a commodity i
- a demand D_i for each commodity
- a capacity function C on the edges of G such that

$$\sum_{\text{all } i} f_i(e) \leq c_e.$$

Let $f \in [0, 1]$. Then we define the *concurrent max-flow* of a multicommodity flow network to be the largest value of f such that we can route fD_i units of commodity i from s_i to t_i simultaneously for each i. Let $U \subseteq V$ and let $E_U \subseteq E$ be the edges with exactly one endpoint in U. Then we define the *min-cut* of a multicommodity flow network to be

$$\min \left(\frac{\sum_{e\in E_U} c_e}{\sum_{\text{exactly one of } s_i, t_i \in U} D_i} \right).$$

Intuitively, the quantity we are minimizing is the total amount of flow that can cross the cut divided by the amount we would like to be able to route across the cut. If we restrict to a single commodity, this agrees with the usual definition of a min-cut.

We will initially restrict ourselves to a subclass of problems known as *Uniform Multicommodity Flow Problems* (UMFP). In these problems we assume that there is a commodity that needs to be transferred between every pair of vertices (regardless of whether they are adjacent) and that the demand for every commodity is the same (so we may assume it is one). There are additional no restrictions on the underlying graph or the edge capacity function for a UMFP. For a UMFP, the demand across a cut is the number of different pairs that can be formed by taking one vertex in U and one vertex in $V - U = \overline{U}$. This number is the product $|U||\overline{U}|$. For an arbitrary cut given by the pair $(U, \overline{U})$, we will refer to the quantity

$$\frac{\sum_{e \in E_U} c_e}{|U||\overline{U}|}$$

as the ratio cost of the cut.

For single commodity flow problems, we know that the max-flow is equal to the min-cut. In this generalization, the concurrent max-flow is still bounded above by the min-cut, but they are no longer necessarily equal. This is demonstrated by the following example:

Let G be a 3-regular graph on n vertices with unit edge capacities for which the demand across any cut, $(U, \overline{U})$ for $U \subseteq V$ is at least $c \min\{|U|, |\overline{U}|\}$ for some constant $c > 0$. For a uniform multicommodity flow problem on such a graph, this property makes sure that the number of edges in any given cut is greater than a constant proportion of the number of vertices in the smaller side of the cut.

By definition of min-cut, which we will denote M, and the fact that we have a UMFP,

$$M = \min\left(\frac{\sum_{e \in E_U} c_e}{|U||\overline{U}|}\right).$$

Applying the relationship we have between the number of edges in a cut and the number of vertices in the two sections of the partition, we have

$$M \geq \min\left(\frac{c \min\{|U|, |\overline{U}|\}}{|U||\overline{U}|}\right).$$

This is equivalent to

$$M \geq \min\left(\frac{c}{\max\{\|U|, |\overline{U}|\}}\right).$$

The largest possible proper subset U of V has $n - 1$ elements, so this is just

$$M = \frac{c}{n-1}.$$

Now that we have found the size of the min-cut, we turn our attention to the max-flow of the network.There are at most $n/2$ nodes within distance $\log(n) - 3$ of any v in V. To see this, note that there are 3 vertices at distance 1 from v and that each of these vertices has two other neighbors. This means that at a distance 2 there are at most $3 * 2$ vertices (fewer if two vertices at the same distance happen to be adjacent). For each step we move away from v after the first, we reach at most a factor of two more vertices. After $\log(n) - 3$ steps we have thus reached at most

$$3 * 2^{\log(n)-4} = \frac{3n}{2^4} < \frac{n}{2}$$

vertices.

This is true for each vertex v in the graph, so at least half of the $\binom{n}{2}$ possible pairs of vertices must be at a distance at least $\log(n) - 2$. If we can route a fraction f of the demand between two such vertices, it must take up a total capcity of at least $\log(n) - 2$. Thus the pairs that are a distance at least $\log(n) - 2$ take up a total capacity of at least

$$\binom{n}{2}\frac{f(\log(n) - 2)}{2}$$

when we are able to route a fraction f of the demand.

Since the total capacity of a 3-regular graph with unit capacities is $3n/2$. Using this total capacity to bound the capacity used by vertices that are far apart, we get

$$\binom{n}{2}\frac{f(\log(n) - 2)}{2} \leq 3n/2.$$

Plugging in the $\frac{n(n-1)}{2}$ for $\binom{n}{2}$ and isolating f gives

$$f \leq \frac{6}{(n-1)}(\log(n) - 2).$$

We know that the min cut $M = \frac{c}{n-1}$, so we have

$$f \leq \frac{6M}{c(\log(n) - 2)} = O\left(M/\log(n)\right).$$

This means that the concurrent flow value is at least an $O(\log(n))$ factor smaller than the min-cut in this example.

In the following theorem, Leighton and Rao show that this is essentially the worst it can possibly get.

Theorem 1 (Leighton, Rao(1999))**.** *For any uniform multicommodity flow problem with n vertices,*

$$\Omega\left(\frac{MINCUT}{\log(n)}\right) \leq MAXFLOW \leq MINCUT.$$

The example given earlier shows that there are UMFPs that attain a $\log n$ factor difference between the max-flow and min-cut, so this theorem is sharp. Before we begin the proof of this theorem, we need some more notation. There is already have a capacity function C defined on the edges of the graph. The max-flow question for a multicommodity flow problem can be solved by a linear program that introduces a distance function d on the edges as well. The dual linear program (which can be used to find the min-cut exactly in single commodity problems) introduces a *distance constraint*

$$\sum_{u,v \in V} d(u, v) \geq 1$$

where V is the set of vertices in the network and we are summing over all unordered pairs and reaches an optimal solution when

$$W = \sum_{e \in E} C(e)d(e)$$

is minimized. Since this is the dual of the linear program that finds a max-flow, the optimal value of W will also be the value of the max-flow. W is known as the *total weight* of the distance function and the optimal value of W is denoted f. We will prove the following lemmas in terms of a generic W with the intent of applying them to a distance function that achieves the flow f.

Lemma 1. *For any graph $G = (V, E)$ with an arbitrary capacity function c, any $\Delta > 0$ and any distance function with total weight W, it is possible to partition G into components with radius at most Δ so that the capacity of the edges connecting nodes in different components is at most $4W \log n/\Delta$.*

Proof. Let $C = \sum_{e \in E}$ denote the total capacity on the edges of G.

First consider the case where $\Delta \leq 4W \log n/C$. If we partition the graph so that every vertex is its own component, the total capacity between the different partitions is just C, the total capacity in the graph. The condition on Δ then gives us the desired $C \leq 4W \log n/\Delta$ and the radius of each component is $0 \leq \Delta$ so this partition satisfies the theorem.

Now we may assume $\Delta > 4W \log n/\Delta$. For this case we will construct a graph with a simpler distance function, find a partition of this graph and then use it to obtain a partition of the original graph. Construct the graph G^+ from G by replacing each edge of G by a path of $\lceil Cd(e)/W \rceil$ edges. Let the capacity of each edge in G^+ be the capacity of the corresponding edge in G and let the distance of each edge in G^+ be 1. Note that there are two kinds of vertices in G^+, those that correspond to vertices in G and those introduced on the new paths.

Choose an arbitrary vertex v_1 of G^+ that corresponds to a node in G. For each $i > 0$ define $H_{1,i}$ to be the subgraph of G^+ induced by vertices that can be reached by starting at v and traversing at most i edges. Let $C_0 = 2C/n$ and for $i > 0$ define C_i to be the total capacity of the edges in $H_{1,i}$. Let j denote the smallest value of $i \geq 0$ for which $C_{i+1} < (1 + \epsilon)C_i$ where $\epsilon = W \log n/\Delta C < 1/4$ (by the condition we are assuming on Δ). Note that this condition ensures that the total capacity on the edges between $H_{1,j}$ and the rest of G^+ is no more than ϵ. Furthermore, for large enough i we will eventually reach every vertex in that connected portion of the graph, so that $H_i = H_{i+1}$ and the desired condition will eventually be reached. Let $H_1 = H_{1,j}$.

Remove H_1 from the G^+ and repeat this process on the smaller graph. If one still exists, pick an arbitrarty vertex v_2 corresponding to a vertex in G and let $H_{2,i}$ be the graph induced by vertices of $G^+ - H_1$ at distance at most i from v_2. Use the same condition on i to define H_2. Iterate this process until removing the current collection of H_i from G^+ produces a subgraph with no vertices corresponding to the original vertices in G. This gives us a partition of the vertices in G^+.

Let C^+ denote the total initial capacity of G^+. Each edge e of G was expanded to a path of length $\lceil Cd(e)/W \rceil$ and each edge on that path was assigned the same capacity as e originally had. This means that

$$C^+ = \sum_{e \in E} C(e) \left\lceil \frac{Cd(e)}{W} \right\rceil.$$

Note that $\lceil x \rceil \leq x + 1$. Applying this, distibuting $C(e)$ and breaking up the sum gives

$$C^+ \leq \sum_{e \in E} C(e) + \frac{C}{W} \sum_{e \in E} C(e)d(e).$$

Applying the definitions of W and C, this means that $C^+ \leq 2C$.

We noted earlier that the capacity of the edges leaving component H_i is at most ϵ times the capacity of the edges in that component, or $\epsilon(2C/n)$ if the component has only one vertex. Furthermore, the total capacity of edges between components will be at it's largest if each H_i only contains a single vertex from G. Combining these two observations, we see that the total capacity leaving the components of G^+ is at most

$$\epsilon \left(C^+ + n\frac{2C}{n} \right) \leq \epsilon(2C) + 2\epsilon C = 4\epsilon C = \frac{4W \log n}{\Delta}.$$

Every vertex of G corresponds to a vertex in one of the H_i, so the H_i induce a partition of the vertices of G. Consider the resulting components. Any edge e that connects two of these components in G must correspond to a path of edges of capacity $C(e)$ that connects two components in G^+. This means that the capacity of the edges between the components of G is at most the capacity of the edges leaving components of G^+. That is, the capacity of the edges linking components of G is at most $\frac{4W \log n}{\Delta}$ as desired.

Consider any single component H_i in G^+ and suppose it has radius j. If $j = 0$ then we know the cooresponding component in G also has radius 0 and we are done. Otherwise, $j > 0$ and so, by construction, H_j must have total capacity at least $(1+\epsilon)^j(2C/n)$. The capacity in this single component cannot be larger than the total capacity of G^+, so we have

$$(1+\epsilon)^j(2C/n) \leq 2C.$$

Isolating j, we see that $j \leq \log(n)/\log(1+\epsilon)$. We noted earlier that $\epsilon < 1/4$, so $\log(1+\epsilon) > \epsilon$ (To see this, consider the function $\log(1+x) - x$. It is zero at 0 and 1 and analyzing its derivative shows that it is otherwise positive in that interval). This means that

$$j \leq \frac{\log n}{\epsilon}.$$

Now, any path of length l in G^+ must correspond to a path of length at most Wl/C in G. Thus if the radius of H_i is j, the radius of the corresponding component in G must be at most

$$Wj/C \leq \frac{W \log n}{C\epsilon} = \Delta.$$

Thus we have found a partition of the vertices of G where the radius of each component is at most Δ and the total capacity on the edges between the components is at most $4W \log n/\Delta$

□

Corollary 1. *For any graph G and any distance function with total weight W, we can either*

1. *find a component with radius at most $\frac{1}{2n^2}$ that contains at least 2/3 of the vertices in G or*
2. *find a cut of G with ratio cost $O(W \log n)$.*

Proof. Let $\Delta = 1/2n^2$. By Lemma 1, we can find a partition of the vertices of G so that each component has radius at most $1/2n^2$ and the edges between components have total capacity at most $4W \log n/\Delta = 8Wn^2 \log n$. If one of these components have $2n/3$ vertices, we are done. Otherwise we can find a way to divide the components into two sets so that each set has at least $n/3$ vertices. The edges between these two sets have capacity at most the capacity between the sets of the partition, or $8Wn^2 \log n$. Since both sides of the cut have at least $n/3$ vertices, the demand between the two sets is at least $(2n/3)(n/3)$. Thus the ratio cost of the cut is at most

$$\frac{8Wn^2 \log n}{(2n/3)(n/3)} = 36W \log(n) = O(W \log n)$$

as desired. □

Lemma 2. *For any graph G, if there is a distance function d with total weight W and a subset of vertices $T \subseteq V$ with $|T| \geq 2n/3$ and*

$$\sum_{u \in V-T} d(T,u) \geq \frac{1}{2n},$$

then we can find a cut with ratio cost $O(W)$.

Proof. Start by constructing the graph G^+ as in Lemma 1. Let G_i^+ denote the graph induced by vertices that can be reached by starting from a vertex corresponding to a vertex in T and traveling along at most i edges. Define V_i to be the set of nodes of G that correspond to nodes in G_i^+. Let $n_i = |V - V_i|$ denote the the number of vertices at a distance more than i from T. Let R_i denote the ratio cost of the cut given by $V_i, V - V_i$ and $R = \min R_i$.

Since n_i nodes are at a distance at least $i+1$ from T in G^+ for all i, we know that

$$\sum_{u \in V-T} d_{G^+}(T,u) = \sum_{i \geq 0} n_i.$$

By the construction of G^+, any path of lenth l corresponds to a path of length at least C/W in G, so we have $d_{G^+}(T,u) \geq (C/W) d_G(T,u)$. Combining these and the condition we put on the distance function in the hypothesis of the theorem gives

$$\sum_{i \geq 0} \geq \frac{C}{W} \sum_{u \in V-T} d_G(T,u) \geq \frac{C}{2nW}.$$

Since $|T| \geq 2n/3$ and $T \subseteq V_i$, the demand accross the cut given by $V_i, V - V_i$ in G is at least $n_i(2n/3)$. We defined R_i to be the cost ratio of this cut and $R \leq R_i$, so the capacity of the cut is at least $2nRn_i/3$. The capacity of the corresponding cut in G_i^+ is also at least this amount. If we recall that the total capacity in G^+ is at most $2C$, we can bound the sum over i of the capacity between the cuts by

$$\sum_{i \geq 0} \frac{2nRn_i}{3} \leq 2C$$

which implies that

$$R \leq \frac{3C}{n \sum_{i \geq 0} n_i}.$$

Applying our bound on the sum of the n_i gives

$$R \leq 6W = O(W),$$

so we have shown that for at least one i, the cost ratio of the corresponding cut is of size at most $O(W)$ as desired.

□

Lemma 3. *Given a graph G and a distance function with total weight W that satisfies the distance constraint imposed by the dual of the max-cut linear program, we can find a cut with ratio cost $O(W \log n)$.*

Proof. Use Lemma 1 with $\Delta = 1/2n^2$ to partition the graph. By Corollary 1 we can either find a cut with cost ratio $O(W \log n)$, in which case we are done, or we can find a component T with radius at most $1/2n^2$ that contains at least $2n/3$ vertices.

Now we will show that the distance funtion satisfies the hypothesis of Lemma 2. Since T has radius $1/2n^2$, the distance between any two vertices of T is at most $1/n^2$. This means that the distance between any two vertices u, v in the graph is at most the distance followed if we start at u, move to its nearest neighbor in T, move to v's nearest neighbor in T and then move back out to v. Symbolically, this is

$$d(u, v) \leq d(T, u) + d(T, v) + 1/n^2.$$

Now, since we know our distance function satisisfies the distance constraint, we have

$$\begin{aligned} 1 \leq \sum_{\{u,v\} \in V} d(u, v) &= \frac{1}{2} \sum_{(u,v) \in V} d(u, v) \\ &\leq \frac{1}{2} \sum_{(u,v) \in V} \left(d(T, u) + d(T, v) + 1/n^2 \right) \\ &< n \sum u \in V - T d(T, u) + 1/2. \end{aligned}$$

This gives

$$\sum u \in V - T d(T, u) > 1/2n$$

so we can indeed apply Lemma 2. Applying Lemma 2 gives us a cut with ratio cost $O(W)$, which is certainly $O(W \log n)$. □

Leighton and Rao's theorem now follows from Lemma 3 if we start with a distance function obtained by finding an optimal solution to the dual of the linear programming formulation of the max-flow problem. Such a function has total weight equal to the max-flow and certainly satisfies the hypotheses, so we get a cut with ratio cost $O(\text{MAXFLOW} \log n)$.

In their paper, Leighton and Rao go on to extend this theorem to product multicommodity flow problems (PMFP's). In these problems there is a function π on the vertices and the demand between two vertices u, v is given by $\pi(u)\pi(v)$. When π is 1 on all inputs, a UMFP is recovered. The key ideas in the proof of this more general theorem are similar to the ideas in the proof we explain in these notes, but the details are rather messier.

3.1 Applications to Approximation Algorithms

While multicommodity max-flow can be solved in polynomial time through a linear programming model, finding the multicommodity min-cut, also known as the *sparsest cut* of a graph is NP-hard (Matula and Shahrokhi, 1986). The proofs of the lemmas used to prove the multicommodity max-flow, min-cut theorem in these notes can be adapted into polynomial time algorithms that will produce a cut with ratio cost $O(f \log n)$. This means that these methods give an $O(\log n)$ approximation for sparsest cut. This approximation algorithm for sparsest cut can then be extended to approximation algorithms for a wide variety of other graph partitioning problems.

These approximation algorithms for graph partitioning problems lead to approximation algorithms for a whole host of other problems NP-hard problems that can be approached through divide and conquer methods. These include uniprocessor scheduling and VLSI layout problems in addition to a varity of graph embedding problems.