18.415/6.854 Advanced Algorithms September 10, 2008

Lecture 3

Lecturer: Michel X. Goemans

1 Introduction

Today we continue our discussion of maximum flows by introducing the fattest path augmenting algorithm, an improvement over the Ford-Fulkerson algorithm, to solve the max flow problem. We also discuss the minimum cost circulation problem.

2 Maximum Flow

In a maximum flow problem, the goal is to find the greatest rate (flow) at which material can be sent from a source s to a sink t. Several problems can be modeled as a max-flow problem, including bipartite matching, which will be discussed today. We will also discuss flow decomposition and the fattest augmenting path algorithm.

2.1 Maximum Cardinality Matching in Bipartite Graphs

A *bipartite graph* is a graph $G = (V, E)$ whose vertex set V can be partitioned into two disjoint sets, A and B, such that every edge connects a vertex in A to one in B. A *matching* M is a subset of E such that the endpoints of all the edges in M are distinct. In other words, two edges in M cannot share a vertex. We are interested in solving the following problem: Given an undirected bipartite graph $G = (V, E)$ where $V = A \cup B$, find a matching M of maximum cardinality.

We can formulate this maximum cardinality matching problem as a max-flow problem. To do that, consider the network shown in Figure 1.

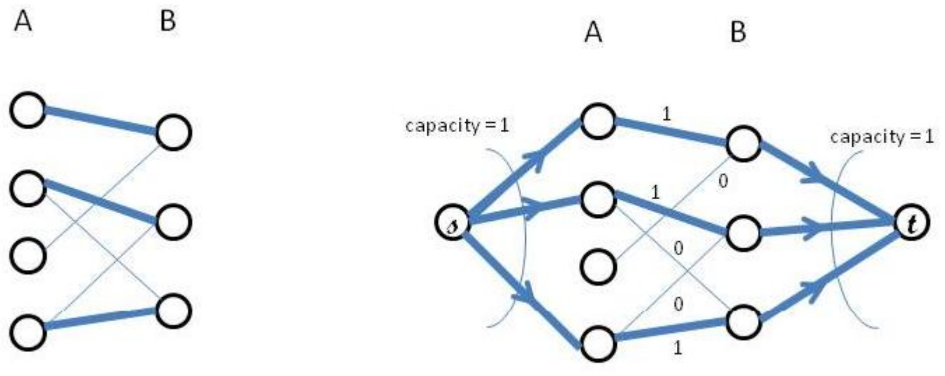

Figure 1: The figure on the left represents a matching in a bipartite graph. The figure on the right shows how the bipartite graph can be converted into a max-flow network by imposing a capacity of 1 on arcs out of s and into t.

The network is constructed as follows: We orient each edge in G from A to B and assign them a capacity of 1 (any capacity greater than 1 works too). We also add two new vertices, s and t, and arcs from s to every vertex in A, and from every vertex in B to t. All the new arcs are given unit capacity.

Theorem 1 *Let $G = (V, E)$ be a bipartite graph with vertex partition $V = A \cup B$, and let $G' = (V', E')$ be the capacitated network constructed as above. If M is a matching in G, then there is an integer-valued flow f in G' with value $|f| = |M|$. Conversely, if f is an integer-valued flow in G', then there is a matching M in G with cardinality $|M| = |f|$.*

Proof: Given M, define a flow f in G' as follows: if $(u, v) \in M$, then set $f(s, u) = f(u, v) = f(v, t) = 1$ and $f(u, s) = f(v, u) = f(t, v) = -1$. For all other edges $(u, v) \in E'$, let $f(u, v) = 0$. Each edge $(u, v) \in M$ corresponds to 1 unit of flow in G' that traverses the path $s \to u \to v \to t$. The paths in M have distinct vertices, aside from s and t. The net flow across the cut $(A \cup s : B \cup t)$ is equal to $|M|$. We know that the net flow across any cut is the same, and equals the value of the flow. Thus, we can conclude that $|M| = |f|$. To prove the converse, let f be an integer-valued flow in G'. By flow conservation and the choice of capacities, the net flow in each arc must be -1, 0 or 1. Let M be the set of edges (u, v), with $u \in A$, $v \in B$ for which $f(u, v) = 1$. It is easy to see, by flow conservation again, that M is indeed a matching and, using the same argument as before, that $|M| = |f|$. □

Since all the capacities of this maximum flow problem are integer valued, we know that there always exists an *integer-valued* maximum flow, and therefore the theorem shows that this maximum flow formulation correctly models the maximum cardinality bipartite matching.

2.2 Flow Decomposition

In an (raw) s-t flow, we have the following building blocks:

- Unit flow on an s-t directed path.
- Unit flow on a directed cycle.

Any (raw) s-t flow can be written as a linear combination of these building blocks.

Theorem 2 *Any (raw) s-t flow r can be decomposed into at most m flows along either paths from s to t or cycles, where m is the number of edges in the network. More precisely, it can be decomposed into at most $|\{e : r(e) > 0\}| \leq m$ paths and cycles.*

Proof: By tracing back the flow on an edge e and tracing forward the flow on e, we either get an s-t path T, or a cycle T with $r(e) > 0$ for all $e \in T$. Denote the min flow on T by $\Delta(T)$:

$$\Delta(T) = \min_{e \in T} r(e).$$

We want to decrease the flow on T such that at least one edge goes to 0 (by subtracting out $\Delta(T)$), and keep doing that until there are no more edges with non-zero flows. More precisely, the following algorithm extracts at most m paths and cycles.

(i) While there is a directed cycle C with positive flow:

  (a) Decrease the flow on this cycle by $\Delta(C)$

  (b) Add this cycle as an element of the flow decomposition

(ii) (The set of arcs with positive flow now form an acyclic graph.) While there is a path P from s to t with positive flow:

(a) Decrease the flow on this path by $\Delta(P)$.

(b) Add this path as an element of the flow decomposition.

Each time we decrease the flow on a path or a cycle T, we zero out the flow on some edge. When we do this, the new raw flow is $r^{\text{new}}(e) = r(e) - \Delta(T)$ if $e \in T$, or $r(e)$ otherwise. Since there are $|\{e : r(e) > 0\}| \leq m$ edges with positive flow in the graph, there will be at most that number of decreases in the flow, and consequently, at most that number of paths or cycles in the flow decomposition. $\square$

2.3 Fattest Augmenting Path Algorithm (Edmonds-Karp '72)

Flow decomposition is a key tool in the analysis of network flow algorithms, as we will illustrate now.

As we saw in the last lecture, the Ford-Fulkerson algorithm for finding a maximum flow in a network may take exponential time, or even not terminate at all, if the augmenting path is not chosen appropriately. We proposed two specific choices of augmenting paths, both due to Edmonds and Karp, that provide a polynomial running time. One was the shortest augmenting path, the other was the *fattest* augmenting path or *maximum-capacity augmenting path*: the augmenting path that increases the flow the most. This is the variant we analyze now.

For an augmenting s-t path $P \in G_f$, define

$$\varepsilon(P) = \min_{(v,w)\in P} u_f(v,w)$$

where the u_f are the residual capacities. The minimum residual capacity $\varepsilon(P)$ (the bottleneck) is the maximum flow that can be pushed along the path P. We wish to find the fattest augmenting path P such that $\varepsilon(P)$ is maximized. The fattest augmenting path P can be efficiently found with Dijkstra's algorithm in $O(m + n \log n)$ time [1].

Theorem 3 *Assuming that capacities are* integral *and bounded by U, the optimal flow for a network can be found in $O(m \log(mU)) = O(m \log(nU))$ iterations of augmenting along the fattest path.*

Proof: Start with a zero flow, $f = 0$. Consider a maximum flow f^*. Its value is at most the value of any cut, which is bounded by mU:

$$|f^*| \leq mU.$$

Consider the flow $f^* - f$ (this is, $f^*(e) - f(e)$ for all edges e) in the residual graph G_f with residual capacities $u_f = u - f$.

We can decompose $f^* - f$ into $\leq m$ flows using flow decomposition. As a result, at least one of these paths carry a flow of value at least $\frac{1}{m}(|f^*| - |f|)$. Suppose now that we push $\varepsilon(P)$ units of flow along the fattest path in the residual graph G_f and obtain a new flow f^{new} of value:

$$|f^{\text{new}}| = |f| + \varepsilon(P).$$

Since the fattest path provides the greatest increase in flow value, we must have that $\varepsilon(P) \geq \frac{1}{m}(|f^*| - |f|)$. Thus we have the following inequality

$$|f^{\text{new}}| \geq |f| + \frac{1}{m}(|f^*| - |f|),$$

[1] Actually, it can be found in $O(m)$ time under the condition that we have the capacities sorted beforehand, see the forthcoming problem set.

which implies

$$\begin{aligned} |f^*| - |f^{\text{new}}| &= |f^*| - |f| + |f| - |f^{\text{new}}| \\ &\leq \left(1 - \frac{1}{m}\right)(|f^*| - |f|)\,. \end{aligned}$$

After k iterations, we get a flow $\hat{f}$ such that

$$|f^*| - |\hat{f}| \leq \left(1 - \frac{1}{m}\right)^k mU.$$

Eventually $|f^*| - |\hat{f}| < 1$ which implies $f^* = \hat{f}$ since, for integral capacities, all intermediate flows will be integral. Since $(1 - \frac{1}{m})^m \leq \frac{1}{e}$ for all $m \geq 2$, the number of iterations required for the difference to go below 1 is

$$k = m \log(mU).$$

□

Combining the results mentioned above we have the following corollary.

Corollary 4 *We can find a maximum flow in an integer-capacitated network with maximum capacity U in $O((m + n \log n) m \log(nU))$ time* [2].

3 Minimum Cost Circulation Problem (MCCP)

A *circulation* is simply a flow where the net flow into every vertex (there are no sources or sinks) is zero. Notice that we can easily transform an $s - t$ flow to a circulation by adding one arc from t to s (with infinite capacity) which carries a flow equal to the $s - t$ flow value.

Definition 1 *A circulation f satisfies*

(i) ***Skew-Symmetry****: $\forall\,(v, w) \in E,\ f(v, w) = -f(w, v)$.*

(ii) ***Flow Conservation****: $\forall\, v \in V,\ \sum_w f(v, w) = 0$.*

(iii) ***Capacity Constraints****: $\forall\,(v, w) \in E,\ f(v, w) \leq u(v, w)$.*

Definition 2 *A cost function $c : E \mapsto \mathbb{R}$ assigns a cost per unit flow to each edge. We assume the cost function satisfies skew symmetry: $c(v, w) = -c(w, v)$. For a set of edges C (e.g. a cycle), we denote the total cost of C by :*

$$c(C) = \sum_{(v,w) \in C} c(v, w).$$

Definition 3 *The goal of the Minimum Cost Circulation Problem (MCCP) is to find a circulation f of minimum cost $c(f)$ where*

$$c(f) = \sum_{(v,w)} c(v, w) f(v, w).$$

The MCCP is a special case of a Linear Programming (LP) problem (an optimization problem with linear constraints and a linear objective function). But while no strongly polynomial time algorithms are known for linear programming, we will be able to find one for MCCP.

[2] Using the previous footnote, we can do this in $O(m^2 \log(nU))$ time.

3.1 Vertex Potentials

Before we can solve MCCP, it is necessary to introduce the concept of *vertex potentials*, or simply *potentials.*

Definition 4 *A vertex potential is a function $p : V \mapsto \mathbb{R}$ that assigns each vertex a potential. The vertex potential defines a reduced cost function c_p such that*

$$c_p(v, w) = c(v, w) + p(v) - p(w).$$

Proposition 5 *The function c_p satisfies the following properties:*

(i) ***Skew-Symmetry****: $c_p(v, w) = -c_p(w, v)$.*

(ii) ***Cycle Equivalence****: for a cycle C, $c(C) = c_p(C)$; i.e., the reduced cost function agrees with the cost function.*

(iii) ***Circulation Equivalence****: for all circulations, the reduced cost function agrees with the cost function, $c(f) = c_p(f)$.*

Proof: The first property is trivial. The second property follows since all the potential terms cancel out. And we'll prove the third property. By definition

$$\begin{aligned} c_p(f) &= \sum_{(v,w)} (c(v, w) + p(v) - p(w))(f(v, w)) \\ &= c(f) + \sum_v p(v) \sum_{w:(v,w)\in E} f(v, w) - \sum_w p(w) \sum_{v:(w,v)\in E} f(v, w). \end{aligned}$$

Now by flow conservation, the inner sums are all zero. Hence $c_p(f) = c(f)$. (The third property also follows easily from flow decomposition, as the decomposition of a circulation only contains cycles and thus the cost and the reduced cost of a circulation are the same because of (ii).) □

3.2 Klein's Cycle-Cancelling Algorithm

We present a pseudo-algorithm for removing negative-cost cycles. While there exists a negative-cost cycle C in G_f, push a flow ε along the cycle C, where ε is the minimum residual flow:

$$\varepsilon = \min_{(v,w)\in C} u_f(v, w).$$

Of course, this doesn't lead to a straight-forward implementation, since we haven't specified which negative-cost cycle to select or how to find them. We should also consider whether the algorithm is efficient and whether it will terminate. We'll answer these questions in the next lecture. However, we will show now that if it terminates, then the circulation output is of minimum cost.

3.3 Optimality Conditions

We now present a theorem that specifies the conditions required for f to be a minimum cost circulation.

Theorem 6 (Optimality Condition) *Let f be a circulation. The following are equivalent:*

(i) f is of minimum cost.

(ii) There exists no negative-cost cycle in the residual graph G_f.

(iii) There exists a potential function p such that for all $(v, w) \in E_f$, $c_p(v, w) \geq 0$.

Proof: To show that (i) implies (ii), we'll prove the contrapositive. Suppose there exists a negative cost cycle C in the residual graph G_f where f is the optimal circulation. Denote by C' the reverse cycle (i.e. following the arcs in the reverse order). We define a new circulation f' for any edge e as follows. If $e \in C$, $f'(e) = f(e) + \varepsilon$. And if $e \in C'$, then $f'(e) = f(e) - \varepsilon$. Otherwise, let $f'(e) = f(e)$.

Then we compute the cost of this new flow as

$$\begin{aligned} c(f') &= c(f) + (\varepsilon)(c(C)) + (-\varepsilon)(-c(C)) \\ &= c(f) + 2\varepsilon c(C) \\ &< c(f), \end{aligned}$$

where the last step follows since C is a negative cost cycle. Thus we've shown that f is indeed not optimal. Hence (i) implies (ii).

Now we show that (ii) implies (iii). Add zero-cost (or of arbitrary cost) arcs from a new vertex s to every vertex in G_f (this is to make sure that s can reach every vertex in V). Define a potential p such that $p(v)$ is the length of the shortest simple path from s to v. Then, since there are no negative cost cycle, we have the optimality conditions for the shortest-path lengths:

$$p(w) \leq p(v) + c(v, w) \ \forall\, (v, w) \in E_f,$$

as one way to go from s to w is to go to v by a shortest path and then go directly to w.

Here, we have implicitly used the fact that G_f has no negative cost cycles. For if the shortest path from s to v already goes through w then adding (v, w), we create a cycle C (and the resulting path is not simple). However, this cycle can't be of negative cost by assumption. Thus, by removing it, we obtain a simple path to w of cost less or equal to $p(v) + c(v, w)$. Rearranging the inequality gives the desired result

$$c_p(v, w) \geq 0 \ \forall\, (v, w) \in E_f.$$

Now we prove that (iii) implies (i) by showing the contrapositive. Suppose we have an optimal circulation f^* and a suboptimal one f: $c(f^*) < c(f)$. Consider the cost of the circulation $f^* - f$:

$$\begin{aligned} c(f^* - f) &= c_p(f^* - f) \\ &= \sum_{(v,w) \in E} c_p(v, w)[f^*(v, w) - f(v, w)] \\ &= 2 \sum_{(v,w): f^* - f > 0} c_p(v, w)[f^*(v, w) - f(v, w)] \\ &\geq 0 \end{aligned}$$

by (iii). Note that in the second to last step, we utilized the skew-symmetry of the cost of reverse arcs (with flows of opposite parity). But since f^* is supposed to be strictly better than f, we have a contradiction. □

References

[EK72] Jack Edmonds, and Richard M. Karp, *Theoretical improvements in algorithmic efficiency for network flow problems*, Journal of the ACM 19 (2): 248–264, 1972.

[Klein67] Klein, M. *A primal method for minimum cost flows with application to the assignment and transportation problem.* Management Science 14: 205-220, 1967.