6.854 Advanced Algorithms
Lecture 8: 10/1/2003 Lecturer: Erik Demaine, David Karger
Scribes: Tudor Leu

Blocking flows (continued). Minimum cost maximum flow. Minimum cost circulation

8.1 Blocking flows

8.1.1 Capacitated graphs

We are going to discuss about extending the use of the blocking flow algorithm from unit graphs to graphs with integral capacities. This can be achieved by scaling, as discussed in the previous lecture: start with a zero-capacity network and build up to the actual capacities one bit at a time.

The running time of the blocking flow algorithm is given by the number of advances, augmentation and retreats done on edges in each stage. We had obtained an $O(m)$ running time bound per stage for unit graphs. However, the analysis for unit graphs is not functional in the case of larger capacities, since augmentation paths may contain edges with capacities greater than 1. Those edges could be used for more than one augmenting path, and therefore they cannot be deleted immediately after augmentation, as it was the case in the unit graph.

A new analysis of the running time starts from two observations. First, there are at most n steps for each blocking flow phase in the scaling algorithm. This is because at each step the distance from source to sink in the residual graph increases by 1, and that distance cannot grow to more than n. The other observation is that each new scaling phase can create at most m units of flow in the residual graph, since at most 1 unit of capacity is added to each edge.

We start by analyzing the cost of retreats in the blocking flow stage. At most m retreats occur in one step of a stage, so overall $O(mn)$ retreats occur the entire blocking stage.

One augment operation costs at most n, since the augmentation is performed along a path in the residual graph and the path is at most n long. Since each scaling phase creates at most m units of flow in the residual graph, there are at most m augment operations in a phase. Therefore, all augments take $O(mn)$.

The last type of operations to be analyzed is advances. Each advance results in either an augment or a retreat, therefore advances also take $O(mn)$ time over a blocking flow stage.

Since there are $\log U$ scaling phases, the total running time of the algorithm is $O(mn \log U)$.

8.1.2 A strongly polynomial blocking flow algorithm

Each time an augmenting path is found, one edge on that path uses up all its capacity for the new incremental flow. That edge disappears in the residual graph, and the augmenting path breaks into two paths. The last part goes all the way to the sink and could be used as the second part of another augmenting path. We would like to not traverse the entire path in order to update the residual capacities, and we would like to use the resulted path segments in the search for new augmenting paths. The path segments can combine and form trees, as shown in the figure below.

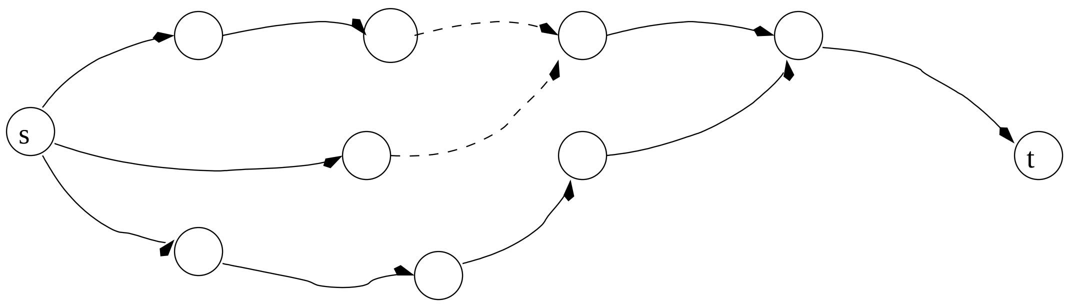

Figure 8.1: Augmenting paths which were updated and a new augmenting path which links to an old one.

Tarjan and Slater built a data structure named *dynamic trees* which maintains a set of trees and can be used to perform in $O(\log n)$ the following operations: link two trees, break a tree in two parts, find the minimum edge on a path, decrease the capacities on a path and find the root of a tree.

The blocking flow algorithm is changed then to take advantage of the new data structure. As before, we search for augmenting paths to the sink. The search is performed on unexplored edges, but whenever we reach a tree, we jump to its root in order to get closer to the sink. Retreats work as before: if stuck, go back and remove the edge. Augmentations are changed: first, the minimum edge is found, then the capacities are decreased and that edge is removed, breaking the current path in two parts.

There are at most m retreats, since each edge allows for only one retreat. Also, there are $O(m)$ augments, since each of them removes an edge. Then each step in the blocking step algorithm takes $O(m \log n)$. and since there are at most n steps (we argued this in the previous section), the total running time of our algorithm is $O(mn \log n)$. This time can be improved to $O(mn \log_{m/n} n)$.

8.2 Costs in flow networks

8.2.1 Introduction. Minimum cost maximum flow

We previously discussed maximum flow in a network. Today we add one parameter to a flow network, a cost per unit of flow on each edge: $c(v, w) \in \mathbf{R}$, where $(v, w) \in E$.

Definition 1 *The* **cost of a flow** f *is defined as:*

$$c(f) = \sum_{e \in E} f(e) \cdot c(e)$$

A **minimum cost maximum flow** of a network $G = (V, E)$ is a maximum flow with the smallest possible cost. This problem combines maximum flow (getting as much flow as possible from the source to the sink) with shortest path (reaching from the source to the sink with minimum cost).

Note that in a network with costs the residual edges also have costs. Consider an edge (v, w) with capacity $u(v, w)$, cost per unit flow $c(v, w)$ and net flow $f(v, w)$. Then the residual graph has two edges corresponding to (v, w). The first edge is (v, w) with capacity $u(v, w) - f(v, w)$ and cost $c(v, w)$, and second edge is (w, v) with capacity $f(v, w)$ and cost $-c(v, w)$.

Observation 1 *Any flow can be decomposed into paths (some of which can be cycles). We define the cost of a path p as $c(p) = \sum_{e \in p} c(e)$ and express the cost of a flow f as $c(f) = \sum_{p \in P} c(p)f(p)$, where P is the path decomposition of f.*

8.2.2 The min-cost circulation problem

Consider a network without a source or a sink. We can define a flow in this network, as long as it is balanced at every node in that network. This kind of flow is called a **circulation**. The cost of a circulation is defined identically with the cost of a flow.

Observation 2 *Any circulation can be decomposed entirely into cycles. The cost of a circulation f can be expressed as the sum of the costs of all cycles in a decomposition of f.*

A **minimum cost circulation** is a circulation of the smallest possible cost. Note that there is no restriction on the flow through the network. For example, if all costs are positive, the minimum circulation has no flow on all edges. On the other hand, if there are negative cost cycles in the network, the minimum circulation has negative costs and flow has to exist on the edges of the cycle.

Claim 1 *Finding the minimum cost maximum flow of a network is an equivalent problem with finding the minimum cost circulation.*

Proof: First, we show that min-cost max-flow can be solved using min-cost circulation. Given a network G with a source s and a sink t, add an edge (t, s) to the network such that $u(t, s) = mU$ and $c(t, s) = -(C + 1)n$. The minimum cost circulation in the new graph will use to the maximum the very inexpensive newly added edge. Any path from s to t forms a negative cost cycle together with (t, s), since $-c(t, s)$ is greater than the cost of any such path. This guarantees that we obtain a maximum flow from s to t "included" in the circulation of the new network. Among all maximum flows, this one is also of minimum cost. All the maximum flows use (t, s) at the same capacity, so they use the edge (t, s) at the same cost. This means that the minimum cost circulation has to be minimum cost on the section from s to t, which makes the max-flow also min-cost.

Another reduction from min-cost max-flow to min-cost circulation is to find any maximum flow in the network, regardless of the costs, then find the min-cost circulation in the residual graph. We claim that the resulted flow is a min-cost max-flow. This is because *the difference between two max-flows is a circulation*, and the cost of that difference circulation is the difference between the costs of the two max-flows. Given f, the initial max-flow, and f^*, the resulting maximum flow, $f - f^*$ is a min-cost circulation in the residual network G_f iff f^* is a min-cost max-flow.

The second part of the proof is showing that min-cost circulation reduces to min-cost max-flow. Consider a network G for which we want to find a min-cost circulation. Add a source s and a sink t to the network, without any edges to the rest of the network. The maximum flow in this network is 0, therefore the min-cost max-flow is actually a min-cost circulation.

We conclude then that min-cost max-flow and min-cost circulation are equivalent problems. ■

8.2.3 Optimality criteria

We are interested to find criteria which determine whether a circulation is a min-cost circulation.

Theorem 1 *A circulation is* **optimal (min-cost)** *iff there are no negative cost cycles in the residual network.*

Proof: First, suppose that a circulation f is not optimal, and let $f*$ be an optimal circulation in a network G. We will show that G_f has a negative cost cycle. The difference $f^* - f$ is a circulation, therefore it has a cycle decomposition. Because the cost of f^* is smaller than the cost of f, $f^* - f$ is a circulation of negative cost, and it is also feasible in G_f. At least one of the cycles in the decomposition has to be negative, therefore G_f contains a negative cost cycle.

To prove the other implication, suppose a residual network G_f has a negative cycle. Then f is not a min-cost circulation, because that cycle can be added to f, forming a new circulation of smaller cost. ■

The optimality criteria is checked by looking for negative cost cycles. This can be done with the Bellman-Ford shortest-path algorithm, which can handle negative cost edges (unlike Dijkstra's algorithm), but runs in $O(mn)$.

Price function

We can analyze the optimality of a circulation using a price function. Think of the flow units as widgets that are given away at the source and they are paid for at the source. There is a market for widgets at intermediate vertices.

We can define then a price function p for the vertices of the network. At the source, $p(s) = 0$. Consider an edge (v, w) which has residual capacity. The price $p(w)$ is *feasible* if $p(w) \leq p(w) + c(v, w)$.

Definition 2 *The* **reduced cost** *of an edge* (v, w) *is* $c_p(v, w) = c(v, w) + p(v) - p(w)$.

We can think of the reduced cost as the cost of buying a widget at v, shipping it to w and selling it there.

Using this definition, we can say that a price function is feasible for a residual graph if no residual edge has a negative reduced cost.

Observation 3 *Prices do not affect the value or structure of a minimum cost circulation.*

As we discussed before, a circulation is decomposed into cycles. Cycle costs do not change if we compute them as the sum of reduced costs of the edges, since the price terms around the cycle cancel out.

Claim 2 *A circulation is optimal iff there is a feasible price function in the residual graph*

Proof:
1) If there is a feasible price function for the residual graph, then the circulation is optimal.
If there is a feasible price function, then no residual edge has negative reduced cost. Then there is no negative cost cycle in the residual graph, therefore the circulation is optimal.

2) If the circulation has minimum cost, then there is a feasible price function for the residual graph. Add a source s' to the residual graph, along with edges of cost 0 to all other vertices. Compute shortest paths $d(v)$ from s' to each vertex v. The distances may be negative, but they are finite, since there are no negative cost cycles (the circulation is optimal).

We claim that we can use the distances as prices. Since d is the shortest-path distance function, $d(w) \leq d(v) + c(v, w)$ if (v, w) is in the residual graph G_f, so d is a feasible price function. ■

8.2.4 Algorithms

Cycle cancelling

To find the minimum cost circulation, look for negative cycles and saturate them until done. Negative cycles can be found with the Bellman-Ford algorithm in $O(mn)$ time.

The number of iterations is less than the cost of the min-cost circulation (taken with a plus sign), because each negative cycle decreases the cost by at least one unit. The minimum cost of a circulation is bounded by $-mUC$, so the total time of this algorithm is $O(m^2nUC)$. Using a scaling algorithm for shortest paths, we can obtain a running time of $O(m^2\sqrt{n}CU \log C)$.

Shortest Augmenting Path

This algorithm finds the min-cost max-flow on a unit graph with no negative costs. To find the min-cost max-flow, we can always pick the shortest, i.e least expensive, augmenting path. We can use Dijkstra's algorithm to find the shortest path, and we need to find at most n shortest paths. In order to keep the residual costs positive, we can use prices and keep reduced costs non-negative. The reduced cost shortest path is still the shortest path in the residual graph.

If we use reduced costs to determine the shortest augmenting path, no negative cycles are created. Augmentation creates only 0-cost reduced cost residual edges.