COMPSCI 532: Design and Analysis of Algorithms August 28, 2015

Lecture 2

Lecturer: Debmalya Panigrahi *Scribe: Allen Xiao*

1 Overview

In the previous lecture, we introduced the maximum flow problem and algorithms based on Ford-Fulkerson augmenting paths. Ford-Fulkerson ran in pseudo-polynomial time, but by carefully choosing paths the Edmonds-Karp algorithm achieved a strongly polynomial algorithm.

This time, will present and analyze Dinitz's blocking flow algorithm for maximum flow. Another one of the first strongly polynomial maximum flow algorithms, it has faster asymptotic running time than Edmonds-Karp ($O(mn^2)$ rather than $O(m^2n)$).

2 Blocking Flows

Blocking flows, in a way, use multiple augmenting paths simultaneously. We first need to introduce a few concepts:

Definition 1. *The **layered graph** of a graph is the decomposition of its vertices into **layers** by distance from the source. Each layer L_i contains vertices with $d(s,v) = i$.*

Notice that $d(s,t)$ is equal to the number of layers. The graph may have edges within layers or to previous layers, but the edges from L_i to L_{i+1} are the ones we focus on. Again, when we talk about finding augmenting paths in such a graph, we mean we are finding a path in residual graph G_f.

Definition 2. *An arc is an **admissible arc** if it is between L_i and L_{i+1} for some distance i.*

Definition 3. *A path is an **admissible path** if every edge is admissible.*

Admissibility is an idea which reoccurs in flow algorithms, although in various forms. In almost every case, we restrict augmentations to these admissible edges. In this case, notice that admissible paths are in fact shortest – every edge goes to a vertex of greater distance. We will refer to the subset of admissible edges by E_a and its induced subgraph by G_a.

Definition 4. *Flow f is a **blocking flow** if every admissible path has at least one arc saturated by f.*

Note that a blocking flow is not always a maximum one, although the maximum flow is a blocking flow. The primary property we use from blocking flows is in the following lemma:

Lemma 1. *Augmenting by a blocking flow f' increases the s-t distance in the residual network.*

Proof. Suppose flow f was augmented by blocking flow f'.

Any s-t path Γ in $G_{f+f'}$ (e.g. the shortest) must use some edges that were non-admissible in G_f, since all paths which didn't were blocked. Let (v,w) be one such edge, and in G_f distances:

$$d(s,v) \geq d(s,w)$$

since (v,w) was not admissible in G_f.

Γ has at least one edge which does not advance forward in the G_f layers (admissible edges are the only ones to do so). It follows that Γ has length strictly greater than the s-t distance in G_f, which is equal to the number of layers in G_f. □

Corollary 2. *At most n blocking flows yield a maximum flow.*

Blocking flows effectively provide a lower iteration count compared to the previous flow algorithms we covered. In Ford-Fulkerson, this was $O(|f|)$. In Edmonds-Karp, this was $O(mn)$. In blocking flow, we have $O(n)$. For this to be worth it, the extra time we need to find a blocking flow must not be too much worse than the time to find an augmenting path.

3 Dinitz's Algorithm

The description is similar to what we have seen previously with Ford-Fulkerson, replacing the augmenting path with a blocking flow.

Algorithm 1 (Dinitz 1970)

```
1: $f(v,w) \leftarrow 0 \quad \forall (v,w) \in E$
2: while $G_f$ has an $s$-$t$ path do
3:     Find a blocking flow $f'$ in $G_f$.
4:     Augment by $f'$.
5: end while
```

To find a blocking flow, we use DFS on admissible edges by continuously applying two operations, starting at the source:

1. ADVANCE: Take an admissible edge in the forward direction.
2. RETREAT: If there is no outgoing admissible edge at the current vertex, go back along the edge we came from, and delete it.

Once the sink is reached, we add the flow of that path to our blocking flow, update the residual capacities, and repeat from the source. We stop once a search retreats all the way back to s.

Of course, this requires that our admissible graph is acyclic. This is true of most fast algorithms for finding blocking flow.

3.1 Runtime on Unit Capacity Graphs

We first examine the runtime when all capacities are unit. One of the keys to the analysis is that the backward edges created by an admissible augmentation are never admissible themselves.

After advancing across an edge, we either find an admissible path to t or retreat back. If we find an admissible path, the augmentation always saturates the edge, since capacities are unit. In both cases the edge is deleted. At most $O(m)$ ADVANCEs and RETREATs occur before a blocking flow is found. The total runtime is $O(mn)$ by Corollary 2.

3.2 Runtime on Capacitated Graphs

The analysis is similar to the unit capacity case. Although augmentation will not saturate every edge on a path, it will saturate at least one. We can still claim that the number of ADVANCEs and RETREATs before augmentation occurs is $O(m)$, and therefore the time to find a blocking flow is $O(m^2)$. Finally, the total runtime is $O(m^2n)$.

This runtime is the same as Edmonds-Karp, and not the improvement we claimed in the introduction. In fact, an *amortized analysis* is what gives us $O(mn^2)$. The algorithm remains the same.

3.3 Alternative Bound for Unit Capacity

By examining distances, we can improve the runtime in the unit capacity case from $O(mn)$ to $O(m^{3/2})$. This is an improvement, since $m = O(n^2)$.

After D iterations, the maximum residual flow is at most m/D, since it is decomposable into paths of at least length D. Each following iteration will send at least one unit of flow, so the remaining number of iterations is at most m/D. Let T be the iteration count:

$$T \leq D + \frac{m}{D}$$

Choose $D = \sqrt{m}$ to obtain the iteration bound of $O(\sqrt{m})$. Again, we can find blocking flows in the unit capacity case in $O(m)$ time, giving us a running time of $O(m^{3/2})$. A similar result can be found using $d = n^{2/3}$.

3.4 Amortized Analysis of Dinitz

Recall that every ADVANCE across an edge either ends in augmentation or RETREAT across that edge. Here, we will charge each ADVANCE to an augmentation or a RETREAT, depending on which occurred.

1. Since RETREAT deletes the edge, there are $O(m)$ per blocking flow. Every RETREAT has exactly one ADVANCE charged to it, so the total amount of charge to RETREATs is $O(m)$.

2. Since each augmentation deletes at least one edge, there are $O(m)$ per blocking flow. However, there can be $O(n)$ ADVANCEs charged to an augmentation, since there are at most n edges in an s-t path. The total amount of ADVANCE charge on augmentations is $O(mn)$.

Adding these up, we have $O(mn)$ time per blocking flow, and $O(mn^2)$ for the algorithm, as desired.

4 Summary

In this lecture, we introduced blocking flows and Dinitz's blocking flow algorithm, then analyzed Dinitz for integral and unit capacities. In one analysis we improved the runtime bound using amortized analysis. In the other, we used a pattern that we will see again in Goldberg-Rao: the residual flow is bounded by the average volume between layers, and the number of layers (shortest distance) increases each iteration.

This algorithm is frequently referred to as *Dinic*'s algorithm – although the author's name is indeed Yefim Dinitz. Dinitz has written a document [Din06] describing the algorithm and its interesting backstory in further detail. The original paper [Din70] was published in the USSR in 1970.

References

[Din70] E. A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Doklady Akademii Nauk SSSR*, 194(4):1277–1280, 1970.

[Din06] Yefim Dinitz. Dinitz' algorithm: The original version and Even's version. In *Theoretical Computer Science, Essays in Memory of Shimon Even*, pages 218–240, 2006.