Max Flow with Level Graphs

CMSC 641 Design & Analysis of Algorithms

Defn. Let f be a flow in a flow network G. The level graph LG_f is a breadth-first search graph of the residual graph G_f with back edges & "sideways" edges deleted. (Cross edges from level i to level $i+1$ are kept.)

Modified Edmonds-Karp

Given: flow network $G=(V,E)$ and $c: E \to \mathbb{R}$

initial flow $f=0$

Construct residual graph G_f

Construct level graph LG_f

(If t is not reachable from s, stop.)

Add path flow of any shortest path in LG_f

Update capacities in LG_f, delete saturated edges

Update total flow, f

Repeat until t is disconnected from s (blocking flow)

Repeat

iteration

Phase

Defn: Let $\delta_f(a,b)$ = min segment distance from a to b in G_f.

Lemma1: If f' is obtained from f by augmenting thru shortest paths, then $\forall a \in V$

$$\delta_f(a,t) \le \delta_{f'}(a,t) \text{ and } \delta_f(s,a) \le \delta_{f'}(s,a)$$

(proven last time)

<u>Lemma 2</u>: Let f' be the new flow after a phase that started w/ flow f. Then, $\delta_{f'}(s,t) \geq \delta_f(s,t)+1$.

<u>Pf:</u> Let $d = \delta_f(s,t)$

In $LG_{f'}$, there must be a path from s to t w/ distance $\geq d$. (By Lemma 1 w/ $a=s$.)

Some edge $u \to v$ on this path must not be in LG_f

$$s \rightsquigarrow u \to v \rightsquigarrow t$$

since otherwise, the phase did not end w/ a blocking flow

Thus, (u,v) must be a sideways or a back edge in the BFS of $LG_f \Rightarrow \delta_f(s,u) \geq \delta_f(s,v)$.

$$\begin{aligned}
\delta_{f'}(s,t) &\geq \delta_{f'}(s,u) + 1 + \delta_{f'}(v,t) \\
&\geq \delta_f(s,u) + \delta_f(v,t) + 1 \\
&\geq \delta_f(s,v) + \delta_f(v,t) + 1 \\
&\geq d+1 \quad \text{since } \forall a \in V,\ d \leq \delta_f(s,a) + \delta_f(a,t)
\end{aligned}$$

$\boxtimes$

Running time for Edmonds-Karp

Within each iteration:

- $O(E)$ time to find an augmenting path
- $O(E)$ time for updates

\# of iterations per phase is $O(E)$

- each iteration saturates & deletes 1 edge
- $|E_f| \leq 2|E|$

\# of phases $\leq |V|$

- $\delta_f(s,t)$ increases by 1 after each phase

Total running time $O(VE^2) \approx O(V^5)$

Dinic's algorithm: find multiple augmenting paths to save time

Algorithm 18.1 (Dinic [29])

Initialize. Construct a new level graph L_G. Set $u := s$ and $p := [s]$. Go to **Advance**.

Advance. If there is no edge out of u, go to **Retreat**. Otherwise, let (u, v) be such an edge. Set $p := p \cdot [v]$ and $u := v$. If $v \neq t$ then go to **Advance**. If $v = t$ then go to **Augment**.

Retreat. If $u = s$ then halt. Otherwise, delete u and all adjacent edges from L_G and remove u from the end of p. Set $u :=$ the last vertex on p. Go to **Advance**.

Augment. Let Δ be the bottleneck capacity along p. Augment by the path flow along p of value Δ, adjusting residual capacities along p. Delete newly saturated edges. Set $u :=$ the last vertex on the path p reachable from s along unsaturated edges of p; that is, the start vertex of the first newly saturated edge on p. Set $p :=$ the portion of p up to and including u. Go to **Advance**.

Claim: Each phase of Dinic's Algorithm takes $O(VE)$ time

Pf: Let $n(G)$ = # of vertices
$e(G)$ = # of edges
$|p|$ = length of path from s to current node

Consider potential function $\Phi = n(G) \cdot e(G) - |p|$

Consider potential function $\Phi = n(G) \cdot \ell(G) - |p|$

Initialize: $O(VE)$ amortized time

Advance: $|p|$ increases by 1, Φ decreases by 1

amortized time $= 0$

Retreat: at least 1 edge removed, which frees up $n(G)$ credits. Also, $|p|$ reduced by 1.

Pays for cost of removing edges.

Augment: at least 1 edge removed, giving $n(G)$ credits.

Pays for updating capacities & shortening p.

Total time for Dinic's alg $O(VE) \times |V|$ phases

$$= O(V^2E) \approx O(V^4)$$

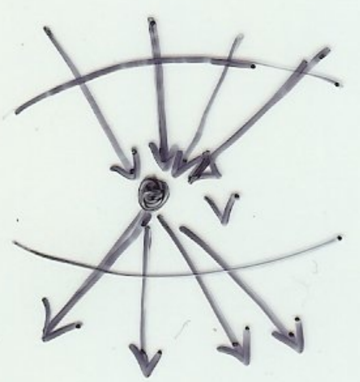
Malhorta, Pramodh-Kumar, Maheshwari

$O(V^3)$ time alg. matches best alg in textbook

Uses Fibonacci heaps, easier to explain

Uses LG_f as before.

Defn: capacity for vertices



$\sum c(u,v)$

$\sum c(v,u)$

$cap(v) = \min\left(\sum c(u,v), \sum c(v,u)\right)$

1. Find a vertex v of minimum capacity d according to Definition 18.2. If $d = 0$, do step 2. If $d \neq 0$, do step 3.
2. Delete v and all incident edges and update the capacities of the neighboring vertices. Go to 1.
3. Push d units of flow from v to the sink and pull d units of flow from the source to v to increase the flow through v by d. This is done as follows:

   Push to sink. The outgoing edges of v are saturated in order, leaving at most one partially saturated edge. All edges that become saturated during this process are deleted. This process is then repeated on each vertex that received flow during the saturation of the edges out of v, and so on all the way to t. It is always possible to push all d units of flow all the way to t, since every vertex has capacity at least d.

   Pull from source. The incoming edges of v are saturated in order, leaving at most one partially saturated edge. All edges that become saturated by this process are deleted. This process is then repeated on each vertex from which flow was taken during the saturation of the edges into v, and so on all the way back to s. It is always possible to pull all d units of flow all the way back to s, since every vertex has capacity at least d.

   Either all incoming edges of v or all outgoing edges of v are saturated and hence deleted, so v and all its remaining incident edges can be deleted from the level graph, and the capacities of the neighbors updated. Go to 1.

Amortized time per phase

$O(E+V)$ to construct LG_f, initialize heap

$O(V \log V)$ for $|V|$ delete min's

$O(E)$ to delete saturated edges &
perform decreasekey on affected vertex
($O(1)$ time to decrease key in Fib Heap)

V^2 "visits" to partially filled edges

- 1 edge per node per iteration
- # of iterations $\le |V|$ since one vertex deleted
- update edge capacities & vertex capacities (decrease key) in $O(1)$ amortized time.

Time per phase: $O(V^2)$ Total time: $O(V^3)$