CS 163 & CS 265: Graph Algorithms

Week 8: Flow

Lecture 8b: Algorithms

David Eppstein
University of California, Irvine

Winter Quarter, 2024

Greedy algorithm idea

Start with a flow that is not maximum (the all-zero flow)

Repeat: find a way to increase flow along a single path

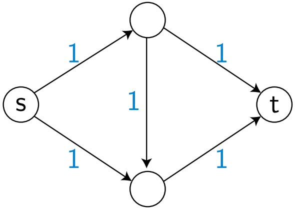
Prove that if we ever get stuck, we can find a cut
with cut capacity $=$ flow amount

Since max flow $\leq$ min cut $\leq$ this cut, and we have a flow that equals the cut, our flow must be maximum

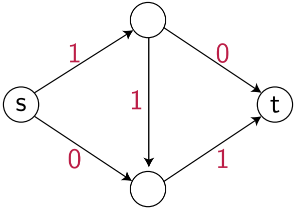
Does it work?

Can't increase flow on saturated edges
(on which we have already used up all the capacity)

But if we only look for paths of unsaturated edges
we can get stuck at a non-maximum flow



capacities



flow amounts

In this example, flow amount is only 1, but maximum flow is 2

How can we get there from here?

A local view of how flow changes along paths

Define the balance of a vertex to be flow in minus flow out
(we want this to be zero at all non-terminal vertices)

Sending x more units of flow along a path containing edge $u \to v$, decreases balance at u by x, and increases the balance at v by x

Both changes in balance are cancelled by the next edge of the path

We would get exactly the same effect on balance at u and v if we send x fewer units of flow along an edge $v \to u$

So find generalized paths containing these backwards edges, and reduce flow on them instead of increasing flow on forward edges

Residual capacity

Suppose we are trying to increase the flow from u–v
(as part of a larger path along which we are increasing flow)

- If the input has an edge $u \to v$ with capacity c_{uv} and a smaller flow amount f_{uv}, we can add up to $c_{uv} - f_{uv}$ more units of flow on that edge
- If the input has an edge $v \to u$ with flow amount f_{vu}, we can reduce the flow by up to f_{vu} units

Both have the same effect on balance (per unit of flow changed)

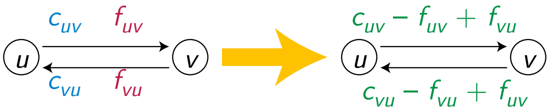
We can do both at the same time

Residual graph

We can represent the amount of additional flow possible by another flow network, the residual graph

Residual capacity from u to $v = c_{uv} - f_{uv} + f_{vu}$

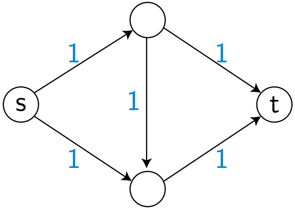
(with zeros when one of these two edges doesn't exist)



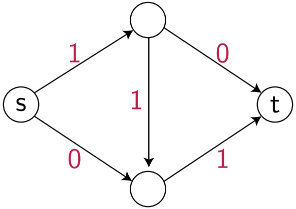
Standard simplifications:

- Only include edges with nonzero residual capacity
- Remove residual edges into s or out of t
  (because they are useless)

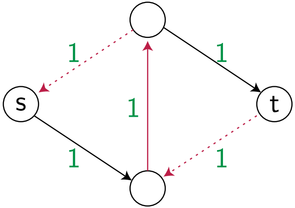
Example



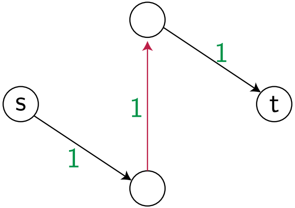
capacities



flow amounts



residual graph
(capacity minus flow)



residual path

Augmenting path (Ford–Fulkerson) algorithm

Start with all flow amounts zero, residual graph = input graph

While residual graph has a path P from s to t
(after simplification, so path has nonzero capacity):

- Find width w of P
  (the smallest residual capacity of an edge)
- Increase flow along P by w units
  (either by adding to capacity of forward edges
  or subtracting from reversed edges)
- Recompute the residual graph

[Ford and Fulkerson 1956]

If this algorithm terminates it finds a minimum cut

Only stops when there are no residual paths from s to t

Form a cut (partition of the vertices into two subsets S and T):

- $S =$ everything that can be reached from s by a residual path
- $T =$ everything else

By construction, there are no (nonzero) residual edges from S to T

$\Rightarrow$ every input edge from S to T is saturated,
and every edge from T to S has no flow
(otherwise there would still be some residual capacity)

$\Rightarrow$ net flow across the cut $=$ capacity of the cut

$\Rightarrow$ we have found a flow and a cut that are equal

$\Rightarrow$ they are both optimal

Algorithmic proof of integer flow property

Suppose all capacities are integers

Then all flow amounts will be integers, and the flow will increase by at least one each time we find an augmenting path

Whatever the maximum flow amount is, it is $\leq \sum c_{uv}$, the sum of all capacities of all edges

So the algorithm will terminate with a maximum flow in which all flow amounts are integers, after it finds at most $\leq \sum c_{uv}$ paths

Algorithmic proof of max-flow min-cut theorem

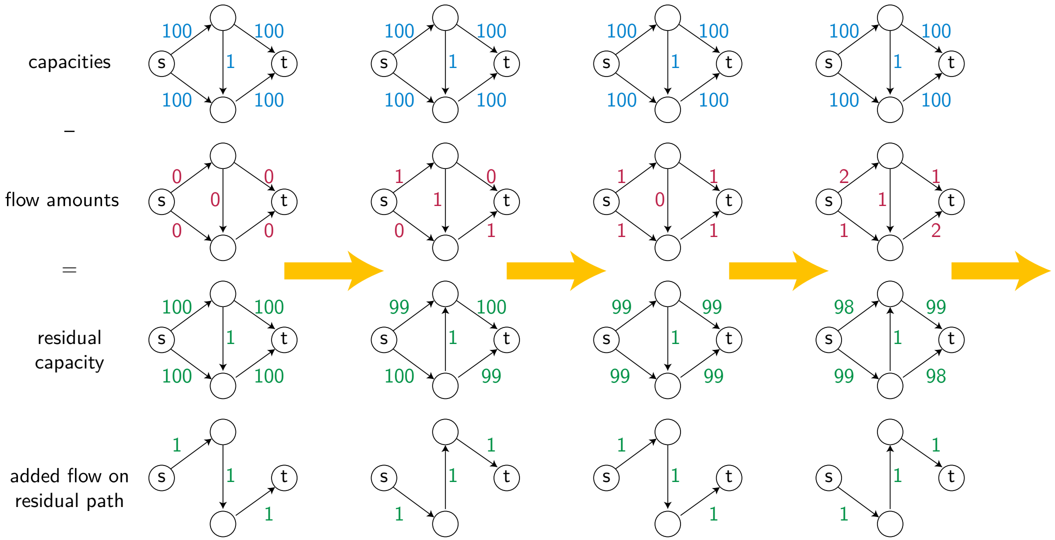
We can prove from general principles (because it's a linear program with a bound of $\sum c_{uv} < \infty$ on its optimal value) that a maximum flow exists

Initialize the flow algorithm to this maximum flow instead of the all-zero flow

It must instantly terminate (because the flow cannot be improved)

When it does, it has found a cut equal to the flow

But choosing bad paths can take a long time



Augmenting on widest paths

Suppose widest residual path from s to t has width w
$\Rightarrow$ wider residual edges do not connect s to t
$\Rightarrow$ for some cut, all residual capacities are $\leq w$
$\Rightarrow$ maximum flow amount is $\leq mw$

So widest path finds at least a $1/m$ fraction of the optimal flow
$\Rightarrow$ After k paths, remaining flow is smaller by $\leq (1-1/m)^k$ factor

If capacities are integers and total flow amount is F, then after $O(m \log F)$ paths we have reduced the remaining flow to a number less than one, which can only be zero

Time/path $= O(m) \Rightarrow$ total $O(m^2 \log F)$ [Edmonds and Karp 1972]

"Weakly polynomial": cannot be improved to a function only of graph size, not F [Queyranne 1980]

Augmenting by shortest paths (Dinic's algorithm)

Repeat:

- Use BFS to find (unweighted) distances from s to other vertices in residual graph
- Repeatedly find augmenting shortest paths (according to these distances), saturating at least one edge/path, until all shortest paths are blocked by saturated edges
- Recompute the new residual graph for the new flow

Each iteration of the loop can be performed in time $O(m \log n)$ using dynamic tree data structures to process each path very quickly (logarithmic time/path) [Sleator and Tarjan 1983]

Distance from s to t always increases $\Rightarrow < n$ iterations
Total time $O(mn \log n)$, strongly polynomial

Close to best known bound for strongly polynomial algorithms

Recent breakthrough in flow algorithms

Requires capacities to be integers $\leq U$

Can handle minimum-cost maximum flow with integer costs $\leq C$

Main idea: augment along approximate minimum-ratio cycles
(like tramp steamer problem) using randomized embeddings into tree metrics and efficient data structures to find each cycle quickly

Time: $m^{1+o(1)} \log U \log C$ with high probability

[Chen et al. 2022]

Morals of the story

Residual graph describes how much more we can increase the flow

Max flow can be found by increasing flow on residual paths

Min cut can be found by splitting vertices into
the ones reachable from s by residual paths, and the rest

It matters which path you choose;
widest paths and shortest paths are good choices

References I

Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *Proc. 63rd IEEE Symposium on Foundations of Computer Science*, 2022.

E. A. Dinic. An algorithm for the solution of the problem of maximal flow in a network with power estimation. *Doklady Akademii Nauk SSSR*, 194:754–757, 1970.

Jack R. Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972. doi:10.1145/321694.321699.

L. R. Ford, Jr. and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956. doi:10.4153/CJM-1956-045-5.

Maurice Queyranne. Theoretical efficiency of the algorithm "capacity" for the maximum flow problem. *Mathematics of Operations Research*, 5(2):258–266, 1980. doi:10.1287/moor.5.2.258. URL https://www.jstor.org/stable/3689153.

References II

Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983. doi:10.1016/0022-0000(83)90006-5.