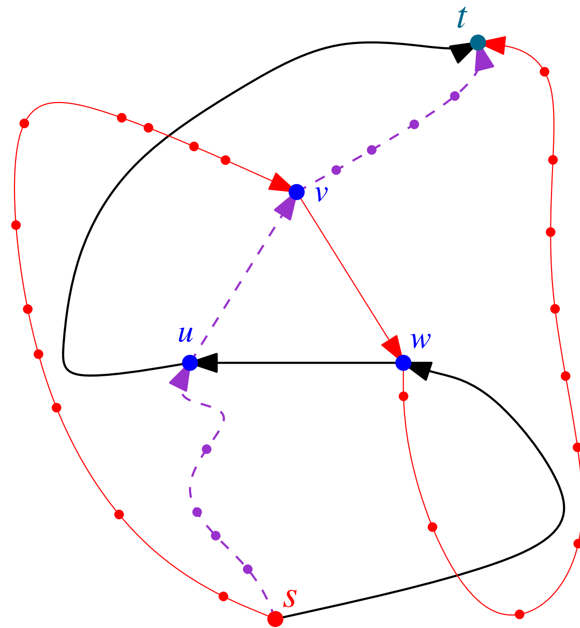
Chapter 17

Network Flow VI - Min-Cost Flow Applications

By Sariel Har-Peled, December 7, 2009[1] Version: 0.1

17.1 Efficient Flow

A flow f would be considered to be ***efficient*** if it contains no cycles in it. Surprisingly, even the **Ford-Fulkerson** algorithm might generate flows with cycles in them. As a concrete example consider the picture on the right. A disc in the middle of edges indicate that we split the edge into multiple edges by introducing a vertex at this point. All edges have capacity one. For this graph, **Ford-Fulkerson** would first augment along $s \to w \to u \to t$. Next, it would augment along $s \to u \to v \to t$, and finally it would augment along $s \to v \to w \to t$. But now, there is a cycle in the flow; namely, $u \to v \to w \to u$.



One easy way to avoid such cycles is to first compute the max flow in G. Let α be the value of this flow. Next, we compute the min-cost flow in this network from s to t with flow α, where every edge has cost one. Clearly, the flow computed by the min-cost flow would not contain any such cycles. If it did contain cycles, then we can remove them by pushing flow against the cycle (i.e., reducing the flow along the cycle), resulting in a cheaper flow with the same value, which would be a contradiction. We got the following result.

[1]This work is licensed under the Creative Commons Attribution-Noncommercial 3.0 License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc/3.0/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

Theorem 17.1.1 *Computing an efficient (i.e., acyclic) max-flow can be done in polynomial time.*

(BTW, this can also be achieved directly by removing cycles directly in the flow. Naturally, this flow might be less efficient than the min-cost flow computed.)

17.2 Efficient Flow with Lower Bounds

Consider the problem AFWLB (acyclic flow with lower-bounds) of computing efficient flow, where we have lower bounds on the edges. Here, we require that the returned flow would be integral, if all the numbers involved are integers. Surprisingly, this problem which looks like very similar to the problems we know how to solve efficiently is NP-Complete. Indeed, consider the following problem.

Problem: Hamiltonian Path

> *Instance:* A directed graph G and two vertices s and t.
> *Question:* Is there a Hamiltonian path (i.e., a path visiting every vertex exactly once) in G starting at s and ending at t?

It is easy to verify that Hamiltonian Path is NP-Complete[2]. We reduce this problem to AFWLB by replacing each vertex of G with two vertices and a direct edge in between them (except for the source vertex s and the sink vertex t). We set the lower-bound and capacity of each such edge to 1. Let H denote the resulting graph.

Consider now acyclic flow in H of capacity 1 from s to t which is integral. Its 0/1-flow, and as such it defines a path that visits all the special edges we created. In particular, it corresponds to a path in the original graph that starts at s, visits all the vertices of G and ends up at t. Namely, if we can compute an integral acyclic flow with lower-bounds in H in polynomial time, then we can solve Hamiltonian path in polynomial time. Thus, AFWLB is NP-Hard.

Theorem 17.2.1 *Computing an efficient (i.e., acyclic) max-flow with lower-bounds is* NP-Hard *(where the flow must be integral). The related decision problem (of whether such a flow exist) is* NP-Complete.

By this point you might be as confused as I am. We can model an acyclic max-flow problem with lower bounds as min-cost flow, and solve it, no? Well, not quite. The solution returned from the min-cost flow might have cycles and we can not remove them by cycling the cycles. That was only possible when there was no lower bounds on the edge capacities. Namely, the min-cost flow algorithm would return us a solution with cycles in it if there are lower bounds on the edges.

17.3 Shortest Edge-Disjoint Paths

Let G be a directed graph. We would like to compute k-edge disjoint paths between vertices s and t in the graph. We know how to do it using network flow. Interestingly, we can find the shortest k-edge disjoint paths using min-cost flow. Here, we assign cost 1 for every edge, and capacity 1

[2] Verify that you know to do this — its a natural question for the exam.

for every edge. Clearly, the min-cost flow in this graph with value k, corresponds to a set of k edge disjoint paths, such that their total length is minimized.

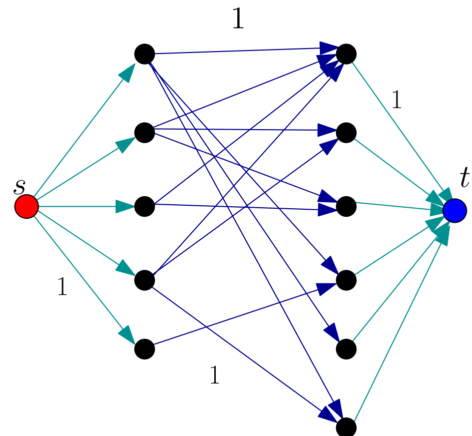
17.4 Covering by Cycles

Given a direct graph G, we would like to cover all its vertices by a set of cycles which are vertex disjoint. This can be done again using min-cost flow. Indeed, replace every vertex u in G by an edge $(u' \rightarrow u'')$. Where all the incoming edges to u are connected to u' and all the outgoing edges from u are now starting from u''. Let H denote the resulting graph. All the new edges in the graph have a lower bound and capacity 1, and all the other edges have no lower bound, but their capacity is 1. We compute the minimum cost circulation in H. Clearly, this corresponds to a collection of cycles in G covering all the vertices of minimum cost.

Theorem 17.4.1 *Given a directed graph* G *and costs on the edges, one can compute a cover of* G *by a collection of vertex disjoint cycles, such that the total cost of the cycles is minimized.*

17.5 Minimum weight bipartite matching

Given an undirected bipartite graph G, we would like to find the maximum cardinality matching in G that has minimum cost. The idea is to reduce this to network flow as we did in the unweighted case, and compute the maximum flow – the graph constructed is depicted on the right. Here, any edge has capacity 1. This gives us the size ϕ of the maximum matching in G. Next, we compute the min-cost flow in G with this value ϕ, where the edges connected to the source or the sing has cost zero, and the other edges are assigned their original cost in G. Clearly, the min-cost flow in this graph corresponds to a maximum cardinality min-cost flow in the original graph.



Here, we are using the fact that the flow computed is integral, and as such, it is a $0/1$-flow.

Theorem 17.5.1 *Given a bipartite graph* G *and costs on the edges, one can compute the maximum cardinality minimum cost matching in polynomial time.*

17.6 The transportation problem

In the ***transportation problem***, we are given m facilities $f_1, \ldots, f_m$. The facility f_i contains x_i units of some commodity, for $i = 1, \ldots, m$. Similarly, there are $u_1, \ldots, u_n$ customers that would like to buy this commodity. In particular, u_i would like to by d_i units, for $i = 1, \ldots, n$. To make things interesting, it costs c_{ij} to send one unit of commodity from facility i to costumer j. The natural question is how to supply the demands while minimizing the total cost.

To this end, we create a bipartite graph with $f_1, \ldots, f_m$ on one side, and $u_1, \ldots, u_n$ on the other side. There is an edge from $\left(f_i \to u_j\right)$ with costs c_{ij}, for $i = 1, \ldots, m$ and $j = 1, \ldots, n$. Next, we create a source vertex that is connected to f_i with capacity x_i, for $i = 1, \ldots, m$. Similarly, we create an edges from u_j to the sink t, with capacity d_i, for $j = 1, \ldots n$. We compute the min-cost flow in this network that pushes $\phi = \sum_j d_k$ units from the source to the sink. Clearly, the solution encodes the required optimal solution to the transportation problem.

Theorem 17.6.1 *The transportation problem can be solved in polynomial time.*