Min Cost Flows and Shortest Paths in Graphs with Negative Edge Lengths

Jonathan Turner

January 17, 2013

In the *minimum cost flow problem*, we are given a flow graph $G = (V, E)$ in which each edge has a real-valued *cost*, in addition to its capacity. The cost of an edge represents the *cost per unit flow* on that edge, so if $cost(u, v) = 3$ and $f(u, v) = 5$ then the edge (u, v) contributes 15 to the overall cost of the flow. Figure 1 is an example of an instance of the min cost flow problem. Note that edge costs may be negative. In general, edge

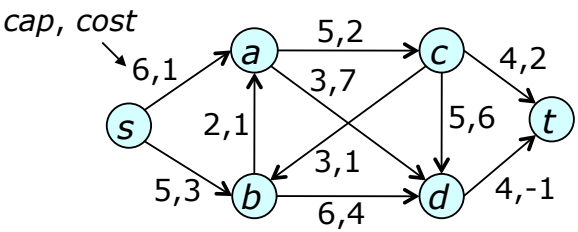

Figure 1: Example of min cost flow problem

costs are defined to be *skew symmetric*, that is $cost(u, v) = -cost(u, v)$. The cost of a flow is defined

$$cost(f) = \sum_{f(u,v)>0} cost(u, v)f(u, v) = \sum_{u,v} cost(u, v)f(u, v)/2$$

So in Figure 2, the cost of the flow is 47.

The residual graph is defined as in the max flow problem. The edge costs in the residual graph are the same as the edge costs in the flow graph.

The *cost of a path p* is defined to be the sum of its edge costs. These definitions ensure that the cost of a path in the residual graph is equal to the cost of adding one unit of flow to all of the path's edges in the flow graph.

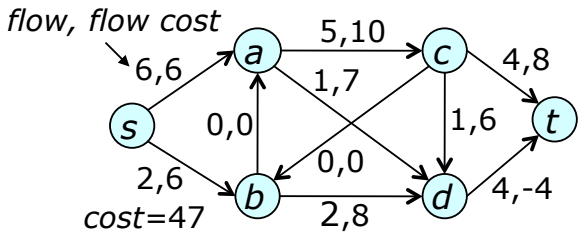

Figure 2: Flow cost example

We say that f is a *minimum cost flow* if among all flows with value $|f|$, f has minimum cost. In the *minimum cost, maximum flow problem* the objective is to find a maximum flow of minimum cost.

There are two general strategies for finding minimum cost flows. The *cost reduction strategy* converts an existing flow into a flow of lower cost by finding negative cost cycles in the residual graph and adding flow to those cycles. The *minimum cost augmentation strategy* repeated finds augmenting paths of minimum cost. Both of these methods require finding least-cost paths in a graph with negative edge costs, so before proceeding further, we turn to the problem of how to find such paths.

Recall the general scanning and labeling method for finding shortest paths. Each vertex can be one of three states, *unlabeled*, *labeled* or *scanned*. Initially, the source vertex s is labeled and all others are unlabeled. We then repeat the following step as long as possible.

> *Scanning step.* Select a labeled vertex u and convert it to the scanned state. For all edges (u, v) with $dist(v) > dist(u) + length(u, v)$, make $p(v) = u$ and $dist(v) = dist(u) + length(u, v)$ and convert v to the labeled state.

Note that vertices can alternate between the scanned and labeled states. To obtain an efficient algorithm for graphs with negative edge lengths, we store the labeled vertices in a queue and select the first vertex in the queue in each scanning step. This is referred to as the *breadth-first scanning algorithm*.

procedure breadthFirst(graph $G = (V, E)$, **vertex** s,
    mapping p : **vertex** $\rightarrow$ **vertex**);
  vertex v; **list** *queue*;
  for $v \in V \Rightarrow dist(v) := \infty$; $p(v) :=$ *null*; rof;
  $dist(s) := 0$; *queue* $:= [s]$; // contains all labelled vertices
  do *queue* $\neq [] \Rightarrow$

```
        v := queue(1); queue := queue[2..];
        for [v, w] ∈ out(v) ⇒
            if dist(v) + length(v, w) < dist(w) ⇒
                p(w) := v; dist(w) := dist(v) + length(v, w);
                if w ∉ queue ⇒ queue := queue & [w]; fi;
            fi;
        rof;
    od;
end;
```

In order to bound its running time, we divide the execution of the breadth-first scanning algorithm into *passes*. Pass 0 ends after vertex s is scanned the first time. Pass j ends after every vertex that was on the queue at the end of pass $j-1$ has been scanned. Note that with this definition, every vertex is scanned at most once per phase and so each edge is examined at most once per phase. This gives us a running time of $O(m)$ per phase.

To complete the running time analysis, we need to bound the number of phases. We can do this by showing that if there is a path p from s to v with k edges, then at the end of pass $k-1$, $dist(v) \leq length(p)$. This is clearly true if $k = 1$, since at the end of pass 0, all edges from s have been examined. Assume then, that at the end of pass $k-1$, $dist(u) \leq length(p)$ for all vertices u and all paths p with k edges. Let q be any path to a vertex v with $k+1$ edges. Let u be the predecessor of v on this path and let p be the portion of the path from s to u. We know that u was placed on the queue before the end of pass $k-1$ with a distance label $dist(u) \leq length(p)$. Consequently, the next time that u is scanned (which must happen before the end of pass k), the edge (u, v) is examined and if $dist(v) > dist(u) + length(u, v)$, it will be assigned the value $dist(v) = dist(u) + length(u, v) = length(q)$. Thus, we have

Theorem 1 *If no negative cycle is reachable from s, the breadth-first algorithm runs in $O(mn)$ time stopping after at most pass $n-1$; otherwise it never halts.*