Min-Cost Flow, shortest A-Path Algorithm

Given a flow network G that also has arc costs \( w(i,j) \), we want to find a min-cost flow (this could be a maximum flow, a flow of a specific value, or the largest flow not exceeding a target). We can solve all of these versions by starting with the zero flow and repeatedly finding the cheapest (shortest with respect to \( w(i,j) \)) A-path in the residual graph and augmenting along it. In general, the residual graph will have negative arcs (either from negative \( w(i,j) \) values, or from back edges whose weights are the negative of the original weights). To avoid using an expensive shortest path algorithm (e.g. Bellman-Ford, BF) for negative weights, we adjust the weights at each step to ensure they are all positive, so we can use Dijkstra to find shortest A-paths.

The adjusted weights will be formed by computing distance labels \( d(v) \) for each vertex \( v \). These are the shortest distance from \( s \) to \( v \) in \( G_f \). We then use

\[
\hat{w}(i,j) \leftarrow w(i,j) + d(i) - d(j)
\]

These values are always non-negative (by the properties of shortest paths), and if an edge \((i,j)\) is on a shortest path from \(s\) to \( j \), then \( \hat{w}(i,j) = 0 \).

**Detailed algorithm:**

Start with the zero flow \( f \) (so \( G_f = G \)),

Compute the initial distance labels \( d(v) \) in \( G_f \); (use BF if some \( w(i,j) < 0 \), else Dijk.)

\[
\hat{w}(i,j) \leftarrow w(i,j) + d(i) - d(j);
\]

**Repeat**

Find a shortest A-Path \( P \) in \( G_f \) with respect to the \( \hat{w}(i,j) \) values; (Dijk since no negative edges)

Augment along \( P \) and update flow \( f \);

Update \( G_f \);

Compute distance labels \( d(v) \) in \( G_f \) using \( \hat{w}(i,j) \) values; (can use Dijk since no negative edges)

\[
\hat{w}(i,j) \leftarrow w(i,j) + d(i) - d(j) \quad \text{(update edge weights)};
\]

Update weights in \( G_f \) to reflect new \( \hat{w}(i,j) \) values.

**Until** No A-Path \( P \) found.

The termination condition above is for a max-flow. If we want a target flow we stop instead when the flow value reaches the desired level (or if above it, simply change the final augmentation to be lower). Similarly for a target budget.

The run time is \( O(mn) \) for an initial run of BF, and then each loop iteration is dominated by the time for Dijkstra ( \( O(m \log n) \) or so; all other operations are \( O(m) \)). If we have a unit capacity network, the max flow is \( n \) and we get a roughly \( O(mn) \) run time. For general capacities the Repeat loop has no good bound.