The All-Pairs Shortest Path Problem

Jonathan Turner

January 30, 2013

In the all-pairs shortest path problem, we are interested in finding shortest paths between all pairs of vertices. One way to do this is by repeatedly applying an algorithm for the single-source problem. So for example, when working with graphs that have negative edge lengths, we could apply the breadth-first scanning algorithm n times, giving us an $O(mn^2)$ algorithm. It turns out that we can do substantially better than this using algorithms explicitly designed for the all-pairs problem. We'll study two such algorithms. The second uses a technique that we can also use to improve the performance of algorithms for the min-cost flow problem.

The first algorithm we'll study is called *Floyd's algorithm* and is based on a simple recurrence. Let $G = (V, E)$ and let $V = \{v_1, v_2, \ldots, v_n\}$. Define $dist_i(x, y)$ to be the length of a shortest path from x to y that passes only through vertices in the set $\{v_1, v_2, \ldots, v_i\}$. Given this definition, it's easy to see that the following recurrence is true.

$$\begin{aligned} dist_0(x,y) &= \begin{cases} length(x,y) & \text{if } (x,y) \in E \\ \infty & \text{otherwise} \end{cases} \\ dist_{i+1}(x,y) &= \min\{dist_i(x,y), dist_i(x,v_{i+1}) + dist_i(v_{i+1},y)\} \end{aligned}$$

Since $dist_n(x, y)$ is the shortest path distance from x to y, we can compute shortest path distances by applying the recurrence repeatedly, for all pairs of vertices. The algorithm can be stated as follows

- Initialize $dist(x, y)$ to $length(x, y)$ if there is an edge from x to y, zero if $x = y$ and ∞ otherwise.
- Apply the following step to each vertex v

  Labeling Step. If $dist(v, v) < 0$, abort: there is a negative cycle. Otherwise, for each pair x, y with $dist(x, y) > dist(x, v) + dist(v, y)$, replace $dist(x, y)$ with $dist(x, v) + dist(v, y)$.

When the algorithm terminates, the $dist(x, y)$ is the length of a shortest path from x to y, assuming that the graph has no negative cycles. Each labeling step takes $O(n^2)$ time, giving an overall running time of $O(n^3)$. This makes it generally faster than the breadth-first scanning algorithm and even for graphs with non-negative edge lengths, it can out-perform Dijkstra's algoirthm when the graphs are very dense.

To obtain the actual paths in addition to the shortest path distances, we compute an auxiliary mapping $midpoint(x, y)$. These values are initialized to *null* and updated using the following version of the labeling step.

> *Labeling Step.* If $dist(v, v) < 0$, abort: there is a negative cycle. Otherwise, for each pair x, y with $dist(x, y) > dist(x, v) + dist(v, y)$, replace $dist(x, y)$ with $dist(x, v) + dist(v, y)$ and replace $midpoint(x, y)$ with v.

A shortest path from x to y can be computed by a simple recursive algorithm using the *midpoint* mapping.

As noted earlier, for graphs with non-negative edge lengths we can solve the all-pairs problem by running Dijkstra's algorithm n times. It turns out that we can also use Dijkstra's algorithm for graphs with negative edge lengths, if we first transform the edge lengths to make them non-negative, while *preserving the relative lengths of paths* between all pairs of vertices. In order to find these new edge lengths, we must solve one single-source problem on a graph with negative edge lengths.

The transformation is computed as follows. Given a directed graph $G = (V, E)$ with edge lengths $length(x, y)$, imagine that we augment G by adding a source vertex s with a zero cost edge to each vertex in V. Now, let $dist(x)$ be the length of a shortest path in the modified graph from s. Now define new edge lengths

$$length'(x, y) = length(x, y) + dist(x) - dist(y)$$

Note that the shortest path tree property implies that the right-hand side of this equation is non-negative, hence the transformed lengths are also. An example of the edge length transformation appears in Figure 1. Note that this transform does not preserve the lengths of paths. For example, in the original graph, the path a, b, d has length 0, while in the transformed graph it has length 3. However, we'll see shortly, that it changes the lengths of all paths from a vertex x to a vertex y by the same amount, and this means that the relative path lengths are preserved. So for example, the path a, c, d in the origninal has length 3, while in the transformed graph, it has length

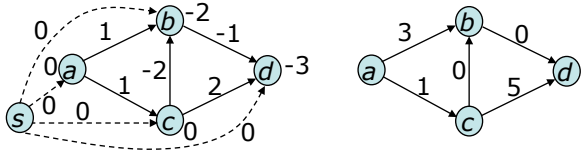

Figure 1: Example of edge length transform

6. So in both paths from a to d, the length has increased by 3. Note that different pairs of vertices may have their path lengths shifted by different amounts.

We can compute the transformed edge lengths without actually augmenting G with an extra vertex. We simply use a modified version of the breadth-first scanning algorithm in which all vertices are inserted into the queue at the start with an initial distance of 0 and a null parent pointer. We then proceed in the normal way.

Note that even though we are using breadth-first scanning to obtain the transformed edge lengths, we only have to do this one time. Once we have the transformed edge lengths, we can apply Dijkstra's algorithm to compute shortest paths from every vertex and the distances in the original graph can be obtained by reversing the transform. This gives us a runtime of

$$O(mn + n(m + n \log n)) = O(mn + n^2 \log n)$$

assuming that Dijkstra's algorithm is implemented using Fibonacci heaps. Note that for sparse graphs, this can be substantially faster than Floyd's algorithm.

Let's finish up by showing that the edge transform really does work as advertised. In particular, we want to show that for any path p, joining a vertex x to a vertex y, that

$$\mathit{length}'(p) = \mathit{length}(p) + \mathit{dist}(x) - \mathit{dist}(y)$$

Note that if this is true, all path lengths from x to y are shifted by the same amount.

For paths of length 1, the equation above is just the definition of the edge length transform. So, let's proceed by induction, by assuming that the equation is true for all paths of length k. If p is a path of length $k+1$ then its first k edges form a path q of length k. If v is the last vertex on q,

$$
\begin{aligned}
length'(p) &= length'(q) + length'(v, y) \\
&= (length(q) + dist(x) - dist(v)) \\
&\qquad +(length(v, y) + dist(v) - dist(y)) \\
&= length(q) + length(v, y) + dist(x) - dist(y) \\
&= length(p) + dist(x) - dist(y)
\end{aligned}
$$

Hence we have the following theorem.

Theorem 1 *For any edge (u, v), $length'(u, v) \geq 0$ and for every path p from a vertex x to a vertex y, $length'(p) = length(p) + dist(x) - dist(y)$.*