From Boston to San Francisco: A Survey of Shortest Paths Algorithms in Planar Graphs

Stuart Baker, Ömer Cerrahoğlu, Sebastian Claici

December 3, 2014

Abstract

1 Introduction

2 Background

3 Single source shortest paths

We call an edge uv relaxed if $d(v) \leq d(u) + c(u, v)$. We call the assignment

$$d(v) \leftarrow \min\{d(v), d(u) + c(u, v)\}$$

the relaxation of vertex v. We know that the labels give a correct shortest path distances if the shortest-path conditions are satisfied:

- $d(s) = 0$,
- every label $d(v)$ is an upper bound on the $s - v$ distance,
- every edge is relaxed.

3.1 Nonnegative edge weights

For a planar graph with nonnegative edge weights, Dijkstra's algorithm runs in $O(n \log n)$ as $m \leq 3n - 6$. It is possible to improve this to $O(n)$. To get there, recall that and r-division of a planar graph is a partition of the graph into $\Theta(n/r)$ regions of size $O(r)$ with boundary size $O(\sqrt{r})$. An r-division of a planar graph can be computed in linear time.

A simple $O(n\sqrt{\log n \log\log n})$ emerges quite beautifully just from the r-division if we set $r = \frac{\log n}{\log\log n}$. The algorithm follows a divide-and-conquer approach in which each region is processed first, followed by a clean-up phase where the results are merged.

Algorithm 1 Shortest paths in each region R

```
for all Regions $R$ do
    for all Boundary nodes $v \in R$ do
        Compute SSSP from $v$ in $R$
        Store $(u, v)$ distances for any two boundary nodes $u$, $v$
    end for
end for
```

The first step is to compute the single-source shortest paths for each boundary node in each region R (algorithm 1). We can now replace each region R by a complete graph on R's boundary nodes with shortest paths distances between any two nodes. Call this auxiliary graph G'. The second phase of the algorithm is to compute the SSSP from s in G'. This gives the true shortest paths from s to all the boundary nodes. Finally, we must tidy up by finding the distances from s to the nodes inside each region (algorithm 2).

Algorithm 2 Clean up: shortest paths from s to inside of each region R

```
for all Regions $R$ do
    for all Boundary nodes $v \in R$ do
        Set $d(v) = d_{G'}(s, v)$
        Compute SSSP from $v$ in $R$
    end for
end for
```

To analyze the algorithm, we will need a few pieces of information:

- Total number of boundary nodes is $O(\sqrt{r})O(n/r) = O(n/\sqrt{r})$.
- Number of nodes in G' is $O(n/r)O(\sqrt{r}) = O(n/\sqrt{r})$.
- Number of edges in G' is $O(n/r)O(r) = O(n)$.

Using $r = \frac{\log n}{\log\log n}$, the first phase is bounded above by

$$O\left(n\frac{\sqrt{\log\log n}}{\sqrt{\log n}}\log n\right) = O(n\sqrt{\log n \log\log n}),$$

the second phase is an SSSP in a size $O(n\frac{\sqrt{\log\log n}}{\sqrt{\log n}})$ graph, and thus also $O(n\sqrt{\log n \log\log n})$, while the tidying up is a series of SSSPs in each of the regions, and has the same bound as the first phase—$O(n\sqrt{\log n \log\log n})$. The total time bound ends up $O(n\sqrt{\log n \log\log n})$.

The simple algorithm shown above is an improvement over Dijkstra's algorithm, but one can do better. In fact, we can achieve linear time by recursively subdividing the graph. Without loss of generality, assume the graph is directed, and that each node has at most two incoming and two outgoing edges. We call a region atomic if it contains only one edge uv. A nonatomic region will have as children subregions that are contained within it.

For each region R, we maintain a priority queue $Q(R)$ that stores the subregions of R if R is nonatomic, or the single arc uv is R is atomic. The algorithm ensures that for every region R, the minimum element of $Q(R)$ is the minimum label $d(v)$ over all edges vw in R that remain to be processed.

Algorithm 3 alg:process

```
  **if** $R$ contains only $uv$ **then**
  **else**
  **end if**
```

3.2 Arbitrary edge weights

4 Multiple source shortest paths

5 Extensions to higher genus