Networks and Flows on Graphs

Shortest Path Problems

Bashar Dudin

April 16, 2018

EPITA

What is it about?

Shortest path problem

Given a weighted directed graph a shortest path problem is about looking at paths having minimal lengths (weighted lenghts)

- between two given vertices (single-pair shortest path problem)
- from a given source to any other vertex in the graph (single-source shortest path problem)
- between all pairs of vertices (all-pairs shortest path problem)

What is it about?

Shortest path problem

Given a weighted directed graph a shortest path problem is about looking at paths having minimal lengths (weighted lenghts)

- between two given vertices (single-pair shortest path problem)
- from a given source to any other vertex in the graph (single-source shortest path problem)
- between all pairs of vertices (all-pairs shortest path problem)

What is it about?

Shortest path problem

Given a weighted directed graph a shortest path problem is about looking at paths having minimal lengths (weighted lenghts)

- between two given vertices (single-pair shortest path problem)
- from a given source to any other vertex in the graph (single-source shortest path problem)
- between all pairs of vertices (all-pairs shortest path problem)

In order to be able to answer any of these problems the digraphs we're working with need not have any negative valued circuits (in case of single source, ones accessible from the source).

What is it about?

Shortest path problem

Given a weighted directed graph a shortest path problem is about looking at paths having minimal lengths (weighted lenghts)

- between two given vertices (single-pair shortest path problem)
- from a given source to any other vertex in the graph (single-source shortest path problem)
- between all pairs of vertices (all-pairs shortest path problem)

In order to be able to answer any of these problems the digraphs we're working with need not have any negative valued circuits (in case of single source, ones accessible from the source).

Remark : One could look for longest paths in a digraph. For graphs having no positive-valued circuits one can adapt the algorithms solving shortest path problems

Single-source shortest path problem : Ford's Algorithm

Input: G a digraph having source s, n vertices and no negative-valued circuits accessible from s
write v the weight map from the set of arrows to integers

Output: minimal lengths of paths from s

number vertices arbitrarily $v_0, \ldots, v_{n-1}$ ensuring s is the first
initialize the vertex v_0 with $\lambda_0 = 0$ and all others with $\lambda_i = +\infty$

for _ from 0 to $n-1$ **do**
    for all arrows (v_ℓ, v_j) **do**
        if $\lambda_j > \lambda_\ell + v(v_\ell, v_j)$ **then**
            $\lambda_j \leftarrow \lambda_\ell + v(v_\ell, v_j)$
        end if
    end for
end for
return the list of λ_is with the corresponding vertices

By adding an extra check to this algorithm one can also detect possible negative-valued circuits, without having to check it beforehand

Ford's Algorithm : building a tree of shortest paths

Input: G a digraph having source s, n vertices and no negative-valued circuits accessible from s
write ν the weight map from the set of arrows to integers

Output: minimal lengths of paths from s together with a tree T of shortest paths

number vertices arbitrarily $v_0, \ldots, v_{n-1}$ ensuring s is the first
initialize the vertex v_0 with $\lambda_0 = 0$ and all others with $\lambda_i = +\infty$
T is a tree having a single vertex s
for _ from 0 to $n-1$ **do**
    for all arrows (v_ℓ, v_j) **do**
        if $\lambda_j > \lambda_\ell + \nu(v_\ell, v_j)$ **then**
            $\lambda_j \leftarrow \lambda_\ell + \nu(v_\ell, v_j)$
            $T[j] \leftarrow \ell$
        end if
    end for
end for
return the list of λ_is with the corresponding vertices and T

The tree T is built up by attaching to each vertex k its predecessor $T[k]$, if there is any. To get a shortest path between s and a vertex v follow the unique path in T linking both.

Validity and complexity of Ford's algorithm

Let V be the set of vertices of G and A its set of arrows. Line 2 of Ford's algorithm takes $|V|$ operations. One goes $|V|$ times through the loop at line 3, for each iteration, one goes $|A|$ times through loop at line 4. In total, we get a complexity for Ford's algorithm of $O\big(|V| + |V||A|\big) = O\big(|V||A|\big)$

Validity

An elementary shortest path from s to a vertex v has length at most $|V| - 1$. Given such an elementary path $a_0, \ldots, a_m$ with $a_0 = s$ and $a_m = v$ each sub-path from s to any intermediate vertex is also a shortest path (otherwise we can make a shorter one from s to v). Following Ford's algorithm along the previous path, one can show, by induction that λ_m can only be equal to the (weighted) length of $a_0, \ldots, a_m$, i.e. the shortest path from s to v.

Single-source shortest path problem : Djikstra's algorithm

Input: G a digraph having source s, n vertices and positive weight function ν

Output: minimal lengths of paths from s to each vertex of G

1: V is the set of vertices of G except for s
2: initialize s with $\lambda_s = 0$, each of its successors v with $\lambda_v = \nu(s, v)$
and all other v with $\lambda_v = +\infty$
3: **while** $V \neq \emptyset$ **do**
4: find v such that λ_v is minimal among elements in V
5: **for all** arrows (v, v_j) **do**
6: **if** $\lambda_j > \lambda_v + \nu(v, v_j)$ **then**
7: $\lambda_j \leftarrow \lambda_v + \nu(v, v_j)$
8: **end if**
9: **end for**
10: $V \leftarrow V \setminus \{v\}$
11: **end while**
12: **return** the list of λs with the corresponding vertices

Building a tree of shortest paths can be adapted from what was done previously in the case of Ford's algorithm.

Validity and Complexity of Djikstra's algorithm

A quick analysis of Djikstra's algorithm, as given here, shows it has complexity $O(|V|^2)$, where V is the set of vertices of G. Indeed, the first loop is taken $|V|$ times, computing the minimum λ_v and going through all your adjacent vertices during each loop, gives a final complexity of $O\Big(|V|\big(|V|+|A|\big)\Big)$ where A is the set of arrows of G.

Remark : Implementation of Djikstra's algorithm can drop complexity to $O\big(|V|\lg|V|+|A|\big)$. This is not very used in practice since involved constants can be huge.

Validity

Since all arrows are positively weighted, the minimal potential length λ_v chosen at each step of line 4 in Djikstra's algorithm would not have changed if we had run Ford's algorithm.

All-pairs shortest path problem : Floyd-Warshall algorithm

We are looking for shortest paths between any two pairs of vertices in a given digraph G with weighted arrows. Such paths exist as long as cycles are not negatively valued.

Number the vertices of G from 1 to n. Let w_{ij}^k be the minimal weight a path from i to j has, assuming intermediate vertices are in $\{1,\dots,k\}$. If there is no such path $w_{ij} = +\infty$. If P is a path joining i to j and having intermediate vertices in $\{1,\dots,k\}$ then

- either P does not go through k
- or P goes only once through k (only positive valued cycles); it is the concatenation of a path from i to k and one from k to j, only supported along $\{1,\dots,k-1\}$.

This means that

$$w_{ij}^k = \min\left(w_{ij}^{k-1}, w_{ik}^{k-1} + w_{kj}^{k-1}\right)$$

All-pairs shortest path problem : Floyd-Warshall algorithm

Input: W the (generalized) adjacency matrix of a digraph G having weighted arrows and only positve weighted cycles

Output: minimal (weighted) lengths of paths between any two vertices of G

1: **for** $k \in \{1, \dots, n\}$ **do**
2: **for** $i \in \{1, \dots, n\}$ **do**
3: **for** $j \in \{1, \dots, n\}$ **do**
4: $W[i,j] \leftarrow \min\Big(W[i,j], W[i,k] + W[k,j]\Big)$
5: **end for**
6: **end for**
7: **end for**
8: **return** W

Building up shortest paths using output of Floyd-Warshall algorithm

The output is a matrix Π where entry at position (i, j) gets a predecessor of j along a shortest path from i to j. To get a full shortest path from i to j read the matrix smartly.

Input: Matrix W of minimal lengths between any two pairs of vertices of G
Output: A matrix Π giving at entry (i, j) a predecessor of j along a shortest path from i to j

```
1:  for j ∈ {1, ..., n} do
2:      for i ∈ {1, ..., n} do
3:          for all arrows (v, j) do
4:              if W[i, j] ≠ +∞ and W[i, j] = W[i, v] + W[v, j] then
5:                  Π[i, j] ← v
6:              end if
7:          end for
8:      end for
9:  end for
10: return Π
```

This is all we'll learn on this topic!