All-Pairs Shortest Path

November 12, 2014

1 Introduction

As many things in the history of analysis of algorithms the all-pairs shortest path has a long history (From the point of view of Computer Science). We can see the initial results from the book *"Studies in the Economics of Transportation*" by Beckmann, McGuire, and Winsten (1956) where the notation that we use for the matrix multiplication alike was first used. However, if want to see something quite older, we can look at the studies in classic graph theory by

- G. Tarry, Le probl'eme des labyrinthes, Nouvelles Annales de Math´ematiques (3) 14 (1895) 187–190 [English translation in: N.L. Biggs, E.K. Lloyd, R.J. Wilson, Graph Theory 1736–1936, Clarendon Press, Oxford, 1976, pp. 18–20].
- Chr. Wiener, Ueber eine Aufgabe aus der Geometria situs, Mathematische Annalen 6 (1873) 29–30, 1873.
- N.L. Biggs, E.K. Lloyd, R.J. Wilson, Graph Theory 1736–1936, Clarendon Press, Oxford, 1976 (**For the theory behind depth-first search techniques**).

2 Possible solutions using

For the problem of finding all the pairs shortest path in a directed graph $G = (V, E)$, we could use some of the classical algorithms $|V|$times:

- **If all the weights are non-negative use the Dijkstra**
  - This has, using Fibonacci Heaps, $O\left(V^2 \log V + VE\right)$ complexity.
  - Which is equal $O\left(V^3\right)$ in the case of $E = O(V^2)$, but with a hidden large constant c.
- **If negative weights are allowed use the Bellman-Ford**
  - Then, we have $O\left(V^2 E\right)$.

- Which is equal $O\left(V^4\right)$ in the case of $E = O(V^2)$

However, it is possible to obtain better performances. Actually, the first time an algorithm solves a problem in polynomial, normally it does not have that good performance. Nevertheless, as in any algorithm development, the people developing that algorithm little by little point to new versions that can have much better performances.

3 Matrix Representation

First, we decide to use a matrix representation of our problem because of the similarities of the first method to a matrix multiplication. For this, we use the following weight values:

$$w_{ij} = \begin{cases} 0 & \text{if } i = j \\ w(i,j) & \text{if } i \neq j \text{ and } (i,j) \in E \\ \infty & \text{if } i \neq j \text{ and } (i,j) \notin E \end{cases} \tag{1}$$

Then, we have

$$W = \begin{pmatrix} w_{11} & w_{22} & \dots & w_{1k-1} & w_{1n} \\ . & . & & & . \\ . & & . & & . \\ . & & & . & . \\ w_{n1} & w_{n2} & \dots & w_{nn-1} & w_{nn} \end{pmatrix}. \tag{2}$$

And, we take the assumption that

- There are not negative weight cycles.

4 Matrix Multiplication Alike Algorithm

Now, using this basic representation, we will develop a dynamic programming solution by simply looking at the Corollary of Lemma 24.1. This Corollary allows us to have following decomposition:

$$i \overset{p'}{\rightsquigarrow} k \rightarrow j \Longrightarrow \delta(i,j) = \delta(i,k) + w_{kj} \tag{3}$$

Thus, if we define the following variable:

- $l_{ij}^{(m)}$ =minimum weight of any path from i to j, it contains at most m edges

Think about this the variable can contain values of paths with less than m edges. Thus, $l_{ij}^{(m)}$ is actually a recursive variable, this could mean that

$$l_{ij}^{(m)} = \begin{cases} \min_k \left\{ l_{ik}^{(m-1)} + w_{kj} \right\} & \text{if you have } m \text{ edges} \\ l_{ij}^{(m-1)} & \text{if you have } m-1 \text{ edges} \end{cases} \tag{4}$$

Using this ideas we design the following recursion:

- Thus, we have that for paths with $m = 0$ edges

$$l_{ij}^{(0)} = \begin{cases} 0 & \textbf{if } i = j \\ \infty & \textbf{if } i \neq j \end{cases}$$

- For m>0, we have that

$$\begin{aligned} l_{ij}^{(m)} &= \min\left(l_{ij}^{(m-1)}, \min_{1 \leq k \leq n} \left\{ l_{ik}^{(m-1)} + w_{kj} \right\} \right) \\ &= \min_{1 \leq k \leq n} \left\{ l_{ik}^{(m-1)} + w_{kj} \right\} \end{aligned}$$

  Why? A simple notation problem.

$$l_{ij}^{(m)} = l_{ij}^{(m)} + 0 = l_{ij}^{(m)} + w_{jj}$$

Before, we get the iterative version of this recursion, it is necessary to know the value of $\delta(i, j)$, and if it is not going to change through the updating process. This can be simply verified by the following equality:

$$\delta(i, j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)} = l_{ij}^{(n+2)} = ...$$

Using this fact, we have then

- The matrix $L^{(m)} = \left(l_{ij}^{(m)} \right)$ contains the paths with at most m edges.
- Then, we can compute first $L^{(1)}$ then compute $L^{(2)}$ all the way to $L^{(n-1)}$ which contains the actual shortest paths because a shortest path in the graph can contain at most *n-1* vertices.

But, wait a minute What is $L^{(1)}$?

- First, we have that $L^{(1)} = W$, since $l_{ij}^{(1)} = w_{ij}$.

Thus, the iterative all shortest path algorithm is :

```
    **Extended-Shortest-Path**$(L, W)$
1.      $n = L.rows$
2.      **let** $L' = \left( l'_{ij} \right)$ **be a new** $n \times n$
```

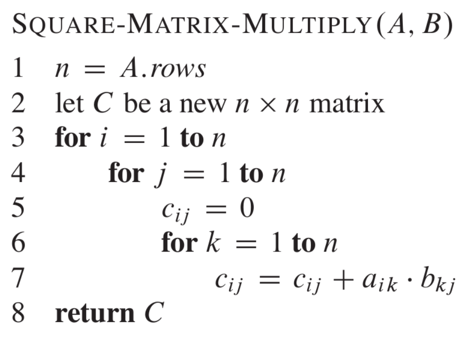
```
3.    for $i = 1$ to $n$
4.        for $j = 1$ to $n$
5.            $l'_{ij} = \infty$
6.            for $k = 1$ to $n$
7.                $l'_{ij} = \min\left(l'_{ij}, l_{ik} + w_{kj}\right)$
8.    return $L'$
```

5 Look Alike Matrix Multiplication Operations

If we think on the mapping of symbols into a different set of symbols, we can interpret certain operations in the all-path algorithm as:

- $L \Longrightarrow A$
- $W \Longrightarrow B$
- $L' \Longrightarrow C$
- $\min \Longrightarrow +$
- $+ \Longrightarrow \cdot$
- $\infty \Longrightarrow 0$

This observation comes from the following (Algorithm 1).



```
SQUARE-MATRIX-MULTIPLY($A, B$)
1  $n = A.rows$
2  let $C$ be a new $n \times n$ matrix
3  for $i = 1$ to $n$
4      for $j = 1$ to $n$
5          $c_{ij} = 0$
6          for $k = 1$ to $n$
7              $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$
8  return $C$
```

Figure 1: Square-Matrix-Multiply(A, B)

We can then have the following final solution for all the paths

$$\begin{aligned}
L^{(1)} = & \quad L^{(0)} \cdot W = & W \\
L^{(2)} = & \quad L^{(1)} \cdot W = & W^2 \\
& \quad \vdots & \\
L^{(n-1)} = & \quad L^{(n-2)} \cdot W = & W^{n-1}
\end{aligned}$$

The final algorithm looks like :

Slow-All-Pairs-Shortest-Paths(W)

```
1.  $n \leftarrow W.rows$
2.  $L^{(1)} \leftarrow W$
3.  for $m = 2$ to $n - 1$
4.      $L^{(m)} \leftarrow$EXTEND-SHORTEST-PATHS$\left(L^{(m-1)}, W\right)$
5.  return $L^{(n-1)}$
```

If $n = V$ this algorithm has a complexity $O\left(V^4\right)$. However, this algorithm can be faster by simply observing that

$$\begin{aligned}
L^{(1)} & \quad = & W \\
L^{(2)} = & \quad W \cdot W = & W^2 \\
L^{(4)} = & \quad W^2 \cdot W^2 = & W^4 \\
L^{(8)} = & \quad W^4 \cdot W^4 = & W^8 \\
& \quad \vdots &
\end{aligned}$$

Thus:

$$2^{\lceil \lg(n-1) \rceil} \geq n - 1 \Longrightarrow L^{\left(2^{\lceil \lg(n-1) \rceil}\right)} = L^{(n-1)}$$

Now, we have

Slow-All=Pairs-Shortest-Paths(W)

```
1.  $n \leftarrow W.rows$
2.  $L^{(1)} \leftarrow W$
3.  $m \leftarrow 1$
4.  while $m < n - 1$
5.      $L^{(2m)} \leftarrow$EXTEND-SHORTEST-PATHS$\left(L^{(m)}, L^{(m)}\right)$
6.      $m \leftarrow 2m$
7.  return $L^{(m)}$
```

<2->Complexity

If $n = V$ we have that $O\left(V^3 \lg V\right)$.

6 A different dynamic-programming algorithm

As in any dynamic programming problem we are required to prove the optimal substructure of the problem.

6.1 The Shortest Path Structure and the Recursion

We know from previous examples that for the problem:

- For a shortest path $p = \langle v_1, v_2, ..., v_l \rangle$, an **intermediate vertex** is any vertex of p other than v_1 or v_l. Then, $v_1 \rightsquigarrow p$ and $p \rightsquigarrow v_l$ are shortest paths.

Thus, we can do the following definition trick:

- $d_{ij}^{(k)}$ =weight of a shortest path between i and j with all intermediate vertices are in the set $\{1, 2, ..., k\}$.

We have two cases

1. If k is not an intermediate vertex. Then a shortest path from vertex i to vertex j with all intermediate vertices in the set $\{1, 2, ..., k-1\}$ is also a shortest path from i to j with all intermediate vertices in the set $\{1, 2, ..., k\}$.
2. If k is an intermediate vertex, then we have that $i \overset{p_1}{\rightsquigarrow} k \overset{p_2}{\rightsquigarrow} j$ with $p_1$and p_2 shortest paths with intermediate vertices in $\{1, 2, ..., k-1\}$.

Thus we have that:

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1 \end{cases}$$

We can use the recursive version:

Algorithm 1 Recursive Version

```
Recursive-Floyd-Warshall(W)
Dn the n by n matrix
for i = 1 to n
        for j = 1 to n
                Dn[i,j] = Recursive-Part(i,j,n,W)
return Dn

Recursive-Part(i,j,k,W)
        if k=0
                return W[i,j]
        if k>=1
                temp1 = Recursive-Part(i,j,k-1,W)
                temp2 = Recursive-Part(i,k,k-1,W)+...
                                Recursive-Part(k,j,k-1,W)
                if temp1<=temp2
                        return temp1
                else
                        return temp2
```

Thus, the final iterative algorithm with complexity $O(V^3)$ looks like:

Algorithm 2 Floyd-Warshall Algorithm

```
FLOYD-WARSHALL($W$)
1 $n = W.rows$
2 $D^{(0)} = W$
3 for $k = 1$ to $n$
4     let $D^{(k)} = \left(d_{ij}^{(k)}\right)$ be a new $n \times n$ matrix
5     for $i = 1$ to $n$
6         for $j = 1$ to $n$
7             $d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$
8 return $D^{(n)}$
```

7 Re-Weigthing

Another thing that can be accomplished to avoid the problem of negative cycles is the the following:

- Given the weight function $w : E \to \mathbb{R}$, define $h : V \to \mathbb{R}$ be any function mapping vertices into real numbers.

We define

$$\widehat{w}(u,v) = w(u,v) + h(u) - h(v)$$

This function has the following lemma:

Lemma. *(Re-weighting does not change shortest paths)*
Given a weighted, directed graph $G = (D, V)$ with weight function $w : E \to \mathbb{R}$, let $h : V \to \mathbb{R}$ be any function mapping vertices to real numbers. For each edge $(u, v) \in E$, define

$$\widehat{w}\left(u,v\right) = w\left(u,v\right) + h\left(u\right) - h\left(v\right)$$

Let $p = \langle v_1, v_2, ..., v_k \rangle$ be any path from vertex 0 to vertex k. Then:

1. *p is a shortest path from 0 to k with weight function w if and only if it is a shortest path with weight function $\widehat{w}$. That is $w(p) = \delta\left(v_o, v_k\right)$ if and only if $\widehat{w}(p) = \widehat{\delta}\left(v_o, v_k\right)$.*
2. *Furthermore, G has a negative-weight cycle using weight function w if and only if G has a negative-weight cycle using weight function $\widehat{w}$.*

Proof. Case 1
We start showing that:

$$\widehat{w}\left(u,v\right) = w\left(u,v\right) + h\left(u\right) - h\left(v\right) \tag{5}$$

For this:

$$\begin{aligned}
\widehat{w}\left(p\right) &= \sum_{i=1}^{k} \widehat{w}\left(v_{i-1}, v_i\right) \\
&= \sum_{i=1}^{k} \left[w\left(v_{i-1}, v_i\right) + h\left(v_{i-1}\right) - h\left(v_i\right)\right] \\
&= \sum_{i=1}^{k} w\left(v_{i-1}, v_i\right) + h\left(v_0\right) - h\left(v_k\right) \\
&= w\left(p\right) + h\left(v_0\right) - h\left(v_k\right)
\end{aligned}$$

Now, observe that $h\left(v_0\right)$ and $h\left(v_k\right)$ do not depend on the path. Thus, $w(p) = \delta\left(v_0, v_k\right) \Longleftrightarrow \widehat{w}(p) = \widehat{\delta}\left(v_0, v_k\right)$.
Case 2
Consider any cycle $c = \langle v_0, v_1, ..., v_k \rangle$, $v_0 = v_k$. Thus

$$\widehat{w}\left(c\right) = w(c) + h\left(v_0\right) - h\left(v_k\right) = w(c) \tag{6}$$

thus c has a negative weight using w if and only if it has negative weight using $\widehat{w}$. □

Thus we have the following:

- Select h such that $w\left(u,v\right) + h\left(u\right) - h\left(v\right) \geq 0$.

Then, we build a new graph G'

- It has the following elements
  - $V' = V \cup \{s\}$**, where s is a new vertex.**
    - $E' = E \cup \{(s,v)\,|v \in V\}$**.**
    - $w(s,v) = 0$ for all $v \in V$, in addition to all the other weights.

Therefore, select h thus $h(v) = \delta(s,v)$. We have the following claim: $w\,(u,v) + h\,(u) - h\,(v) \geq 0$.

By Triangle Inequality:

$$\delta\,(s,v) \leq \delta(s,u) + w(u,v) \tag{7}$$

Then by the way we selected h, we have

$$h(v) \leq h(u) + w\,(u,v)\,.$$

Finally

$$w\,(u,v) + h\,(u) - h\,(v) \geq 0$$

Then, we have the following algorithm:

Algorithm 3 Johnson's Algorithm

```
JOHNSON($G, w$)
compute $G'$, where $G'.V = G.V \cup \{s\}$,
    $G'.E = G.E \cup \{(s,v) : v \in G.V\}$, and
    $w(s,v) = 0$ for all $v \in G.V$
if BELLMAN-FORD($G', w, s$) == FALSE
    print "the input graph contains a negative-weight cycle"
else for each vertex $v \in G'.V$
        set $h(v)$ to the value of $\delta(s,v)$
            computed by the Bellman-Ford algorithm
    for each edge $(u,v) \in G'.E$
        $\widehat{w}(u,v) = w(u,v) + h(u) - h(v)$
    let $D = (d_{uv})$ be a new $n \times n$ matrix
    for each vertex $u \in G.V$
        run DIJKSTRA($G, \widehat{w}, u$) to compute $\widehat{\delta}(u,v)$ for all $v \in G.V$
        for each vertex $v \in G.V$
            $d_{uv} = \widehat{\delta}(u,v) + h(v) - h(u)$
    return $D$
```

This algorithm has the following complexity:

- $\Theta(V+E)$ to compute G'.
- $O(VE)$ to run Bellman-Ford.
- $\Theta(E)$to compute $\widehat{w}$.

- $O\left(V^2 \lg E + VE\right)$ to run Dijkstra's algorithm $|V|$ time using Fibonacci Heaps.
- $O\left(V^2\right)$ to compute D matrix.

Thus, the total is $O(V^2 \lg V + E)$, but if $E = O\left(V^2\right) \Longrightarrow O\left(V^2 \lg V\right)$