All Pair Shortest Path Algorithms

CST302
Algorithm Analysis and Design

Department of Computer Science and Engineering
Government Engineering College Idukki

February 9, 2025

Team Members

- 06, Amritha Remesh, IDK22CS006
- 56, Veena Roy, IDK22CS062
- 57, Vidya Roy, IDK22CS063
- 59, Vivek D V, IDK22CS066

Faculty:Prof.Anish Abraham

Problem Definition

In the all pairs shortest path problem, we want to find the shortest path from every possible source to every possible destination. Specifically, for every pair of vertices u and v, we need to compute the following information:

- dist(u, v) is the length of the shortest path (if any) from u to v.

Algorithm 1: Floyd-Warshall Algorithm

- The Floyd-Warshall algorithm is a dynamic programming algorithm used to discover the shortest paths in a weighted graph, which includes negative weight cycles. [3]
- The Floyd-Warshall set of rules was advanced independently via Robert Floyd and Stephen Warshall in 1962.
- Best for small, dense graphs.

Pseudocode

```
FloydWarshall(V, E, w):
    for u ← 1 to V
        for v ← 1 to V
            dist[u][v][0] ← w(u → v)

    for r ← 1 to V
        for u ← 1 to V
            for v ← 1 to V
                if dist[u][v][r − 1] < dist[u][r][r − 1] + dist[r][v][r − 1]
                    dist[u][v][r] ← dist[u][v][r − 1]
                else
                    dist[u][v][r] ← dist[u][r][r − 1] + dist[r][v][r − 1]
```

Figure: Floyd-Warshall Algorithm

Complexity Analysis

- The Floyd-Warshall algorithm operates using three for loops to find the shortest distance between all pairs of vertices within a graph.
- The **time complexity** of the Floyd-Warshall algorithm is $\mathbf{O(n^3)}$, where ‘n’ is the number of vertices in the graph.
- The **space complexity** of the algorithm is $\mathbf{O(n^2)}$.

Algorithm 2:Johnson's Algorithm

- Johnson's all-pairs shortest path algorithm computes a cost $pi(v)$ for each vertex, so that the new weight of every edge is non-negative, and then computes shortest paths with respect to the new weights using Dijkstra's algorithm. [2]
- Best for sparse graphs with weights.

Pseudocode

```
JohnsonAPSP(V, E, w):
  ⟨⟨Add an artificial source⟩⟩
  add a new vertex s
  for every vertex v
      add a new edge s→v
      w(s→v) ← 0
  ⟨⟨Compute vertex prices⟩⟩
  dist[s, ·] ← BellmanFord(V, E, w, s)
  if BellmanFord found a negative cycle
      fail gracefully
  ⟨⟨Reweight the edges⟩⟩
  for every edge u→v ∈ E
      w′(u→v) ← dist[s, u] + w(u→v) − dist[s, v]
  ⟨⟨Compute reweighted shortest path distances⟩⟩
  for every vertex u
      dist′[u, ·] ← Dijkstra(V, E, w′, u)
  ⟨⟨Compute original shortest-path distances⟩⟩
  for every vertex u
      for every vertex v
          dist[u, v] ← dist′[u, v] − dist[s, u] + dist[s, v]
```

Figure: Johnson's Algorithm

Complexity Analysis

- The running time of this algorithm is dominated by the calls to Dijkstra's algorithm.
- We spend O(V E) time running BellmanFord once, O(V E log V) time running D V times, and O(V + E) time doing other bookkeeping.
- Thus, the overall running time is $O(VE \log V) = O(V^3 \log V)$.

Algorithm 3:Zwick's Algorithm

Let G be a directed graph, where edge weights are integers in $\{-M, \ldots, M\}$, and k be a fixed parameter. We can compute for every pair of vertices u, v, an estimate $\tilde{d}(u, v)$ such that:

$$\tilde{d}(u, v) \geq d(u, v)$$

and with high probability,

$$\tilde{d}(u, v) = d(u, v)$$

for every pair (u, v) where $\ell(u, v) \leq k$. [1]

Pseudocode

Function SHOSHAN-ZWICK-APSP($\mathbf{D}$)

$l = \lceil \log_2 n \rceil$.
$m = \log_2 M$.
for $(k = 1$ to $m + 1)$ **do**
  $\mathbf{D} = clip(\mathbf{D} \star \mathbf{D}, 0, 2 \cdot M)$.
end for
$\mathbf{A}_0 = \mathbf{D} - M$.
for $(k = 1$ to $l)$ **do**
  $\mathbf{A}_k = clip(\mathbf{A}_{k-1} \star \mathbf{A}_{k-1}, -M, M)$.
end for
$\mathbf{C}_l = -M$.
$\mathbf{P}_l = clip(\mathbf{D}, 0, M)$.
$\mathbf{Q}_l = +\infty$.
for $(k = l - 1$ down to $0)$ **do**
  $\mathbf{C}_k = [clip(\mathbf{P}_{k+1} \star \mathbf{A}_k, -M, M) \bigwedge \mathbf{C}_{k+1}] \bigvee [clip(\mathbf{Q}_{k+1} \star \mathbf{A}_k, -M, M) \bar{\bigwedge} \mathbf{C}_{k+1}]$.
  $\mathbf{P}_k = \mathbf{P}_{k+1} \bigvee \mathbf{Q}_{k+1}$.
  $\mathbf{Q}_k = chop(\mathbf{C}_k, 1 - M, M)$.
end for
for $(k = 1$ to $l)$ **do**
  $\mathbf{B}_k = (\mathbf{C}_k \geq 0)$.
end for
$\mathbf{B}_0 = (0 \leq \mathbf{P}_0 < M)$.
$\mathbf{R} = \mathbf{P}_0 \mod M$.
$\Delta = M \cdot \sum_{k=0}^{l} 2^k \cdot \mathbf{B}_k + \mathbf{R}$.
return Δ.

Figure: Zwick's Algorithm

Complexity Analysis

- The algorithm runs in O˜(M · n) time, where is the exponent for the fastest known matrix multiplication algorithm.
- This means that the running time of the algorithm depends on which matrix multiplication algorithm is used.
- The current fastest matrix multiplication algorithm is provided, where $\omega < 2.3727$.

Algorithm 4:Seidel's Algorithm

- Uses fast matrix multiplication. [4]
- **Working:**
  1. Convert adjacency matrix into a distance matrix (0-1 form).
  2. Use **iterative squaring** to refine the shortest path estimates.
  3. Extract the shortest path distances.
- Best for unweighted graphs where edges are either present or absent.

Complexity Analysis

Time Complexity: $O(V^{\omega})$, where ω is the exponent of matrix multiplication (≈ 2.376). Best for unweighted graphs.

Comparison of APSP Algorithms

Algorithm	**Best for**	**Time Complexity**	**Space Complexity**	**Handles Negative Weights?**	**Approach**
Floyd-Warshall	Small, dense graphs	$O(V^3)$	$O(V^2)$	Yes	Dynamic Programming
Johnson's Algorithm	Sparse graphs with weights	$O(V^2 \log V + VE)$	$O(V^2)$	Yes	Bellman-Ford + Dijkstra
Zwick's Algorithm	Large-scale graphs	$O(V^{2.376})$ (Matrix Multiplication)	$O(V^2)$	No	Matrix Multiplication (Strassen's Algorithm)
Seidel's Algorithm	Unweighted graphs	$O(V^\omega), \omega < 2.3727$	$O(V^2)$	No	Fast Matrix Multiplication

References

[1] Pavlos Eirinakis and Matthew Williamson. "On the Shoshan-Zwick Algorithm for the All-Pairs Shortest Path Problem". In: *Journal of Graph Algorithms and Applications* 21 (2016). DOI: 10.7155/jgaa.00410.

[2] Jeff Erickson. *All-Pairs Shortest Paths*. Lecture Notes, University of Illinois at Urbana-Champaign. Unknown. URL: https://jeffe.cs.illinois.edu/teaching/algorithms/book/09-apsp.pdf.

[3] Manuel E. Moreno M. *All-Pairs Shortest Paths*. Lecture Notes, University of Western Ontario. Unknown. URL: https://www.csd.uwo.ca/~mmorenom/HPC-Slides/12-apsp.pdf.

[4] Raimund Seidel. "On the All-Pairs-Shortest-Path Problem". In: *SIAM Journal on Computing* 22.4 (1992), pp. 745–749. DOI: 10.1137/0222049.

The End