All-Pairs-Shortest-Paths in Spark

Charles Zheng, Jingshu Wang and Arzav Jain

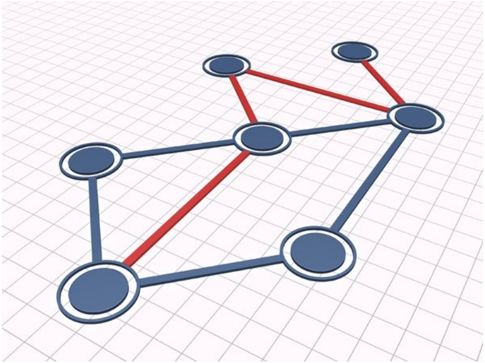
Stanford University

May 31, 2015

Problem

- Weighted graph $G = (V, E)$ with n vertices
- Compute $n \times n$ matrix of distances S where

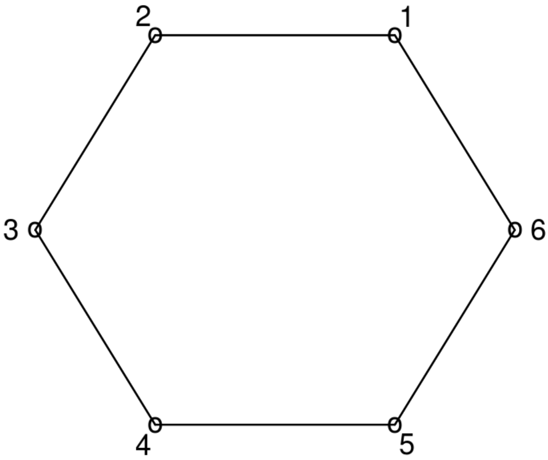
$$S_{ij} = \text{weight of shortest path from } i \text{ to } j$$



Floyd-Warshall

Initial input

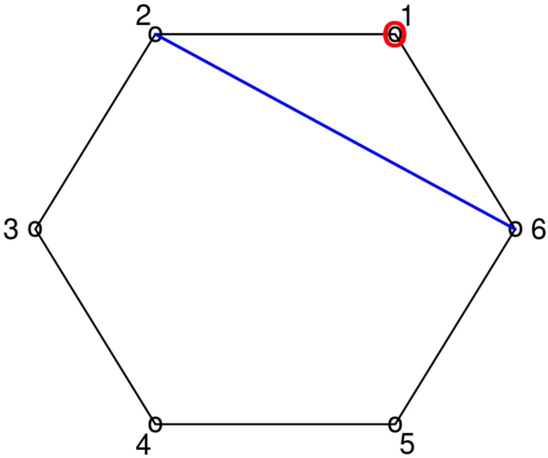
$$\begin{pmatrix} 0 & 1 & & & & 1 \\ 1 & 0 & 1 & & & \\ & 1 & 0 & 1 & & \\ & & 1 & 0 & 1 & \\ & & & 1 & 0 & 1 \\ 1 & & & & 1 & 0 \end{pmatrix}$$



Floyd-Warshall

Iteration 1

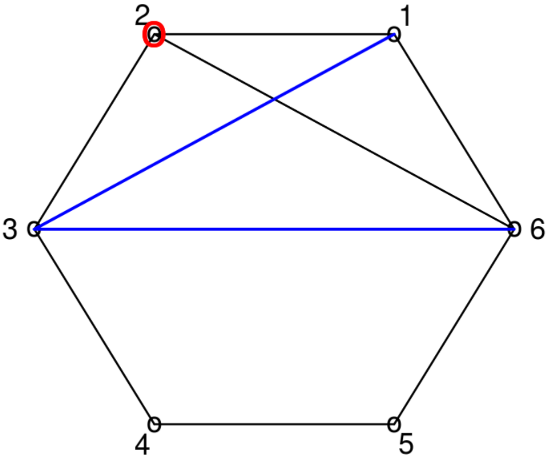
$$\begin{pmatrix} 0 & 1 & & & & 1 \\ 1 & 0 & 1 & & & \color{blue}{\mathbf{2}} \\ & 1 & 0 & 1 & & \\ & & 1 & 0 & 1 & \\ & & & 1 & 0 & 1 \\ 1 & \color{blue}{\mathbf{2}} & & & 1 & 0 \end{pmatrix}$$



Floyd-Warshall

Iteration 2

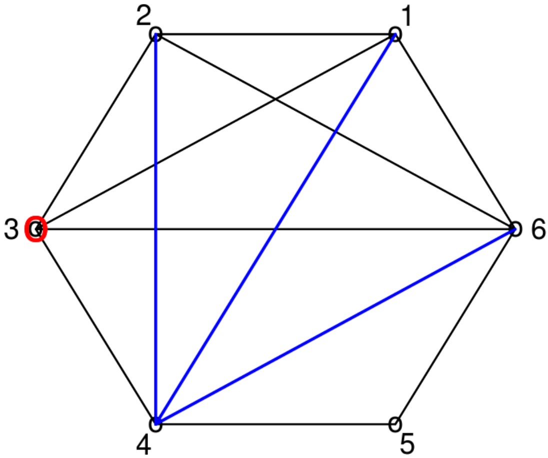
$$\begin{pmatrix} 0 & 1 & \mathbf{2} & & & 1 \\ 1 & 0 & 1 & & & 2 \\ \mathbf{2} & 1 & 0 & 1 & & \mathbf{3} \\ & & 1 & 0 & 1 & \\ & & & 1 & 0 & 1 \\ 1 & 2 & \mathbf{3} & & 1 & 0 \end{pmatrix}$$



Floyd-Warshall

Iteration 3

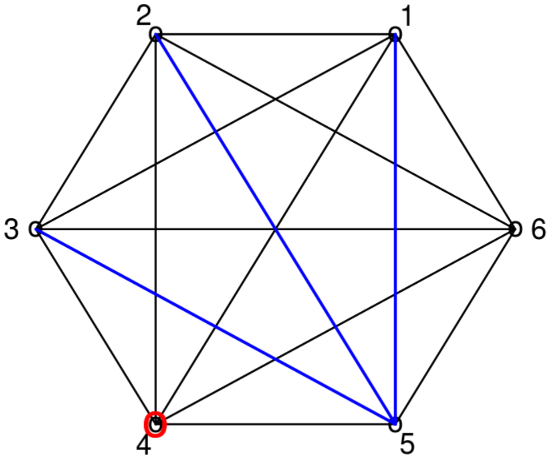
$$\begin{pmatrix} 0 & 1 & 2 & \mathbf{3} & & 1 \\ 1 & 0 & 1 & \mathbf{2} & & 2 \\ 2 & 1 & 0 & 1 & & 3 \\ \mathbf{3} & \mathbf{2} & 1 & 0 & 1 & \mathbf{4} \\ & & & 1 & 0 & 1 \\ 1 & 2 & 3 & \mathbf{4} & 1 & 0 \end{pmatrix}$$



Floyd-Warshall

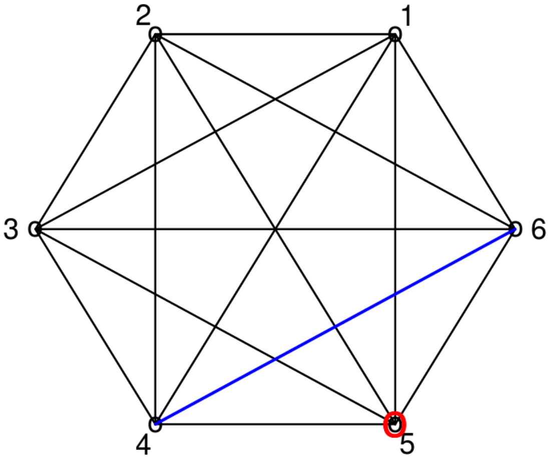
Iteration 4

$$\begin{pmatrix} 0 & 1 & 2 & 3 & \mathbf{4} & 1 \\ 1 & 0 & 1 & 2 & \mathbf{3} & 2 \\ 2 & 1 & 0 & 1 & \mathbf{2} & 3 \\ 3 & 2 & 1 & 0 & 1 & 4 \\ \mathbf{4} & \mathbf{3} & \mathbf{2} & 1 & 0 & 1 \\ 1 & 2 & 3 & 4 & 1 & 0 \end{pmatrix}$$



Iteration 5

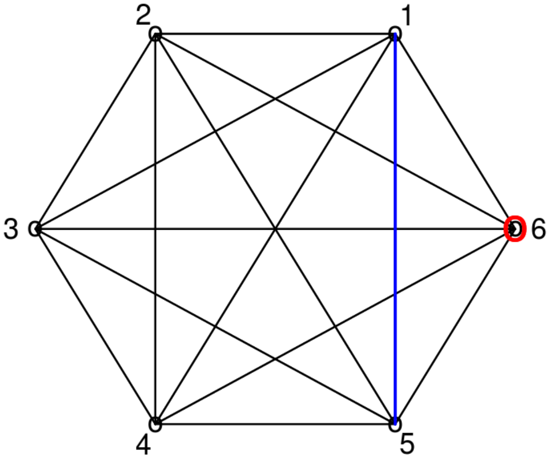
$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 1 \\ 1 & 0 & 1 & 2 & 3 & 2 \\ 2 & 1 & 0 & 1 & 2 & 3 \\ 3 & 2 & 1 & 0 & 1 & \mathbf{2} \\ 4 & 3 & 2 & 1 & 0 & 1 \\ 1 & 2 & 3 & \mathbf{2} & 1 & 0 \end{pmatrix}$$



Floyd-Warshall

Iteration 6, *(terminate)*

$$\begin{pmatrix} 0 & 1 & 2 & 3 & \color{blue}{\mathbf{2}} & 1 \\ 1 & 0 & 1 & 2 & 3 & 2 \\ 2 & 1 & 0 & 1 & 2 & 3 \\ 3 & 2 & 1 & 0 & 1 & 2 \\ \color{blue}{\mathbf{2}} & 3 & 2 & 1 & 0 & 1 \\ 1 & 2 & 3 & 2 & 1 & 0 \end{pmatrix}$$



Floyd-Warshall

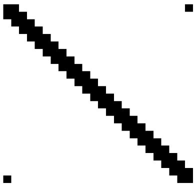
- Cost: $O(n^3)$ operations (single-core)
- Takes *n sequential* iterations

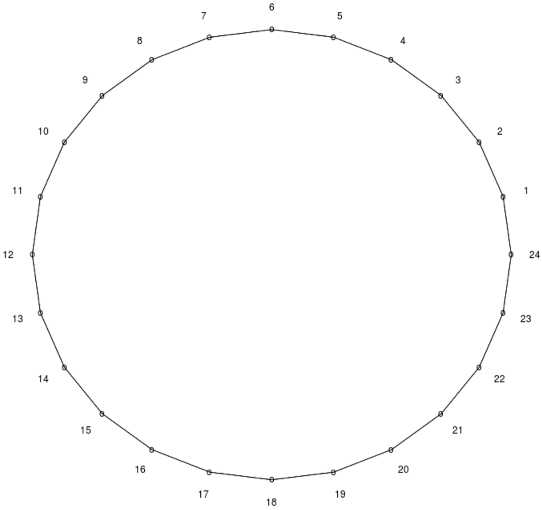
Problems with Floyd-Warshall

- FW updates using 1 vertex at a time
- Result: *n* iterations
- High # iterations = *latency* in distributed setting
- Solomonik et al. (2013) show how to “block” FW iterates
- We modify their block-based approach for Spark

Block APSP

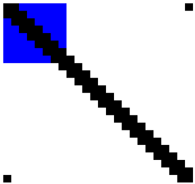
Initial input: $n = 24$, *block size* $= 8$

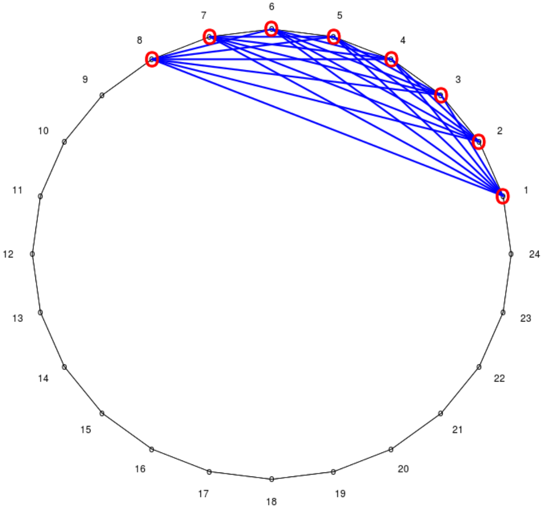




Block APSP

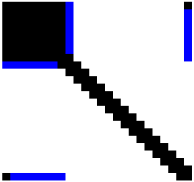
Iteration 1A: Update all paths within block 1 (with FW)

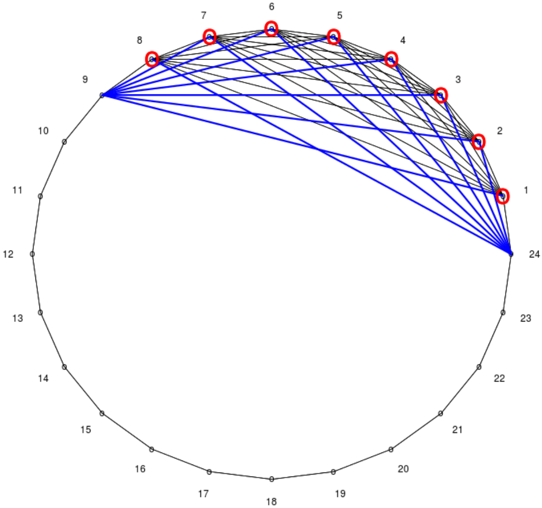




Block APSP

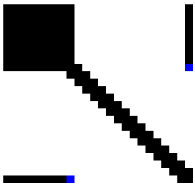
Iteration 1B: Update all paths to/from block 1

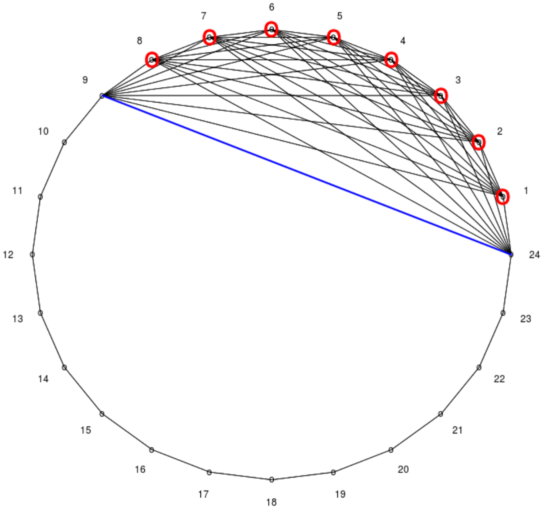




Block APSP

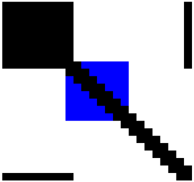
Iteration 1C: Update all paths

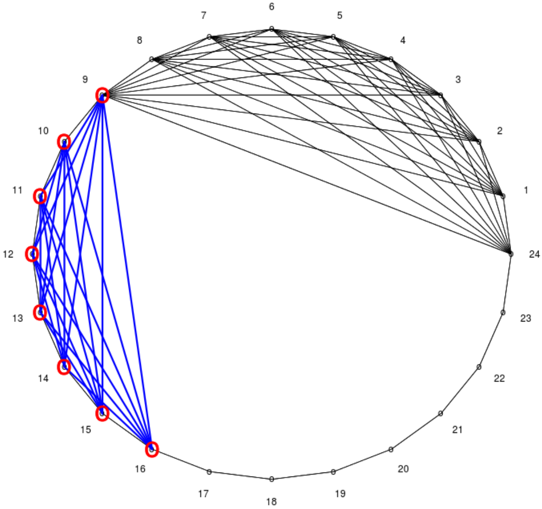




Block APSP

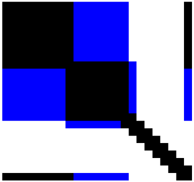
Iteration 2A: Update all paths within block 2 (with FW)

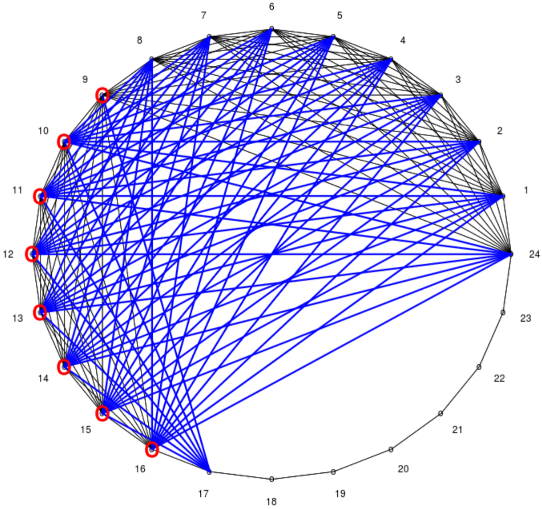




Block APSP

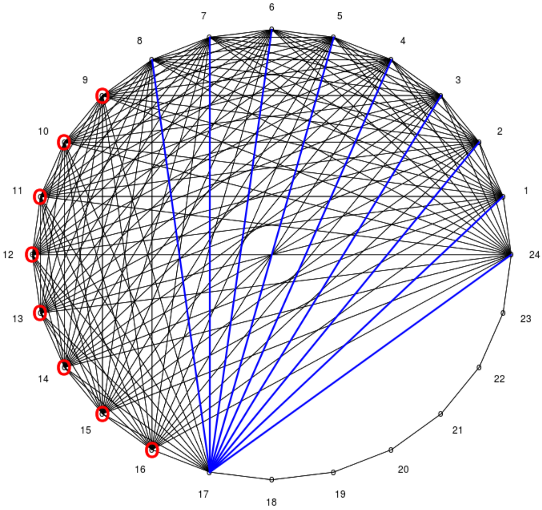
Iteration 2B: Update all paths to/from block 2





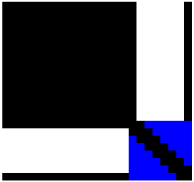
Block APSP

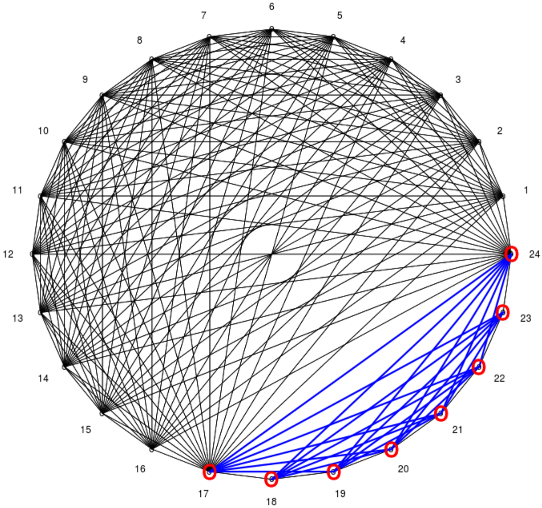
Iteration 2C: Update all paths



Block APSP

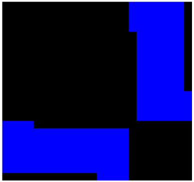
Iteration 3A: Update all paths within block 3 (with FW)

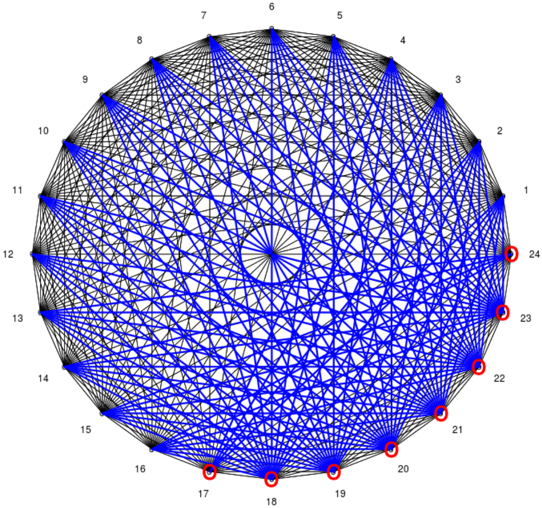




Block APSP

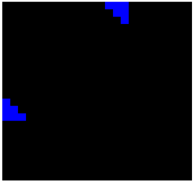
Iteration 3B: Update all paths to/from block 3

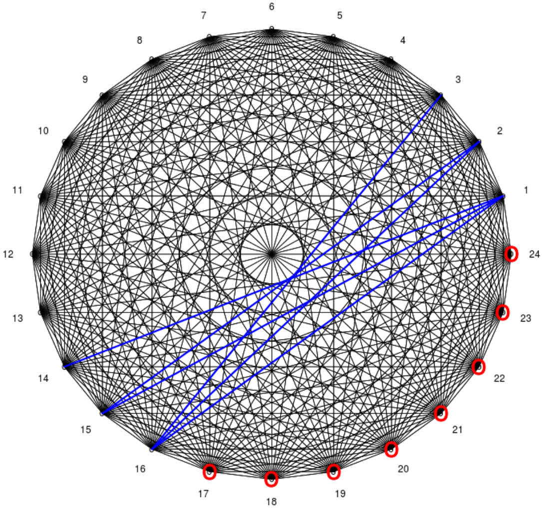




Block APSP

Iteration 3C: Update all paths, *(terminate)*





Block APSP: Single-core

- Block size b, n/b iterations
- A-step (all paths within block): $O(b^3)$
- B-step (all paths to/from block): $O(nb^2)$
- C-step (all paths): $O(n^2 b)$
- Iteration: $O(n^2 b + nb^2 + b^3)$
- Total: $O(\frac{n}{b}(n^2 b + nb^2 + b^3)) = O(n^3 + n^2 b + nb^2)$
- The case $b = 1$ is almost the same as Floyd-Warshall

Distributing Block APSP

Problem setup

- Input format: Given by *dense* adjacency matrix, stored as `BlockMatrix` S with block size b
- Output format: same
- Let $S[i,j]$ denote the i,jth block of S
- Each worker holds k contiguous blocks
- Set block size b so that $k\frac{n^2}{b^2}$ fits in memory

Scaling

- Number of vertices $n \to \infty$
- Number of workers $p = C_w n^2$
- Block size b, blocks per worker k are constant

Distributing Block APSP

For $i = 1, \ldots, n/b$

- A-step: *(update all paths within block)*
  - `filter` + `collect` diagonal submatrix $S[i, i]$
  - Driver computes X = Floyd-Warshall(S) locally
  - Communication cost: $O(b^2)$
- B-step: *(update all paths to/from block)*
  - `filter` + `mapValues`: update b rows and columns using X
  - One-to-all communication (broadcast X)
  - Communication cost: $O(b^2 \log(p))$
- C-step: *(update all paths)*
  - `flatMap` to duplicate b rows/columns and `join` to send updated rows/columns to every block
  - `map` to update each block with updated rows/columns
  - All-to-all communication (join)
  - Communication cost: $O(kb^2 p \log p)$

Distributing Block APSP

Overall cost:

- Total computational cost is $O(n^3)$, divided evenly among workers
- Total communication cost: $O(nb(kp+1)\log p)$
- Total latency/synchronization cost: $O(\frac{n}{b}\log(p))$

Optimal performance when b is as large as possible, e.g.

$$b = \sqrt{\frac{\text{RAM}}{3}}$$

Hence should be much better than Floyd-Warshall ($b = 1$)

Results