All-Pairs-Shortest-Paths in Spark

Charles Zheng, Jingshu Wang and Arzav Jain

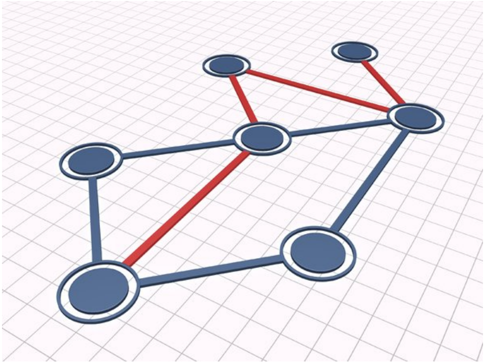
Stanford University

June 1, 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: Single Core

S_{ij}^k - shortest path distance from i to j using intermediate nodes 1 to k

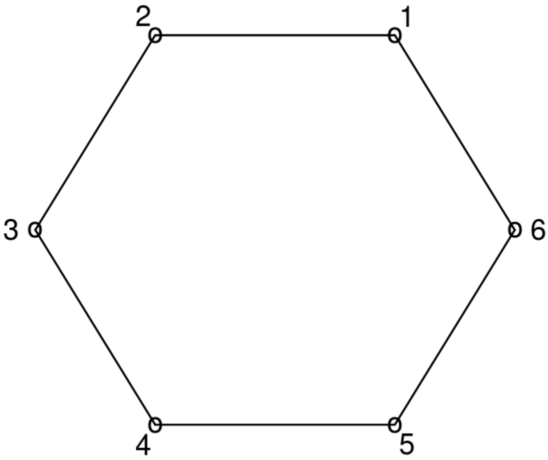
$$S_{ij}^k = \begin{cases} w_{ij} & k = 0 \\ min(S_{ij}^{k-1}, S_{ik}^{k-1} + S_{kj}^{k-1}) & k > 0 \end{cases}$$

$$S \leftarrow min(S, S(:, k) \otimes S(k, :))$$

Floyd-Warshall

Initial input

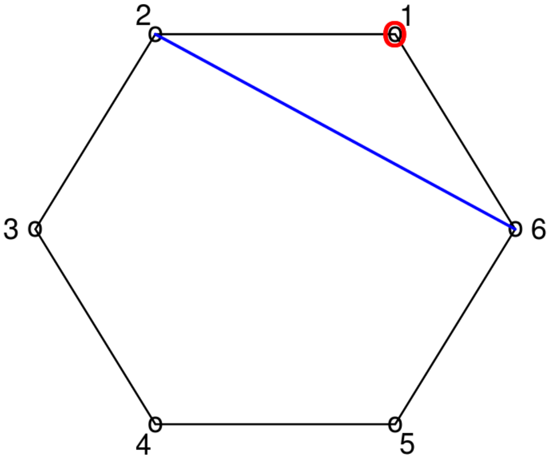
$$\begin{pmatrix} 0 & 1 & & & & 1 \\ 1 & 0 & 1 & & & \\ & 1 & 0 & 1 & & \\ & & 1 & 0 & 1 & \\ & & & 1 & 0 & 1 \\ 1 & & & & 1 & 0 \end{pmatrix}$$



Floyd-Warshall

Iteration 1

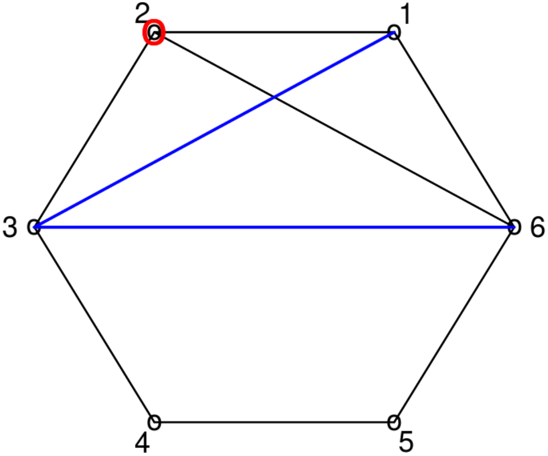
$$\begin{pmatrix} 0 & 1 & & & & 1 \\ 1 & 0 & 1 & & & \mathbf{\color{blue}2} \\ & 1 & 0 & 1 & & \\ & & 1 & 0 & 1 & \\ & & & 1 & 0 & 1 \\ 1 & \mathbf{\color{blue}2} & & & 1 & 0 \end{pmatrix}$$



Floyd-Warshall

Iteration 2

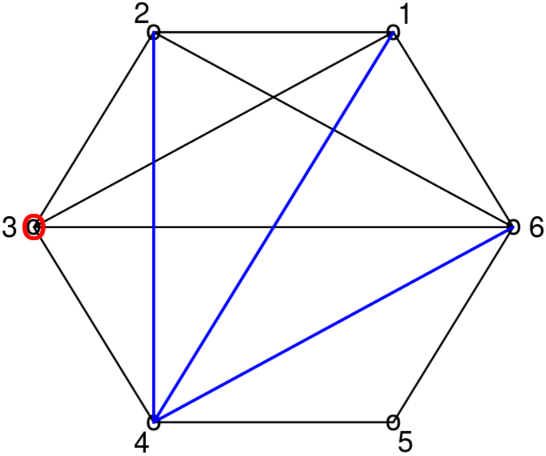
$$\begin{pmatrix} 0 & 1 & \mathbf{2} & & & 1 \\ 1 & 0 & 1 & & & 2 \\ \mathbf{2} & 1 & 0 & 1 & & \mathbf{3} \\ & & 1 & 0 & 1 & \\ & & & 1 & 0 & 1 \\ 1 & 2 & \mathbf{3} & & 1 & 0 \end{pmatrix}$$



Floyd-Warshall

Iteration 3

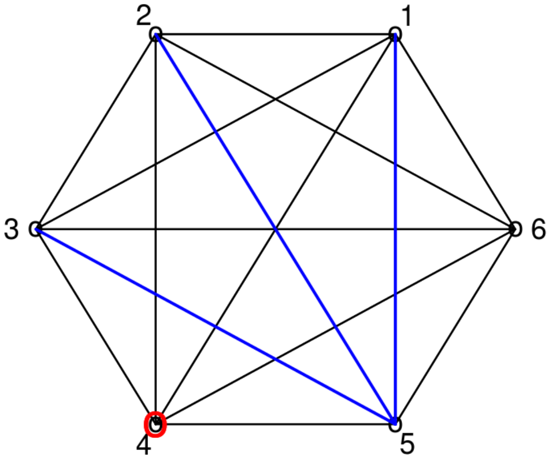
$$\begin{pmatrix} 0 & 1 & 2 & \mathbf{3} & & 1 \\ 1 & 0 & 1 & \mathbf{2} & & 2 \\ 2 & 1 & 0 & 1 & & 3 \\ \mathbf{3} & \mathbf{2} & 1 & 0 & 1 & \mathbf{4} \\ & & & 1 & 0 & 1 \\ 1 & 2 & 3 & \mathbf{4} & 1 & 0 \end{pmatrix}$$



Floyd-Warshall

Iteration 4

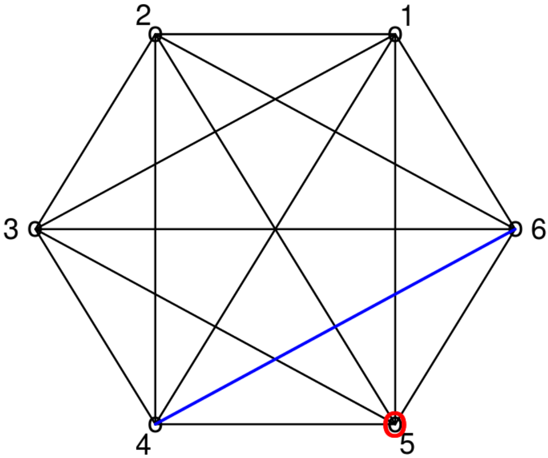
$$\begin{pmatrix} 0 & 1 & 2 & 3 & \mathbf{\color{blue}4} & 1 \\ 1 & 0 & 1 & 2 & \mathbf{\color{blue}3} & 2 \\ 2 & 1 & 0 & 1 & \mathbf{\color{blue}2} & 3 \\ 3 & 2 & 1 & 0 & 1 & 4 \\ \mathbf{\color{blue}4} & \mathbf{\color{blue}3} & \mathbf{\color{blue}2} & 1 & 0 & 1 \\ 1 & 2 & 3 & 4 & 1 & 0 \end{pmatrix}$$



Floyd-Warshall

Iteration 5

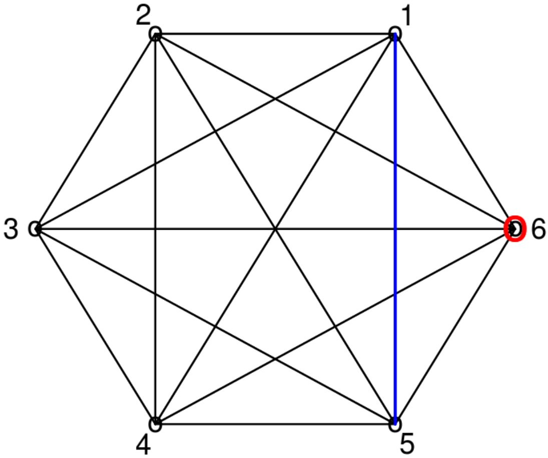
$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 1 \\ 1 & 0 & 1 & 2 & 3 & 2 \\ 2 & 1 & 0 & 1 & 2 & 3 \\ 3 & 2 & 1 & 0 & 1 & \mathbf{2} \\ 4 & 3 & 2 & 1 & 0 & 1 \\ 1 & 2 & 3 & \mathbf{2} & 1 & 0 \end{pmatrix}$$



Floyd-Warshall

Iteration 6, *(terminate)*

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



Floyd-Warshall

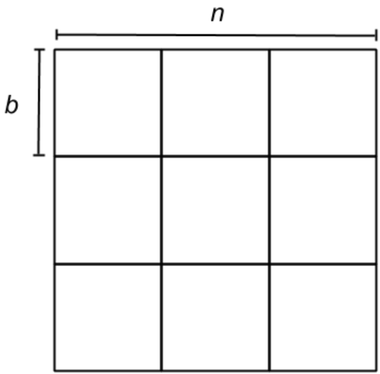
- Cost: $O(n^3)$ operations (single-core)
- Takes *n sequential* iterations

Problems with Floyd-Warshall

- FW updates by considering 1 new 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

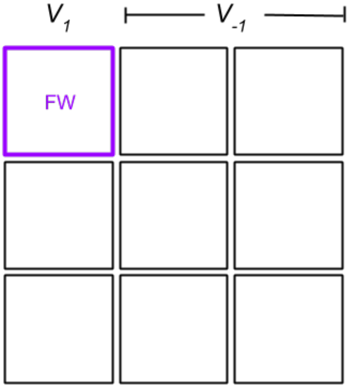
Block APSP

Number of vertices $= n$, Block Size $= b$



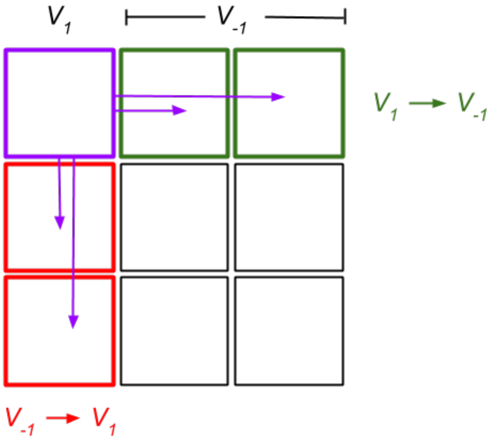
Block APSP

Iteration 1A: Compute APSP within V_1 (block 1 on diagonal)



Block APSP

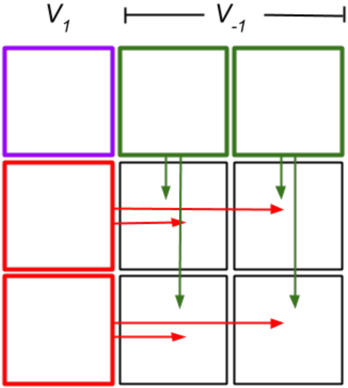
Iteration 1B: Update weights of all paths to/from V_1



Block APSP

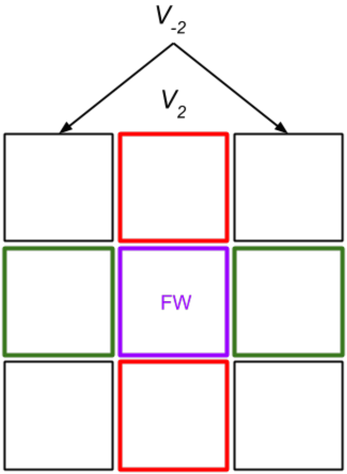
Iteration 1C: Update weights of all paths starting and ending in V_{-1} using

$$S_{ij} \leftarrow min(S_{ij}, S_{ik} \otimes S_{kj}) \quad \text{where } k = 1$$



Block APSP

Iteration 2: Do the same for block 2 on the diagonal



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 through block): $O(n^2 b)$
- 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

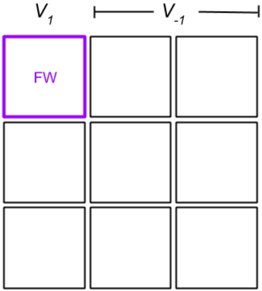
Problem setup

- Input format: Given by *dense* adjacency matrix, stored as `BlockMatrix` S with block size b
- Number of vertices n is large
- Output format: same
- Each block fits in memory

Distributing Block APSP

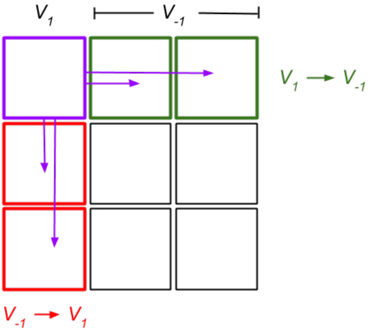
For $i = 1, \ldots, n/b$

- A-step: *(update all paths within block)*
  - One-to-one communication
  - Computation $O(b^3)$
  - Bandwidth $O(b^2)$
  - Runtime $O(b^3)$



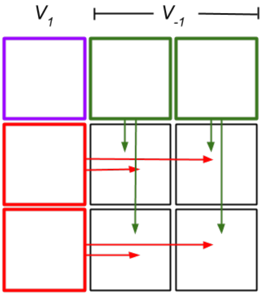
Distributing Block APSP

- B-step: *(update all paths to/from block)*
  - One-to-all communication
  - Computation per worker: $O(nb^2/\sqrt{p})$
  - Bandwidth $O(b^2\sqrt{p})$
  - Runtime $O(\log(p)b^2 + b^2 n/\sqrt{p})$



Distributing Block APSP

- C-step: *(update all paths through block)*
  - All-to-all communication
  - Computation per worker: $O(n^2 b/p)$
  - Bandwidth: $O(nb\sqrt{p})$
  - Runtime: $O(n^2 b/p + nb)$



Distributing Block APSP

Overall cost:

- Total computational cost is $O(n^3 + n^2 b)$ divided evenly among workers plus $O(nb^2)$ on driver
- Total communication cost: $O(n^2\sqrt{p})$
- Total runtime: $O(\frac{n^3}{p} + \frac{n^2 b}{\sqrt{p}} + n^2 + nb^2 + nb\log(p))$

Ignoring latency, optimal $b = 1$
With latency, runtime is

$$\frac{n}{b}L + K\left(nb^2 + \left(n\log(p) + \frac{n^2}{\sqrt{p}}\right)b + \frac{n^3}{p} + n^2\right)$$

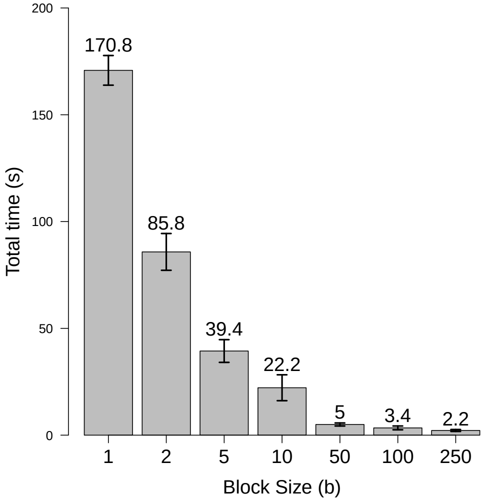
so $b \neq 1$ may be optimal

More Implementation details on Spark

- Grid Partitioner
- Checkpointing

Results

$n = 500$, $p = 4$; Local[4] 8GB



Thank you!