All-Pairs Shortest Paths in Spark

Charles Y. Zheng and Jingshu Wang
Department of Statistics
Stanford University
Stanford, CA 94305
{snarles, jinshuw}@stanford.edu

Arzav Jain
Department of Computer Science
Stanford University
Stanford, CA 94305
arzavj@cs.stanford.edu

1 Path finding in distributed APSP

The output of the original algorithm is a matrix of shortest distances $S \in \mathbb{R}^{n\times n}$ where each S_{ij} is the shortest distance from node i to j. Here we want to add a path lookup function `FindPath(i, j)` which returns for a pair of node (i, j) the shortest path itself from node i to node j.

There are three approaches to consider:

- Calculate `FindPath(i, j)` directly from the distance matrix S
- Store one midpoint for each (i, j) pair as a matrix $M \in \mathbb{R}^{n\times n}$ in the distributed block APSP algorithm, and then calculate `FindPath(i, j)` from M
- For each (i, j) pair, store two midpoints m_1, m_2 in a three-dimensional array $M \in \mathbb{R}^{n\times n\times 2}$ in the distributed block APSP algorithm, and then calculate `FindPath(i, j)` from M.

Both the first two approaches need n iterations in calculating `path(i, j)` for the worst case, while we will show in this session that the third approach guarantees the number of iterations to be at most $\lceil \log_2 n\rceil$ with properly chosen midpoints.

1.1 Creteria for choosing the midpoints

For an (i, j) pair with its shortest path $i \to k_1 \to k_2 \to \cdots \to k_{L-1} \to j$, define its path length as $l_{ij} = L$. We require the midpoints $m_1 = M_{ij1}$ and $m_2 = M_{ij2}$ to satisfy

1. $m_1, m_2 \in \{i, k_1, k_2, \cdots, k_{L-1}, j\}$
2. $l_{ij} = l_{im_1} + l_{m_1m_2} + l_{m_2j}$
3. $\max(l_{im_1}, l_{m_1m_2}, l_{m_2j}) \le \max(l_{ij}/2, 1)$

If M satisfies the above creteria, then the number of iterations in the lookup function `path(i, j)` will be at most $\lceil \log_2 n\rceil$. More details can be found in **??**.

1.2 Algorithm for updating the midpoints in distributed APSP

The initialization of M is

$$M^{(0)}_{ij1} = M^{(0)}_{ij2} = \begin{cases} i & \text{if } (i \to j) \in E \text{ or } i = j \\ \star & \text{if } (i \to j) \notin E \end{cases}$$

where $\star \notin V$ is some symbol to denote an invalid midpoint. To properly update midpoints in our distributed block APSP algorithm, we need to store and update another three-dimensional array $W \in \mathbb{R}^{n\times n\times 3}$ which stores for each (i, j) pair and midpoints (m_1, m_2) the current path lengths l_{im_1}, $l_{m_1m_2}$ and l_{m_2j}. The initialization of W is

$$W^{(0)}_{ij1} = W^{(0)}_{ij2} = \begin{cases} 0 & \text{if } (i \to j) \in E \text{ or } i = j \\ \infty & \text{if } (i \to j) \notin E \end{cases}$$

$$W_{ij3}^{(0)} = \begin{cases} 1 & \text{if } (i \to j) \in E \text{ or } i = j \\ \infty & \text{if } (i \to j) \notin E \end{cases}$$

For a path $i \to \cdots \to j$, denote $v_{ij} = (m_1, m_2, l_{im_1}, l_{m_1m_2}, l_{m_2j})$. Then for joining two paths $i \to \cdots \to k$ and $k \to \cdots \to j$, we define the following function $\texttt{MERGE}(v_{ik}, v_{kj}, k)$ to get v_{ij} for the joint path $i \to \cdots \to k \to \cdots \to j$:

Algorithm 1 Merge midpoints of two adjacent paths

function $\texttt{MERGE}(v_1 = (m_1, m_2, l_1, l_2, l_3), v_2 = (m_4, m_5, l_4, l_5, l_6), m_3)$
    $l = \sum_{i=1}^{6} l_i$
    for $t = 1, 2, 3, 4$ **do**
        if $\sum_{i=1}^{t} l_i \leq l/2$ & $\sum_{i=1}^{t+1} l_i \geq l/2$ **then**
            Break
        end if
    end for
    Return $v = (m_j, m_{t+1}, \sum_{i=1}^{t} l_i, l_{t+1}, \sum_{i=t+2}^{6} l_i)$
end function

We call $v = (m_1, m_2, l_1, l_2, l_3)$ as *flat* if

$$max(l_1, l_2, l_3) \leq \max(1, (l_1 + l_2 + l_3)/2) < \infty$$

Lemma 1.1. *If v_{ik} and v_{kj} are flat and $i \neq k \neq j$, then $v = \texttt{MERGE}(v_{ik}, v_{kj}, k)$ is also flat.*

Proof. Let $v_{ik} = (m_1, m_2, l_1, l_2, l_3)$, $v_{kj} = (m_4, m_5, l_4, l_5, l_6)$, $k = m_3$ and $l = \sum_{s=1}^{6} l_s$. From $i \neq k \neq j$, we have $l \geq 2$.

As v_{ik} and v_{kj} are *flat*, we have $l_1 \leq \max(1, (l_1 + l_2 + l_3)/2) \leq l/2$ and similarly $l_6 \leq l/2$. Thus, there exists $t \in \{1, 2, 3, 4\}$ that both $\sum_{s=1}^{t} l_s \leq l/2$ and $\sum_{s=1}^{t+1} l_s \geq l/2$ holds. Also $l_{t+1} \leq \max(1, (l_1 + l_2 + l_3)/2, (l_4 + l_5 + l_6)/2) \leq l/2$, thus $v = \texttt{MERGE}(v_{ik}, v_{kj}, k)$ is also *flat*. $\square$

We can now modify the original distributed block APSP algorithm to include updating W and M.

Given an $n \times m$ distance matrix A and an $n \times m$ distance matrix B together with the midpoints matrices (W^A, M^A) and (W^B, M^B), define a minimum operation as $(C, W^C, M^C) = \min_P \big((A, W^A, M^A), (B, W^B, M^B)\big)$ by

$$C_{ij} = \min(A_{ij}, B_{ij})$$

$$(M_{ij\cdot}^C, W_{ij\cdot}^C) = \begin{cases} (M_{ij\cdot}^A, W_{ij\cdot}^A) & \text{if } C_{ij} = A_{ij} \\ (M_{ij\cdot}^B, W_{ij\cdot}^B) & \text{if } C_{ij} = B_{ij} \end{cases}$$

Given an $n \times k$ distance matrix A and a $k \times m$ distance matrix B together with the midpoints matrices (W^A, M^A) and (W^B, M^B), define a *min-plus* product $(C, W^C, M^C) = (A, W^A, M^A) \otimes_P (B, W^B, M^B)$ as

$$C_{ij} = \min_{l=1}^{k} A_{il} + B_{lj}$$

$$(M_{ij\cdot}^C, W_{ij\cdot}^C) = \begin{cases} (M_{ij\cdot}^A, W_{ij\cdot}^A) & \text{if } \operatorname{argmin}_l (A_{il} + B_{lj}) = j \\ (M_{ij\cdot}^B, W_{ij\cdot}^B) & \text{if } \operatorname{argmin}_l (A_{il} + B_{lj}) = i \\ \texttt{MERGE}\big((M_{il^*\cdot}^A, W_{il^*\cdot}^A), (M_{l^*j\cdot}^B, W_{l^*j\cdot}^B), l^*\big) & \text{if } l^* = \operatorname{argmin}_l (A_{il} + B_{lj}) \neq i \text{ or } j \end{cases}$$

for $i = 1, \ldots, n$ and $j = 1, \ldots, m$.

Also, $\texttt{APSP}_P(A, M^A, W^A)$ is defined as a modified local APSP method for finding the shortest distance matrix together with the desired midpoints and path lengths matrices.

Here, we give a shorthand description of the modified distributed block APSP including updating W and M, without explicitly specifying the Spark operations.

Algorithm 2 Path-Finding Distributed Block APSP (shorthand)

```
function BLOCKAPSPATH(Adjacency matrix A given as a BlockMatrix with ℓ row blocks
and ℓ column blocks, M^(0), W^(0))
    H^(0) ← (A, M^(0), W^(0))
    for k = 1, ..., ℓ do
        [A-step]
        H^{kk(k)} ← APSP_P(H^{kk(k−1)})
        [B-step]
        for i = 1, ..., ℓ, j = 1, ..., ℓ do in parallel
            if i = k and j ≠ k then
                H^{kj(k)} ← min_P(H^{kj(k−1)}, H^{kk(k)} ⊗_P H^{kj(k−1)})
            end if
            if i ≠ k and j = k then
                H^{ik(k)} ← min_P(H^{ik(k−1)}, H^{ik(k−1)} ⊗_P H^{kk(k)})
            end if
        end for
        [C-step]
        for i = 1, ..., ℓ, j = 1, ..., ℓ do in parallel
            if i ≠ k and j ≠ k then
                H^{ij(k)} ← min_P(H^{ij(k−1)}, H^{ik(k)} ⊗_P H^{kj(k)})
            end if
        end for
        [D-step]
        if k ≡ 0 mod q then
            Checkpoint H^(k)
        end if
    end for
    Return (S, M, W) = H^(ℓ), the APSP result tuple
end function
```

1.3 The path lookup function

After obtaining the the midpoints three-dimensional array M, we can efficiently lookup the shortest path of an (i, j) pair of nodes. The lookup function returns a vector of all the other nodes in the path in order except for the starting node i. Note that if there are multiple shortest paths, the algorithm is only able to find one of them.

Algorithm 3 Lookup the path from one node to another

```
function FINDPATH(i, j)
    if i == j then
        Return NULL
    end if
    if M_ij1 == M_ij2 then
        Return j
    end if
    Return (FindPath(i, M_ij1), FindPath(M_ij1, M_ij2), FindPath(M_ij2, j))
end function
```

As $\max(l_{iM_{ij1}}, l_{M_{ij1}M_{ij2}}, l_{M_{ij2}j}) \leq \max(1, l_{ij}/2)$, the recursion depth of the above algorithm is upper bounded by $\lceil \log_2 l_{ij} \rceil$, which is at most $\lceil \log_2 n \rceil$ for any node pair in the graph.