A Multi-Source Label-Correcting Algorithm for the All-Pairs Shortest Paths Problem

Hiroki Yanagisawa
(IBM Research – Tokyo)
All-Pairs Shortest Paths (APSP) Problem

- Compute shortest path length for every pair of nodes

Input: Graph with edge lengths
- \( n = (\# \text{ nodes}) \)
- \( m = (\# \text{ edges}) \)

Output: Distance matrix

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>from</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>4</td>
<td>0</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>E</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
Repeating Dijkstra’s Algorithm

• Multiple runs of single-source shortest path algorithm
  – We often use Dijkstra’s algorithm
    • $O(mn + n \log n)$ time and $O(m + n)$ space
  – Hereafter, we call this algorithm as $n$-Dijkstra algorithm

![Diagram of a graph with nodes A, B, C, D, E and edges connecting them. The costs of the edges are labeled as 2, 3, 4, 5.]

<table>
<thead>
<tr>
<th>from</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Compute for A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>Compute for B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>Compute for C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>Compute for D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>Compute for E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>to</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Contribution

• Faster algorithm for APSP on sparse graphs
  – 10 times or more faster (with SIMD) than \( n \)-Dijkstra algorithm
  – \( O(m+n) \) working space
  – Essentially equivalent to Hilger’s centralized algorithm (given in 2007)
    • We were not aware of this algorithm (thanks to an anonymous reviewer)

• Its SIMD implementation
  – 2.3 – 3.7 times faster than scalar implementation
  – Hilger did not give SIMD implementation
  – As far as we know, first acceleration with SIMD instructions for sparse graph
    • In contrast to many SIMD implementations for dense graphs
Inefficiency of $n$-Dijkstra

- $n$-Dijkstra algorithm does not use information on the shortest paths from other source nodes.

<table>
<thead>
<tr>
<th>From</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Compute for A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>Compute for B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>Compute for C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>Compute for D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>Compute for E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Independent to each other.
Idea

• Source nodes are close to each other
  ⇒ shortest path trees are similar
  ⇒ we can efficiently compute them at the same time!
Our algorithm

- Multiple runs of multi-source shortest paths algorithm
Dijkstra’s algorithm

- Single-source shortest path

![Graph representation of Dijkstra's algorithm](image)
Extension of Dijkstra’s Algorithm

• Dijkstra’s algorithm
  – Each node is associated with single label
  – Label corresponds to distance label
  – Node with minimum label is extracted from priority queue

• Our algorithm for multi-source case
  – Each node is associated with single key label and distance label for each source node
  – Node with minimum key label is extracted from priority queue
  – Key label is set to the minimum of distance labels
Algorithm for Multi-Source Shortest Path

- Multi-source shortest paths
  - For case with two source nodes
Extension for Many Source Nodes

- Easy to extend for case # source nodes is \( B (> 2) \)
- There are tradeoffs
  - Pros: The # extraction from priority queue may decrease by a factor of \( B \)
    - Only one extraction from priority queue in best case, whereas \( B \) runs of Dijkstra’s algorithm needs \( B \) extractions from priority queue
  - Cons: Each scan operation takes \( O(B) \) time
- Our experiment shows \( B=128 \) is best
Key Selection Rule

• Any key rule outputs correct answer
  – Key label should be closeness from source nodes
  – Minimum key rule is the best one in our experiments
Label-Setting/Correcting Algorithms

- Dijkstra’s algorithm is classified as label-setting algorithm
  - Easy to analyze worst-case computation time

- Our algorithm is classified as label-correcting algorithm
  - Difficult to analyze worst-case computation time
Time Complexity

• Our algorithm terminates in finite time
• No theoretical time bound were given for
  – Minimum key rule
  – Average key rule
  – Maximum key rule
• Hilger gave worst-case running time for another key rule (minimum tentative key)
  – $O(B (m+n \log n))$ time
    • The same as $B$ runs of Dijkstra’s algorithm
  – However slower than minimum key rule (from experiments by Hilger)
SIMD implementation

- Each scan operation can be easily SIMDized

![Diagram showing SIMD implementation process]

1. \(3,4,7,7,6,5,2,1,3,\ldots,5\)
2. \(5,6,9,9,8,7,4,3,5,\ldots,7\)
3. \(4,6,9,7,4,5,4,3,5,\ldots,4\)

**B labels**

**+2 (SIMD)**

**compare (SIMD)**

**Compute minimum (SIMD)**
Our algorithm

- Multiple runs of multi-source shortest path algorithm
Graph Partitioning

- Repeating following procedure
  - Pick up a node and traverse nearby nodes
- BFS is the best (among BFS, DFS, and kNN traverses)
- Times for graph partitioning are negligible

Ex. $B = 3$
Experiment: Single-Thread

- Our algorithm clearly outperforms n-Dijkstra algorithm
- SIMD implementation accelerates scalar version 2.3 – 3.7 times

We used $B = 128$, BFS partitioning, and minimum key rule
Machine: Quad Core Xeon 3.16 GHz on Windows Server 2003
Experiment: Single-Thread (cont.)

- The acceleration is due to the decrease of # operation on priority queue
  - In best case, this number is decreased by a factor of $B$

We used $B = 128$, BFS partitioning, and minimum key rule
Machine: Quad Core Xeon 3.16 GHz on Windows Server 2003
Experiment: Multiple-Thread

- Parallel speedup with multi-thread implementation

We used $B = 128$, BFS partitioning, and minimum key rule.
Machine: Quad Core Xeon 3.16 GHz on Windows Server 2003
Improving Initializations

- Hilger suggested using better initializations yields 1 – 3 times faster running time.
Many-to-Many Shortest Paths

- It is trivial to extend our algorithm for many-to-many shortest paths problem.
Summary

• Results
  – We give fast algorithm for APSP on sparse graph and its SIMD implementation
  – First SIMD acceleration for sparse graph

• Future work
  – Thorough investigations of our algorithm for the many-to-many shortest paths problem