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

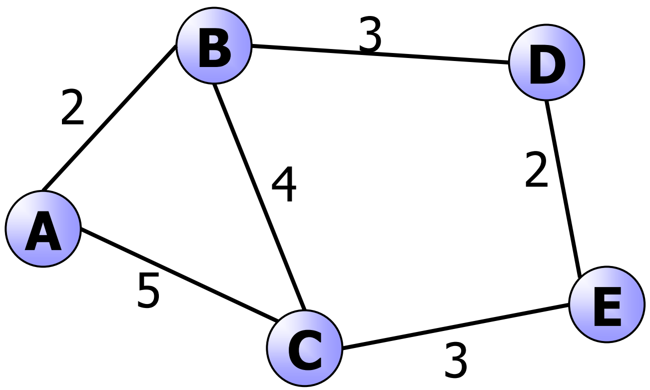
Hiroki Yanagisawa

(IBM Research – Tokyo)

All-Pairs Shortest Paths (APSP) Problem

- Compute shortest path length for every pair of nodes

Input: Graph with edge lengths
- n = (# nodes)
- m = (# edges)



Output: Distance matrix

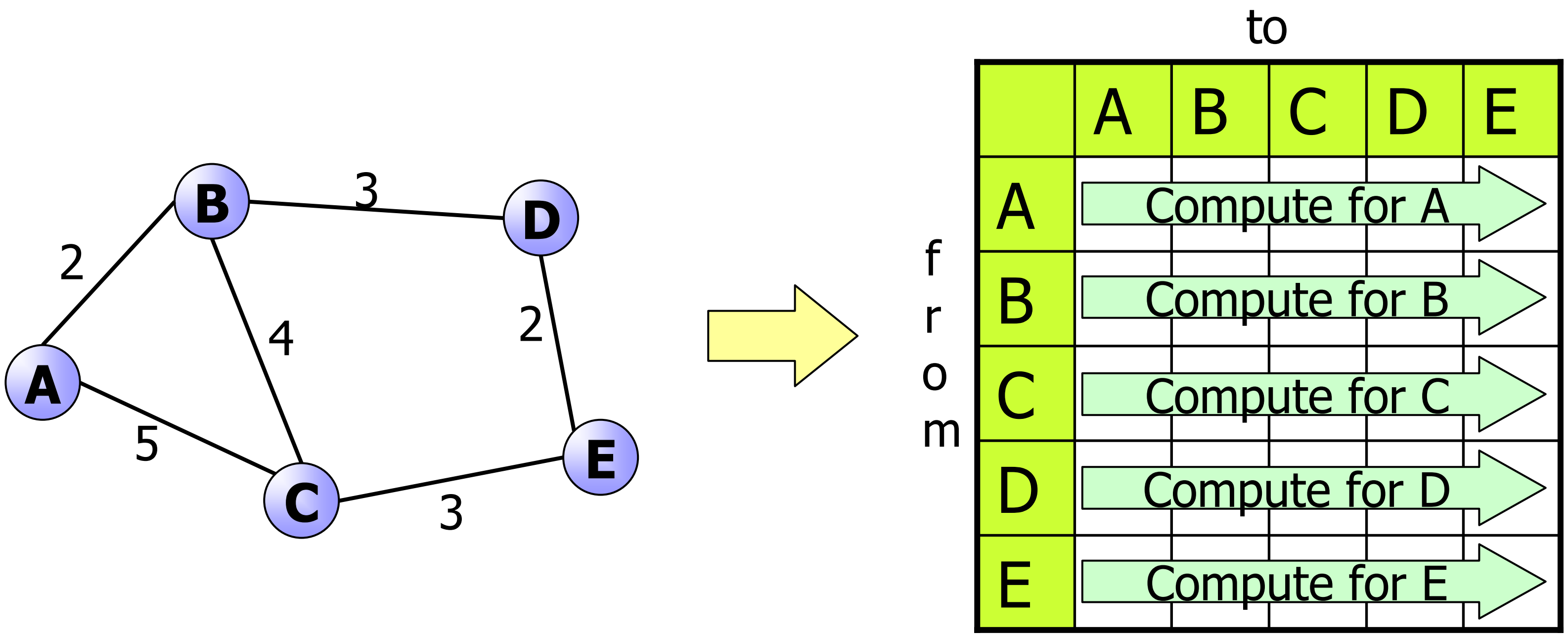
to

from

	A	B	C	D	E
A	0	2	5	5	7
B	2	0	4	3	5
C	5	4	0	5	3
D	5	3	5	0	2
E	7	5	3	2	0

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

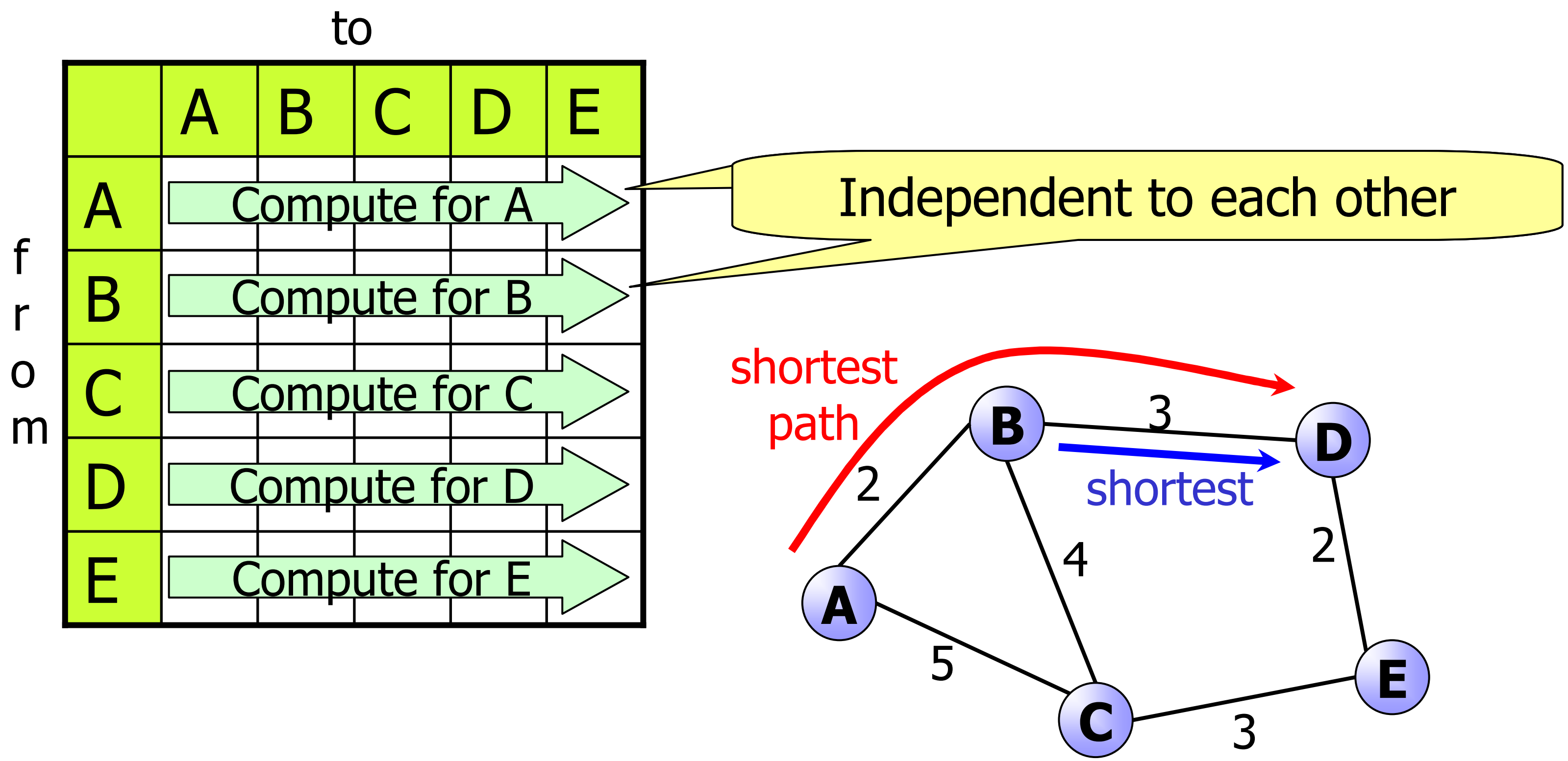


Contribution

- 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

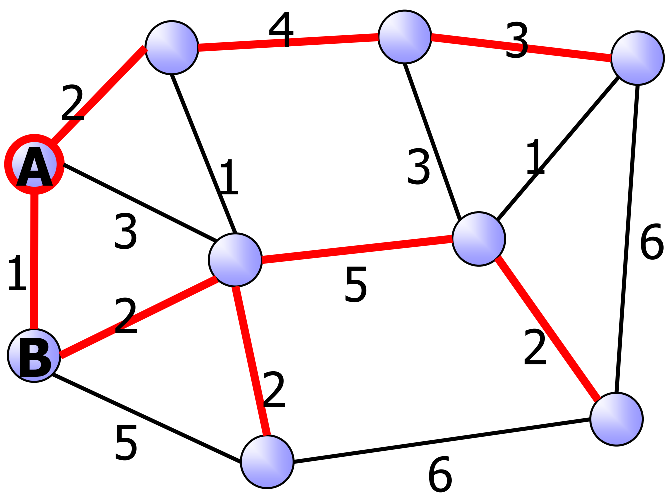


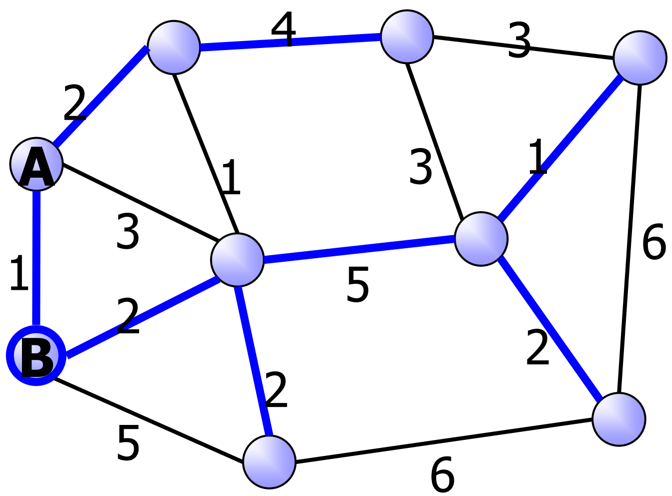
Idea

- Source nodes are close to each other

$\Rightarrow$ shortest path trees are similar

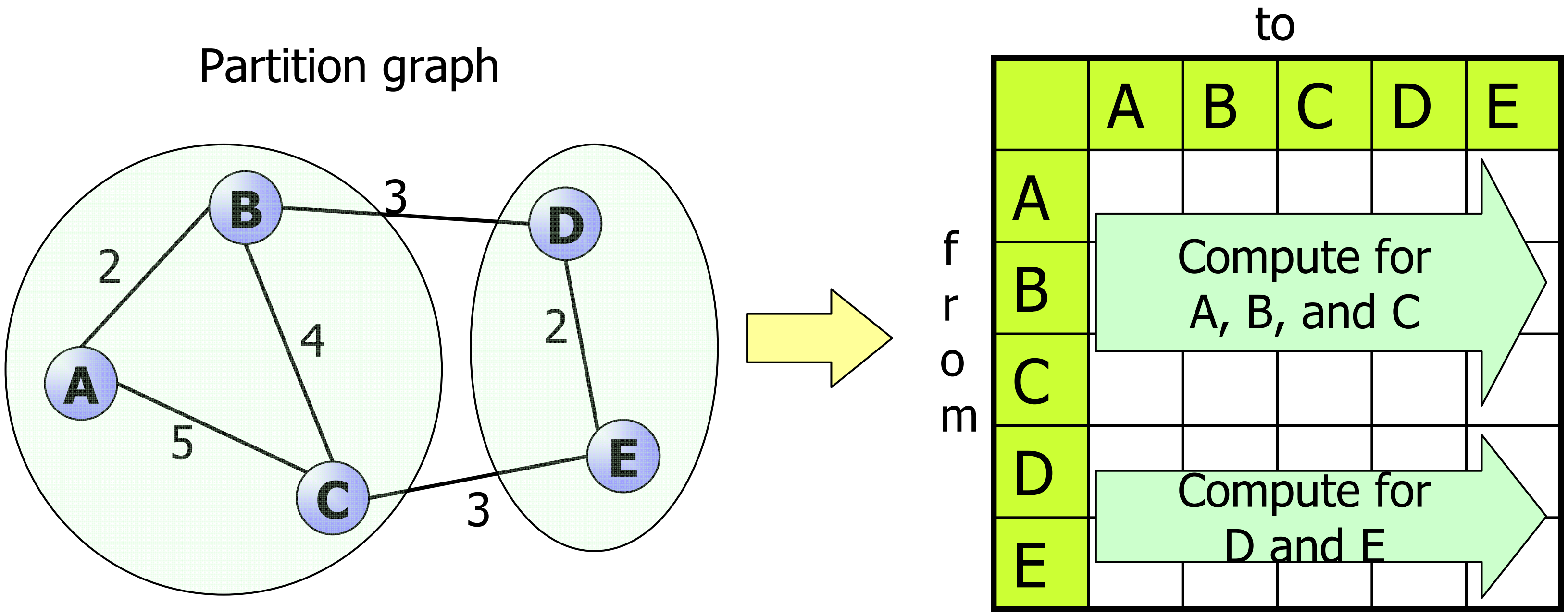
$\Rightarrow$ we can efficiently compute them at the same time!





Our algorithm

- Multiple runs of multi-source shortest paths algorithm

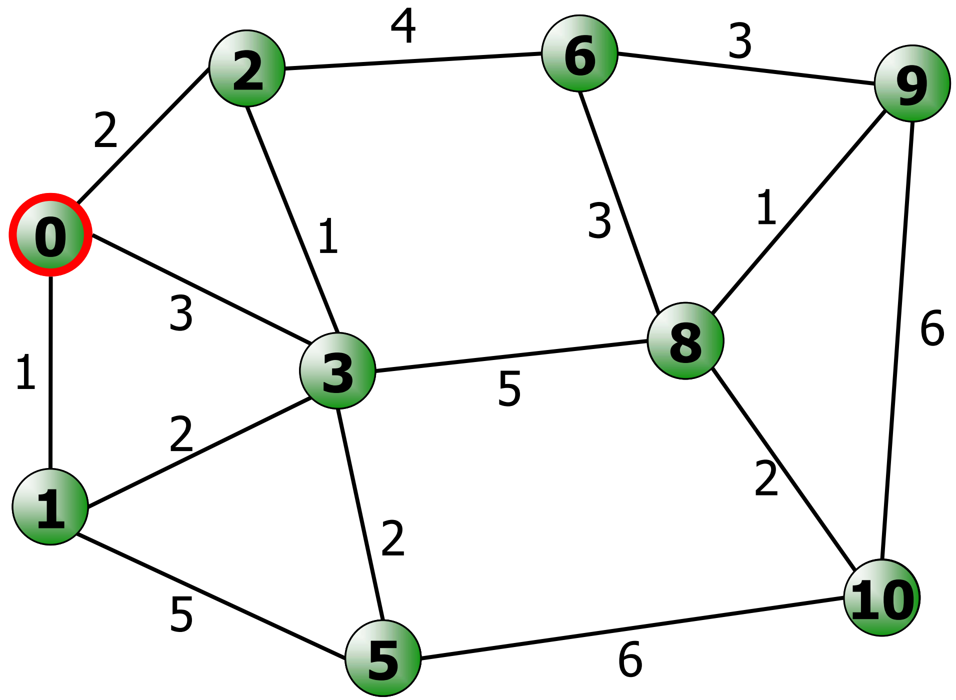


Dijkstra's algorithm

- Single-source shortest path

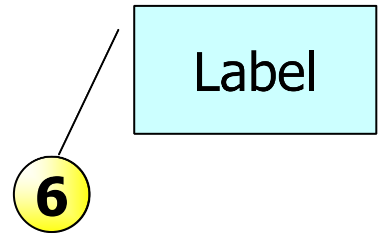
: in priority queue

: unvisited

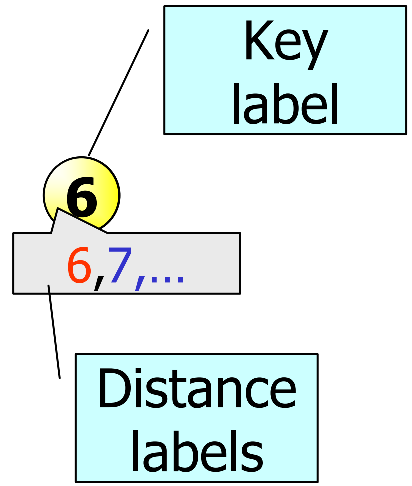


Extension of Dijkstra’s Algorithm

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

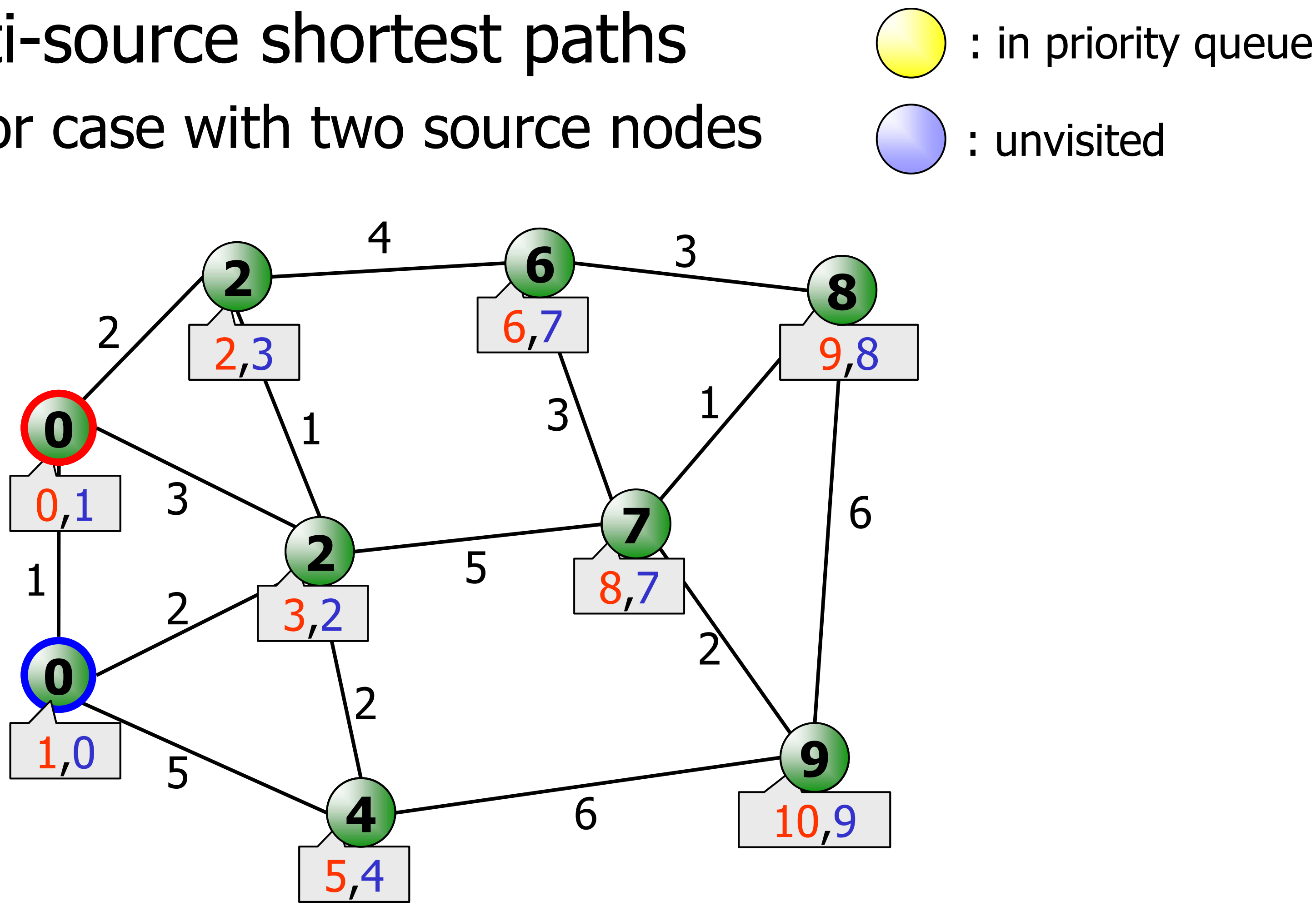


- Our algorithm for multi-source case
  - Each node is associated with single key label and distance label for each source node
  - Node with minimum key label is extracted from priority queue
  - Key label is set to the minimum of distance labels



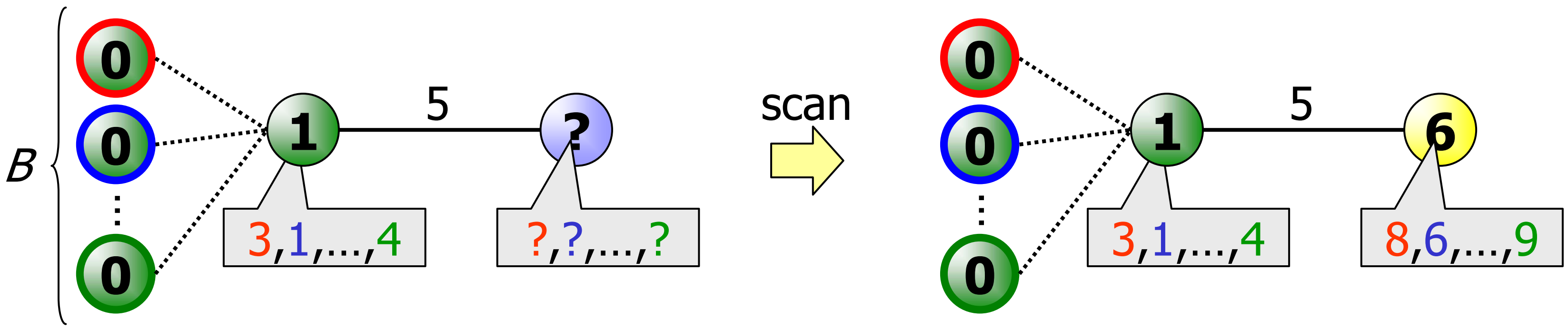
Algorithm for Multi-Source Shortest Path

- Multi-source shortest paths
  - For case with two source nodes



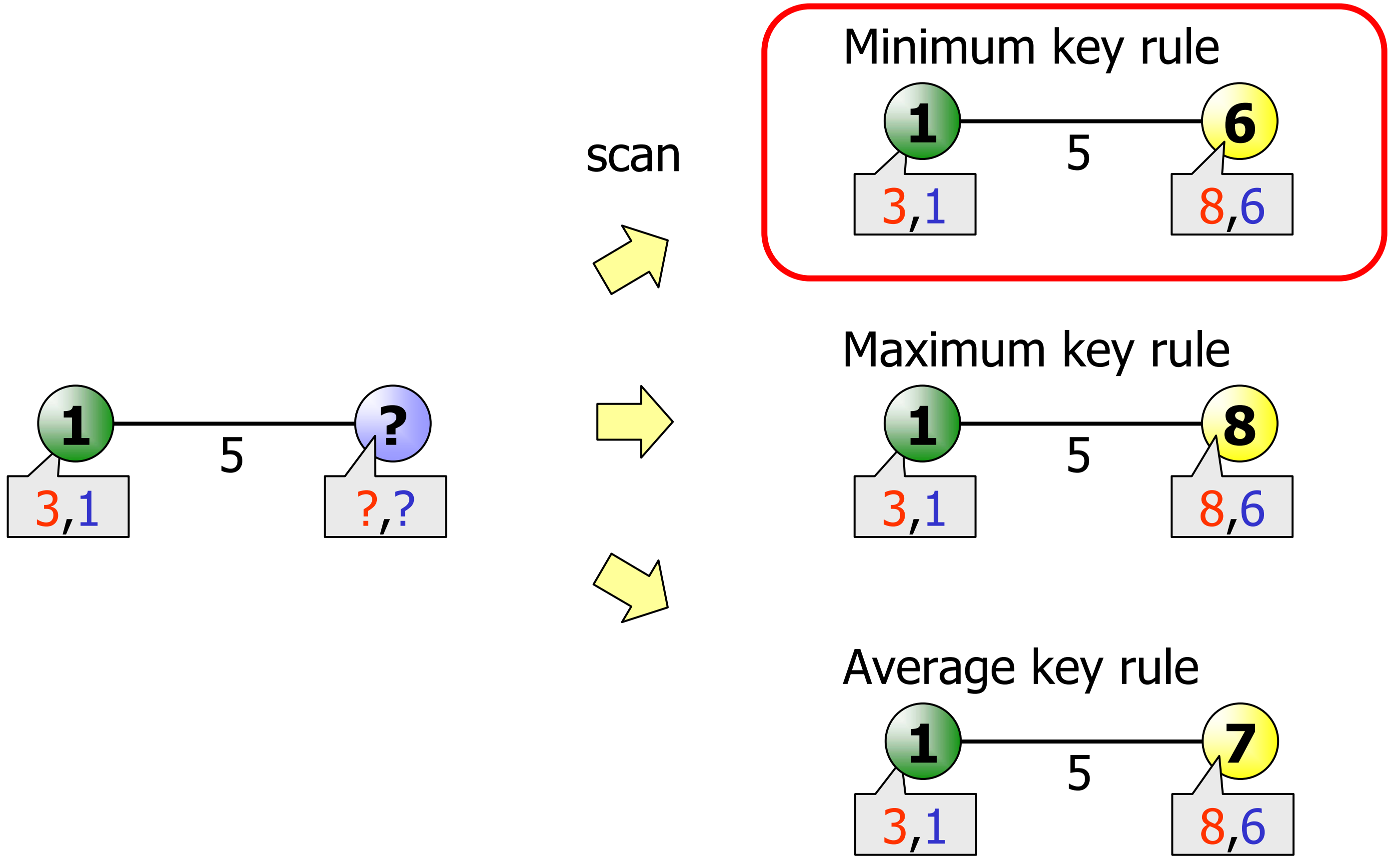
Extension for Many Source Nodes

- 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



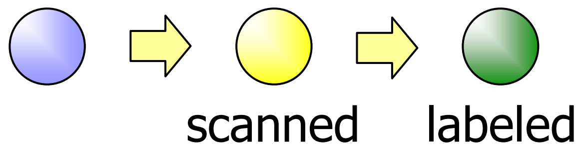
Key Selection Rule

- Any key rule outputs correct answer
  - Key label should be closeness from source nodes
  - Minimum key rule is the best one in our experiments

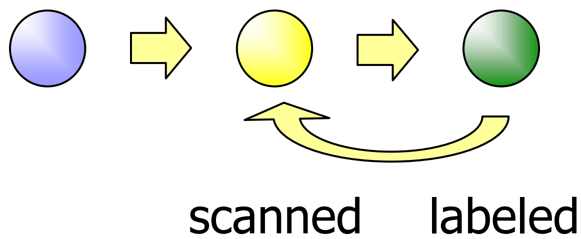


Label-Setting/Correcting Algorithms

- 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

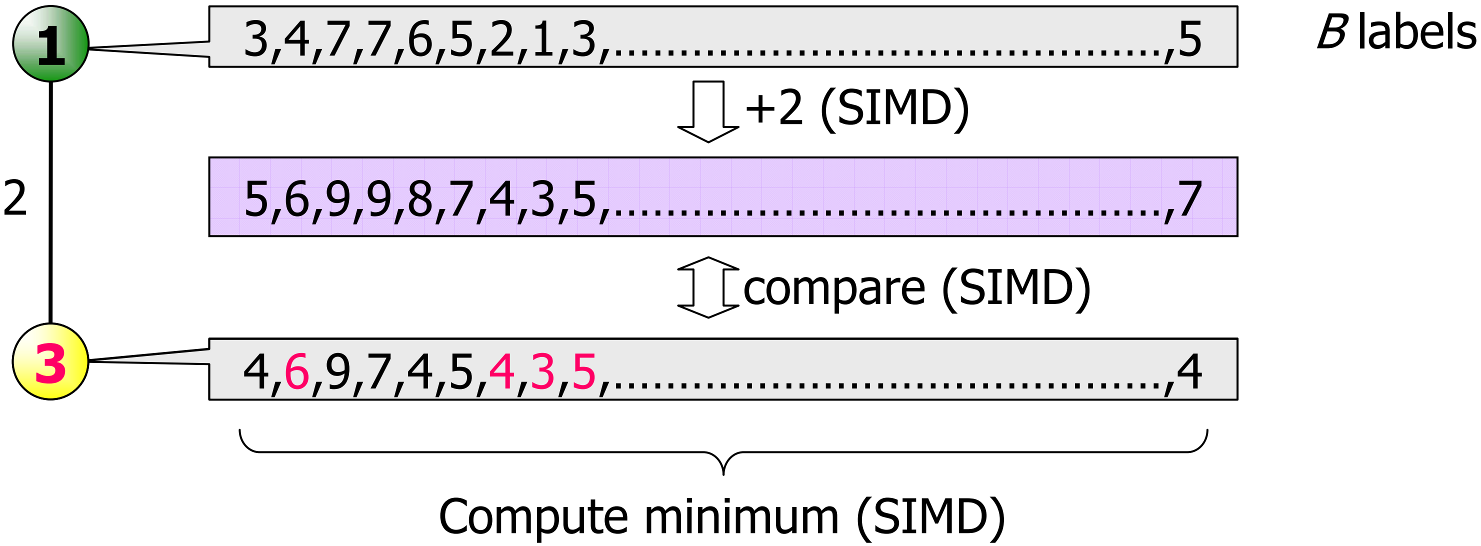


Time Complexity

- 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)

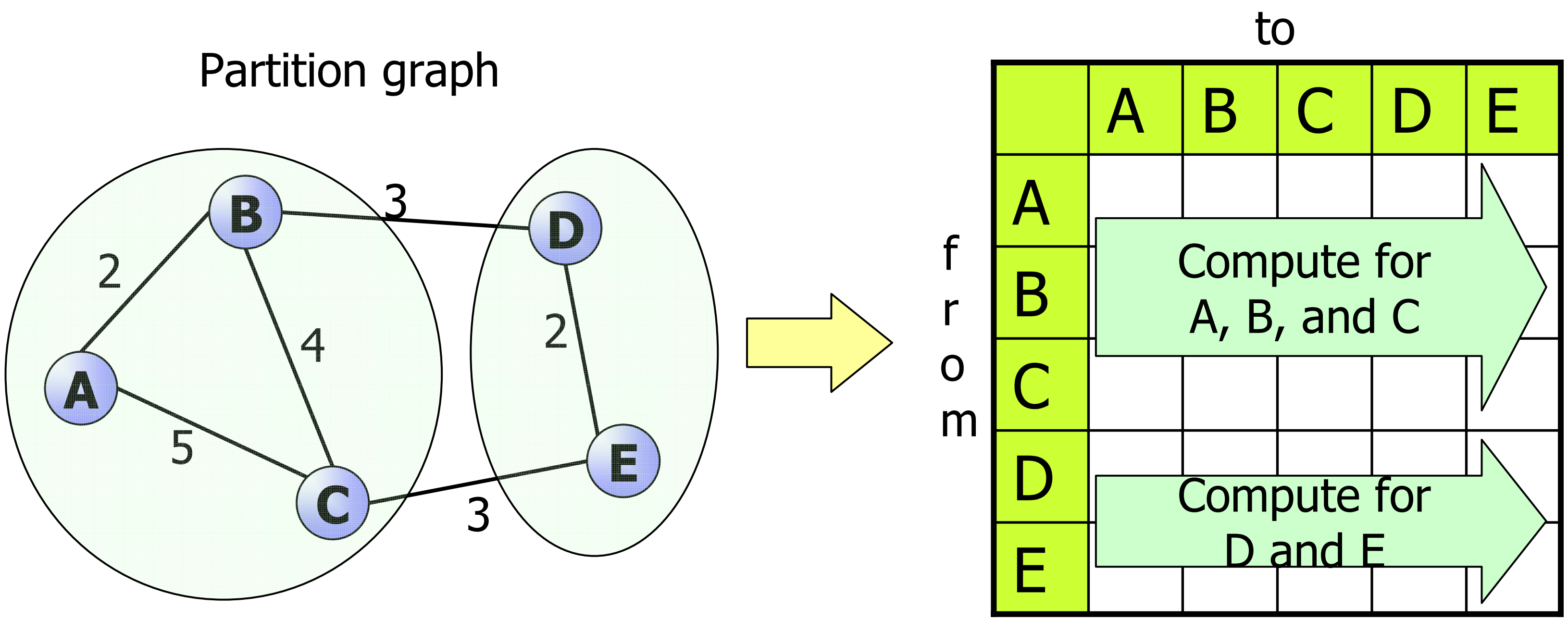
SIMD implementation

- Each scan operation can be easily SIMDized



Our algorithm

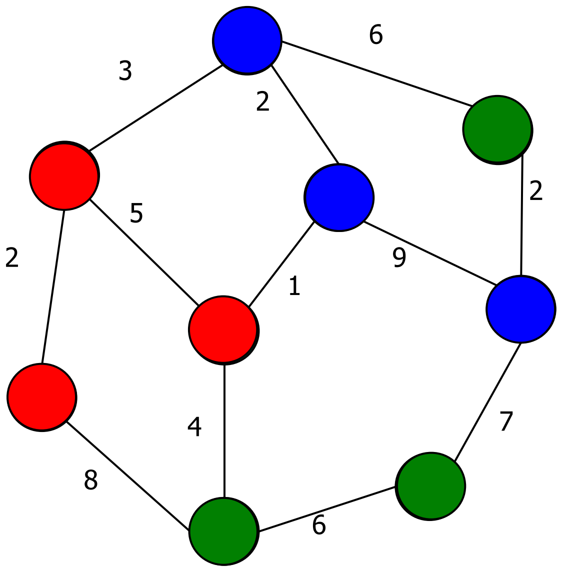
- Multiple runs of multi-source shortest path algorithm



Graph Partitioning

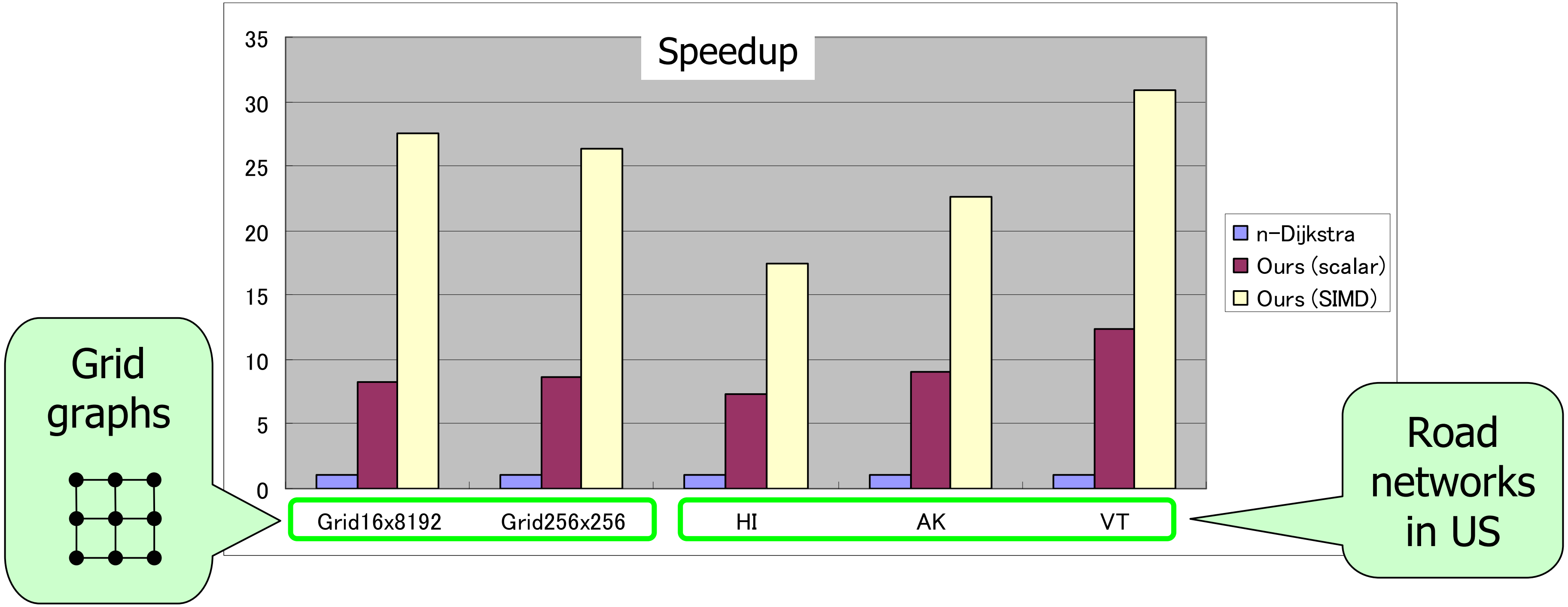
- Repeating following procedure
  - Pick up a node and traverse nearby nodes
- BFS is the best (among BFS, DFS, and kNN traverses)
- Times for graph partitioning are negligible

Ex. B =3



Experiment: Single-Thread

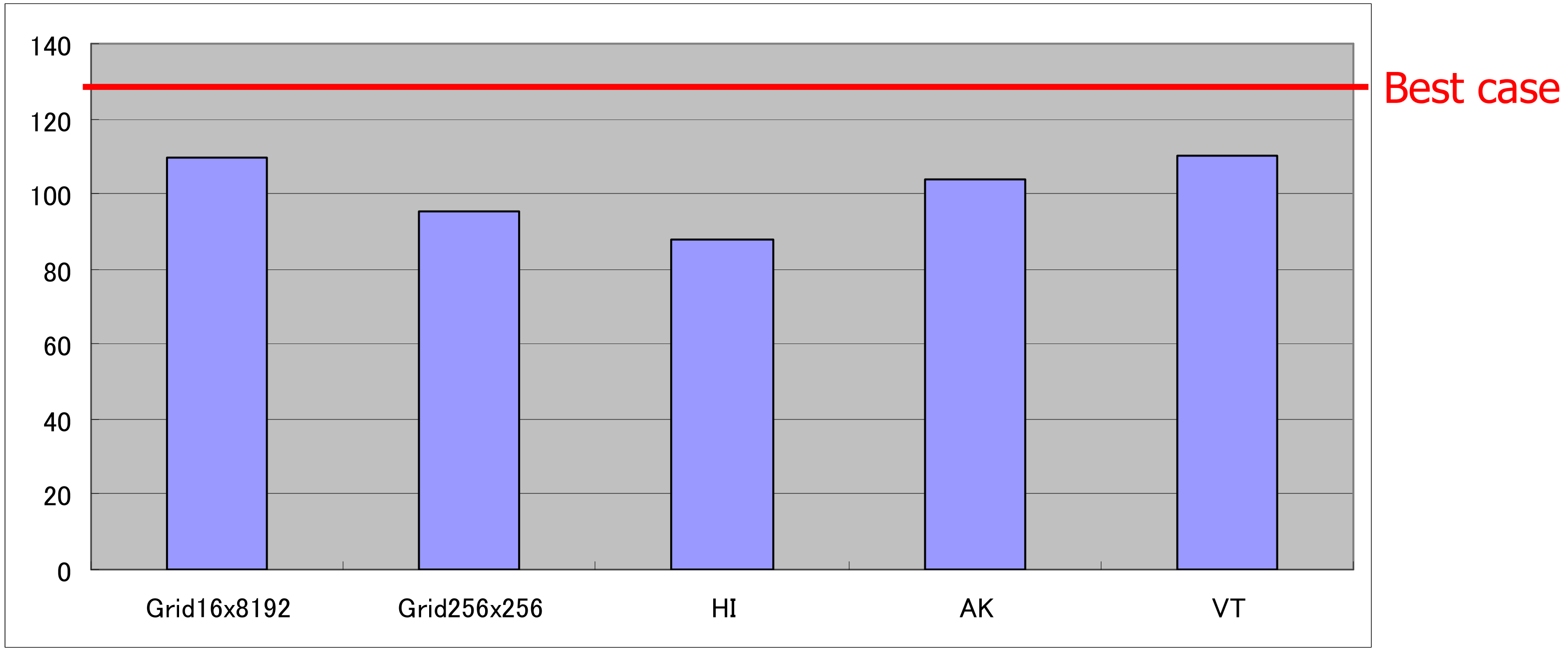
- Our algorithm clearly outperforms n-Dijkstra algorithm
- SIMD implementation accelerates scalar version 2.3 – 3.7 times



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.)

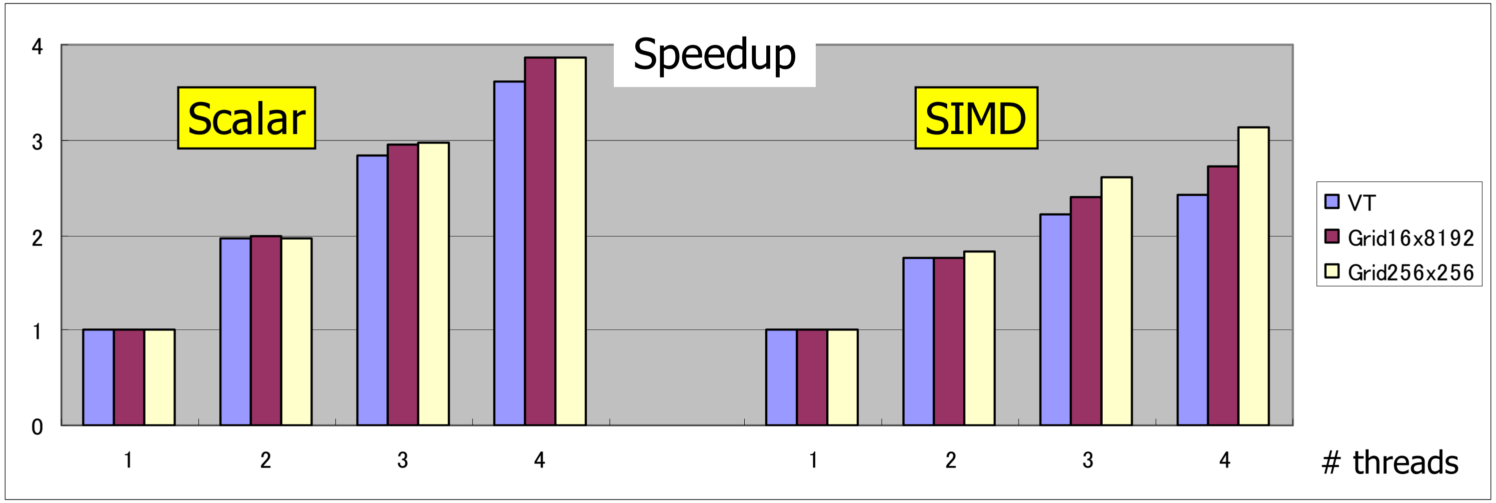
- The acceleration is due to the decrease of # operation on priority queue
  - In best case, this number is decreased by a factor of B



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

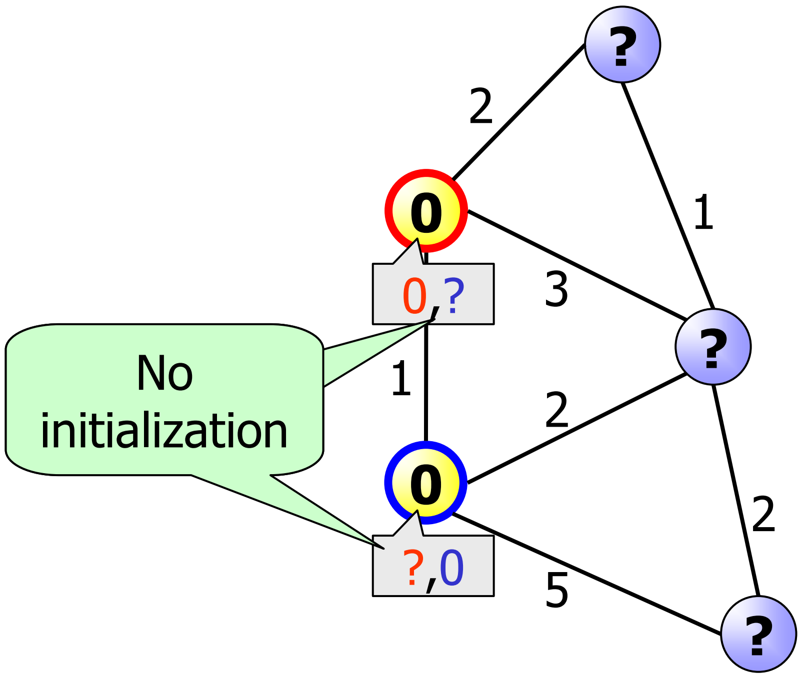


We used $B = 128$, BFS partitioning, and minimum key rule
Machine: Quad Core Xeon 3.16 GHz on Windows Server 2003

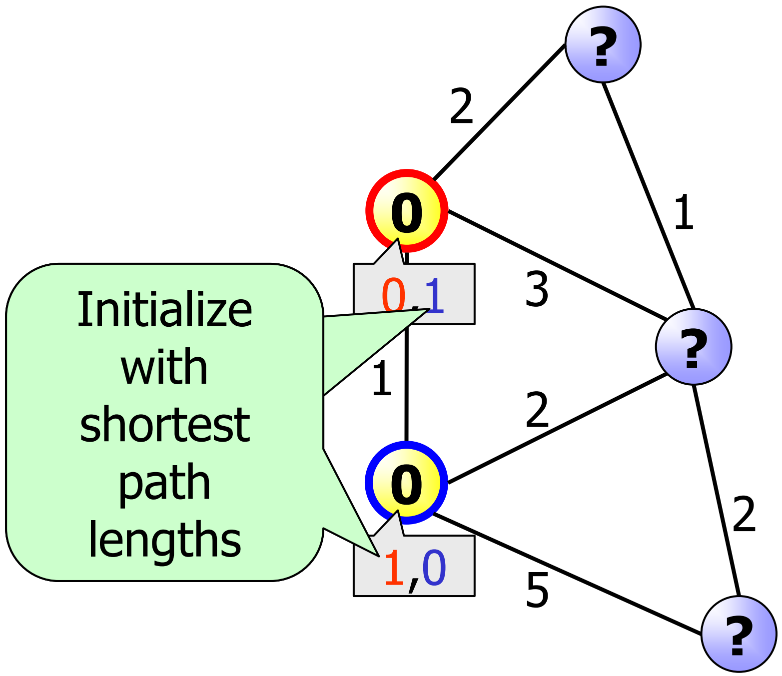
Improving Initializations

- Hilger suggested using better initializations yields 1 – 3 times faster running time

Our algorithm

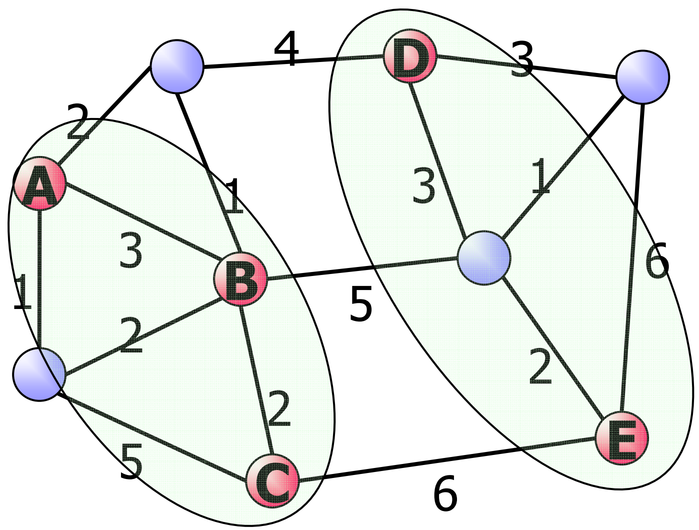


Hilger's algorithm



Many-to-Many Shortest Paths

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



to / from

	A	B	C	D	E
A	0	3	5	6	10
B	3	0	2	5	7
C	5	2	0	7	6
D	6	5	7	0	5
E	10	7	6	5	0

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