# Parallel Implementation of Dijkstra’s and Bellman Ford Algorithm

***Team 26 B***

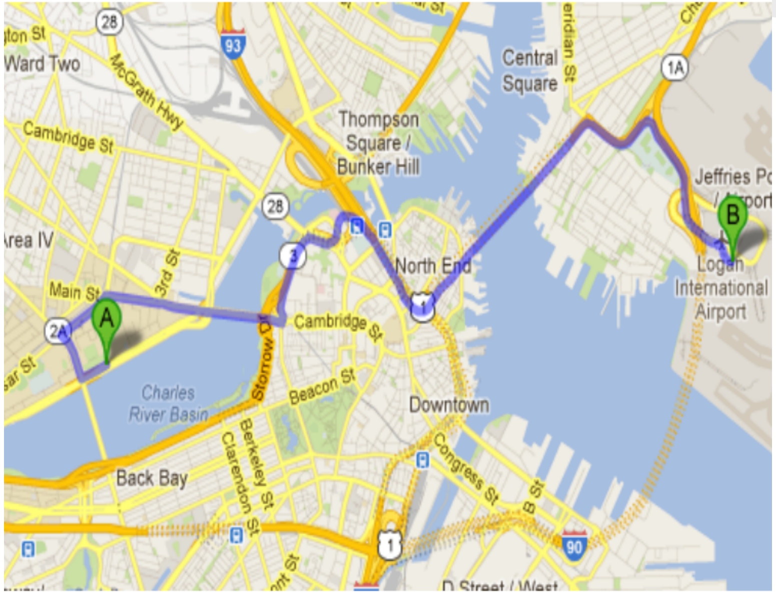
- Priyanka Ramanuja Kulkarni
- Meenakshi Muthuraman
- Aslesha Pansare
- Nikhil Kataria

# Single Source Shortest Path Problem

- The Problem of finding the shortest path from a source vertex

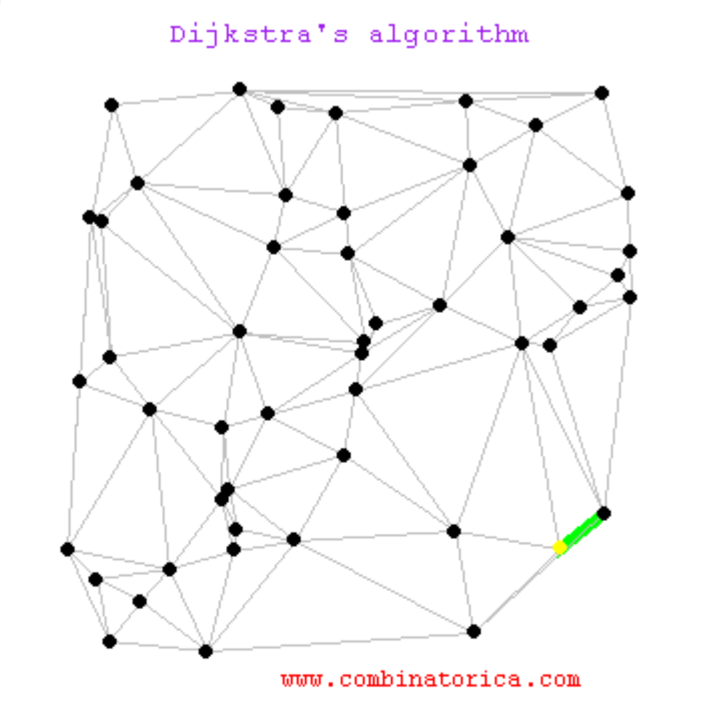
S to all vertices in the graph

- Weighted graph G = (V,E)
- Distance from S to all the vertices



# Dijkstra's Algorithm

- Solution to single source shortest path algorithm in graph theory
  - Both directed and undirected graphs
  - All edges must have non-negative weights
  - Graph must be connected
  - Dijkstra's algorithm runs in O (E*lg|V|)

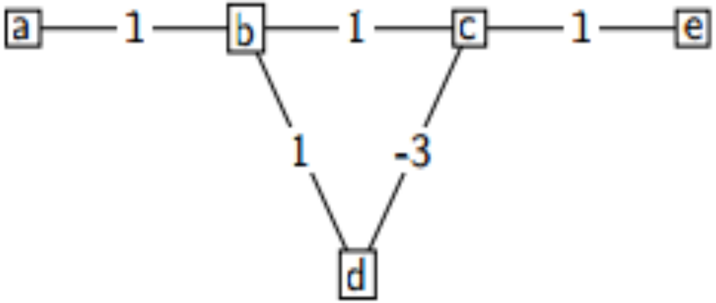


# Pseudocode of Dijkstra’s Algorithm

```
dist[s] ←0                                   (distance to source vertex is zero)
for  all v ∈ V–{s}
        do  dist[v] ←∞                       (set all other distances to infinity)
S←∅                                          (S, the set of visited vertices is initially empty)
Q←V                                          (Q, the queue initially contains all vertices)
while Q ≠∅                                   (while the queue is not empty)
do   u ← mindistance(Q,dist)                 (select the element of Q with the min. distance)
      S←S∪{u}                                (add u to list of visited vertices)
       for all v ∈ neighbors[u]
              do  if   dist[v] > dist[u] + w(u, v)          (if new shortest path found)
                         then      d[v] ←d[u] + w(u, v)     (set new value of shortest path)
                                                            (if desired, add traceback code)
return dist
```

# Negative Cycles

A negative cycle is a cycle in a weighted graph whose total weight is negative



# Bellman Ford’s Algorithm

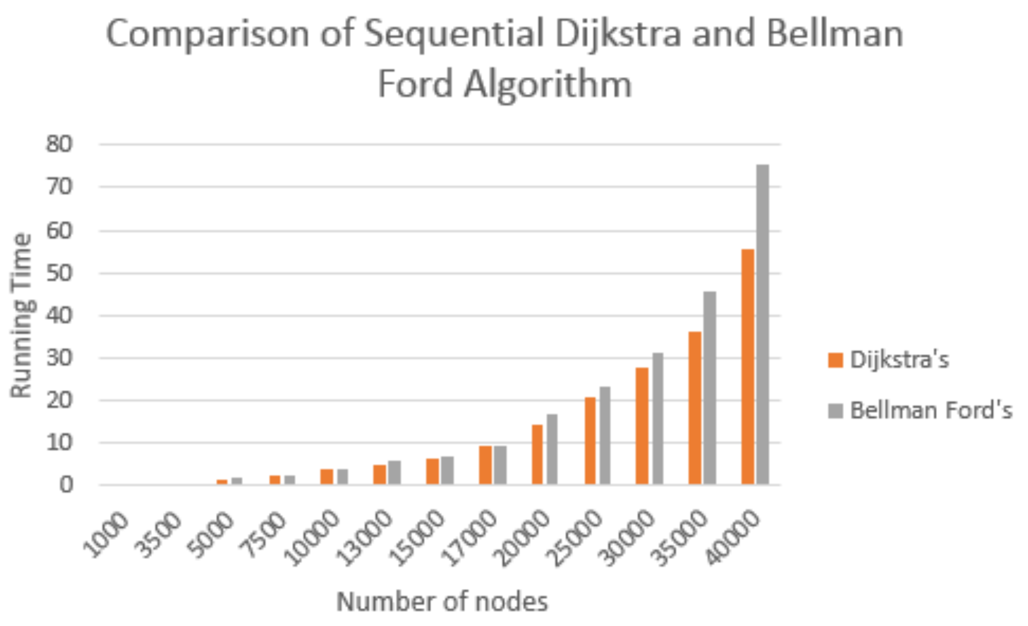
```
BELLMAN-FORD(G,w,s)
INITIALIZE-SINGLE-SOURCE(G,s)
for i = 1 to |G.V|-1
    for each edge (u,v) ∈ G.E
        RELAX(u,v,w)
for each edge (u,v) ∈ G.E
    if v.d > u.d + w(u,v)
        return FALSE
return TRUE
```

- Worst case performance - 0 (|V||E|)
- Space Complexity - 0(|V|)


```
BELLMAN-FORD(G,w,s)
INITIALIZE-SINGLE-SOURCE(G,s)
for i = 1 to |G.V|-1
    for each edge (u,v) ∈ G.E
        RELAX(u,v,w)
for each edge (u,v) ∈ G.E
    if v.d > u.d + w(u,v)
        return FALSE
return TRUE
```

# Comparing Sequential performance

| No of Nodes | Dijkstra's | Bellman Ford's |
|---|---|---|
| 1000 | 0.02354 | 0.02355 |
| 3500 | 0.27148 | 0.27145 |
| 5000 | 0.5337 | 0.5539 |
| 7500 | 1.203 | 1.703 |
| 10000 | 2.1728 | 2.3027 |
| 13000 | 3.7544 | 4.1034 |
| 15000 | 5.04 | 5.71 |
| 17000 | 6.5479 | 6.994 |
| 20000 | 9.1839 | 9.5033 |
| 25000 | 14.4462 | 16.8432 |
| 30000 | 20.73 | 23.2003 |
| 35000 | 27.9595 | 30.9635 |
| 40000 | 36.1225 | 45.8344 |
| 50000 | 55.3941 | 75.4112 |



# Assumptions

- All the edges are undirected
- We consider a sparse matrix for the initial distance vector
- Each vertex is connected to at least 1 other vertex
- The weights of the edges are assigned randomly with a restriction on the x-axis

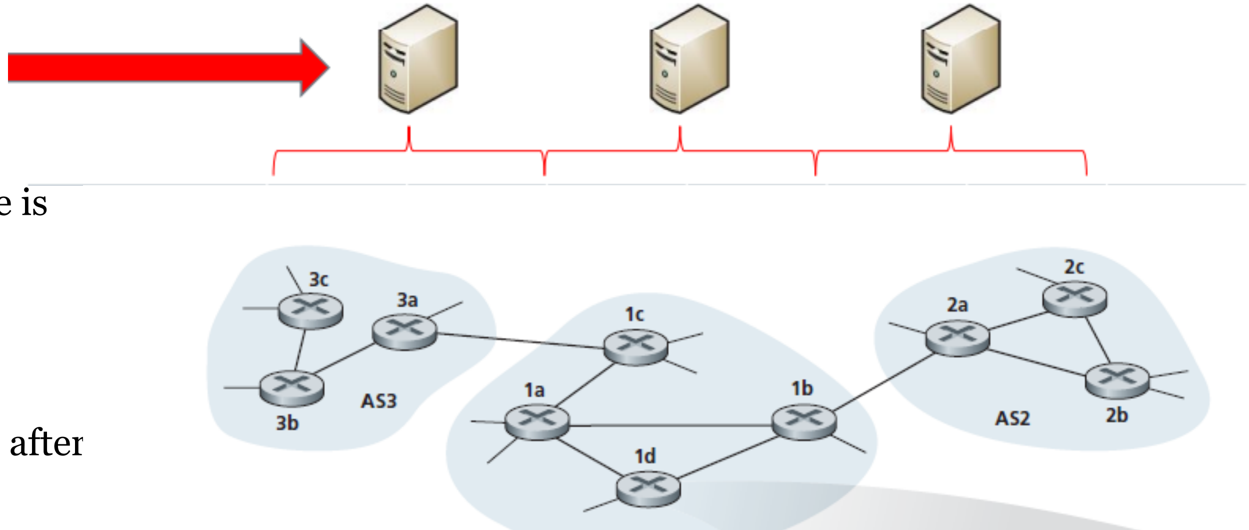
rand()

# Parallel Dijkstra's Algorithm

- On each cluster identify vertices closest to the source vertex
- Use parallel prefix to select the globally closest vertex
- Broadcast the results to all cores
- On each cluster update the distance vectors
- Running time = $O(V^2/P + V*\log(P))$

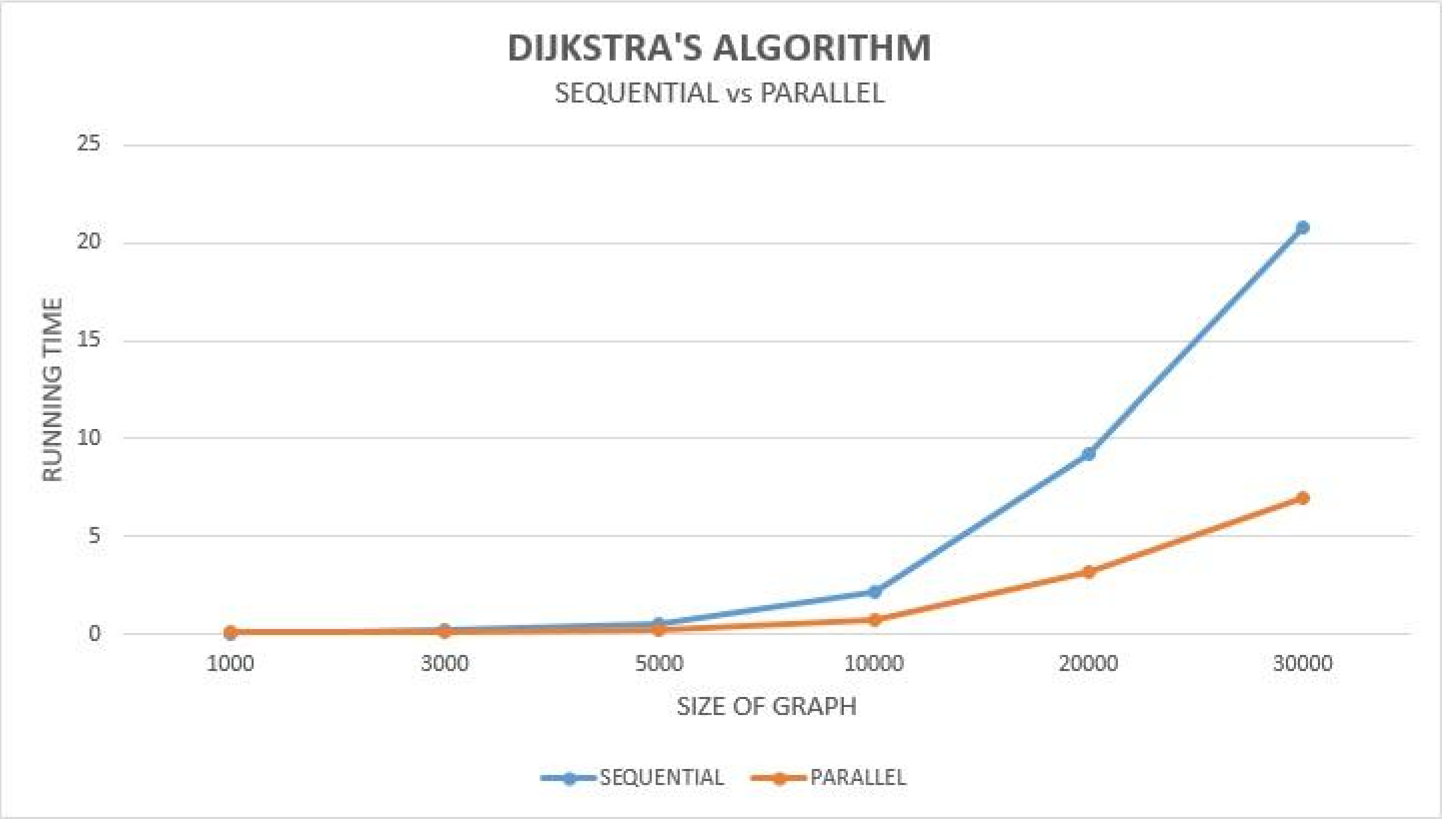
# Parallel Approach

1.Allocate
graph nodes to
different processors.
2.Shortest distance is
computed using
parallel prefix.
3. Broadcast the
Result to all nodes after
closest path.

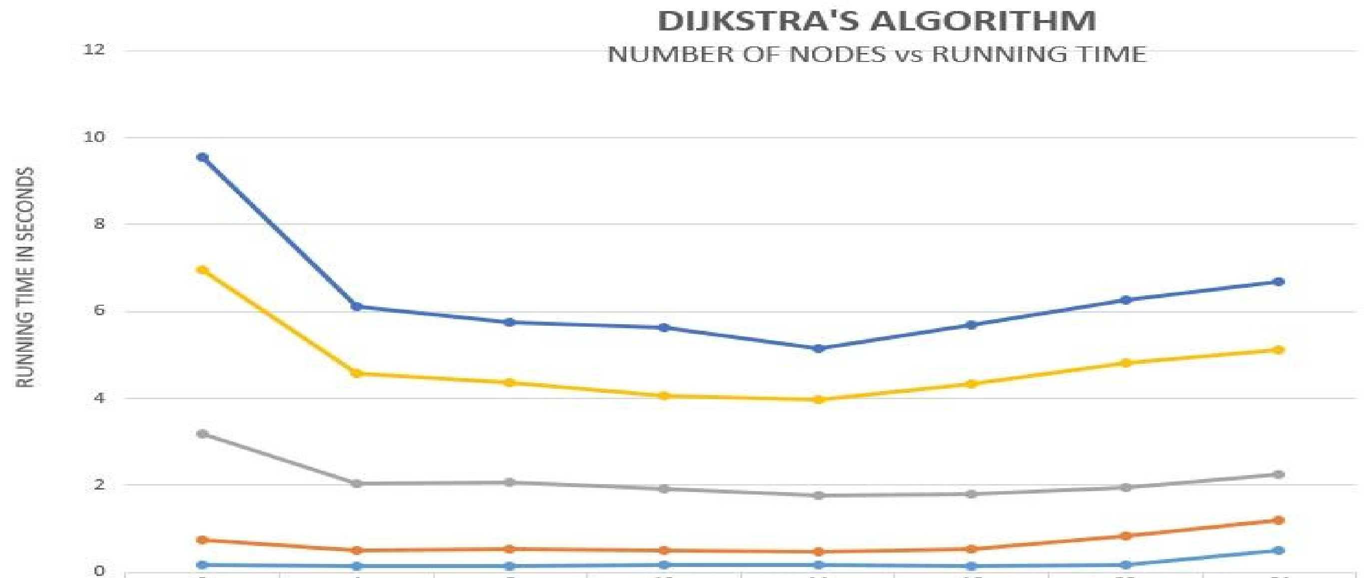



- MPI_Reduce(MPI_IN_PLACE,nextAN,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);
- MPI_Bcast( G->node[0], G->N*G->N, MPI_CHAR, 0, MPI_COMM_WORLD);

# Dijkstra's sequential Vs Parallel Implementation



# Dijkstra's: #nodes v/s Running time




| | 2 | 4 | 8 | 12 | 14 | 16 | 32 | 64 |
|---|---|---|---|---|---|---|---|---|
| 5000 | 0.1748 | 0.1342 | 0.1448 | 0.1611 | 0.1621 | 0.1327 | 0.1686 | 0.511 |
| 10000 | 0.7294 | 0.4963 | 0.5219 | 0.5077 | 0.4823 | 0.5463 | 0.8433 | 1.21 |
| 20000 | 3.1735 | 2.0452 | 2.0573 | 1.9052 | 1.7607 | 1.7905 | 1.944 | 2.2411 |
| 30000 | 6.9632 | 4.571 | 4.3563 | 4.0518 | 3.9665 | 4.319 | 4.8231 | 5.10084 |
| 35000 | 9.5345 | 6.0989 | 5.75 | 5.638 | 5.1331 | 5.69 | 6.25 | 6.675 |

NUMBER OF NODES

5000 10000 20000 30000 35000

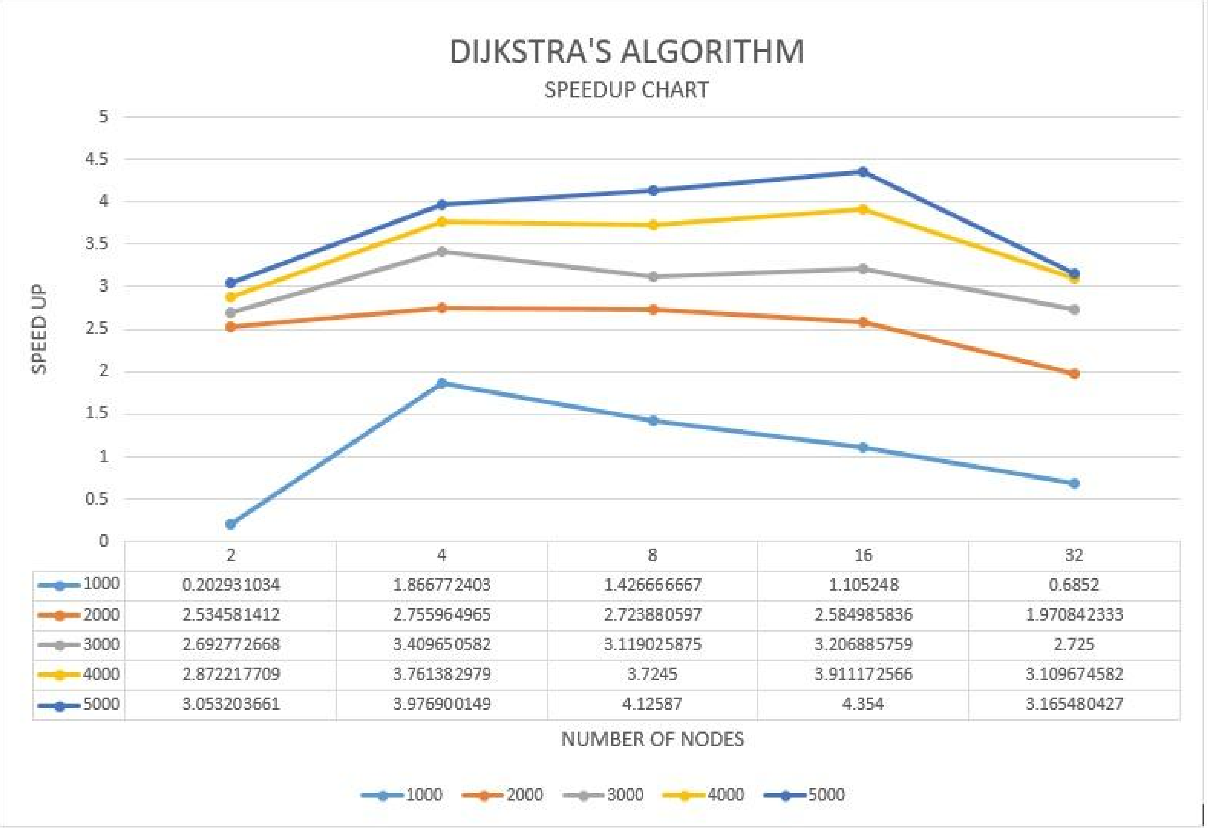
# Djikstra’s: Speed up

Speedup is defined by the following formula:

$$S_p = \frac{T_1}{T_p}$$

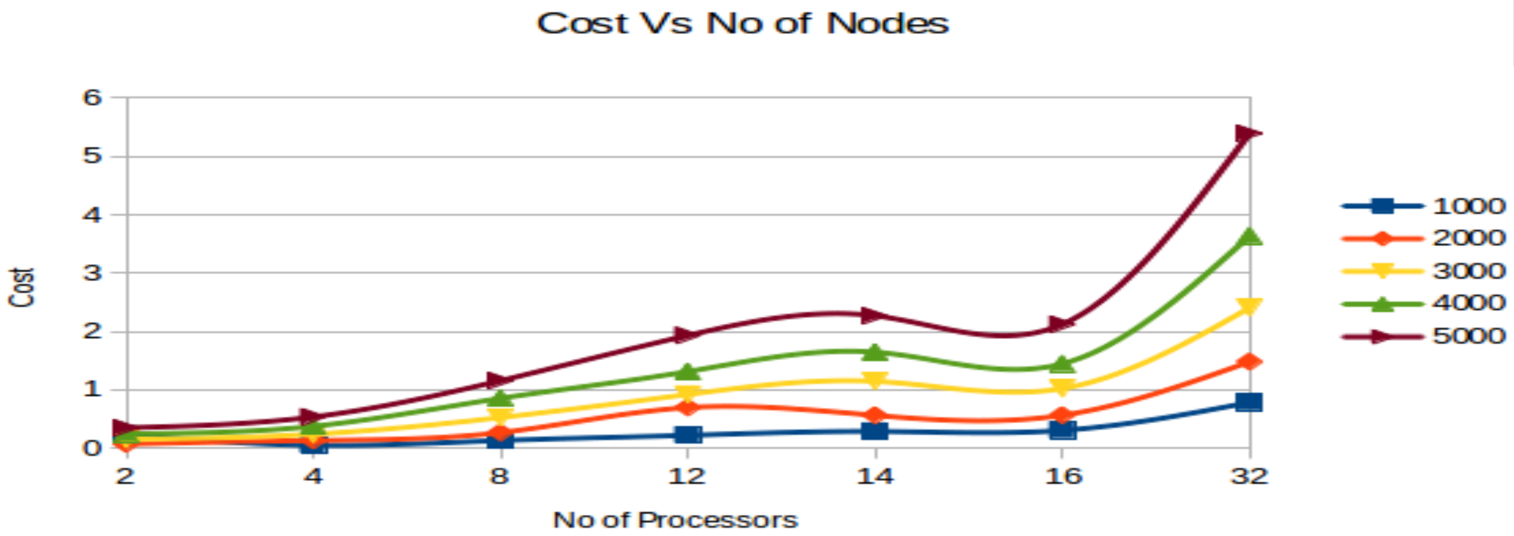
where:

- $p$ is the number of processors
- $T_1$ is the execution time of the sequential algorithm
- $T_p$ is the execution time of the parallel algorithm with $p$ processors



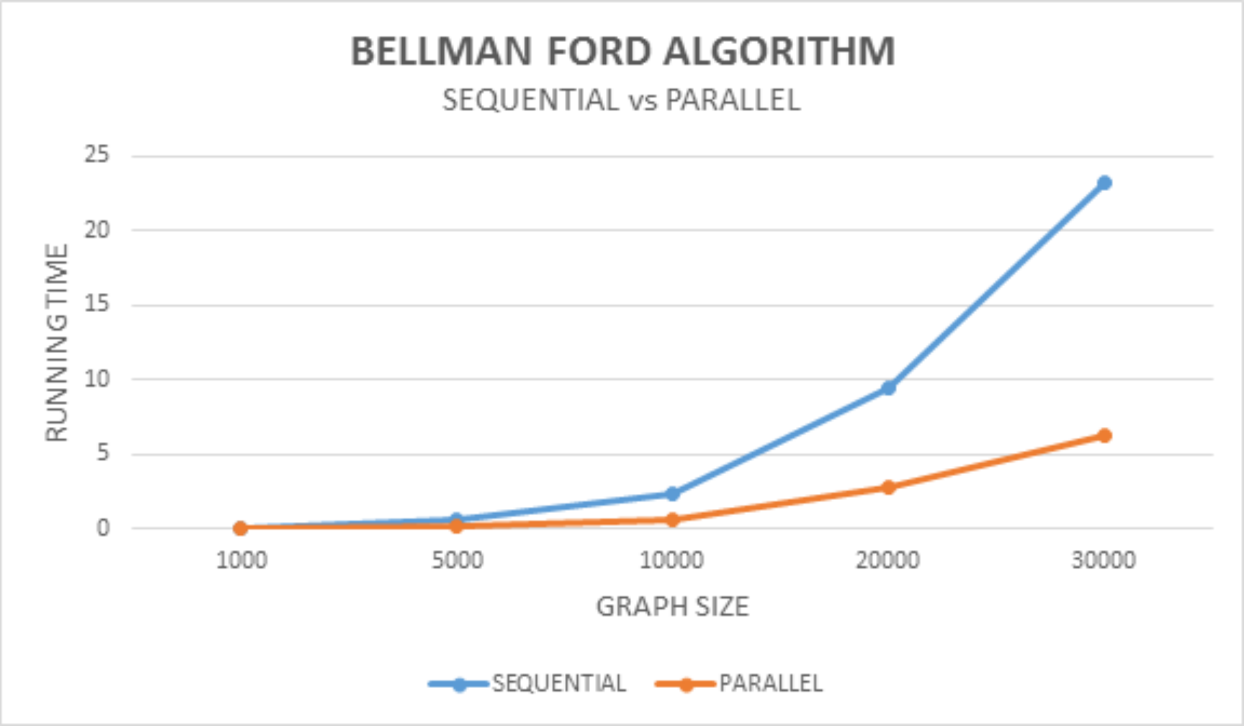

| | 2 | 4 | 8 | 16 | 32 |
|---|---|---|---|---|---|
| 1000 | 0.20293 1034 | 1.86677 2403 | 1.426666667 | 1.105248 | 0.6852 |
| 2000 | 2.534581412 | 2.755964965 | 2.723880597 | 2.584985836 | 1.970842333 |
| 3000 | 2.692772668 | 3.40965 0582 | 3.119025875 | 3.206885759 | 2.725 |
| 4000 | 2.872217709 | 3.76138 2979 | 3.7245 | 3.91117 2566 | 3.109674582 |
| 5000 | 3.053203661 | 3.976900149 | 4.12587 | 4.354 | 3.165480427 |

# Dijkstra’s: Cost Analysis

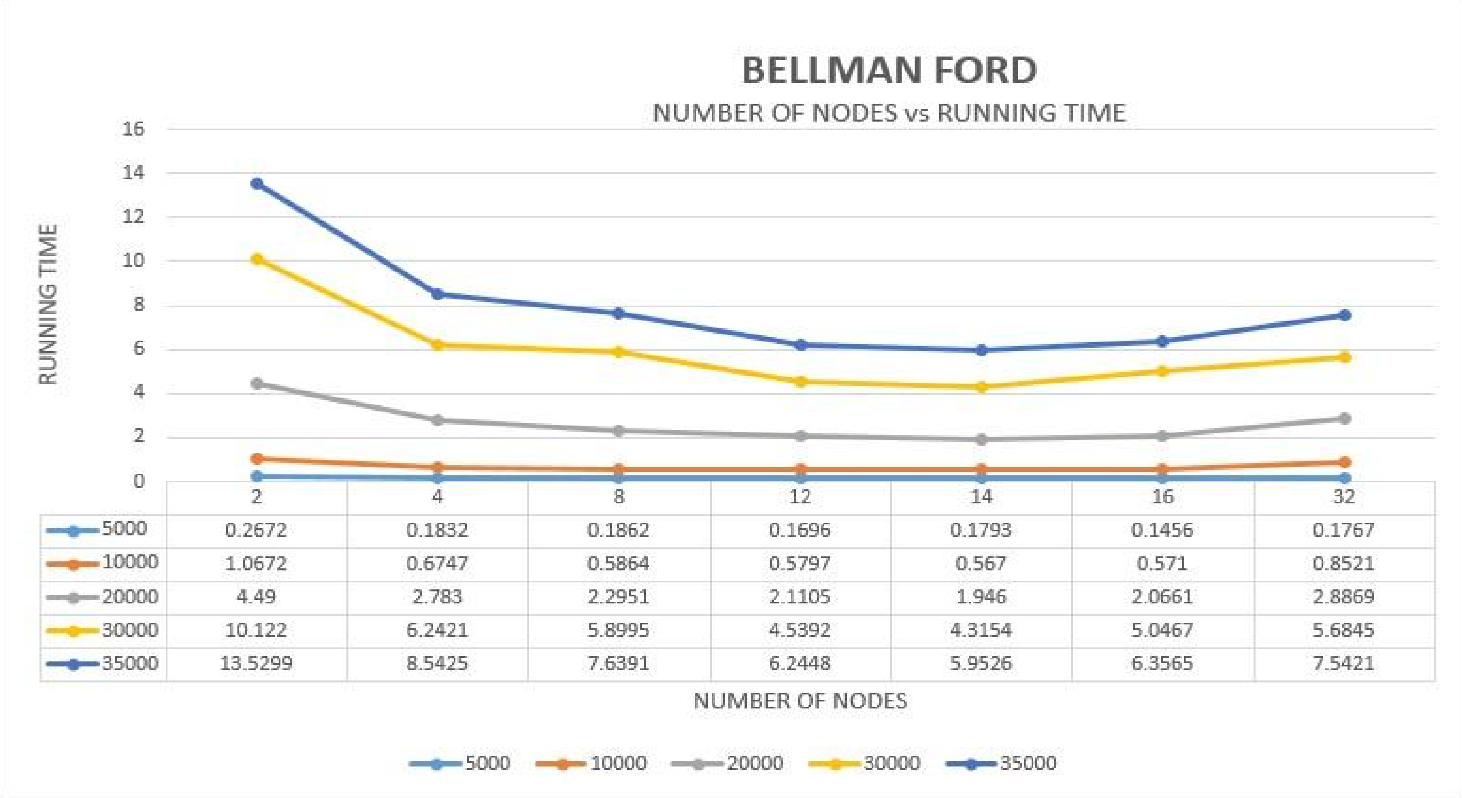



| No of Nodes | 1000 | 2000 | 3000 | 4000 | 5000 |
|---|---|---|---|---|---|
| 2 | 0.232 | 0.072004 | 0.1522 | 0.2462 | 0.3496 |
| 4 | 0.05044 | 0.13244 | 0.2404 | 0.376 | 0.5368 |
| 8 | 0.132 | 0.268 | 0.5256 | 0.8552 | 1.1584 |
| 12 | 0.2208 | 0.6972 | 0.9216 | 1.314 | 1.9332 |
| 14 | 0.2884 | 0.5628 | 1.14632 | 1.6464 | 2.2694 |
| 16 | 0.3056 | 0.5648 | 1.0224 | 1.4464 | 2.1232 |
| 32 | 0.784 | 1.4816 | 2.4064 | 3.6384 | 5.3952 |

# Bellman Ford: sequential vs parallel

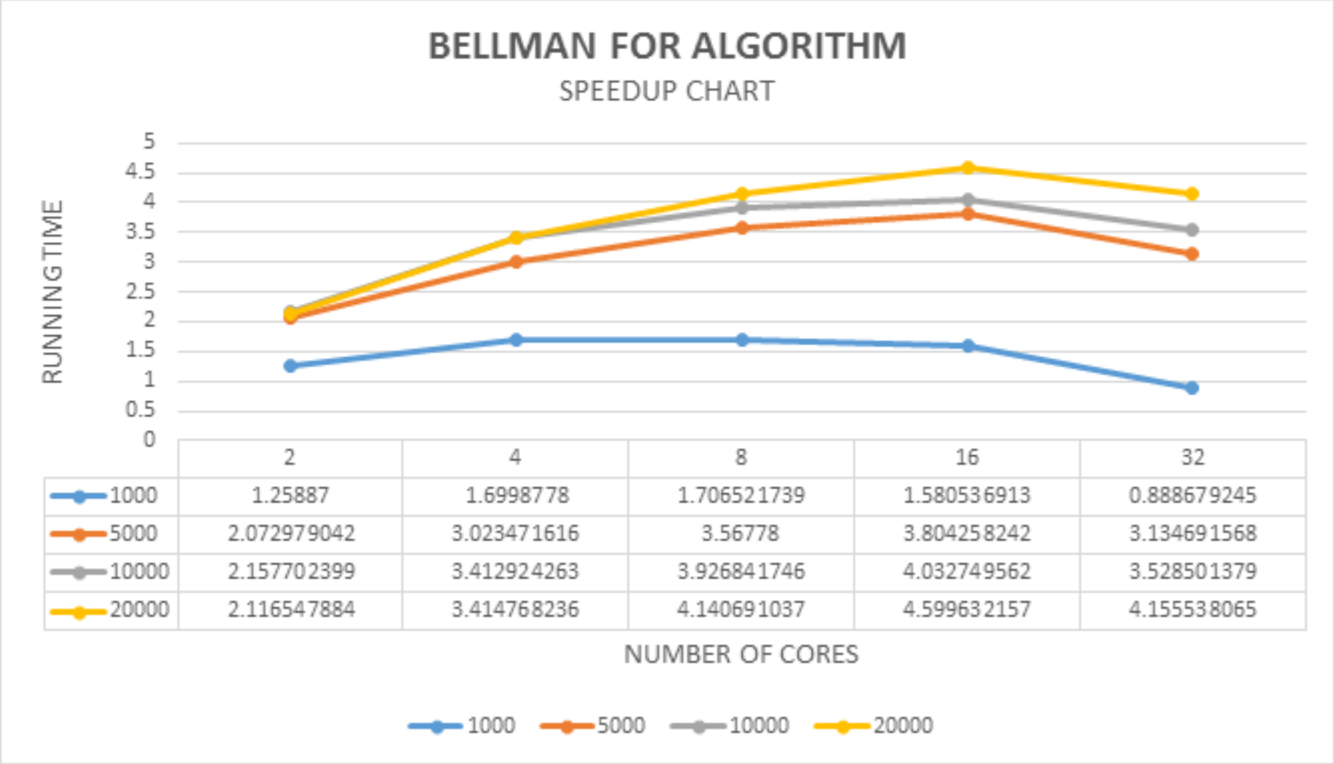


# Bellman Ford: Running Time v/s # of Nodes



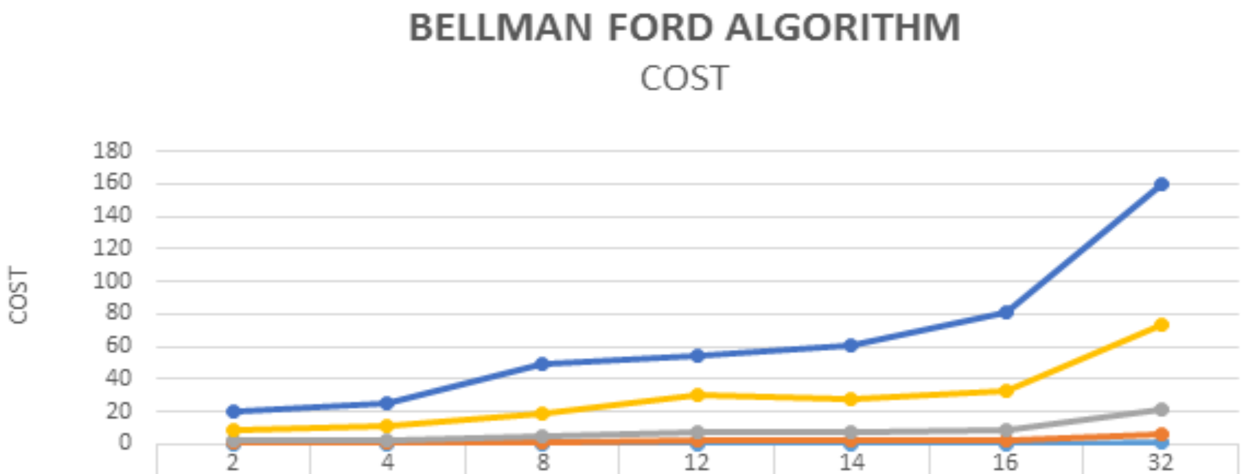

| | 2 | 4 | 8 | 12 | 14 | 16 | 32 |
|---|---|---|---|---|---|---|---|
| 5000 | 0.2672 | 0.1832 | 0.1862 | 0.1696 | 0.1793 | 0.1456 | 0.1767 |
| 10000 | 1.0672 | 0.6747 | 0.5864 | 0.5797 | 0.567 | 0.571 | 0.8521 |
| 20000 | 4.49 | 2.783 | 2.2951 | 2.1105 | 1.946 | 2.0661 | 2.8869 |
| 30000 | 10.122 | 6.2421 | 5.8995 | 4.5392 | 4.3154 | 5.0467 | 5.6845 |
| 35000 | 13.5299 | 8.5425 | 7.6391 | 6.2448 | 5.9526 | 6.3565 | 7.5421 |

# Bellman Ford: Speedup




| | 2 | 4 | 8 | 16 | 32 |
|---|---|---|---|---|---|
| 1000 | 1.25887 | 1.6998778 | 1.706521739 | 1.580536913 | 0.888679245 |
| 5000 | 2.072979042 | 3.023471616 | 3.56778 | 3.804258242 | 3.134691568 |
| 10000 | 2.157702399 | 3.412924263 | 3.926841746 | 4.032749562 | 3.528501379 |
| 20000 | 2.116547884 | 3.414768236 | 4.140691037 | 4.599632157 | 4.155538065 |

# Bellman Ford: Cost Analysis

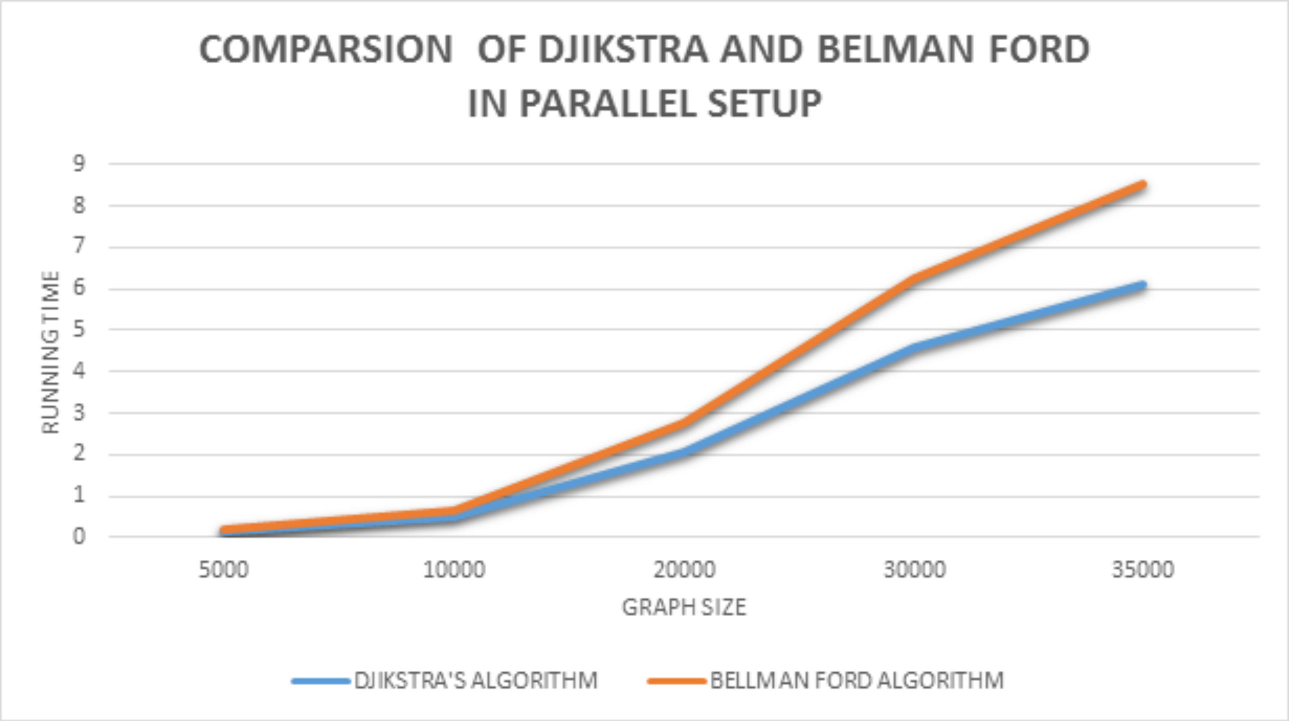



| | 2 | 4 | 8 | 12 | 14 | 16 | 32 |
|---|---|---|---|---|---|---|---|
| 1000 | 0.029 | 0.0836 | 0.1104 | 0.1896 | 0.2436 | 0.2384 | 0.848 |
| 5000 | 0.5344 | 0.7328 | 1.4896 | 2.0352 | 2.5102 | 2.3296 | 5.6544 |
| 10000 | 2.1344 | 2.6988 | 4.6912 | 6.9564 | 7.938 | 9.136 | 20.8832 |
| 20000 | 8.98 | 11.132 | 18.3608 | 30.048 | 27.244 | 33.0576 | 73.1808 |
| 30000 | 20.244 | 24.9684 | 49.7328 | 54.4704 | 60.4156 | 80.7472 | 159.5904 |

NUMBER OF NODES

1000 5000 10000 20000 30000

# Comparing parallel performance



# References


- Implementing Parallel Shortest-Paths Algorithms (1994) by by Marios Papaefthymiou and  Joseph Rodrigue
- Algorithms Sequential & Parallel: A Unified Approach / Edition 2 by Russ Miller and Lawrence Boxer
- An Experimental Study of a Parallel Shortest Path Algorithm for Solving Large-Scale Graph Instances by Kamesh Madduri and David A. Bader, Georgia Institute of Technology and Jonathan W. Berry Sandia National Laboratories and Joseph R. Crobak, Rutgers University
- Parallel Algorithms by Guy E. Blelloch and Bruce M. Maggs, School of Computer Science, Carnegie Mellon University


Q&A