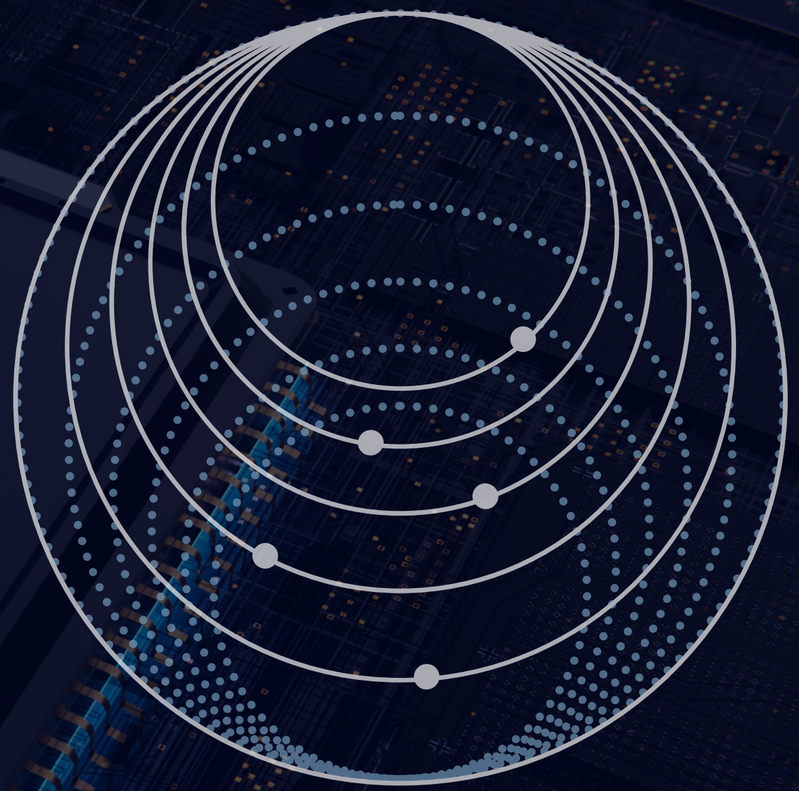
DYNAMIZING DIJKSTRA

A SOLUTION TO DYNAMIC SHORTEST PATH PROBLEM THROUGH RETROACTIVE PRIORITY QUEUE

Presentation By: Adina Adnan Mansoor, Syeda Rija Hasan Abidi,
Syed Zuhair Abbas Rizvi

Project Overview



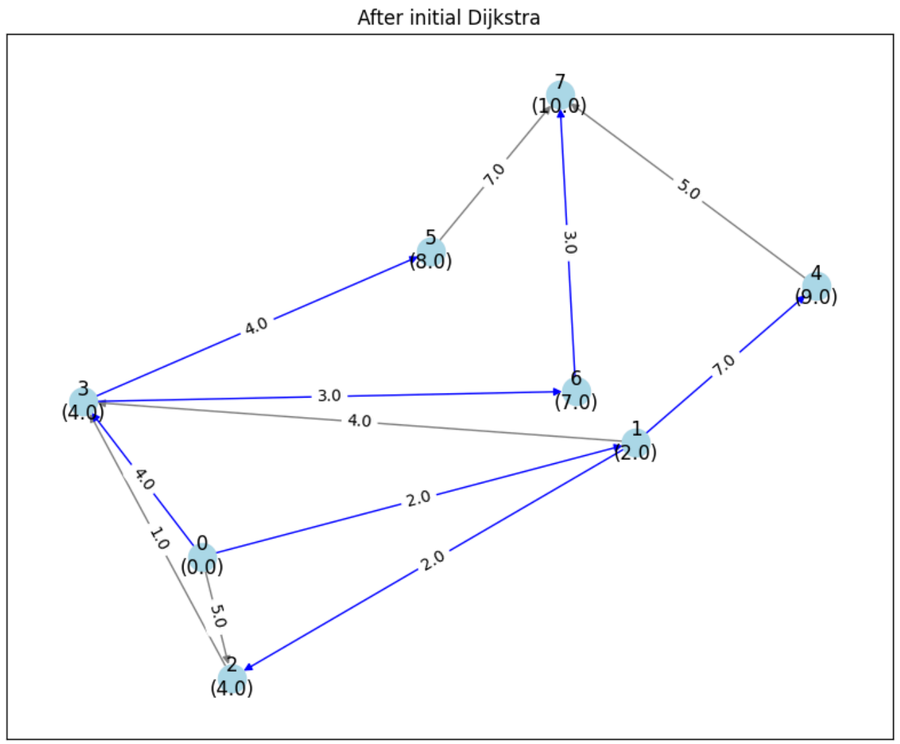
The problem involves finding the shortest path from a source to all other vertices in a graph that changes its weight over time.
This problem is known as the Dynamic Shortest Path Problem (DSPP)

Problem Statement

The Dynamic Shortest Path Problem involves finding the shortest path from a source node to all other nodes in a graph that is subject to frequent updates, such as edge weight changes, edge additions, or deletions.

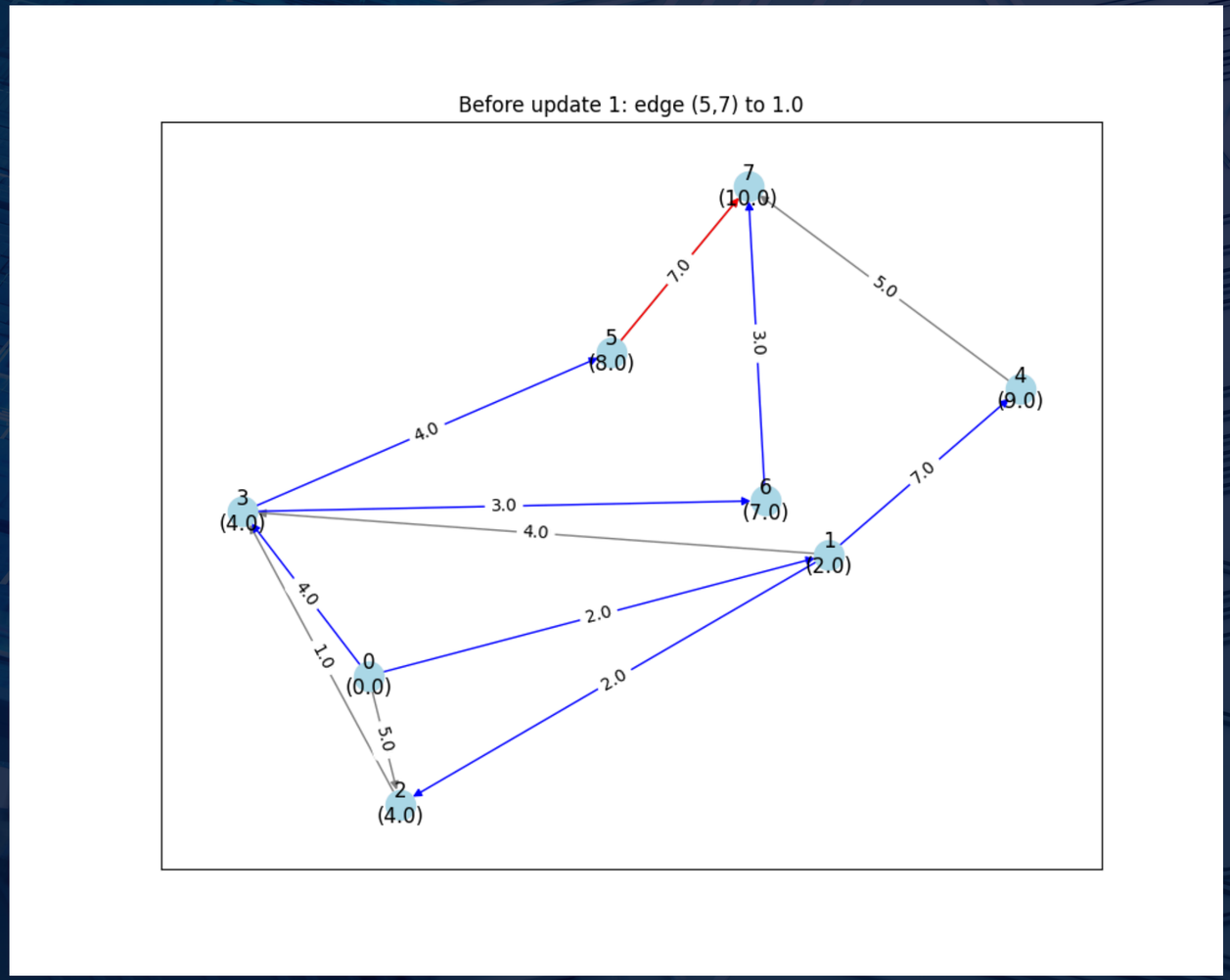
Imagine a road network, where traffic conditions can change dynamically, causing the updates to edge weights. So if the edge update affects any of the shortest path, we have to recalculate the shortest path

Example

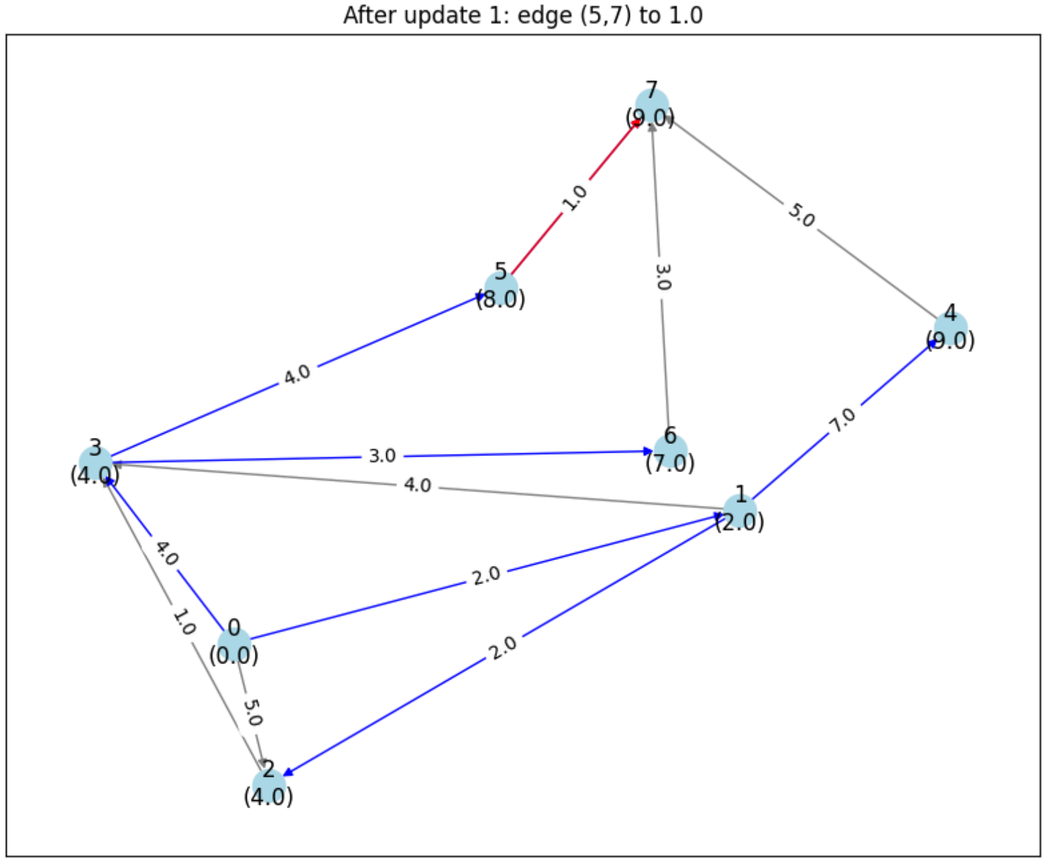


The shortest path from the source (0) to all other vertices has been initialized. The blue edges indicate the shortest path from the source

What if there is an edge update?
In this case, the weight of the edge (5,7) updates from 7 to 1. We will recalculate the shortest paths , to find the new shortest paths



Initially, the shortest path from the source (0) to vertex 7 was via the path 0 → 3 → 6 → 7, with a total weight of 4 + 3 + 3 = 10. After the edge weight between nodes 5 and 7 is updated from 7 to 1, the shortest path is now 0 → 3 → 5 → 7, with a new total weight of 4 + 4 + 1 = 9. Since this new path has a lower weight, the shortest path gets updated accordingly

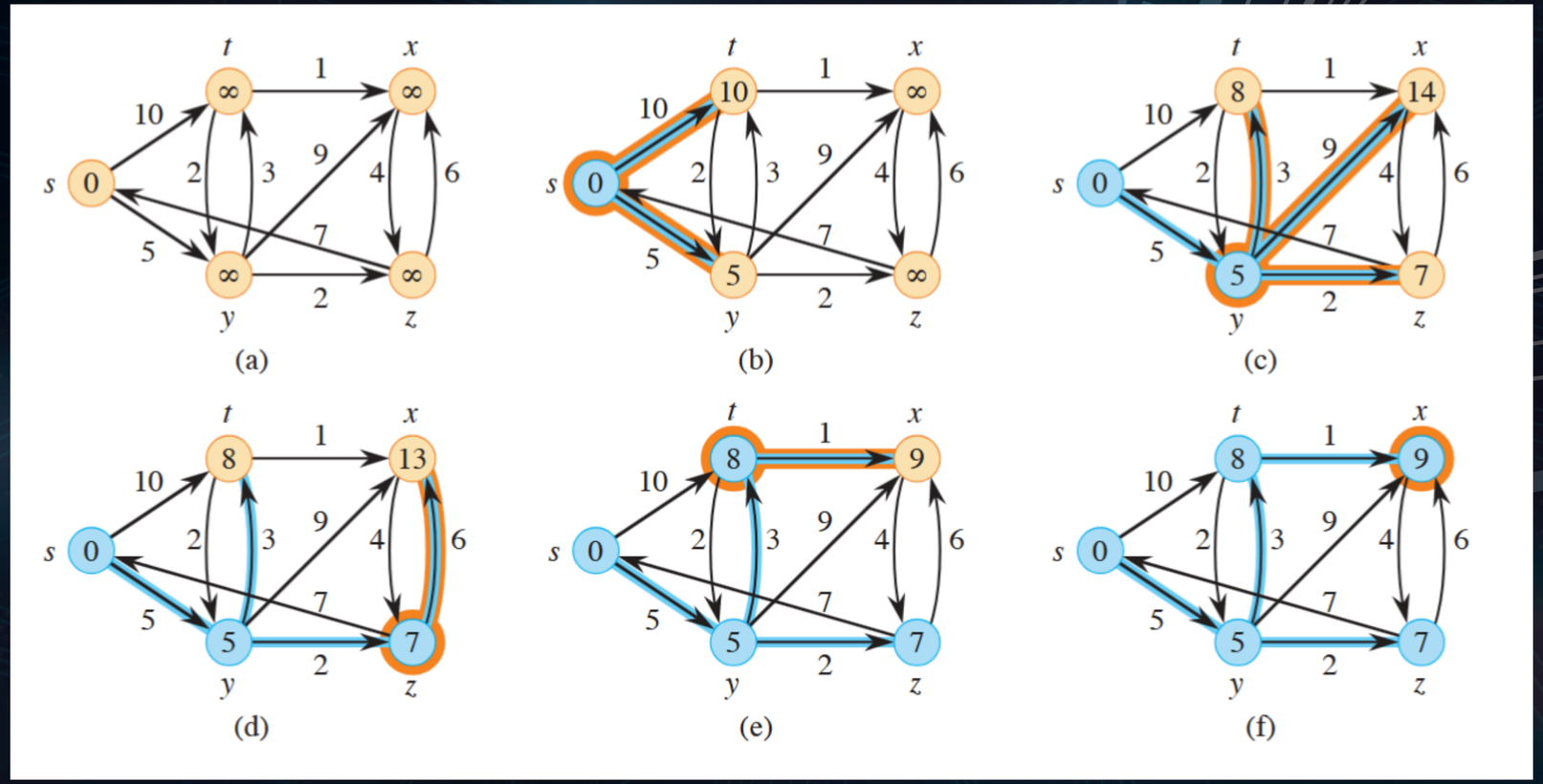


Static Dijkstra

Greedy approach to find the shortest path from a source node to all other nodes in a graph.

```
DIJKSTRA(G, w, s)
 1  INITIALIZE-SINGLE-SOURCE(G, s)
 2  S = ∅
 3  Q = ∅
 4  for each vertex u ∈ G.V
 5      INSERT(Q, u)
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      S = S ∪ {u}
 9      for each vertex v in G.Adj[u]
10          RELAX(u, v, w)
11          if the call of RELAX decreased v.d
12              DECREASE-KEY(Q, v, v.d)
```

Static Dijkstra



Static Dijkstra

Time Complexity:

O(V^2) without PQ, O(ElogV) with PQ

In dynamic environments, where the graph is frequently updated, re-running the algorithm from scratch after every update is inefficient.

Dynamic Dijkstra

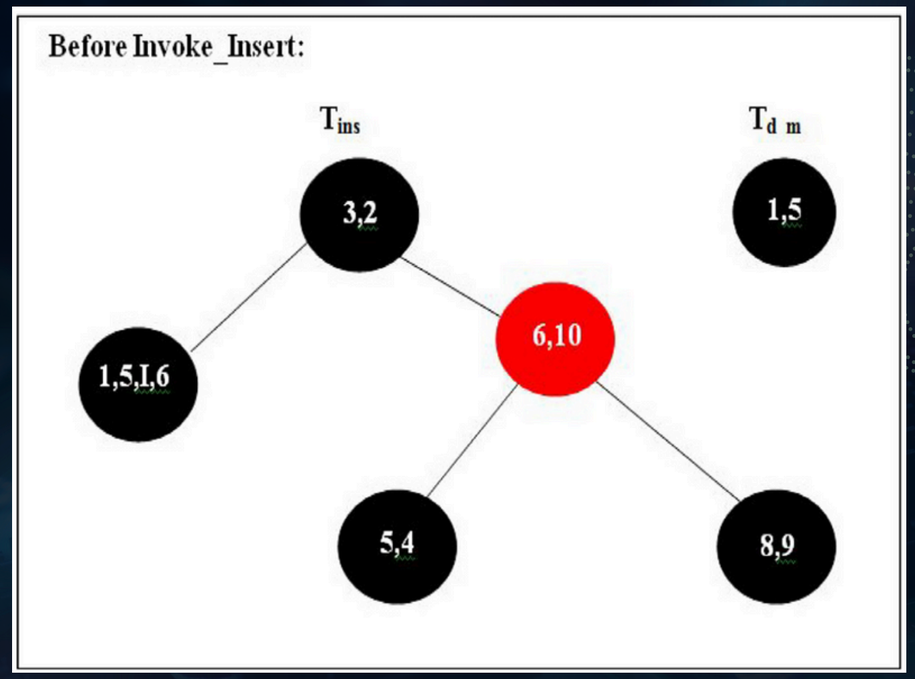
The paper proposed dynamizing the Dijkstra algorithm, using a Retroactive Priority Queue (RPQ), that keeps track of past operations and allows updates to be made selectively to the affected parts of the data structure

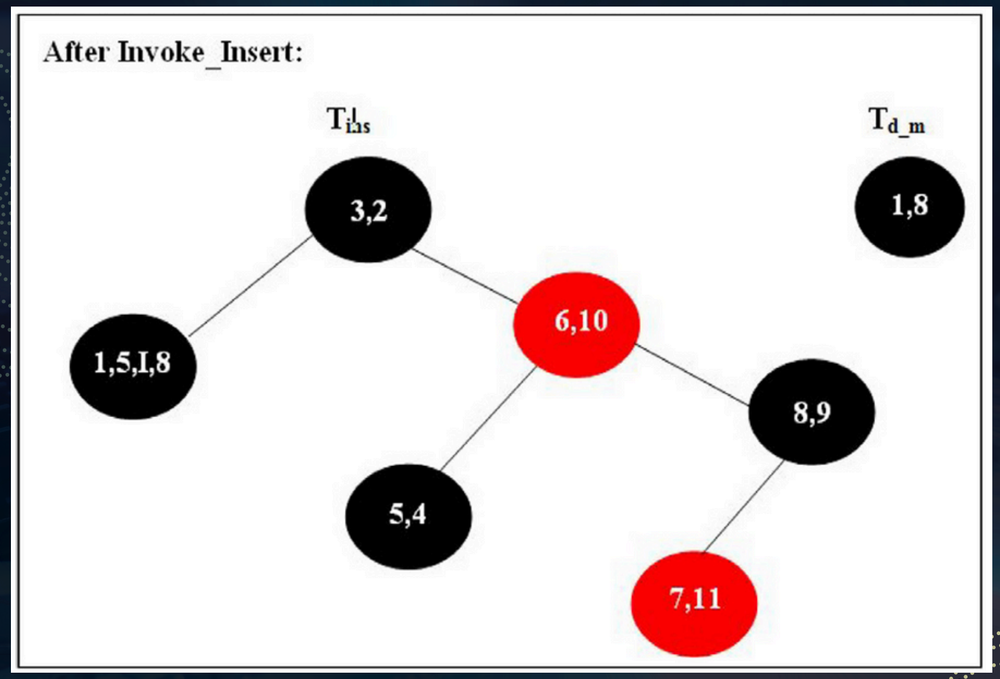
- A Retroactive Priority Queue is a priority queue with retroactive capabilities
- Allows insertion and deletion of elements at specific points in time, maintaining the historical sequence of operations.

What is RPQ??

Dynamic Dijkstra

RPQ WITH RED BLACK TREE





Algorithm Overview

Input: Graph (Adjacency matrix)
Output: Predecessor of each node and shortest distances from source

- Initialization: Run static Dijkstra to find initial predecessors and shortest distances of all nodes from source and insert each update into RPQ
- *Update(u,v,w) | (u,v): edge w: weight*
  - If w<current weight
    - Keep u as v's predecessor and update v's distance from source
  - else
    - Find all past v predecessors and weights through v's states in RPQ
    - If better predecessor for v found - least weight
      - Make this v's predecessor
      - Update distance of v from source through u
  - Propagate changes to v's ancestors (relax outgoing edges) & update their predecessors if needed

Algorithm Overview

Complexity Analysis

- At worst, all vertices are v's ancestors, we may need to update V-1 needs. Traversal in RPQ still only takes O(logV). The total complexity boils down to ***O(ElogV)*** in the worst case.
- In practice, time depends on v's time increases linearly with the number of v's ancestors
- Reducing the updation space to only the affected vertices,

unlike re-running the whole algorithm which guarantees O(V^2) or O(ElogV) like Dijsktra.

Comparison with baseline

Correctness

- Compared results with static Bellman ford and Dijkstra algorithms for same graphs

```
numvertices = 3000
numedges = 1000
numupdates = 200
```

```
Dynamic Algorithm Update Time: 0.0567 seconds
Standard Dijkstra's Update Time: 0.0339 seconds
Bellman-Ford Update Time: 180.3587 seconds

Mismatches (if any):
Vertex     Dynamic          Dijkstra         Bellman-Ford
No mismatches found.
```

Static may be better for small(?) graphs...

Comparison with baseline

```
numvertices = 20000
numedges = 20000
numupdates = 5000
```

```
Dynamic Algorithm Update Time: 18.4621 seconds
Standard Dijkstra's with PQ Update Time: 182.3825 seconds
```

```
Mismatches (if any):
Vertex      Dynamic          Dijkstra
No mismatches found.
```

but dynamic is better for large graphs.

THANK YOU