Incremental Shortest Path

Xi Lin
Mechanical Engineering
Stevens Institute of Technology
Hoboken, USA
xlin26@stevens.edu

***Abstract*—This project implements an incremental all-pair shortest path (APSP) algorithm with optimal worse-case complexity $O(V^2)$, and applies it to solving the APSP problem of directed graphs with different number of vertices and edge densities. The classical Flyod Warshall algorithm with $O(V^3)$ time complexity is also implemented for comparison. Experiment results show that as the number of map points and edge density increase, the incremental algorithm becomes more efficient than the Floyd Warshall algorithm.**

***Index Terms*—shortest path, directed graph**

I. Introduction

All-Pairs Shortest Paths (APSP) problem is one of the most fundamental problems in the area of graph theory, the solution of which contains the shortest path between all pairs of vertices. Floyd Warshall algorithm (Algorithm 1) provides a brute force solution for every pair of vertices via looping through every vertex in the graph as an intermediate point and update the shortest path [1]. However, the time complexity of Floyd Warshall algorithm is $O(V^3)$, where V is the number of vertices, and hence it would be expensive to run on a large scale graph. In addition, usually only a small portion of the graph changes, hence there is no need to recompute the shortest path between every pair of vertices. Dynamic APSP problem has been studied for decades, and existing algorithms could be divided into three categories: Incremental algorithms [2]–[5] only deal with vertex or edge insertions, and edge weight decreases; On the contrary, decremental algorithms [6]–[8] only handle vertex or edge deletion, and edge weight increases; Fully dynamic algorithms [9]–[13] could manage both cases. This project focus on the edge insertion situation only, and I choose to implement the incremental algorithm from [2] with optimal worse-case complexity $O(V^2)$, and examine its performance, as well as that of Floyd Warshall, on the devised APSP problem.

II. Problem Setup

In this project, the APSP problem is embodied as the problem of finding the route of the cheapest overall flight price from any departure city to any destination city in a map. The map could be abstracted as a directed graph $G = (V, E)$, where V is the set of map points that represent cities, and E is the set of edges that represent available airlines. Fig. 1 shows an example solution from the designed GUI, where the route is visualized in the graph, and the price to travel through each edge is also shown.

As the Floyd Warshall algorithm runs a triple loop over all map points, the computation time is proportional to the number of map points, regardless of how many new edges are inserted. On the contrary, for every new edge the incremental algorithm identifies and updates pairs of map points whose cheapest route changes (Algorithm 2). Thus for the incremental algorithm, the number of map points, existing edges, and new edges may affect the number of pairs of map points that need an update in the cheapest route, as well as its computation time. In this project, the effects of first two factors are studied, and the number of new edges scales with the number of map points in a constant factor. Instead of directly varying the number of edges, the value of edge density is used in the experiments, which is defined as the ratio of the number of existing edges to the maximum number of edges. Fig. 2 visualizes graphs with 50 map points and different edge densities. In the following section, the process of generating maps with different numbers of map points and values of edge density, new edge insertion and the performance of two algorithms will be discussed.

Algorithm 1 Floyd Warshall solver

```
initialize all elements in the cost matrix as ∞
initialize all elements in the path matrix (denoted as next) as null
for each edge (u, v) do
    cost[u][v] = price[u][v]
    next[u][v] = v
end for
for each vertex v do
    cost[v][v] = 0
    next[v[v] = v
end for
for each vertex k do
    for each vertex i do
        for each vertex j do
            if cost[i][j] > cost[i][k] + cost[k][j] then
                cost[i][j] = cost[i][k] + cost[k][j]
                next[i][j] = next[i][k]
            end if
        end for
    end for
end for
```

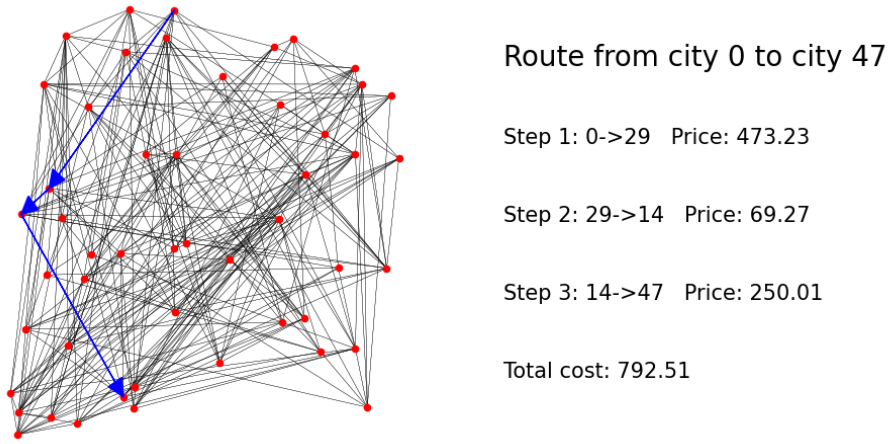

Fig. 1. An example route with cheapest total price from the start to the goal point. Blue arrows mark edges to travel through, and the price of each edge as well as the overall cost are visualized in the GUI.

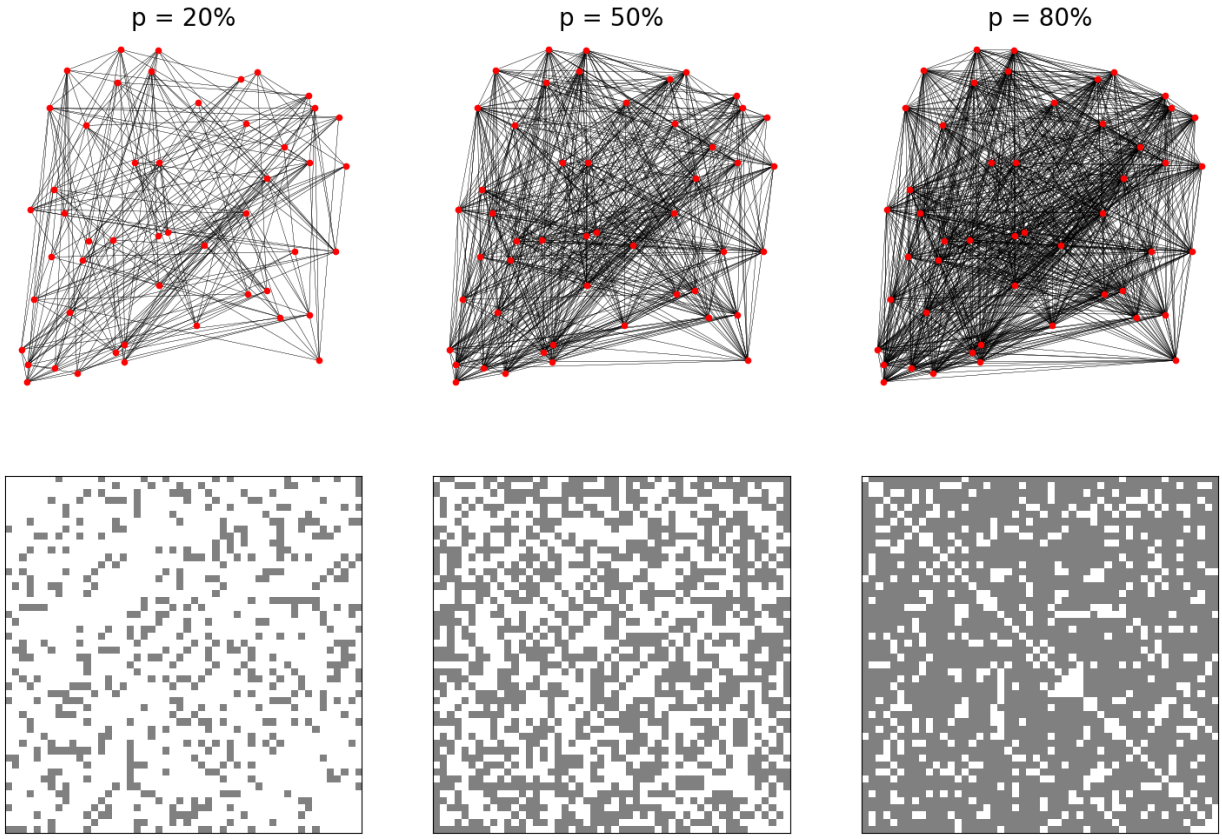

Fig. 2. Maps with 50 points and different edge densities, e.g., p = 20% means that the number of existing edges is 20% of the maximum number of edges. **Upper row:** Plots of the location of map points (shown as red dots) and edges (black lines). **Lower row:** Adjacency matrices, where a grey cube indicates an existing edge between points with index of the corresponding row and the column.

III. Experiment

A. Map generation

The hyperparameters used to generate a map are listed in Table I. When generating a map point, the position (x, y) is sampled from uniform distributions $x \sim U([0.0, w])$ and $y \sim U([0.0, h])$, but its distance to any existing map points could not be smaller than d, otherwise a new sample is drawn. After finishing map point generation, edges between pairs of random map points are created until the ratio of number of existing edges to maximum number of edges reach the designated edge

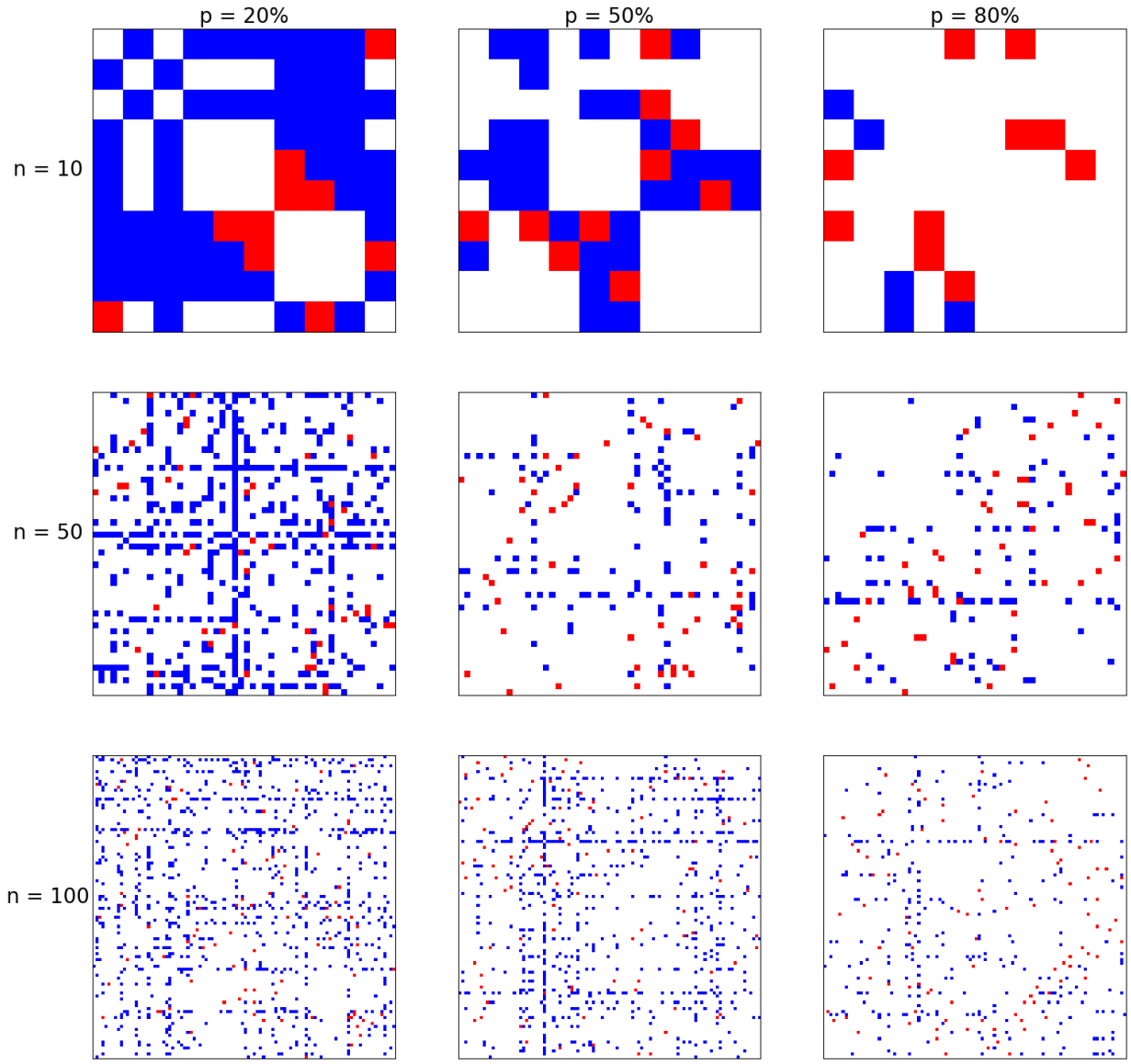

Fig. 3. Newly inserted edges (red blocks) and pairs of map points (blue blocks) that need update in the cheapest route when solving with the incremental algorithm.

TABLE I
MAP HYPERPARAMETERS

Hyperparameter	Notation	Value
map width	w	2000.0
map height	h	2000.0
minimum distance between map points	d	20.0
base price factor	f_p	1.0
price range factor	f_r	0.5

density p: The flight price of an edge $u \to v$ is sampled from the uniform distribution $U([f_p{\cdot}\text{dist}(u,v){\cdot}(1-f_r), f_p{\cdot}\text{dist}(u,v){\cdot}(1+f_r)])$, where $\text{dist}(u,v)$ denotes the distance between u and v. Hence the flight price of edge $u \to v$ and edge $v \to u$ are not necessarily equal.

B. Experiment setup

Since this project focuses on studying the performance of two algorithms on solving the APSP problem with edge insertion only, the APSP solution of the initial map is computed by Floyd Warshall algorithm and given as the prior information, and the experiments only concern about performance of both Floyd Warshall and the incremental algorithm on the following map with newly inserted edges. The experiments iterate over combination of the number of map points, $n = \{10, 50, 100, 500, 1000\}$, and edge density, $p = \{20\%, 50\%, 80\%\}$. The number of newly inserted edges is 0.5 times of the number of map points.

Algorithm 2 Update_affected_pairs

Require: cost matrix, new edge (u,v,price$[u][v]$)
  $S \leftarrow \emptyset$
  if cost$[u][v] >$ price$[u][v]$ **then**
    S = Update_affected_source(cost,(u,v,price$[u][v]$))
    Q.insert(v)
    V.insert(v)
    $P[v] = v$
  end if
  while Q is not empty **do**
    $y = Q$.front()
    Q.pop()
    for each $x \in S[P[y]]$ **do**
      if cost$[x][y] >$ cost$[x][u]$+price$[u][v]$+cost$[v][y]$ **then**
        cost$[x][y]$ = cost$[x][u]$+price$[u][v]$+cost$[v][y]$
        if x is not u **then**
          next$[x][y]$ = next$[x][u]$
        else
          next$[x][y]$ = v
        end if
        if y is not v **then**
          $S[y]$.insert(x)
        end if
      end if
    end for
    for each w s.t. edge(y, w) **do**
      bool c_1 = cost$[u][w] >$ cost$[v][w]$+price$[u][v]$
      bool c_2 = cost$[v][w] ==$ cost$[v][y]$+cost$[y][w]$
      if not V.find(w) and c_1 and c_2 **then**
        Q.insert(w)
        V.insert(w)
        $P[w] = y$
      end if
    end for
  end while

Algorithm 3 Update_affected_sources

Require: cost matrix, new edge (u,v,price$[u][v]$)
  $S \leftarrow \emptyset$
  if cost$[u][v] >$ price$[u][v]$ **then**
    Q.insert(u)
    V.insert(u)
    $S[v]$.insert(u)
  end if
  while Q is not empty **do**
    $x = Q$.front()
    Q.pop()
    for each z s.t. edge (z, x) exists **do**
      bool affected = cost$[z][v] >$cost$[z][u]$+price$[u][v]$
      if not V.find(z) and affected **then**
        Q.insert(z)
        V.insert(z)
        $S[v]$.insert(z)
      end if
    end for
  end while
  return S

C. Experiment results

Fig. 3 visualizes the inserted edges and pairs of map points that require recomputation of the cheapest route between them when solving with the incremental algorithm. When the number of map points is greater than 100, the blocks are too small and sparse to see, hence the corresponding results are not shown in Fig. 3. Note that since Floyd Warshall algorithm needs to recompute the route of every pair of map points each time, the white space in matrices indicates the amount of extra operations that Floyd Warshall algorithm has to perform. It could be seen that as the number of map points or the edge density increases, the distribution of blue blocks is sparser and they occupied less space in the matrix, which means that the proportion of pairs of map points that needs recomputation of route decreases, and the incremental solver is more efficient than the Floyd Warshall algorithm. Fig. 4 shows the change of proportion value of affected pairs with the number of map points and edge density. When $p = 20\%$, The proportion value

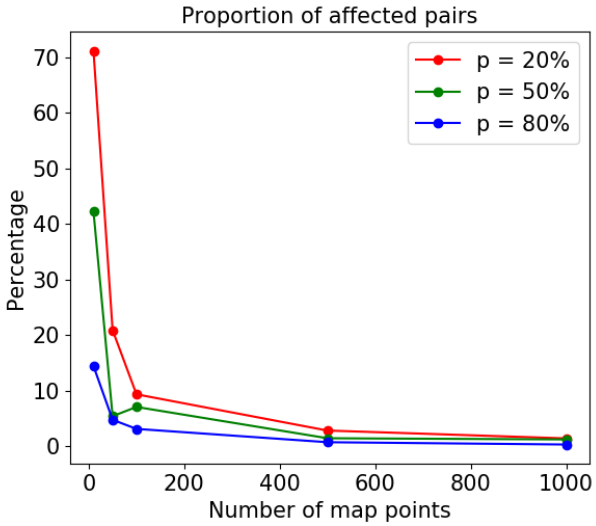

Fig. 4. Proportion of affected pairs of map points when solving with the incremental algorithm.

drop significantly from over 70% to less than 10% as the number of map points increases to 100, and is close to 0 when the number of map points reaches 1000. The other two curves show similar trends, and altogether it could also be noticed that edge density has a greater effect on the proportion value in small maps.

For each newly inserted edge $u \rightarrow v$, the incremental algorithm (Algorithm 2) performs a breadth-first search (BFS) starting from u as the root node, which loops through every edge with the current node as the end point and identify the set of affected source map points; Another BFS is performed

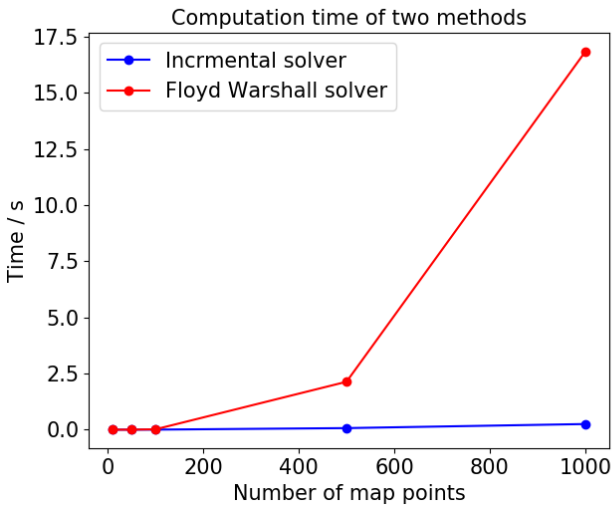

Fig. 5. Computation time of the incremental and Floyd Warshall solver when $p = 20\%$.

with v as the start node to get the set of affected target map points, then the affected pairs are the combination of affected source and target map points. Thus the computation needed for the incremental algorithm depends on the local graph structure surrounds the map point u and v. For a pair of map points, when the number of all map points or the edge density increases, the number of available routes between them is also likely to increase, and it's more possible that the cheapest path is not affected by the insertion of a new edge. Therefore, the number of affected pairs of map points probably decreases as the number of map points or the edge density increases.

Fig. 5 shows the variation of computation time of two algorithms as the number of map points increases. Since the number of operations performed by Floyd Warshall algorithm is not affected by the edge density p, and the difference in computation time of the incremental algorithm when p varies is negligible compared to the difference in computation time between two algorithms, only the results of $p = 20\%$ are shown in the plot. It's clear that the computation time of Floyd Warshall algorithm grows drastically compared to the incremental algorithm, which could be clearly explained by the analysis of the incremental algorithm in former paragraphs.

IV. Conclusion

In this project, I implement an incremental all-pair shortest path (APSP) algorithm from [2], and apply it to solving the problem of finding the route of the cheapest overall flight price for all pairs of points in a map under new edge insertion circumstance. The classical Floyd Warshall algorithm which needs to recompute the cheapest path for every pairs of map points is also implemented for comparison. Experiment results show that as the number of map points and edge density increase, the proportion of pairs of map points that are affected by newly inserted edges and need recomputation of cheapest route decreases, hence the incremental algorithm becomes more efficient than the Floyd Warshall algorithm. For example, to update the routes of cheapest price in the map of 1000 map points when 500 new edges are inserted, the computation time of the Floyd Warshall solver is over 16 seconds, while the incremental solver only needs less than 0.5 seconds.

References

[1] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

[2] Arie Slobbe, Elisabetta Bergamini, and Henning Meyerhenke. *Faster incremental all-pairs shortest paths*. KIT, 2016.

[3] Chaoyi Pang, Ramamohanarao Kotagiri, and Guozhu Dong. Incremental fo (+,¡) maintenance of all-pairs shortest paths for undirected graphs after insertions and deletions. In *International Conference on Database Theory*, pages 365–382. Springer, 1999.

[4] Ganesan Ramalingam and Thomas Reps. An incremental algorithm for a generalization of the shortest-path problem. *Journal of Algorithms*, 21(2):267–305, 1996.

[5] Giorgio Ausiello, Giuseppe F Italiano, Alberto Marchetti Spaccamela, and Umberto Nanni. Incremental algorithms for minimal length paths. *Journal of Algorithms*, 12(4):615–638, 1991.

[6] Aaron Bernstein. Maintaining shortest paths under deletions in weighted directed graphs. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 725–734, 2013.

[7] Surender Baswana, Ramesh Hariharan, and Sandeep Sen. Improved decremental algorithms for maintaining transitive closure and all-pairs shortest paths. *Journal of Algorithms*, 62(2):74–92, 2007.

[8] Surender Baswana, Ramesh Hariharan, and Sandeep Sen. Maintaining all-pairs approximate shortest paths under deletion of edges. In *SODA*, volume 3, pages 394–403, 2003.

[9] Ittai Abraham, Shiri Chechik, and Sebastian Krinninger. Fully dynamic all-pairs shortest paths with worst-case update-time revisited. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 440–452. SIAM, 2017.

[10] Camil Demetrescu and Giuseppe F Italiano. Fully dynamic all pairs shortest paths with real edge weights. *Journal of Computer and System Sciences*, 72(5):813–837, 2006.

[11] Mikkel Thorup. Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In *Scandinavian Workshop on Algorithm Theory*, pages 384–396. Springer, 2004.

[12] Camil Demetrescu and Giuseppe F Italiano. A new approach to dynamic all pairs shortest paths. *Journal of the ACM (JACM)*, 51(6):968–992, 2004.

[13] Monika R Henzinger, Philip Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *journal of computer and system sciences*, 55(1):3–23, 1997.