COMPARATIVE ANALYSIS

NAME - SHIVANSH MANI TRIPATHI
MIS - 112015131

UNDER GUIDANCE OF
DR RANJITH NAIR

DIJKSTRA

VS

A* Algorithm

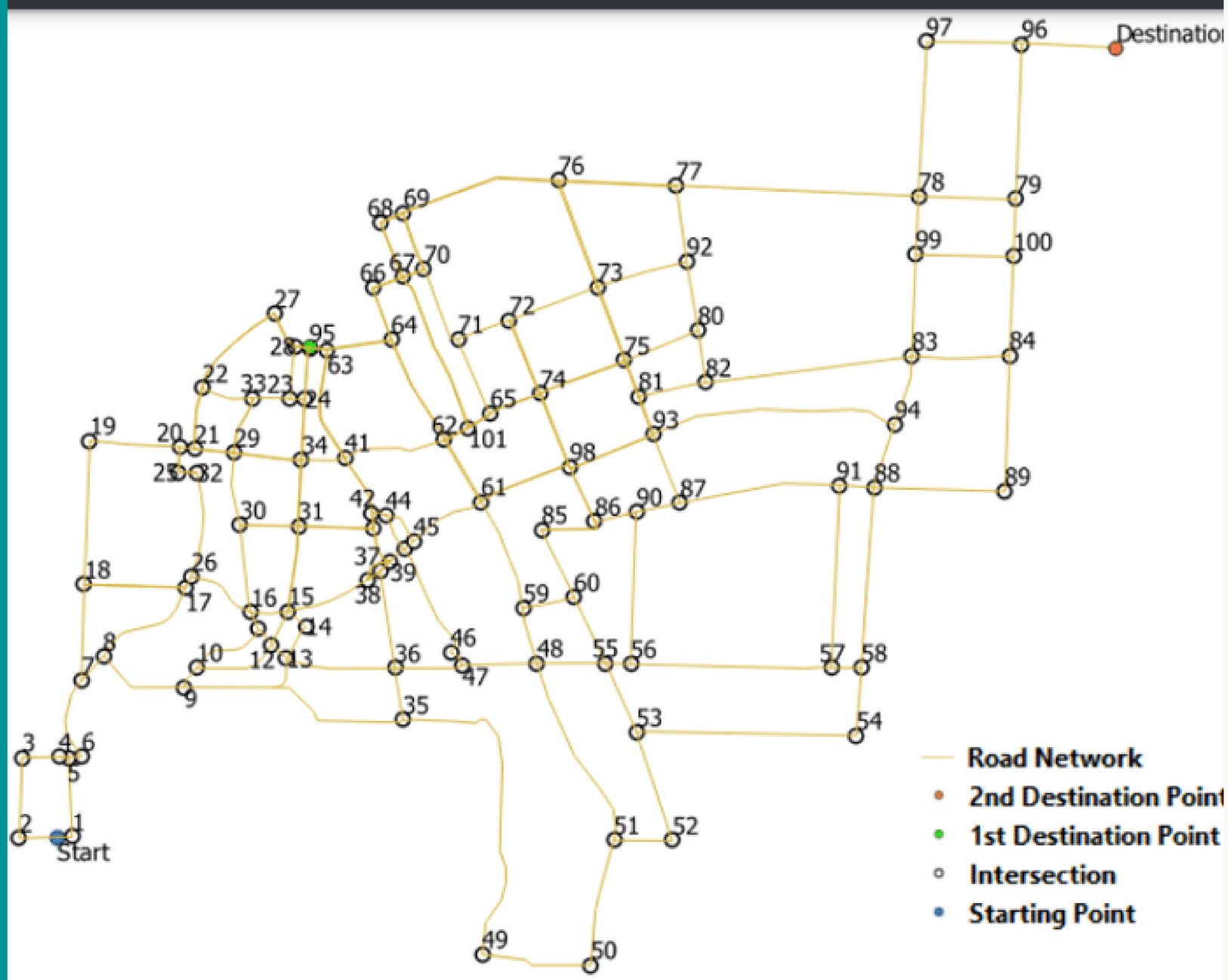
TODAY'S PRESENTATION

POINTS OF DISCUSSION

Finding the shortest path in direction effective is essential.

To solve this shortest path problem, we usually using Dijkstra or A* algorithm. .

Today we will compare those two algorithms in solving this shortest path problem



PROCEDURE

Between the starting point and destination point, we manually generate plenty of points (nodes), which is an intersection between 2 or more roads and find the distance between them (edges).

After that, we apply the data to Dijkstra's Algorithm, A* algorithm, and make a comparison between the two algorithms based on the running time, the number of loops with the number of points as the factor

SAMPLE DATA

Table 1. Distance between points connected by the road

From point	To point	Distance / Weight (in km)
start	2	0.23
start	1	0.084
1	5	0.5
2	3	0.5
…	…	…
101	65	0.14

Table 2. Distance between point to 1st destination point

From	Distance / Weight (in km)
1	3.283108
2	3.441482
3	3.018227
4	2.889003
…	…
34	0.677927

Table 3. Distance between point to 2nd destination point

From	Distance / Weight (in km)
1	7.91655
2	8.185879
3	7.889952
4	7.696507
…	…
101	4.543907

STEPS

DIJKSTRA

1. Set all points distance to infinity except for the starting point set distance to 0.

2. Set all points, including starting point as a non-visited node.

3. Set the non-visited node with the smallest current distance as the current node ”C.”

4. For each neighbor “N” of your current node: add the current distance of “C” with the weight of the edge connecting “C” – “N.”
If it’s smaller than the current distance of “N,” set it as the new current distance of “N.”

5. Mark the current node “C” as visited.

6. Repeat the step above from step 3 until the destination point is visited.

SOURCE CODE

```
while queue:
    queue_count = queue_count + 1 #loop count
    # find min distance which wasn't marked as current
    key_min = queue[0]
    min_val = path[key_min]
    for n in range(1, len(queue)):
        if path[queue[n]] < min_val:
            key_min = queue[n]
            min_val = path[key_min]
    cur = key_min
    queue.remove(cur)

    for i in graph[cur]:
        alternate = graph[cur][i] + path[cur]
        if path[i] > alternate:
            path[i] = alternate
            adj_node[i] = cur
```

Figure 3. Dijkstra's Algorithm Source Code

STEPS

A*ALGORITHM

1. Set all point distance to infinity except for the starting point set distance to 0.

2. Set all points, including start point as a non-visited node. 3. Set the non-visited node with the smallest current distance as the current node ”C.”

4. For each neighbor “N” of your current node: add the current distance of “C” with the weight of the edge connecting “C” – “N” and the weight to the destination point (heuristic). If it’s smaller than the current distance of “N,” set it as the new current distance of “N.”

5. Mark the current node “C” as visited.

6. Repeat the step above from step 3 until one of the neighbors “N” is the destination point.

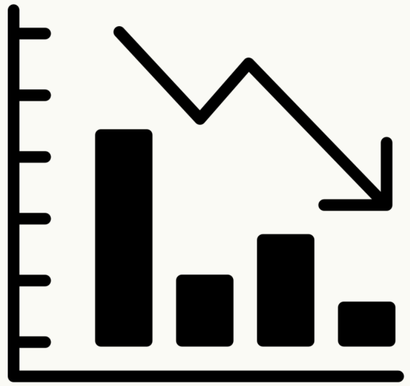
SOURCE CODE

```
while queue and status == 1:
    queue_count = queue_count + 1 #loop count
    # find min distance which wasn't marked as current
    key_min = queue[0]
    min_val = path_heu[key_min]
    for n in range(1, len(queue)):
        if path_heu[queue[n]] < min_val:
            key_min = queue[n]
            min_val = path_heu[key_min]
    cur = key_min
    queue.remove(cur)
    for i in graph[cur]:
        alternate = graph[cur][i] + path[cur]
        alternate_heu = graph[cur][i] + path[cur] + heu[i]
        if path_heu[i] > alternate_heu:
            path_heu[i] = alternate_heu
            path[i] = alternate
            adj_node[i] = cur

            if 'FINISH' in graph[cur]:
                #while_loop_break
```

Figure 4. A* Algorithm Source Code

RESULTS AND DISCUSSION



The experiments were performed on laptop Windows 10 pro with 8GB of RAM and Intel i5-6200 processor with PyCharm version 2018.3.5 64-bit version evaluation copy. 3.1.Starting point to 1st destination point (Total of 36 points include start and destination point) Running Time We take 3 tries for each Dijkstra and A*, running times are shown in the tables.

RESULTS AND DISCUSSION

DESTINATION 1

Table 4. Running Time from starting point to 1 destination point using Dijkstra and A* algorithm

Test Number	Running Time Dijkstra (nanoseconds)	Running Time A* (nanoseconds)	Total Distance (in km)
1	997.200	996.800	4.504
2	998.200	996.900	4.504
3	997.800	997.400	4.504

<- RUNNING TIME

LOOP COUNTS ->

Table 5. Loop counts from starting point to 1st destination point using Dijkstra and A* algorithm

Test Number	Loop Count Dijkstra	Loop Count A*	Total Distance (in km)
1	36	26	4.504
2	36	26	4.504
3	36	26	4.504

RESULTS AND DISCUSSION

DESTINATION 2

Table 6. Running Time from starting point to 2nd destination point using Dijkstra and A* algorithm

Test Number	Running Time Dijkstra (nanoseconds)	Running time A* (nanoseconds)	Total Distance (in km)
1	2,991.400	2,992.000	10.138
2	3,987.800	2,992.800	10.138
3	3,989.100	2,992.100	10.138

<-RUNNING TIME

LOOP COUNTS->

Table 7. Loop counts from starting point to 2nd destination point using Dijkstra and A* algorithm

Test Number	Loop Count Dijkstra	Loop Count A*	Total Distance (in km)
1	103	72	10.138
2	103	72	10.138
3	103	72	10.138

CONCLUSION

In conclusion, the use of Dijkstra's algorithm and A* algorithm in the shortest path is essential will give the same output in no time when being used on the town or regional scale maps.

But on a large scale map, A* will provide the solution faster than Dijkstra.
A* scan the area only in the direction of destination because of the heuristic value that counted in the calculation, whereas Dijkstra searches by expanding out equally in every direction and usually ends up exploring a much larger area before the target is found resulting making it slower than A*.

This can be proven by the loop count of Dijkstra
and A*, the more points (nodes) the higher the difference between the loop count nor the time.

THANK YOU !