Intelligent Transportation Systems Traveling Salesman Problem (ITS-TSP) – A Specialized TSP with Dynamic Edge Weights and Intermediate Cities

Jeffrey Miller, Sun-il Kim, Timothy Menard

University of Alaska, Anchorage

IEEE 13th Intelligent Transportation Systems Conference

September 21, 2010

Outline

- Problem Statement
- Algorithm Description
- Sample Execution of ITS-TSP
- Conclusion

Problem

- What is the *fastest* way to get from my current location to a set of locations (where ordering is irrelevant) before returning to my initial location?
- Note that the order of visiting each of the locations is not important, but just that I visit all of them before returning to my starting point
- Example Application – a flower delivery driver leaving the florist shop needs to make stops at 10 different locations in a day before returning to the shop

Traveling Salesman Problem (TSP)

- Given a graph $G=(V, E)$ with $V=\{v_1,\ldots, v_n\}$ and $E=\{e_1,\ldots,e_p\}$
- Each edge $e_i \in E$ connects two distinct vertices from V and has a weight $w_i > 0$ associated with it
- A start node $n_0 \in V$ represents the start and end node of the overall route
- The TSP traverses all vertices in V exactly once in a manner that minimizes the overall cost of the route, determined by summing the weights of the edges traversed
  - In other words, the TSP finds a Hamiltonian cycle with minimum cost

Intelligent Transportation Systems Traveling Salesman Problem (ITS-TSP)

- Three variations exist differentiating the ITS-TSP from the TSP
  1. Edge weights can change constantly in real-time
  2. Only a subset of the nodes in the graph needs to be traversed (intermediate cities)
  3. A node can be visited more than once if that provides a faster route (since the edge weights can change while traveling)
- So the solution is recomputed after each node in the intermediate set is visited, but the current node then becomes the new start node with the end node still the original start node

Related Work

- Fastest path algorithms have been studied for quite some time
  - Single Source Shortest Path – Dijkstra, Bellman-Ford
  - All Pairs Shortest Path – Floyd-Warshall, Johnson
- With transportation graphs, the edge weights can change quite frequently
  - Dynamic Fastest Path (DynFast) – Demestrescu and Italiano, Miller and Horowitz
- Dynamic Traveling Salesman Problem (D-TSP) allows an optimization when adding nodes to an existing graph
- D-TSP with Time Windows adds a time restriction as to when each node can be visited
- Although there are many variations to the traditional TSP, none of these algorithms takes into account the specific details of the ITS-TSP, which is yet another variation on the TSP

ITS-TSP Pseudo-Code

```
1  -- returns the ROUTE that starts at start_node, ends at end_node, and
2  -- visits all of the nodes in int_nodes, with the minimum overall cost
3  -- NOTE: if we are in the middle of the route, start_node will not be
4  -- the same as end_node
5  ROUTE ITS-TSP(GRAPH g, NODE_SET int_nodes, NODE start_node, NODE end_node) {
6    int_nodes = int_nodes - {start_node} -- in case start_node Є int_nodes
7    PATH_SET original_path_set = {}
8    for each (NODE src Є int_nodes U {start_node}) {
9      for each (NODE dest Є int_nodes) {
10       if (src != dest) {
11         -- using All_Pairs_All_Paths algorithm from [3]
12         -- this variable contains all of the fastest paths between all of
13         -- the nodes in start_node and int_nodes
14         original_path_set = original_path_set U fastest_path(g, src, dest)
15       }
16     }
17   }
18   -- find the fastest route from the original_path_set, which contains
19   -- the fastest paths between start_node, int_nodes, and end_node
20   PATH_SET path_set = original_path_set
21   for each (NODE u Є int_nodes) {
22    ROUTE route = {} U path_set.get_minimum_path(start_node, u)
23     path_set.remove_all_paths_with_src(start_node)
24     path_set.remove_all_paths_with_dest(u)
25     while (path_set != {}) {
26       PATH p = path_set.get_minimum_path()
27       path_set.remove_path(p)
28       -- cycles can exist in overall route traversed,
29       -- but not in route determined based on snapshot
30       if (!route.contains_cycle_with_path(p)) {
31         route = route U p
32         path_set.remove_all_paths_with_src(p.src)
33         path_set.remove_all_paths_with_dest(p.dest)
34       }
35     }
36     -- order paths so dest from one is src of next
37     fin_route[u] = {}
38     PATH curr_path = route.get_path_from_src(start_node)
39     while (curr_path != NULL) {
40       fin_route[u] = fin_route[u] U curr_path
41       curr_path = route.get_path_from_src(curr_path.dest)
42     }
43     -- complete the route with the path from the last node to end_node by
44     -- calling fastest_path function from [3]
45     fin_route[u] = fin_route[u] U fastest_path(g, fin_route[u].dest, end_node)
46     path_set = original_path_set
47   } -- end for each
48   -- find route with minimum cost from the fin_route array
49   ROUTE minimum_route = {}
50   for each (t Є int_nodes) {
51     if (minimum_route == {} OR minimum_route.weight > fin_route[t].weight) {
52       minimum_route = fin_route[t]
53     }
54   }
55   return minimum_route
56 } -- end ITS-TSP function
```

ITS-TSP Algorithm

INPUT = Graph, Set of intermediate nodes to visit, Start node, End node

OUTPUT = Route of minimum cost beginning at start node, visiting all intermediate nodes, terminating at end node

Step 1

- Create a sub-graph of the original graph with nodes comprised of the start node, intermediate nodes, and end node. The weights on the edges will correspond to the minimum cost path between the nodes

Step 2

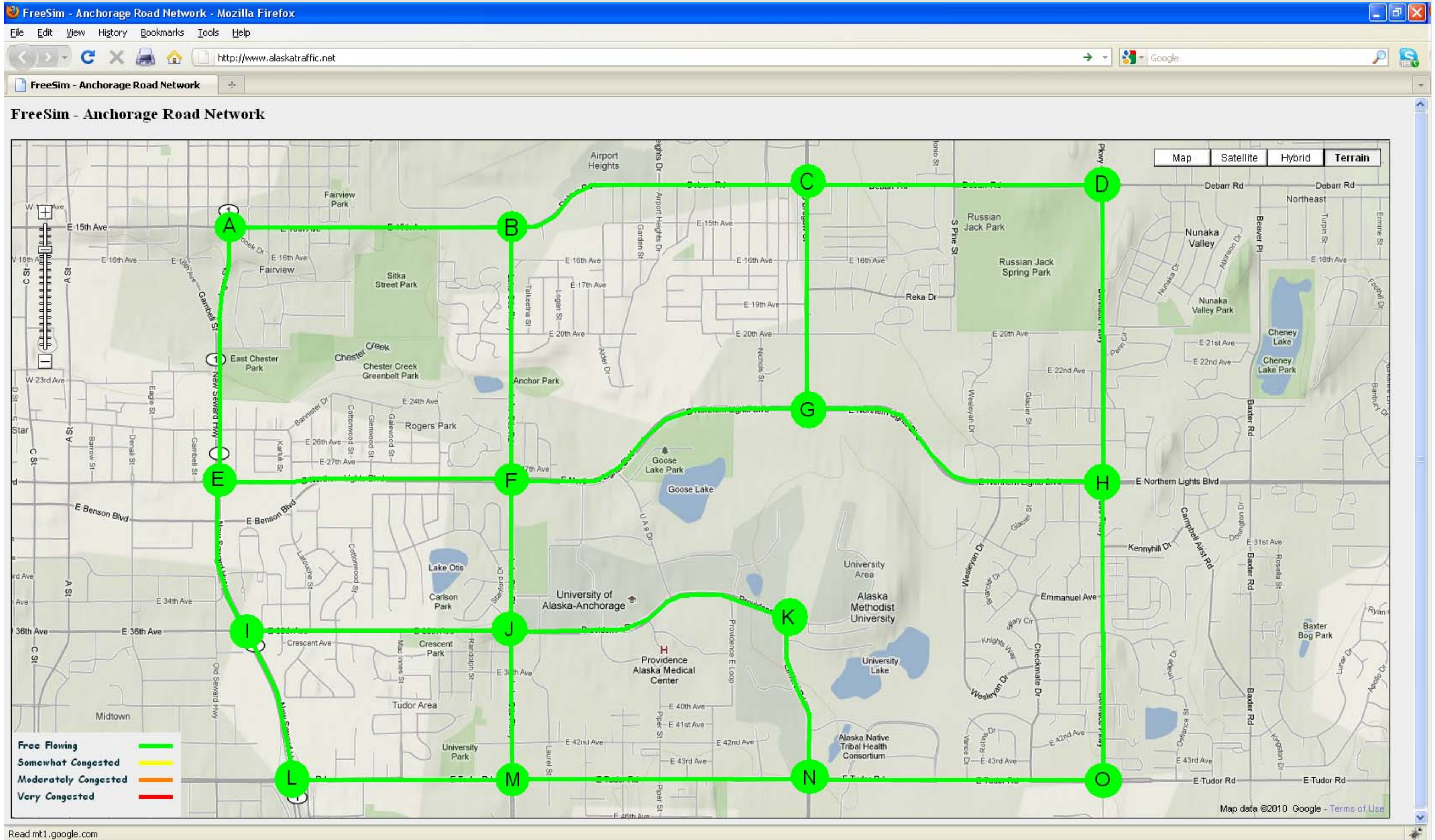
- Iterate over all of the intermediate nodes
- Determine the minimum weight path from start node to intermediate node and add it to the potential fastest route
- From the original set of paths found in step 1, remove all paths that have a source of the start node
- From the original set of paths found in step 1, remove all paths that have a destination of the intermediate node
- As long as the set of paths still contains more paths
  - Determine the path with the minimum overall cost that still remains
  - As long as that path does not create a cycle in our potential fastest route, add it to the potential fastest route
  - From the original set of paths found in step 1, remove all paths that have a source of the source node
  - From the original set of paths found in step 1, remove all paths that have a destination of the destination node
- Order paths in the potential fastest route from start node where destination of one path is source of next path
- Add the fastest path from the destination of the route back to the start node to the potential fastest route
- Store the potential fastest route in an array and reset the potential fastest route to be empty for the next iteration

Step 3

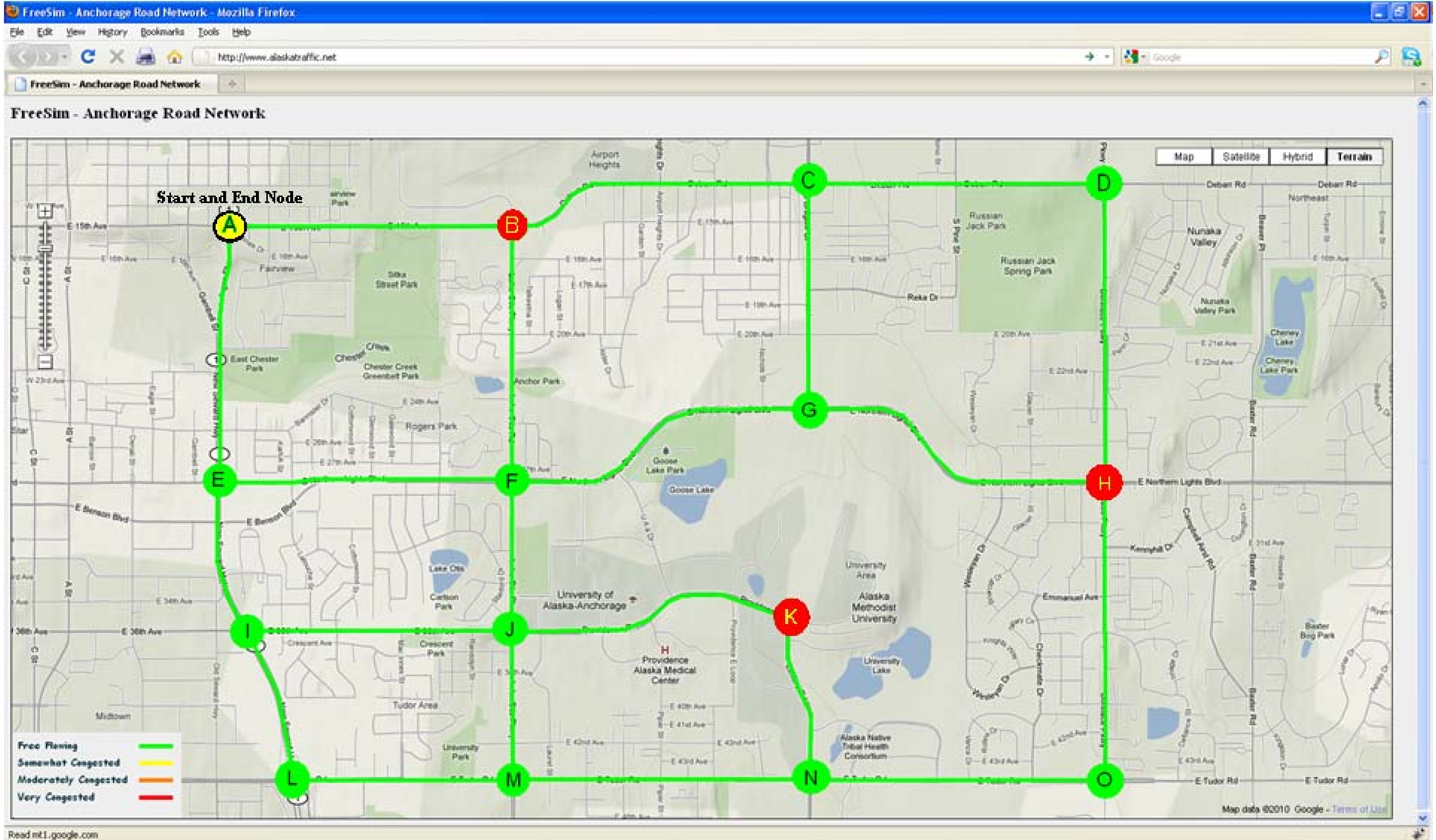
- From all the routes found in Step 2, find the route with minimum weight and return it

Sample Execution on Practical Application

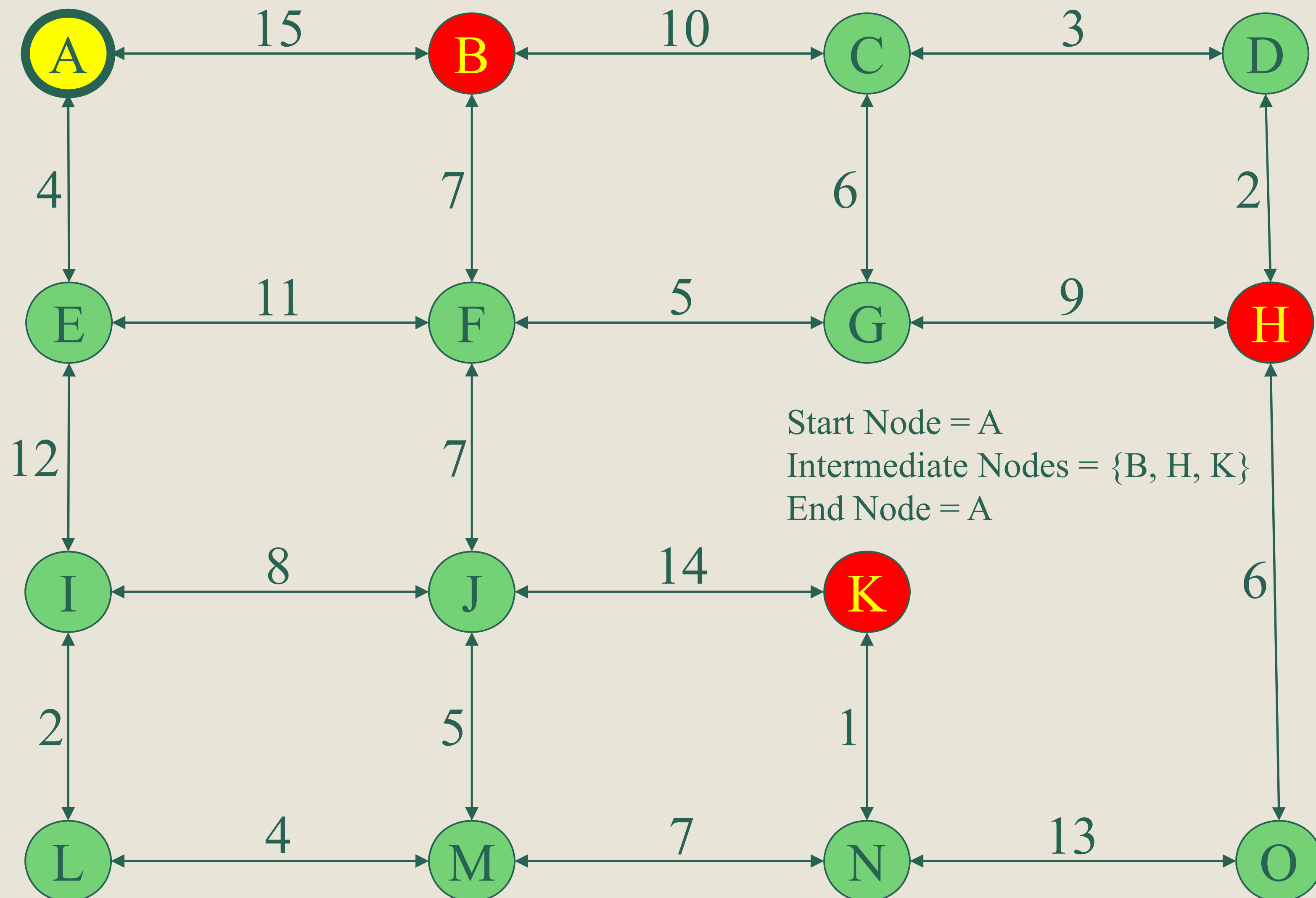
Roadways for Sample Execution



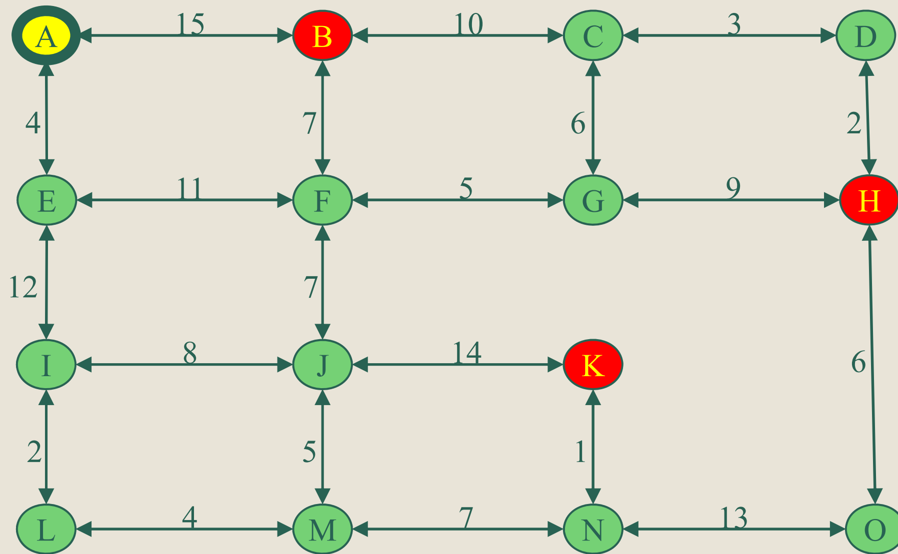
Start Node, End Node, Int Nodes

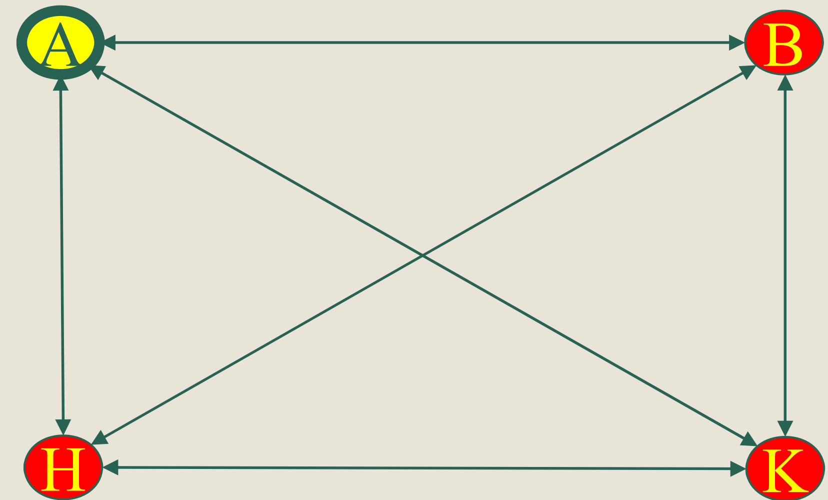


Transportation Graph with Weights



Sub-Graph with Start, Intermediate, and End Nodes



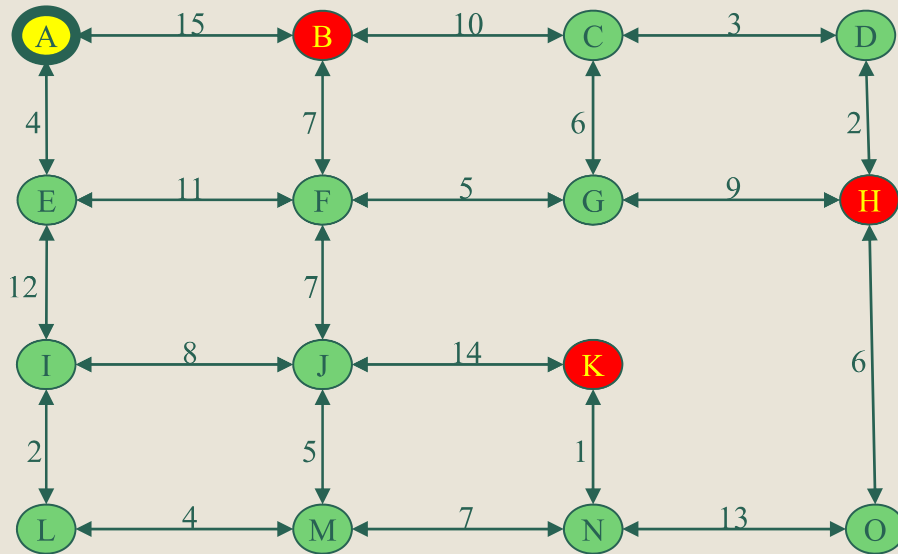


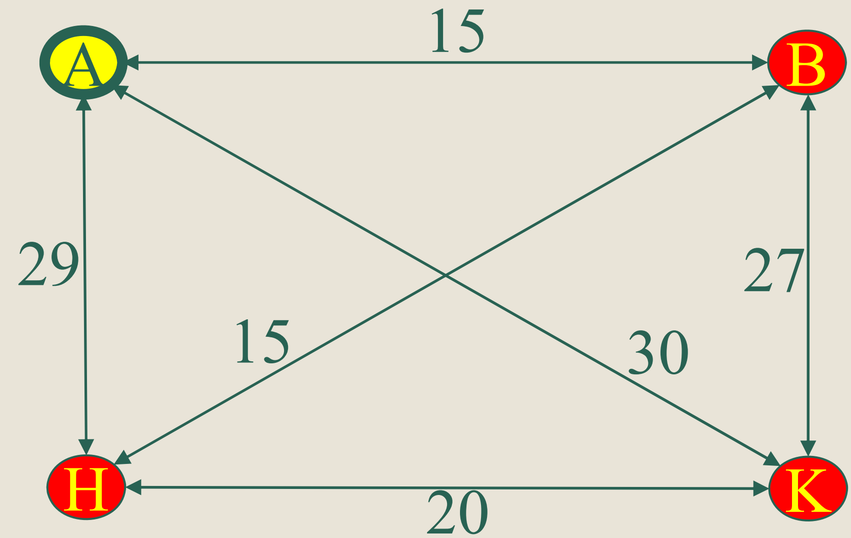
- But since there are multiple paths from one node to any other node, which path do we choose?
  - Starting at A and visiting B, H, and K (in any order) before returning to A has the following potential solutions:

A B H K A	A H B K A	A K B H A
A B K H A	A H K B A	A K H B A

  - How many different paths is this?
    - Factorial with respect to the number of intermediate destinations!

Step 1 at t_1 – Create Sub-Graph

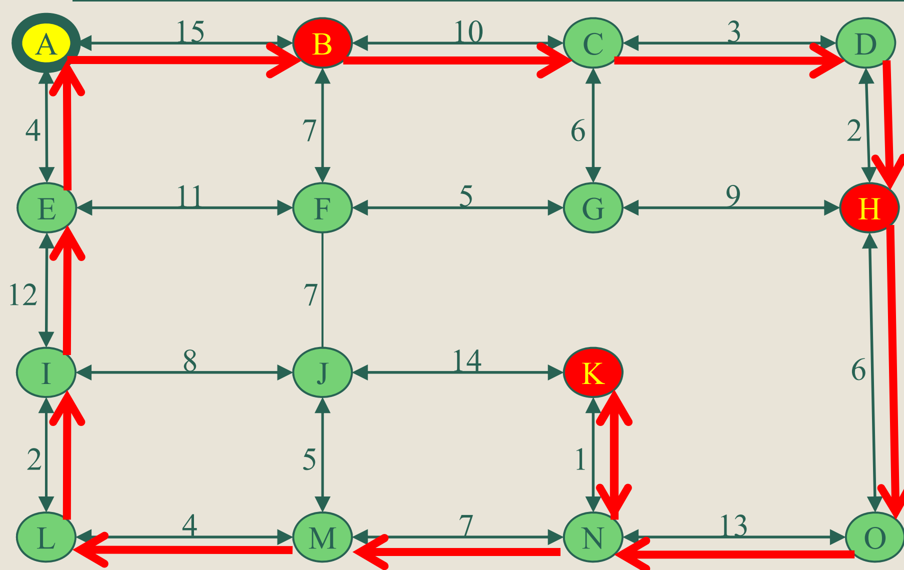


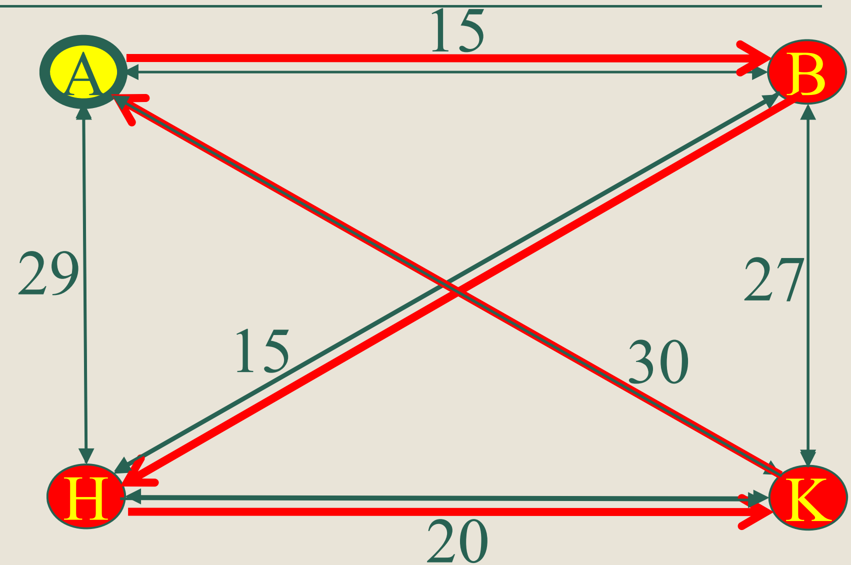


		Source			
		A	B	H	K
Destination	A		15 AB	29 HGFEA	30 KNMLIEA
	B	15 AB		15 HDCB	27 KNMJFB
	H	29 AEFGH	15 BCDH		20 KNOH
	K	30 AEILMNK	27 BFJMNK	20 HONK	

Start Node = A
Intermediate Nodes = {B, H, K}
End Node = A

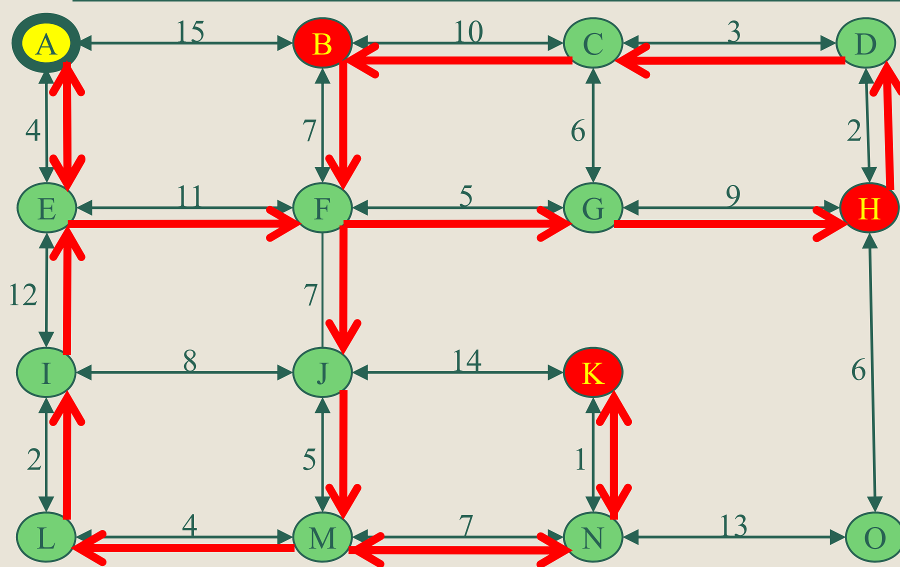
Step 2 at t_1 – Iterate over Intermediate Nodes

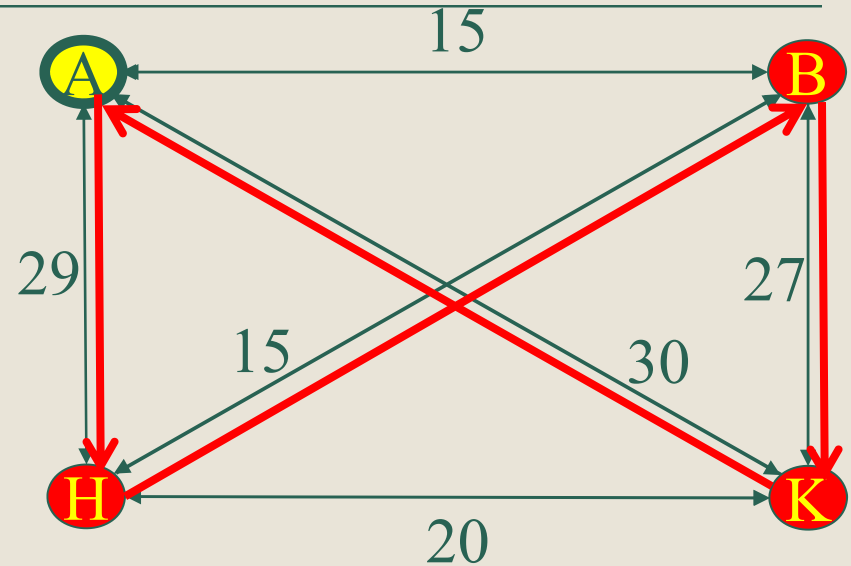




- Starting from A with intermediate node B
  - Add AB to the potential fastest route
  - Remove all edges that emanate from A or terminate at B
  - Choose next shortest path that does not make a simple cycle – BH
  - Remove all edges that emanate from B or terminate at H
  - Choose next shortest path that does not make a simple cycle – HK
  - Remove all edges that emanate from H or terminate at K
  - Since there are no more intermediate nodes, order the paths – AB BH HK
  - Now add the path back to the starting node – AB BH HK KA
    - Total cost = 80
    - Actual path = **A B** C D **H** O N **K** N M L I E **A**

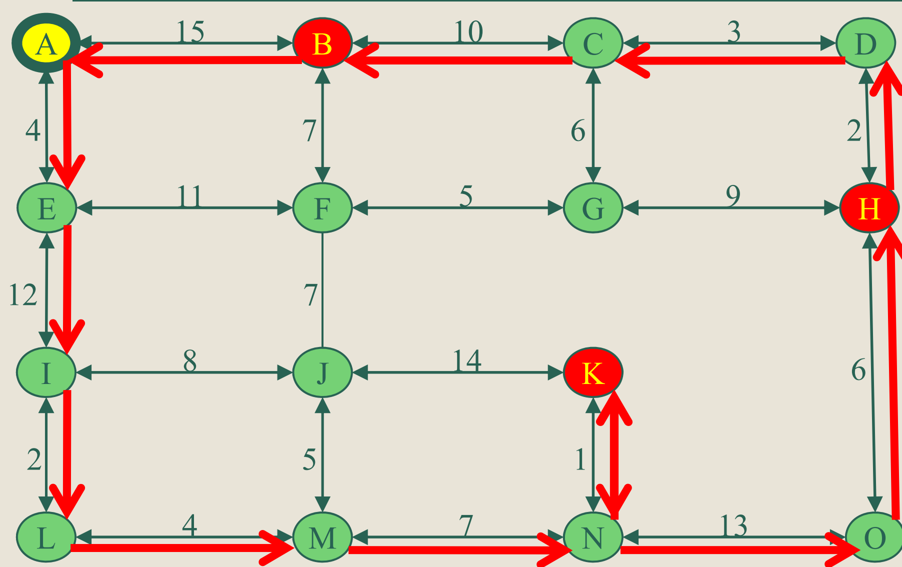
Step 2 at t_1 – Iterate over Intermediate Nodes

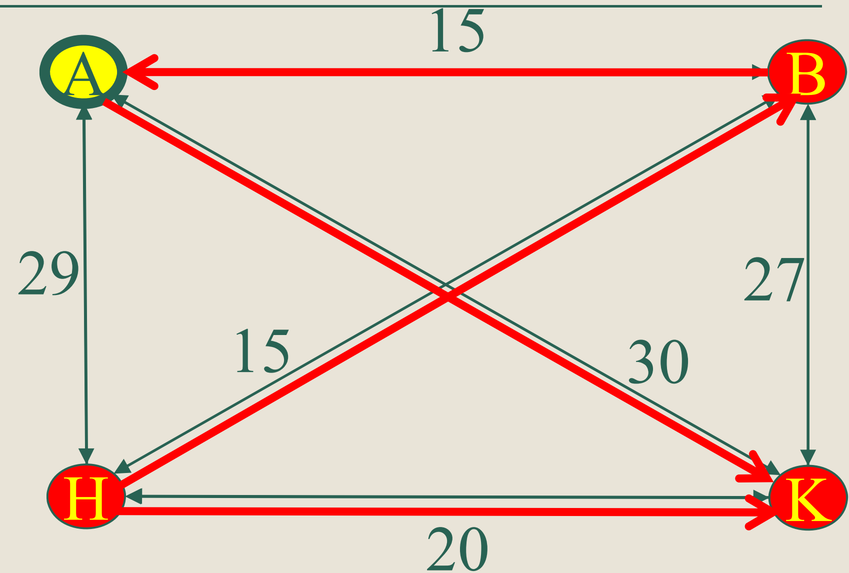




- Starting from A with intermediate node H
  - Add AH to the potential fastest route
  - Remove all edges that emanate from A or terminate at H
  - Choose next shortest path that does not make a simple cycle – HB
  - Remove all edges that emanate from H or terminate at B
  - Choose next shortest path that does not make a simple cycle – BK
  - Remove all edges that emanate from B or terminate at K
  - Since there are no more intermediate nodes, order the paths – AH HB BK
  - Now add the path back to the starting node – AH HB BK KA
    - Total cost = 101
    - Actual path = **A** E F G **H** D C **B** F J M N **K** N M L I E **A**

Step 2 at t_1 – Iterate over Intermediate Nodes



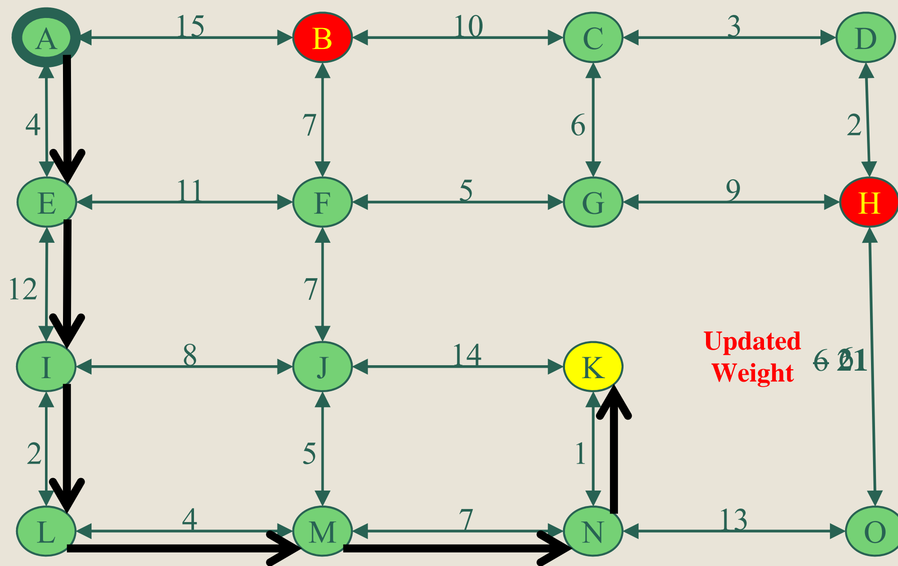


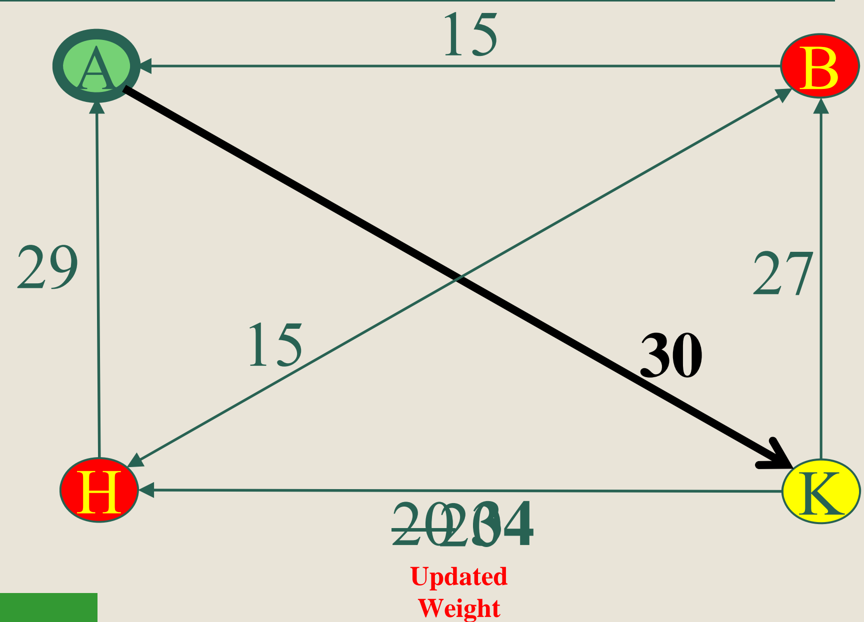
- Starting from A with intermediate node K
  - Add AK to the potential fastest route
  - Remove all edges that emanate from A or terminate at K
  - Choose next shortest path that does not make a simple cycle – HB
  - Remove all edges that emanate from H or terminate at B
  - Choose next shortest path that does not make a simple cycle – KH
  - Remove all edges that emanate from K or terminate at H
  - Since there are no more intermediate nodes, order the paths – AK KH HB
  - Now add the path back to the starting node – AK KH HB BA
    - Total cost = 80
    - Actual path = **A** E I L M N **K** N O **H** D C **B** **A**

Step 3 at t_1 – Choose First Path

- Here are the three routes determined in Step 2 at time t_1
  - **A B** C D **H** O N **K** N M L I E **A** with cost 80
  - **A** E F G **H** D C **B** F J M N **K** N M L I E **A** with cost 101
  - **A** E I L M N **K** N O **H** D C **B A** with cost 80
- Since two routes have the same cost, randomly pick one
  - Since the weights can change constantly, without prediction there is no real strategy for choosing one route over another
- For this example, we will choose the third route, which means we will start going from A to K by A E I L M N K

Step 1 at t_2 – Create Sub-Graph





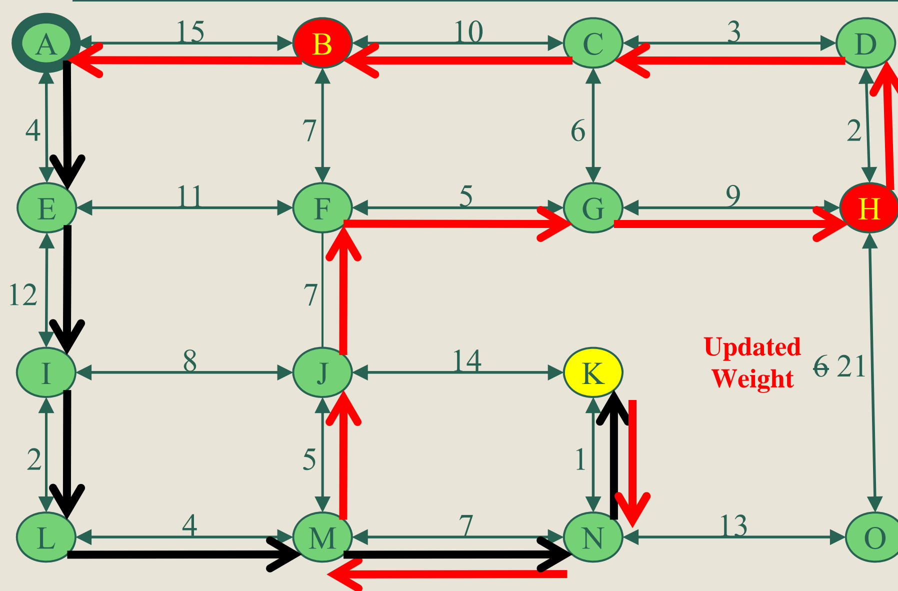
		Source		
		B	H	K
Destination	A	15 BA	29 HGFEA	30 KNMLIEA
	B		15 HDCB	27 KNMJFB
	H	15 BCDH		**34** **KNMJFGH**

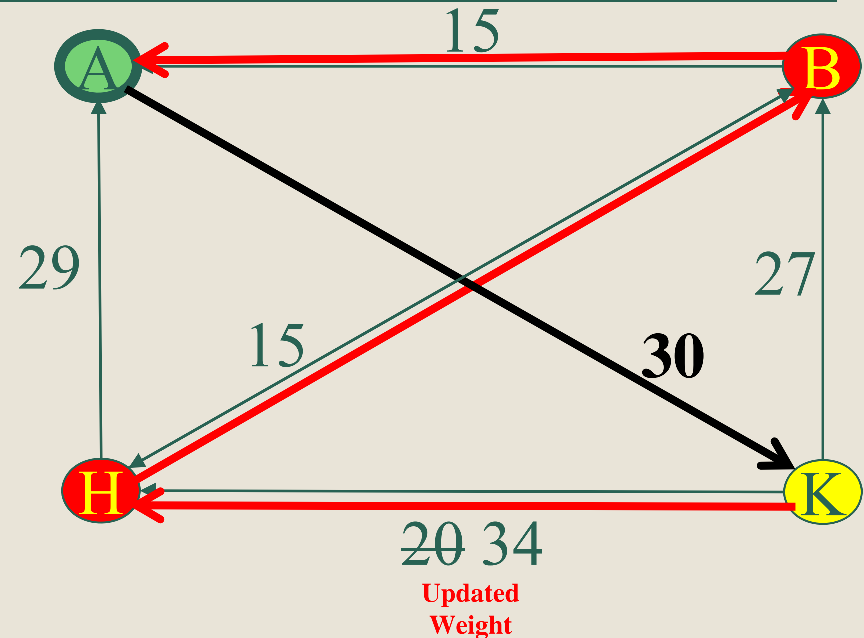
Start Node = K
Intermediate Nodes = {B, H}
End Node = A

Step 2 at t_2 – Iterate over Intermediate Nodes

- Starting from K with intermediate node B
  - Add KB to the potential fastest route
  - Remove all edges that emanate from K or terminate at B
  - Choose next shortest path that does not make a simple cycle – BH
  - Remove all edges that emanate from B or terminate at H
  - Since there are no more intermediate nodes, order the paths – [AK] KB BH
  - Now add the path back to the starting node – [AK] KB BH HA
    - Total cost = 101
    - Actual path = [**A** E I L M N **K**] N M J F **B** C D **H** G F E **A**

Step 2 at t_2 – Iterate over Intermediate Nodes



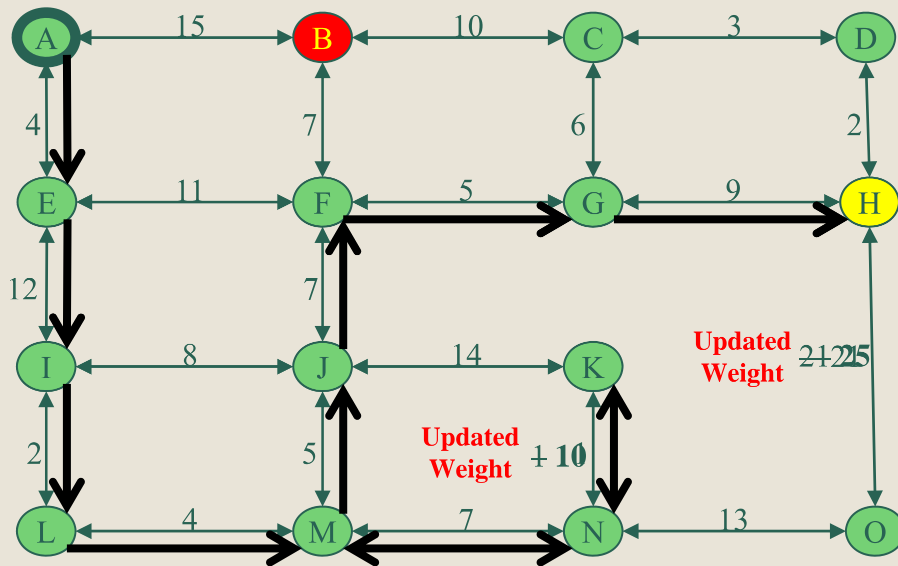


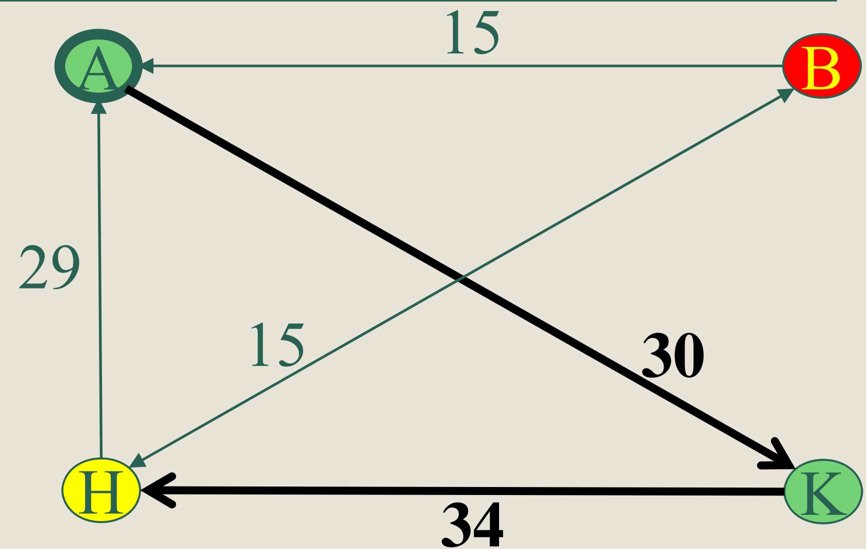
- Starting from K with intermediate node H
  - Add KH to the potential fastest route
  - Remove all edges that emanate from K or terminate at H
  - Choose next shortest path that does not make a simple cycle – HB
  - Remove all edges that emanate from H or terminate at B
  - Since there are no more intermediate nodes, order the paths – [AK] KH HB
  - Now add the path back to the starting node – [AK] KH HB BA
    - Total cost = 94
    - Actual path = [**A** E I L M N **K**] N M J F G **H** D C **B** **A**

Step 3 at t_2 – Choose Second Path

- Here are the two routes determined in Step 2 at time t_2
  - [**A** E I L M N **K**] N M J F **B** C D **H** G F E **A** with cost 101
  - [**A** E I L M N **K**] N M J F G **H** D C **B** **A** with cost 94
- Choose the second path in the route that has the minimum cost
- For this example, we would choose the second route, which means we will start going from K to H by K N M J F G H
  - Note that the cost from K to H (34) was higher than the cost from K to B (27), but we are looking at the overall cost of the route rather than just taking a greedy approach for the next node to visit

Step 1 at t_3 – Create Sub-Graph





		Source	
		B	H
Destination	A	15 BA	29 HGFEA
	B		15 HDCB

Start Node = H
Intermediate Nodes = {B}
End Node = A

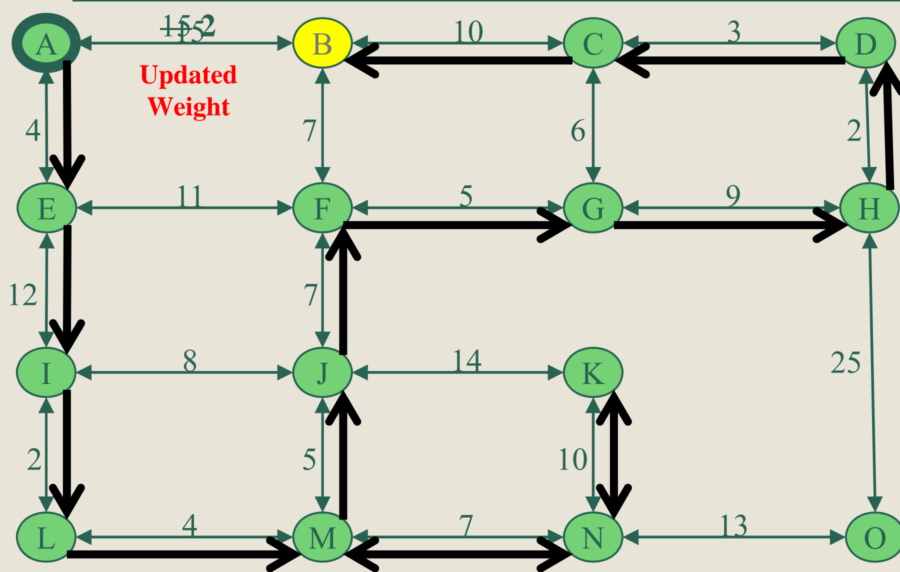
Step 2 at t_3 – Iterate over Intermediate Nodes

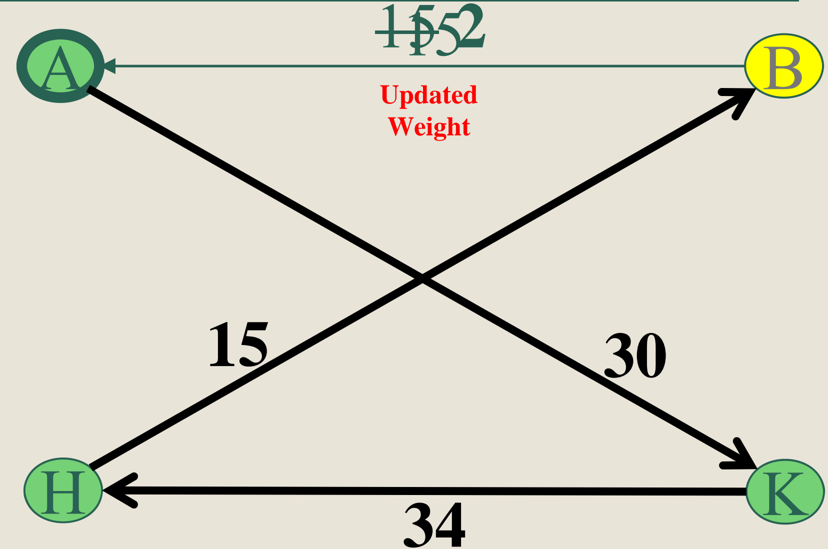
- Starting from H with intermediate node B
  - Add HB to the potential fastest route
  - Remove all edges that emanate from H or terminate at B
  - Since there are no more intermediate nodes, order the paths – [AK] [KH] HB
  - Now add the path back to the starting node – [AK] [KH] HB BA
    - Total cost = 94
    - Actual path = [**A** E I L M N **K** N M J F G **H**] D C **B** **A**

Step 3 at t_3 – Choose Third Path

- Here is the one route determined in Step 2 at time t_3
  - [**A** E I L M N **K** N M J F G **H**] D C **B** **A** with cost 94
- With only one route, we choose the next path in the route
- For this example, we would choose the third route, which means we will start going from H to B by H D C B

Step 1 at t_4 – Create Sub-Graph

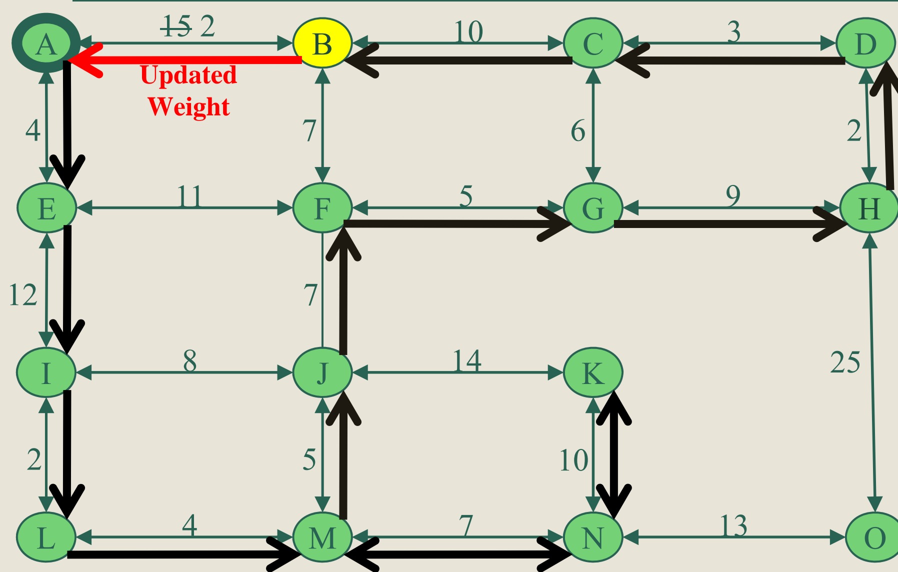


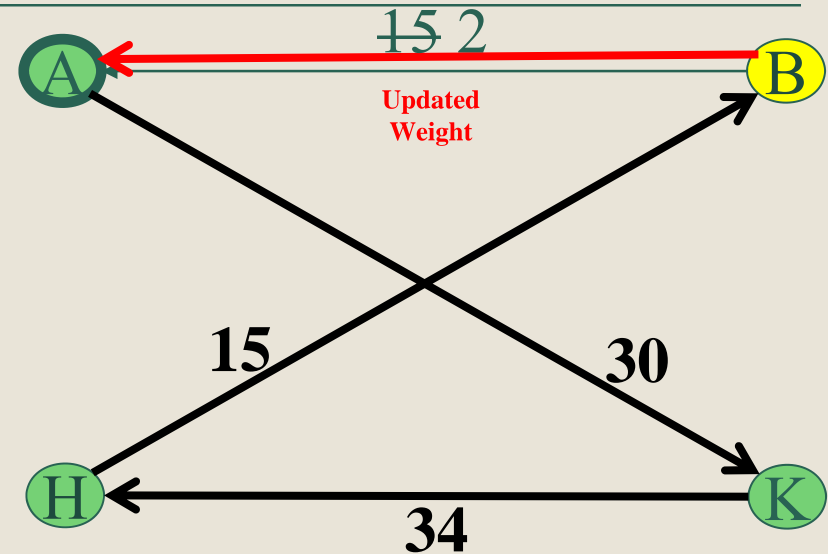


		Source
		B
Destination	A	15 BA

Start Node = B
Intermediate Nodes = {}
End Node = A

Step 2 at t_4 – Iterate over Intermediate Nodes



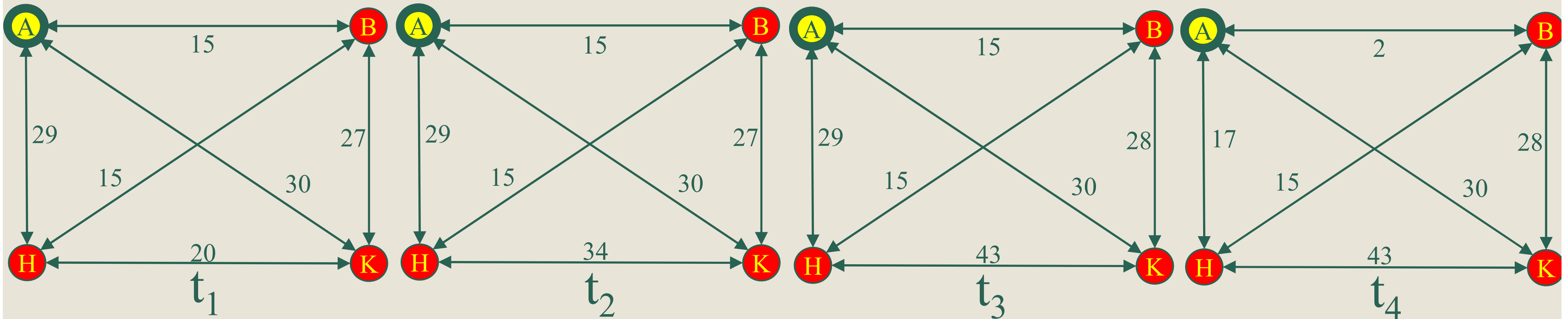


- Starting from B with no intermediate nodes
  - Since there are no more intermediate nodes, the only thing to do is add the path back to the starting node – [AK] [KH] [HB] BA
    - Total cost = 81
    - Actual path = [**A** E I L M N **K** N M J F G **H** D C **B**] **A**

Step 3 at t_4 – Choose Fourth Path

- Here is the one route determined in Step 2 at time t_4
  - [**A** E I L M N **K** N M J F G **H** D C **B** **A**] with cost 81
- With only one route, we choose the next path in the route
- For this example, we would choose the fourth route, which means we will go back to our starting node of A via the path directly from B to A, which completes the ITS-TSP algorithm

All Possible Routes



Time		Route Number					
		1	2	3	4	5	6
	t_1	15 AB AB	15 AB AB	29 AH AEFGH	29 AH AEFGH	30 AK AEILMNK	30 AK AEILMNK
	t_2	15 BH BCDH	27 BK BFJMNK	15 HB HDCB	34 HK HGFJMNK	27 KB KNMNJFB	34 KH KNMJFGH
	t_3	43 HK HDCBFJK	43 KH KJFBCDH	28 BK BFJK	28 KB KJFB	15 BH BCDH	15 HB HDCB
	t_4	29 KA KNMJFBA	17 HA HDCBA	29 KA KNMLIEA	2 BA BA	17 HA HDCBA	2 BA BA
	Total	102	102	101	93	89	81

Start Node = A
Intermediate Nodes = {B, H, K}
End Node = A

ITS-TSP Practical Optimizations

- Realizing that the subset of nodes to visit will typically be much less than the total number of nodes in the network will bring down the practical running time of the algorithm
- If we only consider “reasonable” paths between pairs of nodes, the running time will improve
- Thresholds on edge updates can be used since traffic conditions typically change gradually over time and rarely instantaneously
  - Further, if an edge that is not currently in the fastest path between a pair of nodes has the weight increase, there is no need to re-compute anything
  - Likewise, if an edge that is currently in the fastest path between a pair of nodes has the weight decrease, there is no need to re-compute anything

ITS-TSP Conclusion

- The Intelligent Transportation Systems Traveling Salesman Problem provides a HEURISTIC algorithm for the traditional TSP applied to ITS, which has three variations
  - Edge weights can change constantly
  - Only a subset of nodes in the graph need to be visited
  - Cycles can exist in the overall route since edge weights change
- Without prediction (or even with prediction), the ITS-TSP will always be a heuristic unless we know exactly how traffic will behave in the future, which is impossible
- With a preprocessing step of $O(V^2E!)$, the running time of the ITS-TSP algorithm is polynomial with respect to the number of nodes to visit, with $O(V^3)$ being the worst case if all V nodes need to be visited

Questions?