

Dynamic Fastest Paths with Multiple Unique Destinations (DynFast-MUD) – A Specialized Traveling Salesman Problem with Intermediate Cities

Jeffrey Miller

University of Alaska, Anchorage

IEEE 12th Intelligent Transportation Systems Conference

October 6, 2009

Outline

- Motivation
- Description of Algorithm
- Example of Algorithm
- Running Time and Correctness
- Future Work

Motivation

- Fastest path algorithms have been studied for quite some time
 - Single Source Shortest Path – Dijkstra, Bellman-Ford
 - All Pairs Shortest Path – Floyd-Warshall, Johnson
- But with transportation graphs, the weights on the edges can change quite frequently
 - Dynamic Fastest Path (DynFast) – Demestrescu and Italiano, Miller and Horowitz
- All of the previous algorithms were created for (and work more efficiently) for determining the fastest path between a single source and a single destination

Problem

- What about the case where a driver may want to visit multiple places in their trip before returning to their source?
- For example, what if I want to go to a museum, visit my friend, go to the store, then go back home?
 - What's the fastest way to travel to all of those locations before returning to home?
- 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

Potential Solutions

- The DynFast algorithms could be used, but we would have to consider all of the different permutations of the locations to visit

- For example, if I am starting at A and want to visit B, C, D, and E before returning to A, I would have to consider the following different routes:

ABCDEA	ACBDEA	ADBCEA	AEBCEA
ABCEDA	ACBEDA	ADBECA	AEBDCA
ABDCEA	ACDBEA	ADCBEA	AECBDA
ABDECA	ACDEBA	ADCEBA	AECDBA
ABECDA	ACEBDA	ADEBCA	AEDBCA
ABEDCA	ACEDBA	ADECBA	AEDCBA

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

Potential Solutions

- The Traveling Salesman Problem (TSP) is similar, but has a few important differences
 - The TSP assumes all of the nodes in a graph will be traversed with the lowest overall cost
 - The TSP algorithms do not account for constantly changing edge weights
- The Dynamic TSP allows additional nodes to be added in the middle of the TSP trip, but again visits every node in the graph with minimum cost
- The Probabilistic TSP [Jaillet] provides a solution for visiting a subset of nodes in a graph with the minimum cost
 - However, the solution is determined by using the solution to the TSP then removing nodes from the solution that were not in the subset of nodes to visit

The Solution

- After some initial analysis, it was determined that the optimal solution to determining the route to visit a subset of nodes (addressed by this work) did not necessarily coincide with the ordering of the cities in the TSP (Probabilistic TSP)
- Leveraging the DynFast algorithms, we came up with the Dynamic Fastest Path with Multiple Unique Destinations (DynFast-MUD) algorithm

DynFast-MUD Pseudocode

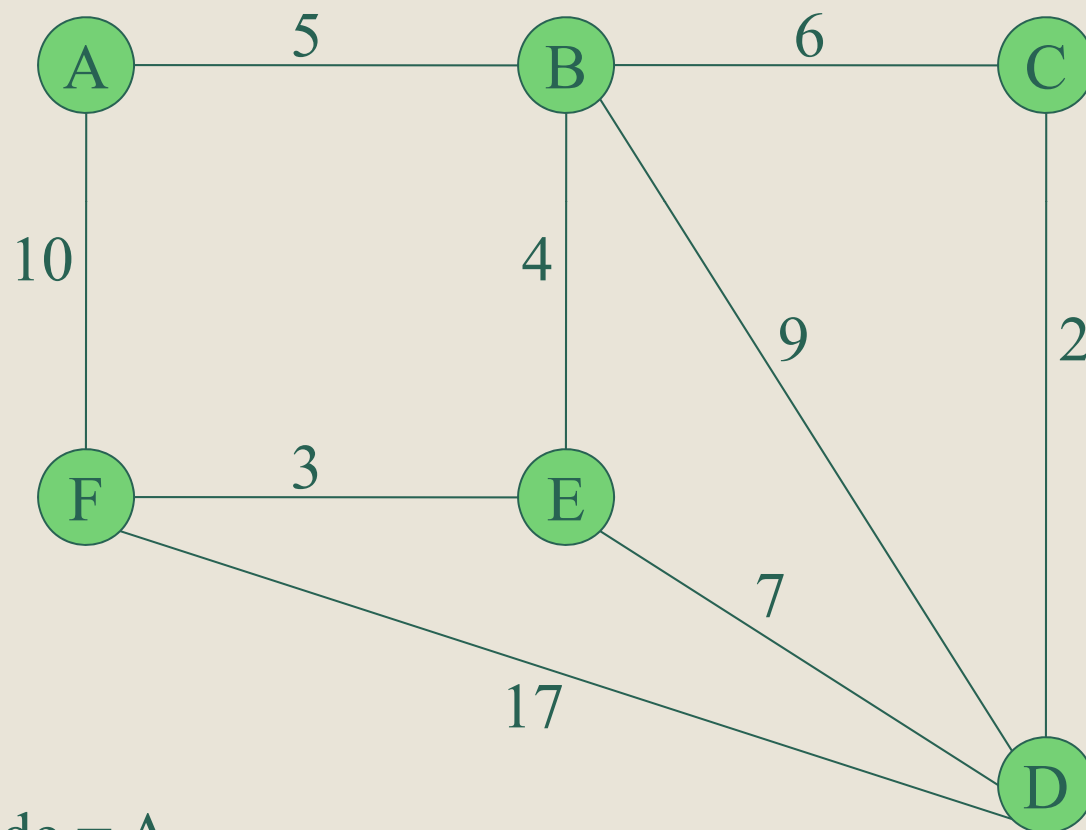
```
1 -- returns the ROUTE that starts at start_node and visits all of the nodes in
2 -- int_nodes, with the minimum overall cost
3 ROUTE DynFast-MUD(GRAPH g, NODE_SET int_nodes, NODE start_node) {
4   if (start_node ∈ int_nodes) {
5     int_nodes = int_nodes - {start_node}
6   }
7   -- get all of the paths connecting all of the NODEs in int_nodes
8   PATH_SET original_path_set = {}
9   for each (NODE src ∈ int_nodes U {start_node}) {
10    for each (NODE dest ∈ int_nodes) {
11      if (src != dest) {
12        -- using All_Pairs_All_Paths algorithm from [3]
13        original_path_set = original_path_set U All_Pairs_All_Paths(g, src, dest)
14      }
15    }
16  }
17 -- iterate through each NODE in int_nodes to find the minimum weight route from
18 -- the start_node with the first NODE visited being each NODE in int_nodes
19 ROUTE fin_route[int_nodes.length]
20 path_set = original_path_set
21 for each (NODE u ∈ 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     -- no simple cycles exist in the route
29     if (!route.contains_cycle_with_path(p)) {
30       route = route U p
31       path_set.remove_all_paths_with_src(p.src)
32       path_set.remove_all_paths_with_dest(p.dest)
33     }
34   }
```

```
35 -- order the paths where the dest from one path is the src of the next
36 -- path, starting with start_node
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 the start
44 -- node by calling fastest_path function from [3]
45 fin_route[u] = fin_route[u] U fastest_path(g, fin_route[u].dest, start_node)
46 path_set = original_path_set
47 }
48 -- find route with minimum cost from the fin_route array
49 ROUTE minimum_route = {}
50 for each (t ∈ 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 }
```

DynFast-MUD Algorithm

- Input: transportation graph, set of intermediate nodes, start node
 - Output: route with lowest overall cost that starts at the start node, visits all the intermediate nodes, then returns to the start node
1. Remove the start node from the intermediate nodes if necessary
 2. Get all of the paths connecting all of the nodes in the intermediate nodes (using All Pairs All Paths algorithm)
 3. Iterate through each node in the intermediate node set
 1. Determine the minimum weight path from the start node to the intermediate node and add it to our potential fastest route
 2. From the original set of paths found in step 2, remove all paths that have a source of the start node
 3. From the original set of paths found in step 2, remove all paths that have a destination of the intermediate node
 4. As long as the set of paths from step 2 above (modified in step 3.2) still contains more paths
 1. Determine the path with minimum overall cost that still remains
 2. As long as that path does not create a cycle, add it to the potential fastest route
 3. From the original set of paths found in step 2, remove all paths that have a source of the source node
 4. From the original set of paths found in step 2, remove all paths that have a destination of the destination node
 5. Order the paths in the potential fastest route from the start node where the destination of one path is the source of the next path
 6. Add the fastest path from the destination of the route back to the start node to the potential fastest route
 7. Store the potential fastest route in an array and reset the route to be empty for the next iteration
 4. Out of all the routes found in step 3, find the minimum weight route and return it

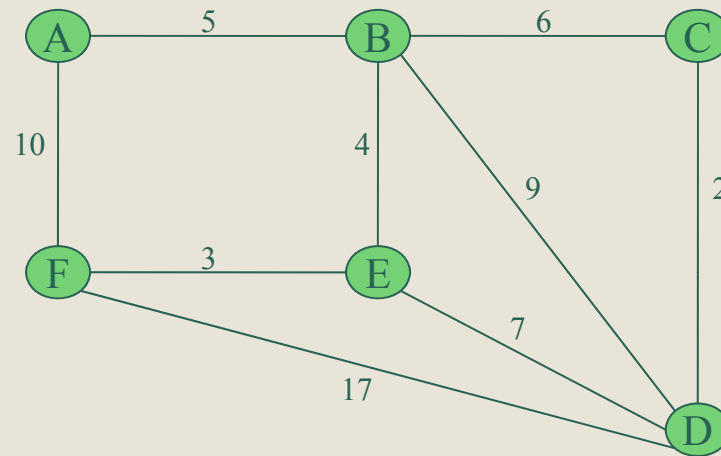
DynFast-MUD Example



Start Node = A

Intermediate Nodes = {B, D, E}

Step 2 – All Pairs All Paths



	B	D	E
A	5 AB	13 ABCD	9 ABE
B		8 BCD	4 BE
D	8 DCB		7 DE
E	4 EB	7 ED	

Step 3 – Iterate through Intermediate Nodes

		Destination		
		B	D	E
Source	A	5 AB	13 ABCD	9 ABE
	B		8 BCD	4 BE
	D	8 DCB		7 DE
	E	4 EB	7 ED	

Intermediate Node B

Route = {AB[5]}

Route = {AB[5], BE[4]}

Route = {AB[5], BE[4], ED[7]}

Order the paths = {AB[5], BE[4], ED[7]}

Add path back to start = {AB[5], BE[4], ED[7], DA[13]}

Potential Routes = {ABEDA[29]}

Step 3 – Iterate through Intermediate Nodes

		Destination		
		B	D	E
Source	A	5 AB	13 ABCD	9 ABE
	B		8 BCD	4 BE
	D	8 DCB		7 DE
	E	4 EB	7 ED	

Intermediate Node D

Route = {AD[13]}

Route = {AD[13], BE[4]}

Choose EB, but that would create a cycle, so remove and try again

Route = {AD[13], BE[4], DB[8]}

Order the paths = {AD[13], DB[8], BE[4]}

Add path back to start = {AD[13], DB[8], BE[4], EA[9]}

Potential Routes = {ABEDA[29], ADBEA[34]}

Step 3 – Iterate through Intermediate Nodes

		Destination		
		B	D	E
Source	A	5 AB	13 ABCD	9 ABE
	B		8 BCD	4 BE
	D	8 DCB		7 DE
	E	4 EB	7 ED	

Intermediate Node E

Route = {AE[9]}

Route = {AE[9], EB[4]}

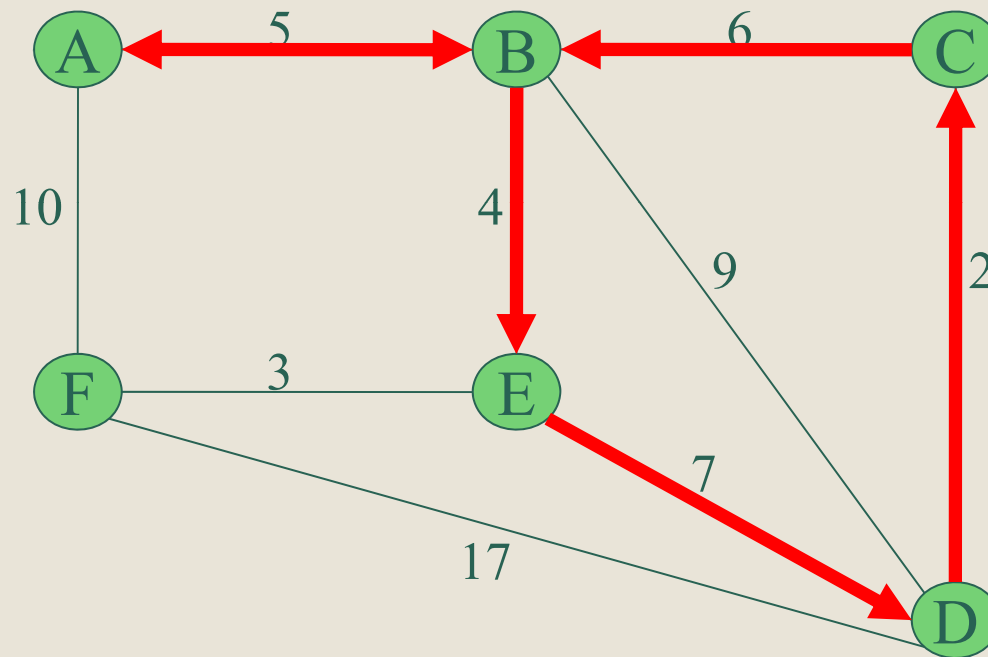
Route = {AE[9], EB[4], BD[8]}

Order the paths = {AE[9], EB[4], BD[8]}

Add path back to start = {AE[9], EB[4], BD[8], DA[13]}

Potential Routes = {ABEDA[29], ADBEA[34], AEBDA[34]}

Step 4 – Choose the Shortest Route



Potential Routes = {ABEDA[29], ADBEA[34], AEBDA[34]}
Fastest Route = ABEDA[29]

DynFast-MUD Running Time

- The preprocessing step for determining all of the paths between all of the pairs of nodes is $O(V^2E)$, where V is the total number of nodes and E is the total number of edges in the graph
 - This is not very significant since it only has to occur one time for any transportation graph
- The algorithm will take $O(n^3)$ time, where n is the number of intermediate nodes to be traversed in the graph

Conclusion

- The DynFast-MUD algorithm provides a way of determining the fastest route through a set of intermediate nodes and returning to the start node in a constantly-updating graph
- The algorithm was proven correct in the paper through the use of two different loop invariants
- The running time of the algorithm is $O(n^3)$, where n is the number of intermediate nodes
 - There is a pre-processing step that required $O(V^2E)$, where V is the number of nodes and E is the number of edges in the graph
 - The traditional TSP has a running time of $O(V!)$

Future Work

- To allow visiting an intermediate node more than once
- To allow ordering of some (but not all) of the intermediate nodes
- To verify the algorithm on a live transportation network
 - In Anchorage, we have tracking devices installed in 50 vehicles and would like to test our algorithm based on live data
 - Using taxi fleets or local shipping companies, we can also test our algorithm

Questions?

