Vehicle Routing in a Dynamic Mesh

Ying Ying Liu
Data and Analytics Practice Department
Manitoba Hydro
Winnipeg, Canada
Email: yingliu@hydro.mb.ca

Parimala Thulasiraman
Department of Computer Science
University of Manitoba
Winnipeg, Canada
Email: Parimala.Thulasiraman@umanitoba.ca

***Abstract*—We model the traffic-aware vehicle routing problem as an online problem and study online algorithms for this problem on a dynamic directed mesh. In this problem, traffic is represented as edge weights. At each time step, edge weights increase or decrease randomly. The goal of a vehicle is to find a path from the top-left corner source node S to the bottom-right corner destination node T, such that the sum of edge weights on the path is minimized. We first study lower bounds on the competitive ratios for any deterministic online algorithm for the problem, and competitive analysis with bounded adversarial traffic, randomization and advice. Following the competitive analysis, we propose nine online algorithms for the problem using deterministic, randomized and advice variants of the Greedy algorithm, Weighted Greedy algorithm, Dijkstra's algorithm, and Ant Colony Optimization (ACO). Our experiment shows that algorithms with advice, representing traffic in the future, find the best solutions among the algorithms in comparison. Our experiment also shows that although simple randomization technique does not help the greedy algorithms in our problem setting, the more sophisticated randomization strategy used by ACO is promising. Compared to ACO, the greedy algorithms are very fast with comparable solution quality. They may be favoured in practice for their speed and simplicity.**

***Index Terms*—Dynamic Mesh; Online Shortest Path; Online Greedy Algorithm; Online Dijkstra's, Online Ant Colony Optimization.**

I. Introduction

Vehicle routing refers to the task of finding the optimal travel path from place A to place B. In classical static routing algorithms, such as Dijkstra's algorithm [1] and A* algorithm [2], this problem is solved by finding the shortest path on a graph representing a road map with the weight of an edge representing the actual geometric distance between two junctions. The static routing algorithms are run once at the path planning stage and do not consider dynamic traffic information such as congestion, accidents and road closure. As vehicle traffic congestion becomes alarming severe in modern metropolitan areas, traffic-aware vehicle routing is one of the important problems in improving quality of life and building smart cities with higher productivity, less air pollution and less fuel consumption.

In this paper, we model the traffic-aware vehicle routing problem as an online problem on a dynamic directed mesh. The contributions of this paper are as follows:

1) We study lower bounds on the competitive ratios for any deterministic online algorithm for the problem, and competitive analysis with bounded adversarial traffic, randomization and advice.
2) We propose, implement, and compare nine online algorithms for the problem using deterministic, randomized and advice variants of the Greedy algorithm, Weighted Greedy algorithm, Dijkstra's algorithm, and Ant Colony Optimization (ACO).

The rest of the paper is organized as follows. Section II provides the formal problem definition and assumptions. Section III reviews related work in the different variants of problems for online vehicle routing. Section IV explains the three strategies for our online algorithm design. Section V provides in-depth theoretical analysis and online algorithm design for vehicle routing in a dynamic mesh. Section VI discusses the experimental results. Section VII concludes this paper and provides thoughts for future work.

II. Problem Statement

The setting of the problem is on an $n \times n$ mesh M with every edge directed from top to bottom and from left to right, with the following assumptions:

- The structure of M remains static.
- Traffic, represented as the set of edge weights W, changes in an online matter at each time step.
- The weight $w \in W$ on an edge e changes between the range of 1 and a constant $\mu > 1$, inclusively.
- The vehicle v knows the structure of M, μ and W at each time step.

The goal of the vehicle v is to find a path P from the top-left corner source node S to the bottom-right corner destination node T, such that the sum of edge weights of P, denoted by $\sum_{e_{ij} \in P} w_{ij}$, is minimized.

We study the directed mesh because it is a basic setting for the online routing problem. Each node of the mesh has at least one and at most two outgoing edges and each edge is on a path from the source node to the destination node, and all the paths from the top-left corner source node S to the bottom-right corner destination node T have exactly $2n$ edges.

III. Literature Review

There are different variants of problems for online vehicle routing. Dynamic path discovery [3] is a well-studied online problem. In this problem, new vertices appear in an online matter, and the goal is to find the shortest path on the graph. In dynamic multiple vehicle routing [4], the requests for service appear in an online matter, and the goal is to dispatch k vehicles to serve the online requests such that the total distance the vehicles travel is minimized. This is similar to the famous k-server problem [5]. In our problem setting, the weights on the edges change in an online matter. This problem is also called the Time-Dependent Shortest Path (TDSP) problem in literature.

Cook and Halsey [6] first studied the TDSP problem and solved it using Dynamic Programming. Dreyfus [7] pointed out that TDSP can be solved by a generalization of Dijkstra's method as efficiently as for static shortest path. Halpern [8] proved that [7] is only true for First-In-First-Out (FIFO) networks. If the FIFO property does not hold in a time-dependent network, then the problem is NP-hard. Dean [9] solved TDSP based on the Bellman-Ford algorithm. Batz et al. [10] solved TDSP using Contraction Hierarchies. Takimoto and Warmuth [11] used a machine learning approach to represent probabilistic weights. Gyrgy et al. [12] proposed a machine learning approach using randomization and advice.

To our knowledge, there are few competitive analyses for the TDSP problem in the literature. In this paper, we begin with a competitive analysis of our online problem setting, followed by algorithm design based on the competitive analysis.

IV. Strategies

A. Competitive Analysis

Competitive analysis [13] is a framework to compare online algorithms. Given a sequence σ of W, let OPT denote the best possible offline solution to the vehicle routing problem. Competitive ratio of an online algorithm A is the maximum ratio between the cost of A and that of OPT over all sequences.

$$cr(A) = max_{\sigma} \frac{cost_A(\sigma)}{cost_{OPT}(\sigma)}) \tag{1}$$

In competitive analysis, we consider the worst-case inputs generated by an *adversary*, which tries its best to make the algorithm inefficient.

B. Competitive Analysis with Randomization

To strive for better competitive ratios, *randomization* [14] is a common strategy for online algorithms. For randomized algorithms, we compare online algorithms against an *oblivious adversary* which knows the code of the algorithm but does not know the run-time random bits used by the algorithm. Randomization helps an online algorithm achieve better competitive ratio when there are more than two ways for the algorithm, and adversary does not know whether a better way is chosen at run time.

C. Competitive Analysis with Advice

Another way for an online algorithm A to achieve a better competitive ratio is through receiving some bits of *advice* from a benevolent oracle [15] . Given sufficient bits, the advice can encode the entire OPT for A. The other end on the advice spectrum is when 1 bit of advice is given. In general, we study the advice strategies with varied sizes and how they can help the algorithm with its competitive ratio.

V. Vehicle Routing in a Dynamic Mesh

A. Lower Bounds

In this section, we derive lower bounds on the competitive ratios of any online algorithms for vehicle routing in a dynamic mesh.

1) Competitive Analysis for Any Deterministic Online Algorithm: Consider a 2×2 mesh in Figure 1, an online algorithm A wants to find a path from node A to D. At time 0, the adversary generates traffic W_0 such that all the edges have weight of 1. Since the two paths A-B-D and A-C-D both have cost of 2, A chooses one of them randomly. In this example, A chooses A-C-D. Once the vehicle v arrives at C at time 1, the adversary generates traffic W_1 such that e_{CD} becomes μ and all other edge weights remain unchanged. Because now C-D is the only possible sub-route for the vehicle to reach D, the vehicle arrives at D at time $\mu + 1$, whereas OPT arrives at time 2. The competitive ratio is $\frac{\mu+1}{2}$.

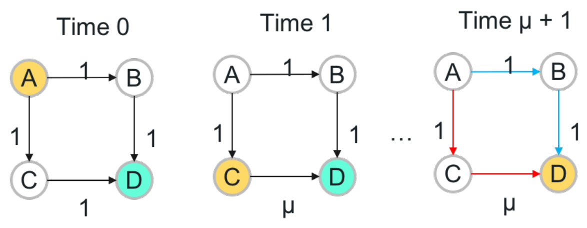

Fig. 1: Adversary Input to a 2 x 2 Mesh

Consider a $n \times n$ mesh in Figure 2, the adversary generates inputs similar to the previous example. At time 0, the adversary generates W_0 with weight 1 on each edge (to make sure OPT also has cost 1 at this time step). At each time step after time 0, the adversary places weight μ on the upcoming two edges of node that the vehicle arrives at, so that when the vehicle arrives at T, $cost_A$ becomes $1 + (2n-1)\mu$. The competitive ratio is $\frac{1+(2n-1)\mu}{2n} \approx \mu$

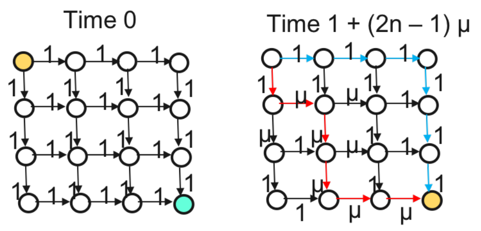

Fig. 2: Adversary Input to a n x n Mesh

2) Competitive Analysis with Bounded Adversarial Traffic: In this section, we analyze the strategy of adversary if the total amount of adversarial traffic is bounded by a factor c of n, that is, $W \leq c * n$. By observation, all the nodes in the mesh have 2 outgoing edges except for nodes at the bottom and the right, which only have 1 outgoing edge. An intuitive strategy for the adversary is therefore to use some of its traffic quota to trick algorithm A to direct the vehicle to these two *critical paths*. Once the vehicle is on a critical path, the adversary can place all of its rest traffic quota on the path. Consider the example in Figure 3 when $c = 2\mu - 1$. The adversary places μ traffic on each right edge of the left-most nodes, tricking algorithm A to choose the bottom edge with less traffic, until the vehicle reaches the bottom critical path. When $c = 2\mu - 1$, competitive ratio is $\frac{1+n\mu}{2n} \approx \mu/2$. The competitive ratio is smaller than $\mu/2$ if $c < 2\mu - 1$ and greater than $\mu/2$ if $c > 2\mu - 1$.

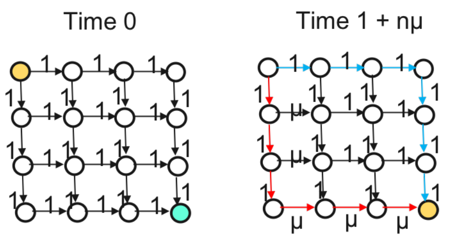

Fig. 3: Adversary Input to a n x n Mesh With $(2\mu - 1)n$ Traffic

3) Competitive Analysis with Randomization: On an $n \times n$ mesh, there are exactly $2n$ edges and $2n + 1$ nodes on any path from S to T. There are at least $n + 1$ nodes (see Figure 3) and at most $2n - 1$ nodes (see Figure 4) that have 2 edges on a path. For each such node, the decision to go right or go down is a binary decision problem. A correct guess has a cost of 1 in the next node, and a wrong guess has cost of μ in the next node. Using randomization in the algorithm, the adversary cannot know which guess the algorithm makes until run time, therefore, the algorithm has a competitive ratio of at most $\frac{\mu\frac{2n-1}{2} + \frac{2n-1}{2} + \mu}{2n} \approx \mu/2$.

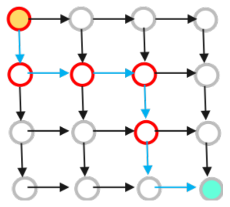

Fig. 4: Nodes With 2 Outgoing Edges in a Path on an n x n Mesh

4) Competitive Analysis with Advice: Following the previous section, since $\Theta(n)$ guesses are required, $logn$ bits of advice is required to find the optimal path. With 1 bit of advice, half of the guesses is correct, and half is wrong, and the algorithm has a competitive ratio of $\mu/2$.

B. Online Algorithms

In this section, we propose nine online algorithms for the vehicle routing problem in a dynamic mesh using determinism, randomization and advice.

```
1: Set current node to S
2: while Destination $T$ not reached do
3:   if Current node has 1 outgoing edge then
4:     Add the edge to path
5:   else
6:     Choose the edge with less weight at current time t
7:     Set the next node to current node when arriving at it
       at time t'
8:   end if
9: end while
10: Output the path
```

Fig. 5: Greedy Algorithm for Vehicle Routing in a Dynamic Mesh

```
1: Set current node to S
2: while Destination $T$ not reached do
3:   if Current node has 1 outgoing edge then
4:     Add the edge to path
5:   else
6:     For edge $i = 0, 1$, select $i$ with $prob_i = 1 - \frac{w_i(t)}{w_0(t)+w_1(t)}$ at current time t, where edge 0 denote
       the bottom edge and edge 1 denote the right edge
7:     Set the next node to current node when arriving at it
       at time t'
8:   end if
9: end while
10: Output the path
```

Fig. 6: Greedy Algorithm With Randomization for Vehicle Routing in a Dynamic Mesh

1) Greedy Algorithm: The most intuitive approach to solve the problem is the Greedy algorithm. The algorithm is extremely simple. At each time step, if a vehicle arrives at a node, it decides next direction with the least traffic at the current time. The pseudo-code of the Greedy Algorithm is shown in Figure 5.

2) Greedy Algorithm With Randomization: The randomization of the Greedy algorithm is straightforward. When a binary decision is needed, the edge with less cost is selected with higher probability. The pseudo code is shown in Figure 6.

3) Greedy Algorithm With Advice: In the previous two algorithms, a binary decision is made based on assessment of edge weights at current time. However, as a vehicle is set off, the traffic may increase or decrease while the vehicle is still on the way. Therefore, the actual arrival time at the next node may be different from the estimated arrival time. In a dynamic mesh, it is common that as the vehicle sets off on one edge, the traffic on the other edge becomes less than the current edge. With advice of log μ that encodes the actual arrival time at the next two nodes, the algorithm is expected to make a better decision. The pseudo code is shown in Figure 7.

4) Weighted Greedy Algorithm: Recall the adversarial strategy in the competitive analysis with bounded adversarial

```
 1: Set current node to S
 2: while Destination $T$ not reached do
 3:     if Current node has 1 outgoing edge then
 4:         Add the edge to path
 5:     else
 6:         for Each edge $i$ do
 7:             $FutureDistance = 0$
 8:             $j = 0$ //incremental time step to look up traffic in the future
 9:             while $FutureDistance < 1$ //default distance of an edge do
10:                 $FutureDistance+ = 1 * \frac{1}{w_i(t+j)}$ //distance of 1 time step with the current speed
11:                 $j++$
12:             end while
13:             $PredictArrivalTime_i = j$
14:         end for
15:         Choose $i$ with less $PredictArrivalTime_i$
16:         Set the next node to current node when arriving at it at time t'
17:     end if
18: end while
19: Output the path
```

Fig. 7: Greedy Algorithm With Advice for Vehicle Routing in a Dynamic Mesh

traffic, in order to avoid being tricked to a critical path, a natural extension to the Greedy algorithm is the Weighted Greedy algorithm where the selection favors nodes in the middle. Specifically, the greedy weight on an edge $i = 0, 1$ becomes $\frac{w_i}{remainingWeightOnTheAxis}$, where edge 0 denote the bottom edge and edge 1 denote the right edge. For example, in time 1 of Figure 2, the vehicle is at the node below the starting node S, and except for the current two edges with weight μ, all other edges have weight 1. The Greedy algorithm selects either the bottom edge or the right edge. But the Weighted Greedy Algorithm select the right edge (edge 1), because $\frac{w_1=\mu}{remainingWeightOnXAxis=n-1} < \frac{w_0=\mu}{remainingWeightOnYAxis=n-2}$. The pseudo-code of Weighted Greedy Algorithm is similar to the Greedy Algorithm except for the weight function.

5) Weighted Greedy Algorithm With Randomization: The pseudo-code of Weighted Greedy Algorithm with randomization is similar to Figure 6 except for the weight function.

6) Weighted Greedy Algorithm With Advice: The pseudo-code of Weighted Greedy Algorithm with advice is similar to Figure 7 except for the weight function.

7) Dijkstra's Algorithm: Dijkstra's algorithm [1] is a classic algorithm for shortest-path problem from a single source to all other nodes in a weighted, directed graph G with nonnegative edge weights [16]. Although it is designed to solve the single-source problem, the output of Dijkstra's includes the solution for single-source single-destination shortest path. Moreover, both problems have the same worst-case asymptotic running time [16]. Dijkstra's starts by initializing an attribute d on all nodes as ∞ except for S with $d = 0$. The Dijkstra's

```
 1: currNode = S
 2: while Destination $T$ not reached do
 3:     At current time t, initialize currNode.d to 0, and all the u.d to $\infty$ for each node u in the area A that is to the right of and below currNode.
 4:     Push currNode and every $u \in A$ to queue Q
 5:     while Q is not empty do
 6:         Pop node u with minimum d from Q
 7:         for Each processor node v do
 8:             if $v.d > u.d + w_{uv}(t)$ then
 9:                 $v.d = u.d + w_{uv}(t)$
10:                 $v.prev = u$
11:             end if
12:         end for
13:     end while
14:     Move currNode to the next node S' identified on the final path from S to T
15:     When vehicle arrives at S' at t', currNode = S'
16: end while
17: Output the path
```

Fig. 8: Dijkstra's Algorithm for Vehicle Routing in a Dynamic Mesh

keeps a priority queue of nodes not being visited, where a node u with the minimum d is visited first. Starting from S, it repeatedly updates d of a node v with the minimum value of $u.d + w_{uv}$ where v is a predecessor node of u. Different from static routing, in the dynamic mesh setting, Dijkstra's algorithm is run each time when the vehicle arrives at a new node, which becomes the new source node. The pseudo-code of Dijkstra's is shown in Figure 8.

8) Dijkstra's Algorithm With Advice: Similar to Figure 7, Dijkstra's Algorithm With Advice uses the predicted actual arrival time from u to v in the future assuming that the time table for future traffic is available for querying. The formula for calculating the predicted actual arrival time is given in line 6 to 14 in Figure 7.

9) Ant Colony Optimization: In addition to improvements in traditional algorithms, stochastic algorithms mimicking the routing of social animals in the dynamic nature have attracted much attention due to their proven efficiency and similarity to the dynamic vehicle routing problem. One popular algorithm is Ant Colony Optimization (ACO) [17], an iterative and evolving optimization heuristic. In nature, ants explore routes from nest to food source and deposit a chemical substance called pheromone, which attracts other ants to follow the same route. Pheromone, if not applied, evaporate over time. Eventually the longer paths lose pheromone concentration and all ants travel on the shortest path. Based on this observation, Dorigo et al. propose the ACO algorithm [17] and the Travelling Salesman Problem (TSP) is the first optimization problem solved by ACO.

When solving the TSP problem, the algorithm considers a TSP with N cities, and scatters m virtual ants randomly on these cities. The algorithm comprises three phases: compu-

```
1:  At time 0: run Dijkstra's algorithm once and get the cost of the shortest path initMinCost. Initialize pheromone $\tau_0 = \frac{1}{initMinCost}$ on all the edges of mesh $M$.
2:  At time 0: choose $numAnts = 2^{n/10}$. Convert id of each ant to bit value, and assign a partial path of length $n/10$ based on the bit values. If the bit is 0, the bottom edge is chosen. If the bit is 1, the right edge is chosen.
3:  for Each ant do
4:      Follow the assigned partial path $p'$
5:      while Destination $T$ not reached do
6:          At current time t, calculate a heuristic value on each outing edge $i$ with the formula $val_i(t) = \frac{pheromone_i}{w_i(t)}$ for $i = 0, 1$
7:          Generate a random number $rand$. If $rand <= 0.5$, choose the edge with greater $val$, otherwise choose edge $i$ with probability $prob_i = \frac{val_i(t)}{val_0(t)+val_1(t)}$ for $i = 0, 1$
8:          When ant reaches next node at time t', update pheromone on the chosen edge $i$ with the formula $pheromone_i = (1-\rho) * pheromone_i + \frac{\rho}{t'-t}$, where $\rho$ is the decay parameter and is set to 0.1 in the experiment.
9:      end while
10: end for
11: Output the best path of all the ants
```

Fig. 9: Online Ant Colony Optimization for Vehicle Routing in a Dynamic Mesh

tation, communication and update. In the computation phase, ACO constructs a tour of minimum length. The ants indirectly communicate with one another through stigmergy by depositing a pheromone concentration on the trail for all other ants to follow. Finally, the ants update the tour by increasing or decreasing (evaporating) the pheromone concentration on trails that were unused or produced a longer tour length. The ants work concurrently and cooperate to find an efficient tour.

For dynamic routing, Zhe et al. [18] develop a variant of ACO algorithm that uses stench pheromone to redirect ants to the second best route if the best route becomes too crowded. The authors incorporate traffic to the cost of each road segment as the total travel time on the segment. José Capela et al. [19] propose a hybrid algorithm of Dijkstra's algorithm and inverted ACO for traffic routing.

In this paper, we modify the basic ACO algorithm [17] for the mesh setting. In the online setting, the iterative process is removed from the algorithm because the ants cannot go back in time. This means that solution quality achieved by global optimization and reinforcement learning is a trade-off in the online setting. The pseudo-code for the ACO algorithm is shown in Figure 9. Since ACO is already a stochastic algorithm, we do not consider the advice variant for it for fair comparison.

VI. Result

A. Implementation Details

The algorithms are implemented in python and the code is available at github.com/yingyingliuCA/ShortestPath_OnMesh. The experiment is conducted on a MacBook Pro with 2.3 GHz Intel Core i5 processor and 8 GB 2133 MHz LPDDR3. The sizes of input $n \times n$ mesh varies from n = 10 to n = 150. For each input mesh, a timetable with random traffic between 1 and $\mu = 5$ on each edge is created, and all the algorithms are run against this timetable. The experiment results are measured as the average results of 3 runs. The algorithms are mainly measured by solution quality. Execution time is also shown for additional analysis.

B. Analysis

Figure 10 shows the costs of different online algorithms on different problem sizes. Unsurprisingly, algorithms with advice outperform other algorithms in general. For 15 input meshes, Greedy with Advice finds paths with the minimum cost among all compared algorithms for eight instances, Weighted Greedy with Advice achieves six times, and Dijkstra with Advice only one time. There is no result for Dijkstra algorithms for n greater than 80 because they become too slow to run. The result shows that Dijkstra's does not have an advantage over Greedy algorithms in our problem setting. This observation can be explained by the mesh setting. In the mesh, every edge is on a path to the destination, therefore, it is not necessary for a Greedy algorithm to traverse the path to the destination to make sure it does not walk into a dead end, as it would on other graph types. In addition, as the traffic changes randomly at each time step, the traversal to the destination in Dijkstra becomes redundant both in terms of solution quality and in terms of execution time, as we will see in the later section.

The two Greedy deterministic algorithms, Greedy and Weighted Greedy, produce good solutions in general. When the problem space becomes larger, Weighted Greedy seems to have a slight advantage over Greedy, probably due to its strategy to stay in the middle and therefore it has more room for exploration.

Among the three algorithms that use randomization, Greedy and Weighted Greedy with randomization do not seem to help the deterministic algorithms. For Greedy, there is no significant difference between the deterministic and randomized versions. For Weighted Greedy, randomization seems to be even worse than the deterministic version. This is probably because randomization offsets the stay-in-middle principle of Weighted Greedy. As randomization is a strategy to improve worst-case caused by an adversarial input, it may not help the algorithm when the input is already random in our problem setting.

On the other hand, ACO is among the best algorithms, even though the global optimization part of the algorithm is removed for the online setting. As shown in Figure 11, when n is greater than 100, ACO continuously finds the minimum cost among the deterministic and randomized algorithms. This is probably due to the unique combination of exploration in the large space and exploitation using collaborative pheromone update strategy.

The execution time of the algorithms is presented in Table I. Algorithms in the Greedy family are the fastest and have robust performance when the problem size becomes larger. Dijkstra's algorithms are slowest as expected because they

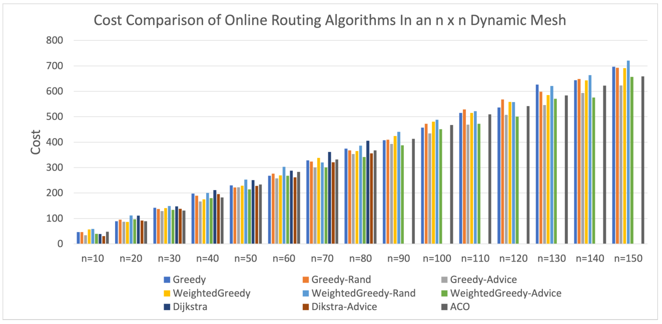

Fig. 10: Cost Comparison of Online Algorithms

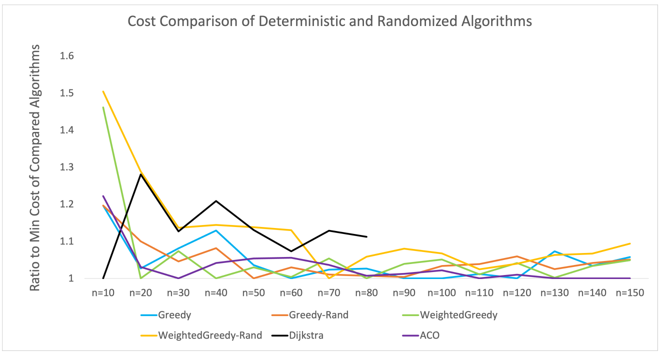

Fig. 11: Cost Comparison of Deterministic and Online Algorithms

need to traverse to the destination in order to update the path each time when a node on the previous path is reached. ACO is the second slowest because of the additional computation by each ant at each node. However, as ACO is inherently parallel, its performance can be improved using parallel computing. On the other hand, Greedy and Weighted Greedy are over 1000X times faster than ACO on average, with comparable solution quality.

VII. Conclusion

In this paper, we study the competitive ratios of the vehicle routing problem in a dynamic mesh and propose online algorithms using determinism, randomization and advice. The deterministic algorithms include Greedy, where the edge with less cost is selected at run time, Weighted Greedy, where the selection greedy favours nodes in the middle, and Dijkstra's algorithm, the classic static algorithm for finding the shortest path on graphs. The randomization algorithms include Greedy-Rand and Weighted Greedy-Rand, where an edge with less cost is selected with a higher probability, and ACO, a nature-inspired algorithm that combines exploitation and exploration. Third, Greedy-Advice, Weighted Greedy-Advice and Dijkstra-Advice are also implemented where the advice is actual travel time in the future on the outgoing edges of the current node.

Our experiment shows that algorithms with advice find the best solutions among the algorithms in comparison. Although such advice is unrealistic in real-life problems, this result shows the justification for high-quality prediction machine learning models for real-life problems. Our experiment also shows that although a simple randomization technique does not help the greedy algorithms, the more sophisticated randomization strategy used by ACO is promising. The drawback of ACO is however speed. Although it can be parallelized, a considerable amount of effort is needed. On the other hand, the greedy algorithms are very fast with comparable solution quality. They may be favoured in practice for their speed and simplicity.

The vehicle routing problem becomes very different than the mesh setting on other graph types. In particular, Greedy algorithms may lose their solution correctness when path traversal is required. In future work, a similar study will be extended to other graph types and real road networks with real traffic.

TABLE I: Execution Time of Online Algorithms

Avg Time (Sec)	*Greedy*	*Greedy Rand*	*Greedy Advice*	*Weighted Greedy*	*Weighted Greedy Rand*	*Weighted Greedy Advice*	*Dijkstra*	*Dijkstra Advice*	*ACO*
n=10	0.000	0.001	0.001	0.001	0.001	0.002	0.058	0.065	0.012
n=20	0.001	0.004	0.001	0.002	0.002	0.008	1.385	1.388	0.162
n=30	0.001	0.007	0.002	0.003	0.002	0.016	9.826	9.556	0.799
n=40	0.002	0.012	0.003	0.004	0.003	0.028	41.300	40.126	2.565
n=50	0.002	0.018	0.004	0.005	0.004	0.047	128.936	126.485	6.769
n=60	0.003	0.025	0.004	0.008	0.005	0.064	325.729	341.760	13.887
n=70	0.003	0.035	0.005	0.010	0.006	0.085	773.689	740.604	29.090
n=80	0.004	0.044	0.006	0.012	0.008	0.111	1591.932	1506.106	49.037
n=90	0.004	0.053	0.006	0.013	0.009	0.135	-	-	80.726
n=100	0.005	0.070	0.007	0.016	0.010	0.175	-	-	131.163
n=110	0.005	0.080	0.008	0.022	0.012	0.207	-	-	203.820
n=120	0.005	0.097	0.009	0.021	0.013	0.245	-	-	306.258
n=130	0.006	0.107	0.010	0.024	0.014	0.294	-	-	444.651
n=140	0.006	0.125	0.010	0.027	0.016	0.329	-	-	637.004
n=150	0.010	0.149	0.011	0.031	0.023	0.386	-	-	949.803

References

[1] E. W. Dijkstra, "A Note on Two Problems in Connetion with Graphs," Numerische mathematik, vol. 1, no. 1, 1959, pp. 269–271.

[2] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," IEEE transactions on Systems Science and Cybernetics, vol. 4, no. 2, 1968, pp. 100–107.

[3] V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia, "A Review of Dynamic Vehicle Routing Problems," European Journal of Operational Research, vol. 225, no. 1, 2013, pp. 1–11.

[4] P. Jaillet and M. R. Wagner, "Generalized Online Routing: New Competitive Ratios, Resource Augmentation, and Asymptotic Analyses," Operations research, vol. 56, no. 3, 2008, pp. 745–757.

[5] E. Koutsoupias, "The k-server Problem," Computer Science Review, vol. 3, no. 2, 2009, pp. 105–118.

[6] K. L. Cooke and E. Halsey, "The Shortest Route through a Network with Time-dependent Internodal Transit Times," Journal of mathematical analysis and applications, vol. 14, no. 3, 1966, pp. 493–498.

[7] S. E. Dreyfus, "An Appraisal of Some Shortest-path Algorithms," Operations research, vol. 17, no. 3, 1969, pp. 395–412.

[8] J. Halpern, "Shortest Route with Time Dependent Length of Edges and Limited Delay Possibilities in Nodes," Zeitschrift fuer operations research, vol. 21, no. 3, 1977, pp. 117–124.

[9] B. C. Dean, "Algorithms for Minimum-cost Paths in Time-dependent Networks with Waiting Policies," Networks: An International Journal, vol. 44, no. 1, 2004, pp. 41–46.

[10] G. V. Batz, D. Delling, P. Sanders, and C. Vetter, "Time-dependent Contraction Hierarchies," in Proceedings of the Meeting on Algorithm Engineering & Expermiments. Society for Industrial and Applied Mathematics, 2009, pp. 97–105.

[11] E. Takimoto and M. K. Warmuth, "Path Kernels and Multiplicative Updates," Journal of Machine Learning Research, vol. 4, no. Oct, 2003, pp. 773–818.

[12] A. György, T. Linder, G. Lugosi, and G. Ottucsák, "The On-line Shortest Path Problem under Partial Monitoring," Journal of Machine Learning Research, vol. 8, no. Oct, 2007, pp. 2369–2403.

[13] A. Borodin and R. El-Yaniv, Online Computation and Competitive Analysis. cambridge university press, 2005.
[14] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson, “On the Power of Randomization in On-line Algorithms,” Algorithmica, vol. 11, 1994, pp. 2–14.
[15] Y. Emek, P. Fraigniaud, A. Korman, and A. Rosén, “Online Computation with Advice,” Theoretical Computer Science, vol. 412, no. 24, 2011, pp. 2642–2656.
[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms. MIT press, 2009.
[17] M. Dorigo and G. Di Caro, “Ant Colony Optimization: a New Meta-heuristic,” in Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406), vol. 2. IEEE, 1999, pp. 1470–1477.
[18] Z. Cong, B. De Schutter, and R. Babuška, “Ant Colony Routing Algorithm for Freeway Networks,” Transportation Research Part C: Emerging Technologies, vol. 37, 2013, pp. 1–19.
[19] J. C. Dias, P. Machado, D. C. Silva, and P. H. Abreu, “An Inverted Ant Colony Optimization Approach to Traffic,” Engineering Applications of Artificial Intelligence, vol. 36, 2014, pp. 122–133.