1. INTRODUCTION

Memoryless online routing (MOR) algorithms are designed for the scenario where each node only stores in its memory the positions of its one-hop neighbors. This special group of algorithms has just local information available to find out its paths. Because of their simplicity and sophistication, memoryless algorithms are very popular in communication networking, urban planning, geographic information systems, geographic robotics, and sensor networks.

In our project we have chosen four popular MOR algorithms that work for Delaunay triangulation (DT). We intended to find the most efficient algorithm among Dijkstra’s, Two-step Routing, Apex-angle Routing and Midpoint Routing based on the length of the path and node cost from starting point to destination. We also calculated average path cost. Our sample consisted of 100 vertices and grid size was 600x600.

2. RELATED WORK

In recent decades memoryless algorithms have received extensive attention and many literature papers have dealt with related geometric difficulty.

Kalyanasundaram and Pruhs [8] present 16-competitive algorithm that explore any unknown plane graph and visit all nodes of the graph, not just destination t.

Past research showed that algorithms can run for Delaunay triangulations. Kranakis et al. [9] presented an algorithm that works for any Delaunay triangulation, and another one that works for any connected plane graph and has O(1) memory. Authors stated that the Compass Routing algorithm works for delaunay triangulation. If we denote v as a current forwarding node, t destination node, and \( w \in N(v) \) (\( N(v) \) set of v’s one-hop neighbours), then algorithm tries to minimizes the \( \text{angle} \ tvw \).

In previous work Cucka et al. [10] observed that the greedy routing algorithm [4] runs better than the compass routing algorithm [4, 8] on random graphs, but in the same time does not do very good on Delaunay triangulations of random point sets.

3. ONLINE ROUTING ALGORITHMS

We selected Dijkstra’s, Two-step Routing, Apex-angle Routing and Midpoint Routing algorithms for our project and this work can be extended in future by adding more online routing algorithms. In this section, we describe about each algorithm in detail.

3.1 The Dijkstra’s Algorithm:

We include Dijkstra’s classic graph-search algorithm to find the shortest routing path for comparison, specifically a single-destination implementation.

1. Mark all except the starting node as unexplored, with an infinite distance. Set the starting node as the current node.
2. Visit all neighbors of the current node and assign a tentative path distance.
3. If the tentative distance for a node is less than its currently assigned value, update the distance value for that node.
4. Once all neighbors are visited, mark the current node as explored and and choose a new unexplored node.
5. Repeat until there are no reachable unexplored nodes.

Traditionally Dijkstra’s includes two performance enhancements not currently implemented in our algorithm: a priority queue to intelligently choose new nodes to visit and early termination once the algorithm has explored enough nodes to be assured of the shortest path to its destination.

To calculate average cost of nodes in each algorithm, we performed 20 tryouts. It turned out that the mean of node cost for Dijkstra’s algorithm is 10.4 as shown on the picture below.

![Graph showing Dijkstra's routing cost comparison](image-url)
3.2 The Two-Step Routing Algorithm:
The idea is to minimize the sum of \( d(v,w) \) and \( d(w,t) \),
where \( w \) is a neighbor of \( v \). Unfortunately there are
cases where the algorithm fails if we do not add
extra condition. And because of that by putting
extra line:
\[
\text{if} (\text{$distance < $start\_distance})
\]
we are fixing the problem. The core of the Two-Step
Algorithm is presented below:

\[
\begin{align*}
\text{$min\_distance} = \text{INF}; \\
\text{$min\_vertex} = \text{null};
\end{align*}
\]

\[
\text{foreach ($neighbors as $neighbor) { if ($neighbor->isEqual($vertex\_finish))}
\]
\[
\text{ $graph->addPath\_Edge( new \\
\text{Edge($vertex\_start, $neighbor)); return; ) else {}
\end{align*}
\]

\[
\begin{align*}
\text{$distance} = \\
\text{$neighbor->distance($vertex\_finish);} \\
\text{if ($distance < $start\_distance) { if ($sum = $distance +$
\text{ $neighbor->distance($vertex\_start);} \\
\text{if ($sum < $min\_distance) { $min\_distance = $sum; $min\_vertex = $neighbor;}}
\end{align*}
\]

In 20 tryouts it turned out that the mean of node
cost for Two-Steps algorithm is 11.36 as shown on the
picture:

3.4 The Midpoint Routing Algorithm:
The idea is to minimize the Euclidean distance to \( m \),
where \( m \) is the midpoint between the current forwarding
node \( v \) and the destination \( t \). The interesting fact is that
the traveler will reach the destination even though the
traveler will only reach the midpoint to the destination
on a tour of a Delaunay triangulation. If we would only
compare the interval to the middle point, there will be
some cases where smallest path will not hold. To avoid
it we need to add extra line to our code:
\[
\text{if ($neighbor->isEqual($destination) )}
\]
The core of the Midpoint Algorithm is presented below:

\[
\text{foreach ($neighbors as $neighbor) { if ($neighbor->isEqual($destination))}
\]
\[
\text{ $graph->addPath\_Edge( new \\
\text{Edge($current, $neighbor));)
\end{align*}
\]

Now our algorithm has the same structure as Two-Steps Routing Algorithm. The core of the
Apex-Angle Algorithm is presented below:

\[
\text{foreach ($current->getNeighbors() as $neighbor) }
\]
\[
\text{ if ($neighbor\_angle Apex\_Angle::
\text{ =get\_Angle($current,$neighbor); $difference = abs( $st\_angle -
\text{ $neighbor\_angle );
\end{align*}
\]

\[
\text{ if ($difference > pi() ) }
\]
\[
\text{ //Difference crosses 0/2pi line $difference = abs( (2 * pi()) +
\text{ $neighbor\_angle);}
\end{align*}
\]

\[
\text{ if( $difference < $min\_difference){ $min\_difference = $difference; $next = $neighbor;}}
\end{align*}
\]

\[
\text{ $path = new Edge($current, $next); $graph->addPath\_Edge($path); $current = $next;}
\end{align*}
\]

In 20 tryouts it turned out that the mean of node cost
for Apex-Angle algorithm is 10.8 as shown on the
picture:
4. RESULTS

To perform this project, we used PHP 5.4.12, Apache 2.4.4, Vim 7.2, GD library, and Windows 7. Source code can be found at https://github.com/CruorVolt/OnlineRouting. We strongly advise to start exploration by checking the README file where all current updated information about the project are located. Running time of the program is $O(n^2)$ but can be improved to $O(n^{1.5})$. Our test latitude was 100 vertices and grid size 600x600. We compared between each other four Memoryless Online Routing Algorithms. Vertices were placed on the graph randomly. Our experimental results provide findings on efficiency, average node cost and average path cost.

The experimental results showed that the most efficient is Dijkstra’s (100%) and the least efficient is Midpoint (89.64%). If the number of nodes is n, then on average it takes n/10 vertices to get from source node to destination node.

In 20 tryouts when a package travels from a source node s to a destination node t, on average the shortest path will consists of 10.45 node. When traveler uses Midpoint algorithm, then the average node cost is the smallest (9.25) and when traveller chooses Two-Step algorithm, the average node cost is the biggest (11.35).

Now let’s considering average path cost. In 100 tryouts when a package travels from a source node s to a destination node t, on average the shortest path will cost 814.54. When traveler uses Dijkstra’s algorithm, then the average path cost is the smallest (782.81) and when traveller chooses Two-Step algorithm, the average path cost is the highest (877.56) as shown below:

5. CONCLUSION

There is a significant difference between each online routing algorithms. We can deduce that [2] there is no upper bound for existing MOR algorithms. The Two-step Routing and the Apex-angle Routing work the best in Euclidean metric. That is why they can be used in the scenarios such as robotics where reducing the Euclidean path length is the primary goal. The Midpoint Routing algorithms perform best in worst cases in Euclidean and link metric. Because of that they fit for the scenarios where unacceptable worst case performance is to be reduced.

6. FUTURE WORK

If time permits, we strongly believe we should consider more MOR algorithms and confront them with each other. we should try embed functionality that allows running more than two algorithms in the same time. We may find new discoveries. We definitely would like to work more on Improvement and making a program more efficient in the Presorted Bowyer-Watson and Dijkstra’s Priority Queue area. What is more, we would.
like to introduce the ability to import and save results in a text file. Our contribution to the geometric field would be highly beneficial.

REFERENCES: