Recovery Methods for Local Geometric Routing Algorithms

Owen Hnylycia
University of Manitoba
Winnipeg, Canada
hnylycio@myumanitoba.ca

***Abstract*— Local geometric algorithms are split into those that produce close to optimal path lengths, and those that guarantee delivery. To obtain an algorithm that has both characteristics two algorithms are combined. Algorithms that guarantee delivery are rarely built with optimal path length as a major consideration, thus there is little data comparing their performance. We examine the performance of different recovery algorithms when they are used as a recovery method for algorithms that produce close to optimal path lengths.**

Keywords— Local geometric routing, void traversal, guaranteed delivery

I. Introduction

Local geometric routing algorithms are those that use the locations of neighbors, and the destination, as well as a small amount of data passed with the message to make forwarding decisions. Local geometric routing algorithms are designed to be used on wireless networks, where the locations of nodes are known, and their distance apart accurately predicts if they can communicate. Routing algorithms can thus be designed using geometric properties of the network. Along with the geometry of the network some additional information is needed to guarantee delivery, this is passed with the message and may be modified before being forwarded to the next node. There are routing algorithms that are able to find routes that are close to the shortest, and those that can guarantee delivery, but no single approach can do both. To achieve routing that both performs well and guarantees delivery two routing approaches can be combined, the default base approach finds short paths, and the second, guaranteed delivery algorithm recovers when the base is unable to find a valid next node. Guaranteed delivery algorithms are not designed with optimal path length as a main target, and thus there is little information available on how they perform.

A. Contributions

Local geometric routing algorithms that can guarantee delivery have many advantages over those that do not, but delivery on its own is often not the goal. Efficient delivery is often more important than guaranteeing delivery, when a base strategy and a recovery strategy are combined the best of both can be achieved. Local geometric routing algorithms that do guarantee delivery are rarely performance tested and are even less often compared against each other. The aim of this paper is to give data to compare algorithms that guarantee delivery in the context that they will be most used, as secondary recovery algorithms for an efficient routing approach. Knowing how to most efficiently enter and exit the recovery algorithm will be important to the implementation. There are advantages when running as a guaranteed delivery algorithm that allow for a reduction in message size and make implementation much easier, as in left routing.

B. Model and Definitions

The nodes in this model only had knowledge of their neighbors, all other information was to be passed with the message. The only preprocessing done on the graph was finding a planar sub graph. The nodes were arranged as a random geometric graph. The random geometric graph is created by defining a region and placing points randomly in that region, each node is then connected following a unit disk. Unit disk graphs are those where each node being connected to the nodes within a pre-specified radius. Networks were generated with a varying number of nodes, between 20 and 500, and total network area, from 20 to 225.

The procedure used to generate the graphs for this analysis, generated random points, connected each point to those within range and removed unconnected sections, if the resulting graph contained enough connected nodes, it would then have routing checks done. General position was not enforced when generating the graphs, and since whole numbers were used for position, general position was often violated. To allow routing that required general position a "bump" function was used to move the points by a deterministic but very small fraction. Once a connected graph had been generated, two random points would be selected as the start and end nodes. With start and end nodes a filter runs to check if the base routing strategy does get stuck routing between those nodes, graphs that do not get stuck were discarded as differences in recovery algorithms would not be found with those. Some graphs were generated without filtering out the ones with complete paths to verify that the results don't vary from those that have been filtered, these were the sparse and dense graph sets. With a valid graph and valid end nodes, other graph data would be calculated like the minimum path and average neighbors. Since most recovery strategies rely on a planar graph one must be generated from the nonplanar base graph. To go from a base network to a planar network, a unit disk graph can be converted to a Gabriel graph [5]. A Gabriel graph is constructed through checking if there are any connected nodes within the diameter circle two nodes being checked.

When implementing a local geometric routing algorithm, it is important to consider what type of graph is being used as delivery is not guaranteed on all graph types with any recovery algorithm. The base routing strategies all work best with as many connections as are available. The only recovery algorithm tested that did not require a planar graph was Void routing. Network density can affect how recovery algorithms perform, as the denser the network the less a recovery strategy is needed and the quicker it will be able to revert to the base strategy. Graphs of varying size and density were generated.

C. *Related Work*

There are algorithms and approaches to combining two routing strategies that have been done previously. For surveys on void recovery see, [2], [10]. GOFAR was the first of the multi-mode routing strategies, it combines greedy routing with a face recovery method. While the recovery algorithms tested here are separate algorithms, there are some approaches that make modifications to the planar recovery graph [11]. Flooding strategies also exist to recover, these methods send multiple messages in an effort to reach the destination [7]. For Survey papers on Local geometric routing in general, [14].

II. Routing Strategies

A. *Base Routing algorithms*

Three base routing algorithms were used for testing, greedy, compass, and greedy compass. Greedy routing selects the next node, as the neighbor that is closest to the end. The decision of when to enter and exit the recovery strategy is easy when one metric is always decreasing. Greedy routing enters the recovery algorithm when no neighbor is closer to the end point than the current node, and it returns to greedy once a node is found that is closer to the end point than the one it entered recovery on. Having a metric that is always moving in the same direction is also a simple way to eliminate loops from a routing strategy, the routing will never return to a node that it has already been to.

Compass routing uses the angle to the end as the metric to pick the next node. Using the line between the current node, and the end, the neighbor which has the smallest angle to this line is where the message is forwarded to. Compass routing will loop. To combat looping, three versions of compass routing were tested. The first method used a simple counter that would be passed with the message, once the message has been passed a set number of time routing would switch to a recovery strategy. The second used memory at each node to track which messages had been seen by that node, once the message had been seen more than once the recovery strategy would be used. Compass routing with a greedy condition was also used, If the node chosen by compass forwarding was further from the end than the current node the recovery strategy would then be used. The discussion on looping will be left for section III.

Greedy compass routing was also implemented. greedy compass selects the greedy option between the best left-hand and righthand compass options. The greedy compass algorithm that was used for this analysis, will only forward to nodes that are closer to the destination than the current node, this eliminates the possibility of looping. Greedy compass used the same decisions on when to enter and exit the recovery strategy as Greedy routing.

B. *Face Routing*

Face routing was the recovery algorithm that was used as a baseline to compare all others against. Face routing was originally proposed in 1987 [6] and was one of the first local geometric routing algorithms. Face routing uses the planar faces to route along. To start the algorithm, a progress line is drawn between the start node and the end node, this line is used to keep track of movement towards end. The algorithm will traverse the first face in the direction of the end node searching for the edge that crosses the progress line closest to the end. Fully traversing each face is necessary to guarantee delivery. Once the whole face has been traversed, the message will travel to the closest crossing edge found, and switch to the next face. The same face traversal and switching will occur until the end node is found. Face routing requires knowledge of its predecessor, the start node or the line between the start node and end, the node that it started the current face at, and the distance at which the closest edge crossed the progress line.

Quick face routing is a variation of face routing that does not guarantee delivery but is faster on its own. While face routing goes around the whole planar face quick face routing switches faces as soon as an edge that crosses the start to end line is found. Many descriptions and examples of face routing are actually of quick face routing, this is an issue as quick face routing does not guarantee delivery.

C. *Left Routing*

Left routing is a new form of recovery routing that we are proposing based on face routing. This routing approach is equivalent if routing leftwards or rightwards around the planar face but will be described as “left routing”. Left routing removes all face switching components of face routing, and just always passes the message to the next lefthand neighbor. Removing the face switching ability then relies on each planar face having a node that satisfies the exit condition of recovery. When combined with greedy routing this would be that there is some node on all planar faces that is closer to the end than the local minimum node that greedy failed to route on. If it is then true, that there is always a node on the planar face that satisfies the greedy exit condition then left routing will guarantee delivery and performance will be the same as face routing. Face routing must route around the whole planar face before switching to the next face, and thus if there is a node on that planar face which satisfies the exit condition, then face routing will always exit before it switches faces.

When a Gabriel graph is constructed as the planar graph of a unit disk graph, there will always be a node on each face that satisfies the exit condition. When beginning to route around the planar face, the left-hand neighbor in relation to the end point is chosen, and thus we will always be on the face that is towards the end point. The arrangement that would create a planar face where one node is closer to the end than all others, is seen in figure 1. If the plane is divided by a line perpendicular to the line created by the endpoint and the local minimum, which goes through the local minimum, there must be at least two nodes on the side of the end node, and they must be connected and further from the end than the local minimum. The two nodes that are connected through the edge that is closer to the end than the local minimum will be removed in the Gabriel graph as they must be on either side of the local minimum. For the two nodes to remain connected, they would have to be beyond the end node.

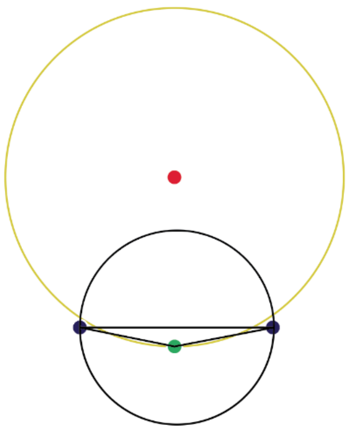

Figure 1: Planar face with one node closer to the end than all others on the face.

D. *Void Routing*

Void routing uses the non-planar graph to apply face routing principles. This approach was proposed by Clouser, et al. [13] in 2013, in that paper they proved that this technique works for quasi-unit disk graphs, unit disk graphs are a subset of quasi-unit disk graphs and thus their approach will work with the model being used here. Void routing uses knowledge of the two-hop neighborhood and an instructional message type to guarantee delivery. Void routing creates a closed void to route around using the intersections of edges, the void that is created by the absence of any edge is what is routed around. To accomplish Void routing, at each node the two-hop neighborhood is checked, if some edge crosses the edge just routed along and is part of the void, the message is sent along that edge. To route along the edge found the node that is first, in the routing direction, is given instruction as to which node is next. Face routing is non-trivial to implement, and void routing is even more difficult, there is added complexity from void crossing points. Two hop edge crossing is also more difficult to implement than it would initially seem. Void routing does require more storage at each node, some additional setup, and more data to be passed with each message. While this was already the fastest recovery algorithm tested, the advantages are greater when routing on a quasi-unit disk graph.

E. *Variations on routing approaches*

Face routing, Left routing and Void routing were the three main recovery algorithms tested, with variations of each also being tested. As already mentioned, Quick face routing which switches faces as soon as it can, was used. A version of left routing that used the non-planar graph and does the calculation to arrive at a planar graph was tested. While there is no proof that the non-planar Left routing strategy guarantees delivery, it did not fail in any of the testing. As exiting the recovery strategy as soon as possible was tested, never exiting the recovery strategy once entered was also tested. Modifications were made to Void routing to apply the same void switching principles as quick face. Another change to quick void was made so that if the two-hop neighbor was connected to the node that would send a directive message, it would send the message directly to that node and replace the predecessor information so that routing could continue.

III. Loops

When choosing a base strategy whether it will loop should be one of the first considerations if guaranteed delivery is desired. Recovering from a loop has not been shown to be possible without additional information either held by each node traversed or passed along with the message. The simplest way to detect a loop is to have a counter of the number of nodes visited and some break point at which the recovery method is engaged. Since the size of the network is not known, a guess must be made to pick the break point. looping until a counter is reached will require more data to be sent with each.

message, and if the break point is set overly high or the loop is encountered early a significant number of resources will be used by the network passing the message around the loop. Another method tested is maintaining some memory in the node, of which messages have been seen, and if a message has been seen over a set number of times, activating the recovery strategy. The number of nodes remembered will depend on the busyness of the network, how many messages are being passed how often? The other question is how many visits before a recovery strategy should be implemented? The answer to the second question is much easier than the first. with compass routing at each node, it will make the same decision as to which neighbor the message will be forwarded to, therefore if the message is at the same node for a second time, the next node will be the same as the first time that message was forwarded.

IV. Testing Results

Greedy routing was the fastest base routing strategy of the three tested. It outperformed the local memory compass strategy by 55% and greedy compass by 7%. While compass routing did have a shorter average path length for each bin, fewer of the paths were

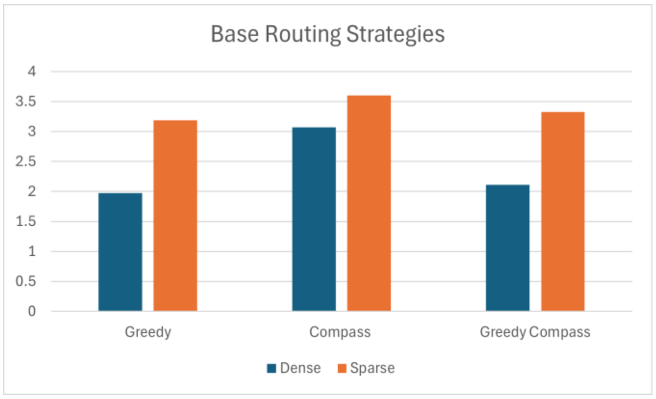

Figure 3: Stretch comparison of Greedy, Compass, and Greedy compass base routing strategies.

shorter.

Splitting the recorded path lengths into two buckets provided a better sense of how the routing was performing, this was done at greater and less than 3 times the shortest path. To inform at what threshold the bin should be, a histogram of the path length

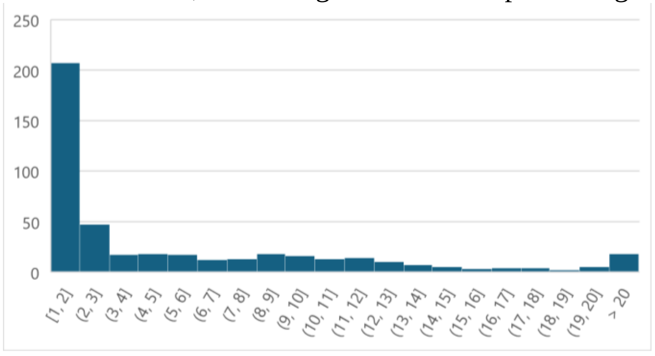

Figure 5: Stretch histogram of greedy base routing, face recovery on the base set of graphs.

frequencies can be made. See figure 3 for the histogram of greedy base, face recovery on the basic set of graphs.

To get a full sense of how each recovery approach performed, looking at the raw results gives the clearest picture. The order of recovery performance from fastest to slowest was Quick Void, Void, Quick Face, Non-planar Left, Face and Left, Full Face. Quick Void outperformed Face by 20% depending on the graph set, and Void outperformed Face by 17% depending on the graph set. The path lengths in the shorter path bin only changed by 3% to 4% and the number of paths in the bins was also quite stable on the same graph set. In the short paths all approaches perform close to the optimal length, and thus there is little room for differences to be seen. Significant performance differences were found in the longer paths. The long paths are when the recovery algorithms run for longer and thus more of the differences are shown.

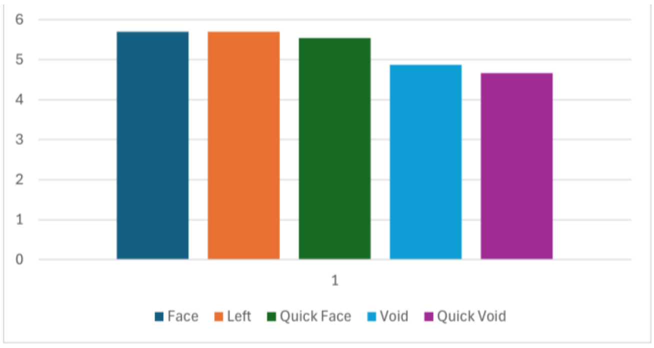

Figure 2: Recovery averages with Greedy base routing on the basic set of graphs.

A simple change that made all recovery operations route quicker was to exit the recovery algorithm when any neighbor of the current node satisfied the exit condition. Using the recovery algorithm as little as possible results in better routing. Exiting the recovery algorithm sooner resulted in a 5% improvement in path length.

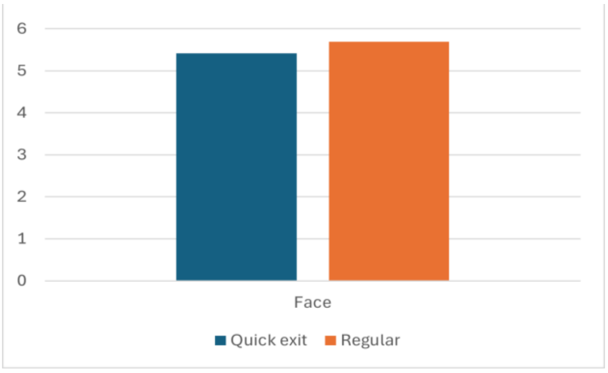

Figure 4: Exiting when any neighbor satisfies the exit condition vs when the node satisfiest the exit condition.

With looping base strategies some method of detecting loops is necessary. Three different loop detection methods were tested with compass routing, a counter, local memory, and a greedy distance measurement. The counter was set to some multiple of the total number of nodes in the graph, once the message had been passed more than that limit the recovery algorithm would be used. For all looping recovery there is no returning to the base strategy as one would have to guarantee that the loop will not be entered again, no methods were found or created that could accomplish this. The Local memory method had each node remember the messages that it had seen and if the message appeared again the recovery algorithm would be used. A different side of greedy compass was used, this version would route normally as compass routing but only to nodes that are closer to the end node than the current one. The modified compass greedy strategy performed best. The hop count approach got close to local memory but was never better than it, the local memory was able to detect a loop as soon as it occurred whereas the hop count approach must balance efficient loop detection with unnecessary use of the recovery algorithm.

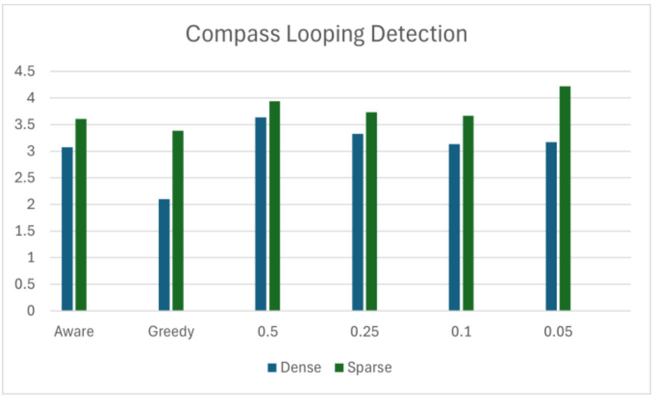

Figure 6: Loop detection comparison of compass routing, with face routing as the recovery.

V. Observations

All algorithms that fully traverse the face will perform the same when used as a recovery algorithm if every face contains a node that exits the recovery strategy. When using greedy routing and a Gabriel graph as the planar sub graph face routing and left routing perform the same because there is always a node that exits the recovery strategy. For a guaranteed delivery algorithm that uses planar faces to route, going around the whole of each planar face is necessary to not get stuck in a loop and thus any face routing strategy will perform the same. To get improved performance routing around voids the routing algorithm will have to change the void or planar face in some way. Non-planar void routing is able to perform better because the void it is routing is smaller than the one that face routing uses, and strategies such as [12], have potential to also be faster as they manipulate the planar void. Methods to exit the recovery algorithm sooner could also bring improvements in routing efficiency.

Although guaranteed delivery is regularly the goal, dynamic network conditions can make it not a realistic expectation. If a routing protocol is routing around a void and edges are added during that process, those edges can enclose a face that did not exist previously and result in an infinite loop. If network conditions are allowed to change while messages are being routed, then even if neither the base nor recovery algorithms loop, loop recovery may still be necessary to implement. Given changing network conditions it may also be necessary to handle the end node being disconnected from the network. The real world is often more chaotic than we would like to admit and mechanisms that are robust to failures compromise performance ideals but can result in better real-world outcomes.

When considering how much extra data can be passed with the message while still being more efficient, using the total data passed makes comparison easy. If an algorithm passes 10% more data with each message, then it must send 10% less messages to have the same routing efficiency. The differences in which routing strategy performs best is dependent on the size of the messages being sent, the larger the message the more tolerant of extra routing it is. When looking to improve the routing efficiency of recovery algorithms it may be useful to focus on which aspect of recovery it is important to improve, the paths that are close to optimal or the paths that are multiples of the shortest. Some of the approaches that are designed to smooth out voids may make the long path recovery on sparse graphs much worse.

VI. Directions for Future Work

While left routing works on Gabriel graphs, when greedy routing is the base strategy, it would be valuable to see if the same approach can be used with other base strategies, and a broader class of graphs. Proving that a calculation on a non-planar graph to achieve a planar face routing is valid would guarantee routing on any graph with only a distance variable needing to be passed with the message. While the Nonplanar left routing never failed to reach the end in testing, we have yet to find a proof that it will always reach the end.

VII. References

[1] C. Yi and P. Wan, "On the Longest Edge of Gabriel Graphs in Wireless Ad Hoc Networks" in IEEE Transactions on Parallel & Distributed Systems, vol. 18, no. 01, pp. 111-125, 2007. doi: 10.1109/TPDS.2007.16

[2] D. Chen and P. K. Varshney, "A survey of void handling techniques for geographic routing in wireless networks," in _IEEE Communications Surveys & Tutorials_, vol. 9, no. 1, pp. 50-67, First Quarter 2007, doi: 10.1109/COMST.2007.358971.

[3] F. Kuhn, R. Wattenhofer, and A. Zollinger, “Worst-case optimal and average-case efficient geometric ad-hoc routing,” _Proceedings of the 4th ACM international symposium on Mobile ad hoc networking &amp; computing_, Jun. 2003. doi:10.1145/778415.778447

[4] F. Zhang, H. Li, A. Jiang, J. Chen and P. Luo, "Face Tracing Based Geographic Routing in Nonplanar Wireless Networks," _IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications_, Anchorage, AK, USA, 2007, pp. 2243-2251, doi: 10.1109/INFCOM.2007.259.

[5] G. Liotta. 1996. “Low Degree Algorithms for Computing and Checking Gabriel Graphs”. Technical Report. Brown University, USA.

[6] G. G. Finn, “Routing and addressing problems in large metropolitan-scale Internetworks”, Mar. 1987. doi:10.21236/ada180187

[7] M. Rekik, N. Mitton, and Z. Chtourou, “Geographic greedy routing with ACO Recovery Strategy Graco,” Ad-hoc, Mobile, and Wireless Networks, pp. 19–32, Jul. 2015. doi:10.1007/978-3-319-19662-6_2

[8] Kranakis, E., Singh, H., & Urrutia, J. (1999). Compass routing on geometric networks. Canadian Conference on Computational Geometry.

[9] P. Bose _et al._, “Online routing in convex subdivisions,” _Algorithms and Computation_, pp. 47–59, 2000. doi:10.1007/3-540-40996-3_5

[10] S. Parvin, M. A. Sarram, G. Mirjalily, and F. Adibnia, “A survey on Void handling techniques for geographic routing in VANET Network,” _International Journal of Grid and Distributed Computing_, vol. 8, no. 2, pp. 101–114, Apr. 2015. doi:10.14257/ijgdc.2015.8.2.10

[11] S. Ruehrup and I. Stojmenovic, “Optimizing communication overhead while reducing path length in beaconless georouting with guaranteed delivery for wireless sensor networks,” IEEE Transactions on Computers, pp. 1–1, Dec. 2013. doi:10.1109/tc.2012.160

[12] S. Tao, A. L. Ananda, and M. C. Chan, “Greedy face routing with face identification support in wireless networks,” Computer Networks, vol. 54, no. 18, pp. 3431–3448, Dec. 2010. doi:10.1016/j.comnet.2010.07.004

[13] T. Clouser, A. Vora, T. Fox, and M. Nesterenko, “Void traversal for efficient non-planar geometric routing,” _Ad Hoc Networks_, vol. 11, no. 8, pp. 2345–2355, Nov. 2013. doi:10.1016/j.adhoc.2013.05.013

[14] X. Guan. “Face routing in wireless ad-hoc networks”. PhD thesis, Univ. Toronto, 2009