1 Paper 4: Karp, Kung GPSR: Greedy Perimeter Stateless Routing for Wireless Networks

This is a widely cited paper, in fact the main one on geographic routing. The basic idea is to always route greedily. When you get to a local point where the greedy breaks, then use a slower but guaranteed-to-work routing protocol to get out of the local bad spot. Once out of the bad spot, continue with greedy routing. Here, they use a node's location info to make greedy choices and do face traversal for the spots where greedy routing fails. The algorithm only uses neighbor information to make the next hop decision, so the scaling is superb in terms of state needed per node. The paper also claims that the algorithm is very good at adapting to dynamic environments.

1.1 Introduction

Considering scaling in terms of:

- number of nodes in the network
- rate of change of topology

The most well understood tricks to help with scalability are hierarchy and caching. Hierarchy does not work too well in dynamic ad-hoc wireless networks because it is based on well-defined and rarely-changing boundaries. Caching is useful (seen in DSR) but it is also not as scalable as using only local information (although GPSR can benefit from piggybacking).

Here, the main idea: use location information to improve scalability. Aim for scalability, for both factors above. Use the following measures for the scalability of a system:

- Routing protocol message cost
- Application packet delivery success rate
- Per-node state

Geographic routing allows routers to be nearly "stateless". Requires propagation of topology info for only a single hop: each node needs to keep track of only its immediate neighbors' positions. "Self-describing nature" of position is the key to geography's usefulness in routing: the position of a packets' destination and positions of the candidate next hops are enough to make correct forwarding decisions.

1.2 Assumptions

- Assume everyone knows their own positions.
- Bidirectional radio connectivity.
- Circular radio range model.
- nodes and movement are in the plane.
- Assume packet sources know the geographical coordinates of packet destinations (could be obtained from a separate location service. Queries to that service could use the same geographical routing service described here.)

1.3 Algorithm

1.3.1 Greedy forwarding

Here we will consider the basic greedy algorithm using location. The originator sends a message tagged with the geographical location of the destination. Each node will forward a SINGLE message (unlike LAR/Geocast) by greedily choosing it's optimal neighbor. Optimal neighbor is the neighbor that is geographically closest to the destination.

For this algorithm to work, nodes need to know who their neighbors are, and what their locations are. Since we are considering mobile networks, this info has to be kept up-to-date. We fix this by having each node periodically transmit a "beacon" message containing it's own identifier and position. Others keep updating their info based on the beacon messages they hear. Use timeout to remove nodes from the neighbor set (if you don't hear for a neighbor in a while, assume they are no longer a neighbor.). Furthermore, we can also use link-level NACK info if available (e.g., in 802.11).

Notice that the state per node needed depends only in the density of nodes in the network, not on the total number of nodes. This is a VERY big difference when compared to DSR (which requires state proportional to all the routes it hears) or AODV (which needs state proportional to the possible number of destination forwarded to, limit of all destination).

Correct choice of frequency of beacon transmissions depends on rate of mobility and range of transmission.

Beaconing mechanism is proactive, but not excessively costly because it is all local. Optimization: Can piggyback beacon info on all other packets, avoid sending some beacons (reset timer whenever it sends the info). Could also make the mechanism reactive, only using it when there is actual data traffic to forward—sounds like a nice idea! But they haven't pursued this.

While this greedy algorithm is fast and requires little state, it can fail (see the standard example in Figure 3.)

1.3.2 Combining greedy and planar perimeters

Well, since greedy can fail, but it's very good in most cases, then let's just use greedy until we cannot anymore, then switch to face routing (Face-II in Bose et al.) for getting through the failure regions of the greedy algorithm. As soon as we can, go back to greedy routing. The algorithm basically works in two separate graphs: the original graph is used for greedy decisions, and a planar graph (constructed as in Bose et al.) is used to do the face routing when greedy fails.

They describe their message format, which includes:

- the location of the destination
- the location where the message entered "perimeter mode" (used to check if we can go back to greedy mode)
- the point shared between current and previous face while doing face traversal (no usually at a node location, rather the point that intersects the line used in face routing)
- the first edge a message traversal while in face traversal
- flag denoting if in greedy mode or perimeter mode.

A message initially starts in greedy mode, with the sender filling in the destination's location field. When a node receives a message in greedy mode, it checks to see if there is a neighbor closer to the destination. If the is, then the message is forwarded. If not, the "perimeter" mode is entered. The node records the entry site of perimeter mode in the message. Each time a new face is entered, a node records the intersection point shared between old an new face. Lastly, a node records the first edge crossed in a face in the message field. While in perimeter mode, each node checks to see if their destination is closer than the entering location of perimeter mode and, if node is closer to destination, the message returns to greedy mode.

In perimeter mode, GPSR, forwards the packet using a simple planar graph traversal like that in Bose et al. (assumes we already have reduced the graph to a planar graph using algorithm like the one in Bose et al.). If the packet enters perimeter mode at node x, and is bound for destination D, they work from the line segment x-D. As in the Bose et al. paper, then traverse progressively closer faces of the planar graph, looking for crossings of the line x-D. This can involve traversing both interior faces and the exterior region. In either case, they claim that the right-hand rule will allow traversal, to the point of finding the other crossing. When they find the next crossing, they switch to the new face on the other side of the crossing point.

Note that if they are traversing the exterior face in the "wrong" direction, then the message will route all the way around the figure. The packet will eventually get to the destination, but it will take a lot of hops to do it. This is why face traversal is only used when greedy fails.

If D isn't reachable, the packet will tour unsuccessfully around a face. This can be detected because of the recorded info about where the packet entered the current face.

The algorithms we saw that used constrained flooding dealt with the high mobility of the network. The greedy algorithm here also deals with high mobility, however the face-traversal algorithm is not very good since it assumes that the planar graph you are routing on will not change before you have reached your destination.

1.3.3 Protocol implementation

Some details here:

- Support for MAC-layer failure feedback
- Reducing rate of sending local beacons by piggybacking
- use promiscuous mode of receiving broadcast packets.

And then there is a planar graph recalculation. How do they handle recalculating the planar graph in the presence of mobility and other changes ?

The GG or RNG graph depends on the current set of neighbors, which is constantly changing. They recalculate upon every acquisition of a new neighbor and every loss of a former neighbor. But this really isn't enough—the choices of links to remove could change even in the face of minute movements that don't change the neighbor relation. This means that their perimeter traversing sub-algorithm will be working with possibly stale planar graphs and in fact fail to find a destination. They state that in the future they want to try to incrementally change the planar graph on receipt of any beacon messages with neighbor locations.

1.4 Simulation results and evaluation

They focus on mobility and compare their algorithm to DSR.

1.4.1 Simulation environment

- ns-2 with CMU wireless extensions Mobility uses random waypoint model (they use the pause length to capture the degree of mobility)
- Nodes initially randomly placed in a square. (seem to be considering rather dense networks. Because they are trying to compare with existing results by Broch et al. on DSR, and that's what they used.)
- Use idealized location database (assume exact target location is known by the sender at sending time).

Evaluated:

- Packet delivery success rate
- Routing protocol overhead
- Optimality of path lengths

1.4.2 Packet delivery success rate

High. Better when they use shorter beaconing intervals, levels off when the interval is comparable to the pause time. They saw no difference when the change B (beacon rate) to greater than 1.5. Note, they say that the dense networks begin simulated will cause very few disconnections of nodes (this should be studied further I think) so few possibilities of using stale planar graphs..

1.4.3 Routing protocol overhead

Constant overhead because of the beacon nodes which are only sent locally.

1.4.4 Path length

Delivers vast majority in optimal number of hops. But that seems to be because greedy routing works almost always in the dense networks they consider. If they have to default to perimeter routing, it seems that they would get long paths sometimes. Once again, simulating with much sparser networks would be helpful in evaluating the algorithm.

1.4.5 Effect of network diameter

They show that the network diameter has little effect on the algorithm. This is because GPSR uses local routing and so route-length does not in any way affect these local actions.

1.4.6 State per router

Trivially low – just neighbor info.

1.4.7 Location database overhead

Adding location registration and lookup traffic would increase the overhead. However, they seem to think it could be a single lookup (DNS style). Some possible optimizations: Queries and updates to location service can themselves be routed using GPSR.

1.5 Related work

Finn actually proposed greedy routing + flooding search when greedy fails, way back in 1987.

Ko, Vaidya LAR: KK describe this work as an "optimization to DSR", since it's about route discovery, not forwarding. Basically, LAR is a strategy for limiting the flooding involved in propagating RREQs. When LAR's forwarding region isn't enough, it reverts to complete flooding. There is quite a difference between the flooding style of KV and the single-packet-propagation style of this paper.

1.6 Future work

- They note the issue about inconsistent estimates at two neighbors, in producing the RNG or GG. Needs to be solved.
- relax the communication model (right now they have a disk model)
- Measure the combined behavior of GPSR and a location database service.
- Compare use of RNG and GG
- Extension to 3D

2 Barriere et al., Robust position-based routing

An interesting paper, with some decent practical motivation. However, the practical significance of their model is questionable.

The basic idea is not to assume exact, known transmission radii as in the unit disk model. Rather, they assume two known numbers $r \leq R$, where:
Each pair (u,v) of nodes that can communicate are at most R apart.
Each pair (u,v) of nodes that are at most r apart can communicate.
However, nodes that have distances in (r,R), might or might not be able to communicate.

By introducing those radii, they attempt to address non-uniformity of antennae radiation patterns, such as the following: However, this non-uniformity tends to be quite limited, but their model might still deal with obstacles that are at long range.

Their generalization looks useful; however, they assume that for each pair (u,v), the existence or nonexistence of an edge does not change—that is, either they can always communicate or they can never communicate. Unfortunately, this decision at the edge of effective range of an antenna is hard to make and even harder to maintain! Anyway, they assume a very static setting. Their justification for this is that changes in communication capabilities will occur more slowly than the communication speeds used in routing, so it's reasonable to assume that they are fixed for the purpose of routing. However, they don't really seem to adequately address how their algorithm would cope with the changes.

Anyway, they give a routing protocol for this setting; it works provided the ratio between R and r is no more than $\sqrt{2}$.

2.1 Introduction

MANET background and motivation. Unknown initial topology. Mobility.

Use of location info, everyone knows their own location.
Some history:
LAR (Ko, Vaidya), is essentially DSR with restricted forwarding zones, limited flooding of RREQ packets.
DREAM protocol (Basagni et al.) uses limited flooding of data packets.
Non-flooding protocols, forward packet to exactly one neighbor
Greedy routing, forward to neighbor closest to target
Compass routing, forard to neighbor along the line whose angle to the direction of the target is smallest
Both can fail
Unit disk model, with a single range R, is commonly used.

Bose et al, and GPSR:
Guarantees delivery
Perimeter routing in the Gabriel graph
Combined with greedy routing, switch back to greedy whenever possible

Claim that this can fail if there is some "instability in the transmission range".
By which they mean that:
The area that a transmission can reach isn't necessarily a disk, and it may change over time, but their paper doesn't seem to handle this case.

Motivation: Obstacles, bad weather

GPSR strategy isn't robust to having such instability:
They say this is the case even when only bidirectional communications are assumed—every edge is either there in both directions or in neither
Gabriel graphs aren't "robust":
Figure 5 (in paper), nice example: u and v don't agree on existence of the vertex w—u sees it and v doesn't.
So, u would make a decision to remove the edge (u,v) on the basis of the witness w, but v would not.
Not easy to resolve this:
Removing might disconnected the network, since v can't communicate with w.
But leaving it can leave crossing edges—see Figure 6, where u sees two witnesses v doesn't see. If the edge uv remains, it crosses (w, w').

So in this paper, they figure out what to do to accomplish routing in such cases.
Their protocol works as long as the ratio $R/r \leq \sqrt{2}$.
This restriction guarantees that for any two hosts u and v, if there is another host w in the disk around the segment (u,v), then at least one of u and v will be aware of w. To see this, draw w anywhere in the circle with diameter (u,v).
Draw the triangle uvw.
Angle w is geq pi/2.
So $d(u, w) + d(w, v) \leq d(u, v)$, which is $\leq R$.

Claim at least one of d(u,w) and d(w,v) is $\leq r$: if not, then we would have the $d(u,w)+d(w,v) > 2r \geq R$, contradiction the assumed bound on R/r.

In order to maximize R/r, we should maximize R and minimize r. Let us maximize R by selecting the distance between nodes $d(u,v) = R$. Then, we require the minimum value of r for which point w is always in range of either u or v, that is, w is either in the circle with radius r around u or v. Figure 1a shows w outside of both circles while in the disk of diameter uv. The minimal value of r (thus the maximal of R/r) for which w is marginally inside all circles is shown in Figure 1b, in which:

$$\frac{R}{r} = \frac{uv}{xv} = \frac{2\rho}{wv} = \frac{2\rho}{\rho\sqrt{2}} = \sqrt{2} \tag{1}$$

They make a funny claim that existence of "such a protocol" if R/r is unbounded is unlikely. By this they could mean that if w is outside the range r of both u and v, then maybe a third node x is a neighbor to both u and v and has w within its range r. Thus, node x could inform u and v about the existence of w.

Protocol uses almost-synchronized clocks.

2.2 Model

Nodes in 2D, know their own coordinates.
Let $r \leq R$ and $R/r \leq \sqrt{2}$
Geometric graph G = (V,E), V = nodes, E = links (static).
Undirected (representing bidirectional communication assumption).
E satisfies:
$(u,v) \in E$ implies $d(u,v) \leq R$
$d(u,v) \leq r$ implies $(u,v) \in E$

2.3 A 3-Phase Routing Scheme

Completion phase
Adds "virtual edges" to the original graph G, yielding a denser super-graph S(G), with better symmetry properties.

Extraction phase
Produces a planar connected graph from S(G), using Gabriel Graph methods as before. Some modifications to the GG extraction method to accommodate the virtual edges.

Routing phase
As usual, using greedy routing augmented with perimeter routing when necessary.

2.3.1 Completion phase

Ball B(c,rho)
$Disk_{u,v}$: the circle having diameter (u,v).

Each node bcasts request for coordinates of its neighbors. Any host receiving such a request responds

to the sender with its coordinates. This yields the graph G. (But I think, given the bidirectional edge assumption, that we don't need the requests above—everyone could just bcast their coordinates and whoever receives them is a neighbor). So now every node knows its neighbors in G.

Now, the nodes start adding "virtual edges". Each node u sets up a list L(u), initially with all its neighbors in G. Marks each such node "unprocessed".

Then u starts processing unprocessed nodes:
For any edge (u,v) where v is unprocessed:
For every w in L(u) that is in $D_{u,v}$, but that is not in B(v,r): (These are nodes that u knows about, in the disk it shares with v, but that v might not know about.)
u sends message (new,w) to v, with w's coordinates.
u sends message (new,v) to w, with v's coordinates.
After sending all these messages, u marks v as processed.

Thus, processing v amounts to telling v about the possible witnesses, and vice versa—telling the possible witnesses about v.

As these messages are sent and received, some nodes may update their adjacency lists to reflect new "virtual neighbors". Specifically, if u receives a message (new,w) from a neighbor v, and w is not already in L(u), then u will classify w as a "virtual neighbor", and (u,w) as a "virtual edge". All it needs to do is add w to L(u) marked as unprocessed.
Well, it needs to do a bit more: It needs to record an actual communication path ve(u,w) in G that it can use to reach w. That will be (u,v,w), which means that u, in order to send a message to w, should send it directly to v, who presumably will be able to take care of sending it to w.

More precisely, u sends a message to a (possibly virtual) neighbor v by calling send(v,M), which does the following:
If v is an actual neighbor of u in G, then u just transmits the message, knowing it will reach v.
Otherwise, v is a virtual neighbor, so u uses ve(u,v) = (u,w,v), and recursively calls send(w,(v,M)).

Now, if w were a real neighbor of u, we would not need the recursive call—u could just send the message to w. But w might itself be a virtual neighbor of u, so the recursive call would involve another lookup, to find ve(u,w) = (u,w',w).
Unwinding the recursion one more step, we get send(w',(w,(v,M))). Etc.
Eventually the recursion will bottom out, and u will get to send an actual message. But by then, the message that is sent will be a parenthesized string representing a route to the real, final descintation v.

When u sends this message, the recipient does ordinary forwarding based on this route:
If the message received is just a basic M, you are the recipient, and handle it accordingly.
Otherwise, the message contains a route representation in addition to a basic M, just strip off the first node in the route, getting a shorter route, and send the shorter route and M to the first node.

Anyway, that was a digression.
If u records a new virtual neighbor in L(u), then it processes it just like a real neighbor, looking for other witness nodes, etc. In this way, all the nodes keep processing the new nodes they hear about, until every

node's list is completely processed.

It would be nice if everyone could complete the completion phase before anyone goes on the the extraction phase; however, because this is a distributed algorithm, no one knows when others are done.
They allow for the possibility that some nodes go on to the extraction phase, which involves removing some edges on the basis of having some redundancy, before everyone is done with the completion phase. This could mean that a node that is engaged in the extraction phase could later find out about new virtual neighbors!
But they claim that this doesn't affect correctness—they can process such new neighbors as before, "in parallel with" performing the extraction work. This is saying that any edges that u has removed on the basis of already-available information, can still be removed on the basis of the later information, with more virtual edges. That sounds reasonable.

Another important claim that they will need later, but don't actually state here:
Every virtual edge (u,v) has $d(u, v) \leq R$. Prove this by induction on the number of virtual edges that have been added.

We should understand the above claim in order to see why they talk about learning about all nodes within radius R, at the bottom of p. 23, col. 1. Since we are only talking about nodes within distance R anyway, it sounds reasonable to try to find all such nodes by some kind of neighborhood search. But they claim that just knowing the nodes and their coordinates isn't enough—that we actually need the routes, which is what their construction yields. (Their construction could be nicely written in precondition/effect form, to better capture the asynchronous nature of the processing.)

2.3.2 Extraction phase

Extract the Gabriel graph as usual, but based on S(G) instead of G.

2.3.3 Routing phase

In fact, they say that this phase also can be interleaved with the first two, without affecting correctness. It seems it would be nice to present the combined protocol in some way, then, with invariants that hold overall. Or does their non-quite-complete decomposition actually lead to nice modularity in proofs?

Bose, GPSR style, using the GG.

2.4 Proof of correctness

Lemma 1: If vertex u learns about a virtual neighbor v, then v eventually learns about u.
Proof of this is based on correctness of the routing via virtual edges, which seems kind of self-evident.

Lemma 2: The completion phase terminates.
i
Lemma 3: If u decides to remove the (possibly virtual) edge (u,v) during extraction, then v eventually removes (u,v) as well.
This is interesting. Basically, u and v eventually get the same information, as described in Lemma 1.

This leads to the same decision about each edge. This describes a kind of "monotonicity" property—more information can enable more actions, but cannot invalidate previous actions.

Lemma 4: The extraction phase terminates.

Next we see where they are using the sqrt(2) ratio assumption.

Lemma 5: If G is connected and R/r leq sqrt(2) then the graph after the extraction phase is planar and connected.
Proof: The connectness seems to follow from standard Gabriel Graph theory

The planarity seems to be where we are using the sqrt(2) assumption. This is a little proof by contradiction: Suppose we have crossing edges, must form a quadrilateral (but not as they say, a parallelogram). They argue about possibilities for edge lengths, given the conditions that would force edge removal. They get a contradiction, using the ratio.
QED Lemma 5

Theorem 1 pulls all the pieces together.

2.5 Discussion

Property 1 gives an upper bound on the total length of a virtual route. The bound is based on R and r, as we would expect. But it also depends on lambda, which is the minimum distance between two hosts anywhere in the network. The reason why lamda should not be extremely small probably stems from the fact that they want to avoid bad network layouts, such as the one presented in Figure 7 in the paper. But their bound has lambda (in fact, squared) in the denominator, so the bound could be quite huge.

The proof looks reasonable, induction.

Property 2 shows a bad configuration, producing long virtual edges. This involves nodes that are very close. Again, this is probably why lambda should have a lower bound.

3 Kuhn et al. Theory and practice

They present a new geometric routing algorithm which is both: Very efficient in practical average-case networks, and Asymptotically worst-case optimal, according to theoretical measures. In doing this, they are able to drop the assumption that appeared in some prior work, that network nodes have a minimum distance between them (this appeared in the previous paper).

They also consider the general issue of defining cost metrics for routing in ad hoc networks—they are able to classify them into two kinds, which yield different kinds of results.

3.1 Introduction

Geometric routing, geocasting. Their geometric algorithm is called GOAFR+ ("gopher-plus"). Uses a combination of greedy and face routing. They use an "early fallback" technique to return to greedy

routing ASAP. Claim this makes it more efficient than similar algorithms in the average (practical) case; yet they are still able to get a good theoretical analysis—asymptotically matching the known lower bound. Their analysis doesn't assume the lower bound on inter-node distances.

They introduce a general notion of a "cost metric", defined in terms of a nondecreasing function of the lengths of edges over which messages are sent. What turns out to be crucial, for the cost of routing, is the behavior of the cost function for edge lengths approaching 0. Linearly bounded cost metrics (those bounded from below by a linear function): Here, a clustering method allows construction of a routing backbone, which gives optimality even for arbitrarily close nodes. Super-linear functions (those that aren't bounded in this way): They give an example showing that no geometric routing algorithm can guarantee paths whose costs are competitive with costs of the optimal path.

3.2 Related work

Greedy routing: Finn,...; doesn't guarantee delivery Compass routing, doesn't guarantee delivery

Geometric routing that does guarantee delivery: Face routing, in Kranakis et al. paper (called Compass Routing II there). Guarantees delivery within O(n) steps, n = total number of nodes. Various others, not as good. Their own Adaptive Face Routing (AFR): Enhances face routing by employment of an ellipse restricting the searchable area. Cost of route is leq (cost of optimal route)2. They also prove a lower bound, saying that this cost is the lowest possible for any geometric routing algorithm. (This result sounds very interesting, but I don't think we have time to track it down.)'

But Face Routing and AFR aren't practical, because they just use face routing—no greedy normal case. Previous proposals to combine greedy routing and face routing don't have good worst-case guarantees.

In a previous paper, these guys proposed an algorithm to combine greedy and face routing, in a worst-case optimal way. However, that algorithm didn't fall back to greedy mode ASAP, so it would not be optimal in practice. So here, they are trying to get early fallback while still having worst-case optimality.

Here, they use a new clustering technique to drop the lower bound assumption for inter-node distances. That is apparently the key to achieving both goals (?)

3.3 Model and preliminaries

2D. Unit disk graph, radius 1.

Cost function c: Maps any d in (0,1] (d is a possible edge length) to a positive real c(d). Nondecreasing. Cost of an edge is then defined to be the cost of its length.

Examples of cost functions: 1 (hop count), d (Euclidean distance), d^alpha (energy); positive linear combinations of these

Cost of path = sum of costs of its edges. Cost of algorithm on a graph G = supremum, over all executions of the algorithm on G, of sum of costs of all edges traversed in the execution.

Their routing algorithm assumes G is planar. ¿From a non-planar graph, a planar graph could be derived using Gabriel Graph techniques. And that doesn't have much effect on the costs—at worst, a constant multiplicative increase.

They also consider unit disk graphs with a constant bound k on the number of neighbors ("bounded-degree unit disk graphs with parameter k")

Geometric routing algorithms: Forward message from a given source s to given destination t over the edges of the network graph, where: Each node knows its own and its neighbors' positions s knows t's position Nodes cannot store info about packets (except temporarily, while processing them). Packet may contain info about only a constant number of nodes.

Assume routing takes place much faster than node movement; they use this to justify assuming the network is stationary.

3.4 GOAFR+

Here, the introduce the algorithm, and show it is worst-case optimal for bounded-degree unit-disk graphs. They do simulations to show it also works well on average.

3.4.1 The GOAFR+ algorithm

Management of the bounding circle C: Increase radius of the circle if we hit its perimeter twice while traversing a face. This means that the only way out of this face is by crossing the perimeter of the circle, thus we increase its radius, so as never to get out of the circle. We decrease the radius as greedy routing successfully approaches the target.

They use parameters ρ_0, ρ, σ.

0. Initialize C to be the circle with center t, radius $\rho_0|st|$. That means that s is inside this circle.

1. Greedy mode: As usual, but "whenever possible", reduce C's radius by a factor of ρ. Though, "whenever possible" is too vague a term. Maybe it's when the current node u is closer to t than s, and we can redefine the circle so that the radius is at least $\rho_0|ut|$. In that way, the node u would be behaving like the original s.

2. Face routing mode, starting from u_i: Start exploring the current face. If you hit the boundary of C for the first time, then turn around and continue within C. If you hit C for the second time, then: If you have found any node v closer to t than u_i is, give up exploring the current face, and resume the search from v, in greedy mode. If you haven't yet found any closer node, then enlarge C by a factor of rho, and continue exploring the current face. If it ever turns out that you have encountered sigma times more nodes on the face's boundary that are closer to t than nodes that aren't, then decide that this is "good enough". Stop the search, pick the node v seen so far that is closest to t, and resume from v in greedy mode

So, this looks like a combination of heuristics.

3.4.2 Asymptotic optimality

Prove asymptotic optimality on bounded degree unit disk graphs. In section 5, they will show how to extend this result to general (unbounded degree) unit disk graphs. They will do this by analyzing properties of cost metrics.

Assume planar graph, obtained from Gabriel Graph construction.

Lemma 4.1: Consider the unit disk graph of a set V of vertices in a bounded convex region R. If this graph is k-bounded degree, then the number of points in V is bounded by an expression in terms of k, the area, and the perimeter of the given region, as follows:

$$|V| \leq (k+1)\frac{8}{\pi}(A(R) + p(R) + \pi) \tag{2}$$

Proof: Simple geometry. They get an upper bound on the number of disks of diameter 1 that suffice to cover R. The bound on number of nodes follows since there can't be more than k+1 points in each disk.

Now they talk more about the way C is adjusted. They say that the circles that are used form a sequence C_0, C_1, C_2,..., where C_0 is the initial circle, and each C_i has radius smaller than the previous

one. The values of r_{C_i} form a geometric progression with ratio rho. However, the radius can both increase and decrease during execution So, the steps taken while in C_i need not occur consecutively.

Lemma 4.2 says that if s and t are connected within a circle C_i, then the algorithm is guaranteed to reach t.

The circle C_i may have to be increased during the execution of the algorithm, because we may fall into some local minimum in which case we may need to increase the radius of the circle in order to backtrack out of the local minimum. We can imagine the circle as a constraint that pushes us towards the target, but we may need to relax this constraint, by increasing the radius, in order to get out of some local minimum. If we still fail to approach the target, then we are disconnected from it.

Lemma 4.3: They bound the cost of all face routing steps when exploring the boundary of a face F within a particular circle C_i. This turns out to be $\leq \gamma c_F$, where γ is some constant, and c_F is the number of edges around the face F.

Proof: They show this first for the hop-count metric.

1. Assume that the boundary of face F is involved in k face routing rounds, and s_j is the node where round j started, for $1 \leq j \leq k$: p_j, q_j are the final values of counters p, q in round j. $p_j + q_j$ is the total number of nodes of the face seen in this round: **all nodes seen in this round are either closer or farther to t than s_j, so the total nodes of the face are** $p_j + q_j$

2. In each round **the face is traversed at most twice**: once to find the node closest to t and then at most once again to get to that node.

3. For step 2c to trigger, it should hold that $\mathbf{p_j > \sigma q_j}$.

4. Let P_j be the set of nodes visited in round j that are closer to t than s_j. Since p_j may count the same node twice (p_j is increased without memory of the nodes visited so far), then $\mathbf{p_j \leq 2|P_j|}$.

5. Let N_j be the set of nodes newly visited in round j. Since we get closer to t with every round, all nodes $\in P_j$ must be newly visited ones, thus $\mathbf{P_j \subseteq N_j}$.

6. Therefore the total cost of face F is:

$$c'_F = \sum_{j=1}^{k} c_{F_j} = \sum_{j=1}^{k} 2(p_j + q_j) \quad (3)$$

$$< \sum_{j=1}^{k} 2(1 + \frac{1}{\sigma})p_j \leq \sum_{j=1}^{k} 4(1 + \frac{1}{\sigma})|P_j| \quad (4)$$

$$\leq \sum_{j=1}^{k} 4(1 + \frac{1}{\sigma})|N_j| \leq 4(1 + \frac{1}{\sigma})c_{l_F} \quad (5)$$

$$because \sum_{j=1}^{k} |N_j| \leq c_{l_F} \quad (6)$$

(total cost is at least as much as the newly found nodes)

Then they claim that the result also holds for other metrics; but for this they rely on a result in section 5.1 that is going to show that all metrics are essentially the same (for bounded-degree unit disk graphs).

Lemma 4.4: Total cost of the steps within circle C_i is $O(r_{C_i}^2)$.

Proof: They use Lemma 4.3. to bound the steps to traverse each face. Then they sum over the number of faces that are traversed in C_i, and break up the sum in terms of the number of edges that are contained within C_i. They add another sum over all edges for the greedy steps (each edge can be traversed at most once in greedy mode). These bounds in terms of edges can be recast in terms of number of vertices, since the graph is planar (use Euler formula). This all gives something proportional to the number of vertices within C_i. Which, by Lemma 4.1, can be bounded in terms of the area, or r2. QED Lemma 4.4

Theorem 4.5: Puts it all together, by summing over all i. They claim that since GOAFR only enlarges the bounding circle if it does not contain a path from s to t, and according to GOAFR's radius update policy with the constant factor ρ, the maximum radius that can be reached is $r_{max} < \rho c_l(p^*)$. The sum of all areas of all circles created gives an upper bound on the path that will be followed. Total area is:

$$\sum_{i=1}^{k} \pi \left[r_{max} \left(\frac{1}{\rho} \right)^i \right]^2 = c\pi r_{max}^2 \quad (7)$$

$$< c\pi(\rho c_l(p^*)) \quad (8)$$

$$\in O(c_l(p^*)^2) \quad (9)$$

As the following figure shows, let p^* be the optimal path between the source being the node on the opposite side of w and destination being the center of circle. Then the cost of any geometric algorithm is $\Omega((p^*)^2)$. See Kuhn, Wattenhofer, Zollinger, "Asymptotically Optimal Geometric Mobile AdHoc Routing", DialM 02, September 28, 2002, Atlanta, for more information.

3.4.3 Average-case efficiency

Simulated, it works well In terms of path cost. Early fallback seems to be the key to getting the good average-case results. On the other hand, the worst-case optimal bound results from the use of the bounding circles.

3.5 Cost metrics

Result 1: All possible cost metrics are equivalent up to constant factors on bounded-degree unit disk graphs.

Result 2: For general unit disk graphs (not nec. bounded degree), there are two kinds of cost functions: Linearly bounded metrics: Hop count, Euclidean distance Here, they show that the algorithm and optimality result can be extended to arbitrary unit disk graphs, using a backbone construction. Superlinear: Energy; but in practice, energy cost never drops below a basic minimum, so this could be modeled differently, and would also be linear. Here, they show a lower bound graph, which proves that there is no geometric routing algorithm whose cost is bounded w.r.t. the shortest path!

3.5.1 Bounded degree unit disk graphs

Lemma 5.1: On bounded-degree unit disk graphs, for every pair of cost metrics c1 and c1, there is a linear function f s.t., for every path, $c_1(p) \leq f(c_2(p))$.

Proof: Very simple. They relate an arbitrary cost metric to the Euclidean distance metric. This involves showing inequalities in 2 directions: (1) c(p) is bounded by Euclidean distance (2) Euclidean distance bounded by c(p).

For (1), they use the k bound to say that, k+1 consecutive hops of p must traverse Euclidean distance at least 1 (remember, no circles!). So, $c_l(p)$, the number of hops in p, is $\leq (k+1)(c_d(p))$, where $c_d(p)$ is the Euclidean distance of p. Now, the cost metric c(p) is upper-bounded by c(1) times the number of hops in p, so this yields an upper bound of c(p) in terms of $c_d(p)$, as needed.

For (2), they argue similarly: Of any portion of p of length at least 1, there must be some edge with length at least 1/(1+k), which would have to be of c-cost at least c(1/(k+1)). Thus, c(p) is geq c(1/(k+1)) times the number of such disjoint paths. But there are a lot of such disjoint subpaths—they can divide up p to produce a number comparable to $c_d(p)$. Multiplying c(1/(k+1)) by $c_d(p)$ yields the lower bound for c(p). QED Lemma 5.1

As a nice corollary, they give Lemma 5.2, which relates the costs of optimal paths between any s and t by linear functions.

3.5.2 General unit disk graphs

Classify cost metrics: Linear: for some $m > 0, c(d) \geq md$, for every d in (0,1] Super-linear: no such m.

1. Linearly bounded cost functions

For an arbitrary unit disk graph G, they first compute a Clustered Backbone Graph:

They start with the backbone graph itself, G': G' is a subgraph of G that: G' is a bounded-degree unit disk graph. The nodes of G' form a connected dominating set of G (every node of G has at least one neighbor in the set of nodes of G').

They use existing techniques to get G'. Then they form the Clustered Backbone Graph by adding in all the other nodes of G, each with one edge to a dominator to which it is connected in G. The CBG needn't be of bounded degree, because the "clusters" associated with the various backbone nodes can consist of any number of nodes.

Lemma 5.3: The number of hops in the CBG's best path between any two nodes, is O(number hops in a best path in G).

Lemma 5.3 talks about the link distance metric, but they claim, in Lemma 5.4, that the result can be extended to any LINEARLY BOUNDED cost metric c.

Proof: The proof proceeds by relating c both ways to c_d, the Euclidean distance metric. One direction, expressed in (5), doesn't need any new assumptions, and the reasoning seems to be similar to before. This says that the c metric is bounded above by c_d.

On the other hand, the final thing they show is that the c metric bounded from below by c_d. Here is where they need the new linear boundedness assumption. QED Lemma 5.4

Finally they describe their complete algorithm for general unit disk graphs, with linear-bounded cost functions: Determine the CBG. Then apply the Gabriel Graph construction to get a planar subgraph of the CBG. They have to argue, at this point, that the resulting graph still has optimal cost functions.

Then apply GOAFR+ to route on the Gabriel Graph—except, at each point, check whether we are yet at a neighbor of the target t. If so, just send the message directly to t.

2. Super-linear cost functions

Finally, they give an example that shows why super-linear cost functions c (which can associate large costs with very short distances), make it impossible to bound the cost of the path generated by ANY geometric routing algorithm in terms of the optimal path. Their claim applies to randomized and deterministic algorithms!

Actually, I thought they were going to show this for an arbitrary super-linear cost function c, but it seem (in paragraph 2 of the proof) that they are fixing c to satisfy certain properties, like c(d) = 1/n for pre-chosen d and n. So the result seems to just say that THERE EXISTS some super-linear c...not something about every super-linear c.

Anyway, the construction looks very simple and plausible. The shortest path from s to t is clearly going to be the only (non-looping) one—the one that goes through w and w'. But any algorithm will have to "guess" which node to try—if the actual node w is chosen adversarily, or randomly, then we can show that an algorithm can be fooled into exploring many side-branches, which can be costly, if c is defined that way.