A Virtual Spanner for Efficient Face Routing in Multihop Wireless Networks*

Héctor Tejeda[1], Edgar Chávez[1], Juan A. Sanchez, and Pedro M. Ruiz[2]

[1] Escuela de Ciencias Físico-Matemáticas, Universidad Michoacana, México.
{elchavez,htejeda}@fismat.umich.mx
[2] Facultad de Informática, University of Murcia, Spain
{jlaguna,pedrom}@dif.um.es.

Abstract Geographic routing for ad hoc and sensor networks has gained a lot of momentum during the last few years. In this scheme routes are created locally by each individual node, just based on the position of the destination and its local neighbors. To do that, a node selects its best neighbor (according to some metric) out of those being closer than itself to the destination. This operation is called *greedy mode*. When a node has no such neighbors, it enters into *face routing* mode. However, for face routing to work properly, the underlying graph needs to be planarized by removing crossing edges, which may eventually be very good from the routing metric point-of-view. In this paper, we introduce a new localized scheme to build a planar *virtual spanner* in a simple and efficient way, with low control overhead. The produced *virtual spanner* allows *face routing* to be executed, without the need to remove any of the original links in the network. Thus, the best links according to the routing metric can still be used, Our simulation results show that by performing face routing over the virtual spanner, we manage to enhance the routing performance both for greedy-face-greedy routing and face routing between a 40 to 60% compared to existing planarity tests.

1 Introduction

Mobile ad hoc networks (often referred to as MANETs) as well as wireless sensor networks consist of wireless nodes that communicate with each other in the absence of a fixed infrastructure. When a node needs to send a message to another host which is outside of its radio range, it uses other intermediate hosts as relay nodes. Those intermediate nodes are dynamically selected by the routing protocol being used. This kind of networks are useful in many scenarios such as disaster relief, battlefield environments, etc.

Among all routing protocols for these networks, geographic routing [BMSU01] has emerged recently as a very efficient way to provide guaranteed delivery routes without flooding the whole network with control messages. However, nodes are required to be able to know their position and, by exchanging control messages,

* Partially supported by CONACyT and the Spanish MEC by means of the "Ramon y Cajal" program and the SAVIA project (CIT-410000-2005-1).

the position of its neighbors. To send a message from the source to the destination, each intermediate node selects locally its *best* neighbor to forward the message towards that destination among those which are closer than itself. Those nodes are often said to provide advance towards the destination. The best node depends on the routing metric. For instance, if we are using hop count as the routing metric, it could be the one which is closest to the destination. This operation is called *greedy mode*. When greedy mode reaches a local minimum (i.e. no neighbor can provide advance towards the destination) then the protocol needs to resort to a recovery mechanism until a node is found which can continue greedy forwarding. This mechanism is *face routing* described in [BMSU01]. The basic idea is that when no progress can be made in greedy mode, packets are sent following the edges of the faces of a planar decomposition of the underlying graph, until greedy mode can again continue, or the destination is eventually reached. This approach combining greedy and face modes when necessary, is commonly known as GFG (Greedy-Face-Greedy) routing [BMSU01]. As we said, the *face routing* part requires the underlying graph to be planar.

There are several methods to extract a planar subgraph from a given Unit Disk Graph (UDG), which models the entire network. A UDG is a graph in which an edge $[u, v]$ exists only if $dist(u, v) \leq r$ being r the radio range. The *Relative Neighborhood graph*, RNG [Tou80] is obtained by applying the RNG test to every edge of the UDG: an edge $[u, v]$ is retained in $RNG(G)$ if there is no vertex z such that $\max\{d_G(u, z), d_G(v, z)\} < d_G(u, v)$. That is, if there is no vertex in the intersection of their disks. The *Gabriel graph*, GG [GS69], applies a slightly different test to every edge of the graph. It retains an edge $[u, v]$ in $GG(G)$ if there is no node in the disk with diameter $\overline{uv}$. Finally, the *Morelia test* [BCG+04] manages to preserve some long edges by using a stronger condition for the removal of edges. An edge $[u, v]$ is not included in $MG(G)$ if there is a couple of points $[x, y]$ so that one of them (or both) is in the disk with diameter $\overline{uv}$ and $[x, y]$ crosses $[u, v]$. Given a UDG G we have $RNG(G) \subseteq GG(G) \subseteq MG(G)$.

The guaranteed delivery provided by *face routing* has a price, which is that the computed routes are generally not optimal. The main reason is that traversing faces to avoid voids, may eventually produce a large deviation from shortest path. Another important reason is that the elimination of links to avoid crossings may degrade the routing performance when the protocol enters into face routing mode. As a matter of fact, long links (which are the ones preferred to reduce hop count) are the ones which are usually eliminated first, because they usually cross many other links.

To mitigate this problem, we propose the creation of a planar virtual spanner of the original graph using a tessellation. Given that crossing edges are forbidden in *face routing* to guarantee correctness of the algorithm, we build our virtual spanner in such a way that guarantees its planarity (there are no crossing virtual edges). Then, when a node enters into *face mode*, it will route using virtual edges, which will then be translated to a path using real nodes. Once the next hop virtual neighbor is selected using face routing, the real nodes will route

the message towards the representative of the selected neighboring tessel. Given that real nodes will route using all available links (no links are eliminated), the performance in face mode of the protocol is enhanced. We shall show this in our simulation results.

The remainder of the paper is organized as follows: Section 2 presents our network model and the problem formulation. Section 3 illustrates how the virtual spanner is built. We explain how to route based on the virtual spanner in section 4. Finally we present some simulation results in section 5 and give some conclusions and future work in section 6.

2 Network Model and Problem Formulation

This section introduces the notation and the model we use throughout the paper. We consider routing algorithms on Euclidean graphs, i.e. weighted graphs where edge weights represent Euclidean distances between the adjacent nodes in a particular embedding in the plane. As usual, a graph G is defined as a pair $G := (V, E)$ where V denotes the set of vertices and $E \subseteq V^2$ denotes the set of edges. The number of nodes is denoted by $n := |V|$ and the Euclidean length of an edge $e \in E$ is denoted by $c_{\mathrm{d}}(e)$. A path $p := v_1, \ldots, v_k$ with each $v_i \in V$ is a list of nodes such that two consecutive nodes are adjacent in G, i.e. $(v_i, v_{i+1} \in E)$. A path p also can be denoted by the corresponding list of edges. In our evaluations we will use the traditional hop count metric. Thus, given a path $p = v_1, \ldots v_k$ the cost of such path is the number of edges traversed.

In this paper we consider the standard UDG model for ad-hoc networks where all nodes have the same transmission range (r). Thus, given two nodes $v_1, v_2 \in V$, the edge $[v_1, v_2] \in E \Leftrightarrow c_{mathrmd}([v_1, v_2]) \leq r$.

As in previous geographic routing works in the literature, we assume that nodes know their positions and those of their neighbors. It is also assumed that sources of data packets know the position of the destination.

3 The Virtual Spanner

We divide the plane in regions with a regular tessellation, which is a tessellation (or planar subdivision) made up of congruent regular polygons. The idea is that an entire region may be represented by a single virtual point, the center of the regular polygon. If we link the centers of the polygons we observe a peculiar behavior: the centers define a dual tessellation that is also planar. The dual of a triangle tessellation is a hexagonal tessellation, while a square tessellation is auto dual.

Only three regular polygons tessellate the Euclidean plane: triangles, squares or hexagons, from elementary geometry. They are depicted in figure 2.

The virtual node for a polygon is chosen as the centroid of the polygon. Two virtual nodes will share an edge if in their respective cells two real nodes are neighbors. Thus, we need to choose a suitable polygon size for building the

virtual graph, so that we achieve a good trade-off between the simplicity to build the virtual graph (guaranteeing that is planar), and the number of cells to be checked in its creation process. We have analyzed three options as we show in figure 1.

a) The transmission radius does not cover all the cell.
b) Any two points in the cell are within radio range.
c) A node in one cell can reach any other node in a neighboring cell.

Case a) complicates the design because it may require multihop routing within a cell. In case c) there may be a very big number of cells in which to look for possible virtual edges. We decided to use case b) because it is the configuration which avoids multihop within a cell in which the number of cells to look for virtual neighbors is low.

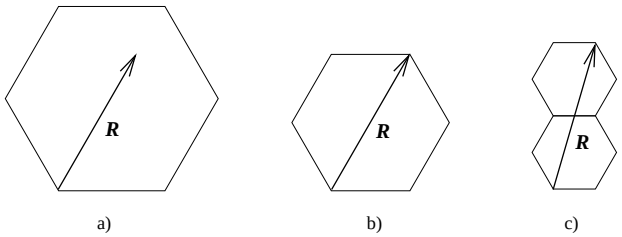
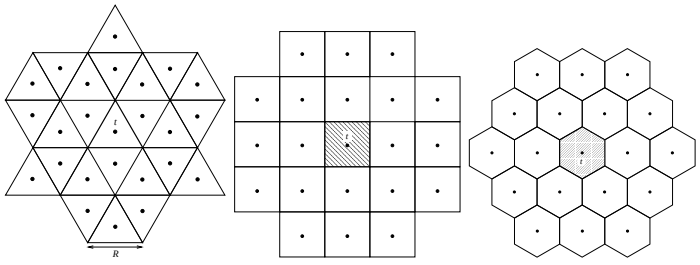


Figure 1: Variation of polygon size with transmission radius fixed.

If the graph is dense enough, there will be at least one node in each cell. Thus, each virtual node will be connected to all neighboring virtual nodes. The resulting virtual graph is exactly the dual of the graph, which is planar. In real situations we cannot guarantee that every cell will have a node. Thus, to preserve connectivity we must find all possible virtual neighbors. They may be in cells which are not contiguous to the current one. In figure 2 we show for each different tessellation (triangular, square and hexagonal) the possible cells that may contain nodes which are neighbors of nodes in the current cell t. The cell t can reach more cells when using a triangular configuration (24 cells). With a square configuration 20 cells are candidates and when using hexagonal cells only 18 cells. Please note that the dual of the virtual graph *may not be planar* if we have void cells and want to preserve connectivity. This crossings can be eliminated using a local test, and the complexity of the test depends on the number of neighboring cells.

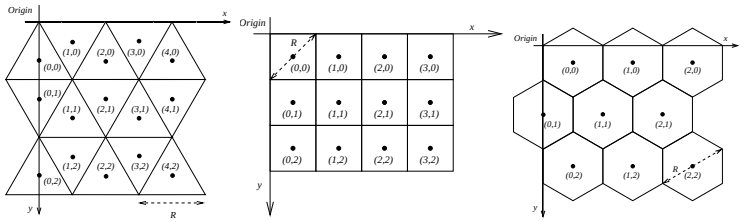
The grid with triangles, squares or hexagons is located arbitrarily in the plane. Each cell is identified by a coordinate pair as is showed in Figure 3. Note that real nodes only need to know the type of tessellation and the communication radius at deployment time. Based on that, and given their current position they can easily compute the coordinates of their centroid. In addition, only with local



(a) Triangles (b) Squares (c) Hexagons

Figure 2: Regular tessellations in the plane and cell centroids. Additionally, the cells shown are those reachable from t using a radius R equal to the diameter of the cell

information about the position of its neighbors they can compute their local view of the virtual graph (virtual edges). This has no additional overhead because position of real neighbors is already known or was computed using beacons.



(a) Triangles (b) Squares (c) Hexagons

Figure 3: Fixing the origin, the virtual coordinates are computed with elementary calculations

We show below some elementary calculations for the node to compute the coordinates of the virtual node for each real node. It only uses the transmission radius R and its position (x, y).

In addition, each real node also needs to compute the virtual edges shared with reachable cells. Every real node can make exactly the same calculations independtly without the need of a central authority or coordination among them. This connectivity test is accomplished in two stages:

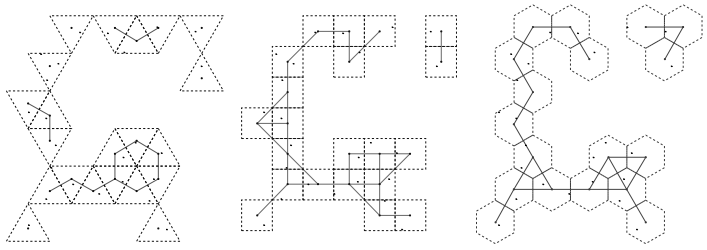
1. Test surrounding cells that are neighbors by their side.
2. Test all other cells that are reachable from current cell but are not neighbors by their side.

Type of tilling	Position of centroid	Tessel
Hexagonal	$y_c \leftarrow 3R(y+1/3)/4$ **if** $y \bmod 2 = 0$ **then** $x_c \leftarrow \sqrt{3}R(x+1/2)/2$ **else** $x_c \leftarrow \sqrt{3}Rx/2$	$y \leftarrow$**truncate** $(3y_n/4/R)$ **if** $y \bmod 2 = 0$ **then** $x \leftarrow$**truncate** $(2x_n/\sqrt{3}/R)$ **else** $x \leftarrow$**truncate** $((2x_n + \sqrt{3}R/2)/\sqrt{3}/R)$
Triangular	$x_c \leftarrow Rx/2$ **if** $(x+y) \bmod 2 = 0$ **then** $y_c \leftarrow \sqrt{3}R(y+2/3)/2$ **else** $y_c \leftarrow \sqrt{3}R(y+1/3)/2$	$x \leftarrow$**truncate** $(x_n/2/R)$ $y \leftarrow$**truncate** $(2y_n/\sqrt{3}/R)$
Square	$x_c \leftarrow (x+1/2)R/\sqrt{2}$ $y_c \leftarrow (y+1/2)R/\sqrt{2}$	$x \leftarrow$**truncate** $(\sqrt{2}x_n/R)$ $y \leftarrow$**truncate** $(\sqrt{2}y_n/R)$

Table 1: Formulas for a node to compute position of its centroid and its tessel

The first stage is easier than the second one because it always produces a planar graph. There are no edge crossings, as it is depicted in figure 4. In the first stage, a virtual edge is added between centroids of two cells adjacent by the side if there are two mutually reachable real nodes, one in each of those cells. Unfortunately, the virtual graph produced after the first stage may not be connected. Thus, we need to apply the second stage to obtain a connected graph without crossings of virtual edges.

For the second part, we start testing if we can add a virtual edge to the centroid of those cells (see figure 2) which are second degree neighbors (side neighbors of our side neighbors). If for one of those, we cannot add the virtual edge (i.e. there is no other real node in that cell directly reachable from any real node in current cell) then we try again with those cells being neighbors by side of this particular cell we couldn't find nodes to add the virtual edge. This condition guarantees that the resulting virtual graph will be planar. Figure 5 shows the resulting virtual graph after both stages.



(a) Triangles (b) Squares (c) Hexagons

Figure 4: First connectivity test. Natural neighbors

As an example, we give the concrete algorithms used in each stage to add edges to the virtual spanner with an hexagonal tilling. The algorithms for squares and triangles are similar and are not included in here due to space limitations.

The algorithm for the first connectivity test using hexagons is given in algorithm 1.

Algorithm 1 Algorithm for the first stage with hexagons

```
1: procedure ReviewHexagonsStage1(I, t)          ▷ I are the neighbor cells
2:     k ← 0                          ▷ by their side to t, they are enumerated from 0 to 5
3:     while k < 6 do
4:         if isThereEdge(t, I_k) then
5:             addEdge(t, I_k)
6:         end if
7:         k ← k + 1
8:     end while
9: end procedure
```

For the second stage with an hexagonal tilling, all reachable cells which are not side neighbors of the current cell (t) are tested. The test needs to take into account existing virtual links which have been added before, to avoid creating a non-planar virtual spanner. The detailed algorithm is given in 2.

As we stated, the goal of this virtual graph is enhancing the performance of face routing. Thus, we will explain in the next section how real nodes do face routing using the virtual graph, whereas we show the performance enhancements achieved later on.

Algorithm 2 Algorithm for the second stage with hexagons

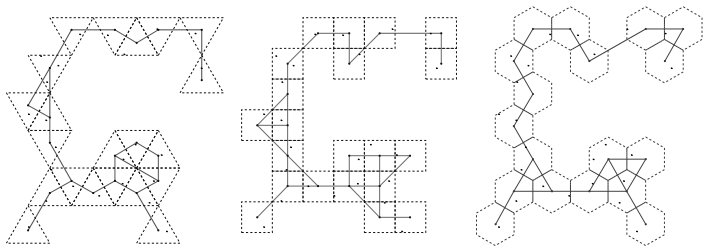
```
1: procedure ReviewHexagonsStage2(I, E, t)          ▷ E are the rest of the cells
2:     k ← 0                                        ▷ reachable by t, enumerated from 0 to 11
3:     while k < 6 do                               ▷ Review for the odd cells from E
4:         a ← 2k
5:         b0 ←!isThereEdge(t, I_k)
6:         b1 ←!isThereEdge(I_k, E_a)
7:         b2 ←isThereEdge(t, E_k)
8:         if b0 AND b1 AND b2 then
9:             addEdge(t, E_k)
10:        end if
11:        k ← k + 1
12:    end while
13:
14:    k ← 0
15:    while k < 6 do                               ▷ Review for the even cells from E
16:        a ← (k + 1) mod 6
17:        b ← (2k + 1) mod 6
18:        b0 ←isThereEdge(t, I_k)
19:        b1 ←isThereEdge(I_k, E_b)
20:        b2 ←isThereEdge(t, I_a)
21:        b3 ←isThereEdge(I_a, E_b)
22:        b4 ←isThereEdge(t, E_b)
23:        if !(b0 AND b1) AND !(b2 AND b3) AND b4 then
24:            addEdge(t, E_b)
25:        end if
26:        k ← k + 1
27:    end while
28: end procedure
```

4 Routing with the virtual graph

When the protocol enters into face mode, we plan to perform face routing based on the virtual spanner. However, only real nodes can process messages. Thus, we need to understand the two points of view of our proposed scheme. On a high level view we use the virtual nodes whenever a planar graph is needed to forward a message using face routing. In the low level view we always use a real node, which needs to send a message towards another real node, based on its relation with the intended virtual node. We explain how this works based on the Face Routing (FR) algorithm [KSU99]. However, any geographic routing algorithm making use of face routing (i.e. relaying in a planar spanner) can be used as well. For instance, in our experiments we use the GFG variant [BMSU01].

A brief description of the proposed algorithm is presented below. At each step of the algorithm the node currently trying to send the packet to the next neighbor in face mode performs the following operations:

1. Based solely on its coordinates, the node finds out his cell and corresponding virtual node.
2. Using the information from neighbors (obtained by any geographic routing protocol using periodic beacons), the node finds which virtual edges exist according to the procedures explained in the previous section. As we explained before, a virtual edge can only exist to a virtual node if there is a real neighboring node in the corresponding cell.
3. In *face mode* the current real node routing in face mode will use the virtual graph to select (according to face routing) the proper virtual edge to follow. Once it is selected, it uniquely defines the cell that needs to be reached using *real nodes*. The node then sends the packet to any real node in the next cell based on some metric. For instance in our simulations we send the packet to the real node which is more distant to the current real node. If the selected cell is not directly reachable, the real node will greedily hand the packet to another node within the same cell, to reach the target cell.
4. Once a real node in the destination cell (the next cell in the path) received the packet it will forward the packet by repeating the process. Inside a cell the packets can be forwarded greedily because all nodes in a cell a mutually reachable.



(a) Triangles (b) Squares (c) Rectangles

Figure 5: Second test. Reachable neighbors

The steps above can be used for traversing the face as is depicted in Figure 6. The source node and the target are labeled with 22 and 24 respectively. The source node is in virtual cell $(4, 5)$. Virtual edges exists between $(4, 5)$ and $(3, 5), (3, 4), (4, 4), (4, 6)$. Using the left hand rule node 22 forwards the packet to cell $(3, 4)$ selecting an arbitrary node in such cell. The sequence of virtual nodes, and the sequence of real nodes are depicted in Figure 6.

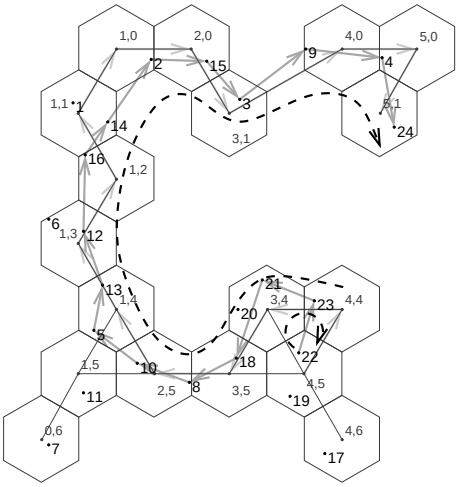


Figure 6: Face routing in virtual hexagonal graph.

5 Experimental Results

In this section we present simulation results to assess the performance of GFG and face routing when run over the different variants of the Virtual Graph. We compare the results with those of the Relative Neighborhood Graph, the Gabriel Graph and the Morelia Graph. We also present real shortest paths computed using Dijkstra's algorithm. Of course, the shortest path cannot be computed using only local information, but provides a good indication of the overall performance of the different proposals.

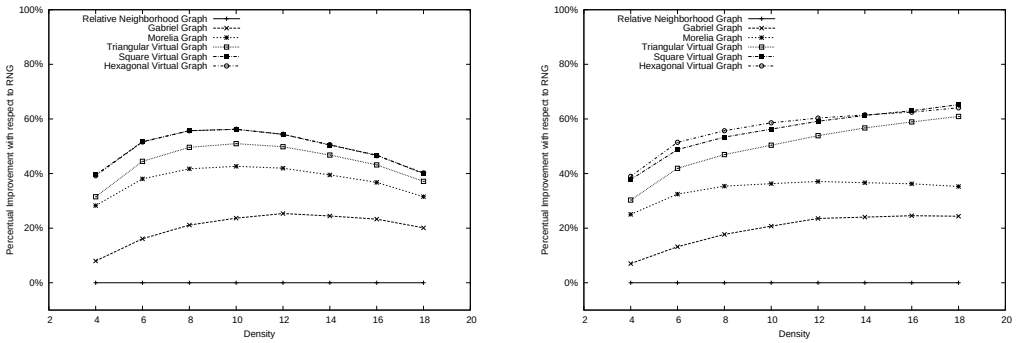
When the protocol enters in face mode, there may be several metrics to decide to which real node within the next cell to send the packet to. In our simulations, we use the euclidean distance, because it is the most common metric used by geographic routing protocols to select neighbors in greedy mode.

5.1 Simulation setup

We used *connected* random unit disk graphs for our simulations. We test our spanners with different densities, from 4 to 18 with increments of 2. Each one of those densities corresponds to a mean number of neighbors. For each density we used 1000 nodes, which were placed randomly in the simulation area. For each scenario we generated randomly 100 different graphs, so we have obtained 800 graphs for simulation. The size of the simulation area was adapted to preserve the density of the network. Finally, for every graph, we select 1000 different (source,destination) pairs. Thus, each point in the graph represents the average over 100000 routing tasks.

5.2 Simulation results

We present in this subsection the results of our simulations for different densities of the graphs.



(a) Greedy-Face-Greedy Routing

(b) Face Routing

Figure 7: Efficiency of the Virtual Spanner against three standard spanner test

Figure 7(a) shows the percentual improvement in terms of the mean number of hops required to route from source to destination for different network densities. As we can see in the graph, the higher the density, the better the performance that the proposed protocol achieves, up to a mean density of 10. The reason is that for those mean densities the amount of routing performed in face mode is high. Thus, our proposed virtual spanners allow for a significant reduction in the hop count. The reason being that the virtual spanner manages to use long edges, while traditional plannarization tests (i.e. GG, RNG and MG) remove them. So, the higher the density the more options has the virtual spanner to select best edges. In addition, the increase in density makes traditional tests to remove more (eventually long) edges. As the mean network density goes beyond 10 neighbors per node, we see that our proposed schemes still outperform traditional tests, although the percentual improvement compared to GG and RNG and MG is is reduced. The reason for that reduction is that for those high densities most of the routing is peformed in greedy mode, thus there is no big difference between approaches. In addition, by having a fixed number of nodes and increasing density means that the overall length of the paths is reduced as density increases. That also affects the reduction in the percentual performance benefit. But, in any case for any density our proposed schemes outperform traditional schemes. For instance, our hexagonal tilling obtains a 40 to 57% improvement compared to RNG for all the ranges of density. The square tilling obtains basically the same results, whereas triangular one has a little bit lower performance, outperforming all of them traditional planarity tests.

To assess the real benefit of the virtual spanner, we performed the same experiments but using only face routing to go from source to destination. As we see in figure 7(b), again a lower density produces longer paths. As before, the reason is that paths become longer because the simulation area is enlarged to accommodate such nodes maintaining the mean density. Figure 7(b) shows that our proposed schemes outperform all other approaches for all densities. In addition, we can see that in this case the gain is higher than with GFG because in this experiment all the routing has been done in face mode regardless of the density of the network.

6 Conclusions and future work

We have shown that with the application of the Virtual Graph for representing the underlying structure of a wireless ad-hoc network we can achieve face routing with a fewer number of hops, outperforming in all cases existing techniques (Relative neighborhood graph, Gabriel graph and Morelia graph).

The proposed virtual spanner can be built locally by nodes based solely on local information about neighbors. Thus, it can be perfectly integrated with any geographic routing protocol such as GFG, face routing, etc. Our proposed virtual spanner based on hexagons manages to reduce by a 40 to 60% the number of hops required to route a message from source to destination both for GFG and face routing protocols. This scheme can be integrated with any geographic routing protocol, and can help at improving the performance of such protocols.

For future work, we are working on the use of different routing metrics which may allow the virtual spanner to improve not only the number of hops but energy consumption and quality of the selected paths.

References

[BCG+04] P. Boone, E. Chavez, L. Gleitzky, E. Kranakis, J. Opartny, G. Salazar, and J. Urrutia. Morelia test: Improving the efficiency of the gabriel test and face routing in ad-hoc networks. *Lecture Notes in Computer Science*, 3104:23–24, January 2004.

[BMSU01] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with guaranteed delivery in ad-hoc wireless networks. *ACM/Kluwer Wireless Networks*, 7(6):609–616, 2001.

[GS69] K. Gabriel and R. Sokal. A new statistical approach to geographic variation analysis. *Systematic Zoology*, 18:259–278, 1969.

[KSU99] E. Kranakis, H. Singh, and J. Urrutia. Compass routing on geometric networks. In *Proc. 11th Canadian Conference on Computational Geometry*, pages 51–54, Vancouver, August 1999.

[Tou80] G. Toussaint. The relative neighbourhood graph of a finite planar set. *Pattern Recognition*, 12(4):261–268, 1980.