Compact and Low Delay Routing Labeling Scheme for Unit Disk Graphs

Chenyu Yan, Yang Xiang, and Feodor F. Dragan

Algorithmic Research Laboratory, Department of Computer Science
Kent State University, Kent, Ohio, U.S.A.
{cyan,yxiang,dragan}@cs.kent.edu

Abstract. In this paper, we propose a *new compact and low delay routing labeling scheme* for *Unit Disk Graphs (UDGs)* which often model wireless ad hoc networks. We show that one can assign each vertex of an n-vertex UDG G a compact $O(\log^2 n)$-bit label such that, given the label of a source vertex and the label of a destination, it is possible to compute efficiently, based solely on these two labels, a neighbor of the source vertex that heads in the direction of the destination. We prove that this *routing labeling scheme* has a constant *hop route-stretch* (= *hop delay*), i.e., for each two vertices x and y of G, it produces a routing path with $h(x, y)$ hops (edges) such that $h(x, y) \leq 3 \cdot d_G(x, y) + 12$, where $d_G(x, y)$ is the hop distance between x and y in G. To the best of our knowledge, this is the first compact routing scheme for UDGs which not only guaranties delivery but has a low hop delay and polylog label size. Furthermore, our routing labeling scheme has a constant length route-stretch.

1 Introduction

A common assumption for wireless ad hoc networks is that all nodes have the same maximum transmission range. By proper scaling, one can model these networks with *Unit Disk Graphs* (*UDGs*), which are defined as the intersection graphs of equal sized circles in the plane [3]. In other words, there is an edge between two vertices in an UDG if and only if their Euclidean distance is no more than one.

Communications in networks are performed using *routing schemes*, i.e., mechanisms that can deliver packets of information from any vertex of a network to any other vertex. In most strategies, each vertex v of a graph has full knowledge of its neighborhood and uses a piece of global information available to it about the graph topology – some "sense of direction" to each destination – stored locally at v. Based only on this information and the address of a destination vertex, vertex v needs to decide whether the packet has reached its destination, and if not, to which neighbor of v to forward the packet. The *efficiency* of a routing scheme is measured in terms of its *multiplicative route-stretch* (or *additive route-stretch*), namely, the maximum ratio (or surplus) between the cost (which could be the *hop-count* or the *length* of a route, produced by the scheme for a pair of vertices, and the cost of an optimal route available in graph for that

F. Dehne et al. (Eds.): WADS 2009, LNCS 5664, pp. 566–577, 2009.

pair. Here, the *hop-count* of a route is defined as the number of edges on it and the *length* of a route is defined as the sum of the Euclidean length of its edges. Using different cost functions, for a given graph G and a given routing scheme on G, one can define two different notions of route-stretch: *hop route-stretch* and *length route-stretch*.

The most popular strategy in wireless networks is the *geographic routing* (sometimes called also the *greedy geographic routing*), where each vertex forwards the packet to the neighbor geographically closest to the destination (see survey [12] for this and many other strategies). Each vertex of the network knows its position (e.g., Euclidean coordinates) in the underlying physical space and forwards messages according to the coordinates of the destination and the coordinates of neighbors. Although this greedy method is effective in many cases, packets may get routed to where no neighbor is closer to the destination than the current vertex. Many recovery schemes have been proposed to route around such voids for guaranteed packet delivery as long as a path exists [4,14,16]. These techniques typically exploit planar subgraphs (e.g., Gabriel graph, Relative Neighborhood graph), and packets traverse faces on such graphs using the well-known right-hand rule. Although these techniques guarantee packet delivery, none of them give any guaranties on how the routing path traveled is "close" to an optimal path; the worst-case route-stretch can be linear in the network size.

All earlier papers assumed that vertices are aware of their physical location, an assumption which is often violated in practice for various of reasons (see [7,15,21]). In addition, implementations of recovery schemes are either based on non-rigorous heuristics or on non-trivial planarization procedures. To overcome these shortcomings, recent papers [7,15,21] propose routing algorithms which assign virtual coordinates to vertices in a metric space X and forward messages using geographic routing in X. In [21], the metric space is the Euclidean plane, and virtual coordinates are assigned using a distributed version of Tutte's "rubber band" algorithm for finding convex embeddings of graphs. In [7], the graph is embedded in R^d for some value of d much smaller than the network size, by identifying d beacon vertices and representing each vertex by the vector of distances to those beacons. The distance function on R^d used in [7] is a modification of the ℓ_1 norm. Both [7] and [21] provide substantial experimental support for the efficacy of their proposed embedding techniques – both algorithms are successful in finding a route from the source to the destination more than 95% of the time – but neither of them has a provable guarantee. Unlike embeddings of [7] and [21], the embedding of [15] guarantees that the geographic routing will always be successful in finding a route to the destination, if such a route exists. Algorithm of [15] assigns to each vertex of the network a virtual coordinate in the hyperbolic plane, and performs greedy geographic routing with respect to these virtual coordinates. However, although the experimental results of [15] confirm that the greedy hyperbolic embedding yields routes with low route-stretch when applied to typical unit-disk graphs, the worst-case route-stretch is still linear in the network size.

In this paper, we propose a new compact and low delay routing labeling scheme for Unit Disk Graphs. We show that one can assign each vertex of an n-vertex UDG G a compact $O(\log^2 n)$-bit label such that, given the label of a source vertex and the label of a destination, it is possible to compute efficiently, based solely on these two labels, a neighbor of the source vertex that heads in the direction of the destination. We prove that this *routing labeling scheme* has a constant *hop route-stretch* (= *hop delay*), i.e., for each two vertices x and y of G, it produces a routing path with $h(x, y)$ hops such that $h(x, y) \leq 3 \cdot d_G(x, y) + 12$, where $d_G(x, y)$ is the hop distance between x and y in G. To the best of our knowledge, this is the first compact routing scheme for UDGs which not only guaranties delivery but has a low hop delay and polylog label size. Furthermore, our routing labeling scheme has a constant length route-stretch. Note also that, unlike geographic routing or any other strategies discussed in [4,7,12,14,15,16,21], our routing scheme is *degree-independent*. That is, each current vertex makes routing decision based only on its label and the label of destination, does not involve any labels of neighbors. The label assigned to a vertex in our scheme can be interpreted as its virtual coordinates. To assign those labels to vertices, we need to know only the topology of the input unit disk graph and relative Euclidean lengths of its edges.

To obtain our routing scheme, we establish a novel *balanced separator* theorem for UDGs, which mimics the well-known Lipton and Tarjan's planar balanced shortest paths separator theorem. We prove that, in any n-vertex UDG G, one can find two hop-shortest paths $P(s, x)$ and $P(s, y)$ such that the removal of the 3-hop-neighborhood of these paths (i.e., $N^3_G[P(s, x) \cup P(s, y)]$) from G leaves no connected component with more than $2/3n$ vertices. The famous Lipton and Tarjan's planar balanced separator theorem has two variants (see [19]). One variant (called *planar balanced* $\sqrt{n}$*-separator theorem*) states that any n-vertex planar graph G has a set S of vertices such that $|S| = O(\sqrt{n})$ and the removal of S from G leaves no connected component with more than $2/3n$ vertices. Another variant (called *planar balanced shortest-paths separator theorem*) states that any n-vertex planar graph G has two shortest paths removal of which from G leaves no connected component with more than $2/3n$ vertices. Although the first variant of the planar balanced separator theorem has an extension to the class of disk graphs (which includes UDGs) (see [1]), the second variant of the theorem proved to be more useful in designing compact routing (and distance) labeling schemes for planar graphs (see [13,22]). To the date, there was not known any extension of the planar balanced shortest-paths separator theorem to unit disk graphs. The paper [11] notes that *"Unfortunately, Thorup's algorithm uses balanced shortest-path separators in planar graphs which do not obviously extend to the unit-disk graphs"* and uses the well-separated pair decomposition to get fast approximate distance computations in UDGs. We do not know how to use the well-separated pair decomposition of an UDG G to design a compact and low delay routing labeling scheme for G. Application of the balanced $\sqrt{\cdot}$-separator theorem of [1] to UDGs can result only in routing (and distance) labeling schemes with labels of size no less than $O(\sqrt{n} \log n)$-bits per vertex. Our separator theorem allows

us to get $O(\log^2 n)$-bit labels which is more suitable for the wireless ad hoc and sensor networks where the issues of memory size and power-conservation are critical.

Our new *balanced shortest-paths—3-hop-neighborhood separator* theorem allows us to build, for any n-vertex UDG $G = (V, E)$, a system $\mathcal{T}(G)$ of at most $2\log_{\frac{3}{2}} n + 2$ spanning trees of G such that, for any two vertices x and y of G, there exists a tree T in $\mathcal{T}(G)$ with $d_T(x, y) \leq 3 \cdot d_G(x, y) + 12$. That is, the distances in any UDG can be approximately represented by the distances in at most $2\log_{\frac{3}{2}} n + 2$ of its spanning trees. An earlier version of these results has appeared in [24] (see Section 3.4 and pages 124 and 125 of Section 3.5.5). Taking the union of all these spanning trees of G, we obtain a *hop* $(3, 12)$*-spanner* H of G (i.e., a spanning subgraph H of G with $d_H(x, y) \leq 3 \cdot d_G(x, y) + 12$ for any $x, y \in V$) with at most $O(n \log n)$ edges. There is a number of papers describing different types of *length-spanners* and *hop-spanners* for UDGs (see [2,8,10,17,18] and literature cited therein). Many of those spanners have nice properties of being planar or sparse, or having bounded maximum degree or bounded length (or hop) *spanner-stretch*, or having localized construction. Unfortunately, neither of those papers develops or discusses any routing schemes which could translate the constant spanner-stretch bounds into some constant route-stretch bounds.

2 Notions and Notations

Let V be a set of $n = |V|$ nodes on the Euclidean plane and let $G = (V, E)$ be the unit disk graph (UDG) induced by those nodes. Let also $m = |E|$. For each edge (a, b) of G, by (a, b) we denote also the open straightline segment representing it, and by $|ab|$ the Euclidean length of the edge/segment (a, b). For simplicity, in what follows, we will assume that any two edges in G can intersect at no more than one point (i.e., no two intersecting edges are on the same straight line), and no three edges intersect at the same point.

For a path P of G, the *hop-count* of P is defined as the number of edges on P and the *length* of P is defined as the sum of the Euclidean length of its edges. For any two vertices x and y of G, we denote: by $d_G(x, y)$, the *hop-distance* (or simply *distance*) in G between x and y, i.e., the minimum hop-count of any path connecting x and y in G; by $l_G(x, y)$, the *length-distance* in G between x and y, i.e., the minimum length of any path connecting x and y in G.

A graph family Γ is said (see [20]) to have an $l(n)$ *bit* (s, r)*-approximate distance labeling scheme* if there is a function L labeling the vertices of each n-vertex graph in Γ with distinct labels of up to $l(n)$ bits, and there exists an algorithm/function f, called *distance decoder*, that given two labels $L(v), L(u)$ of two vertices v, u in a graph G from Γ, computes, in time polynomial in the length of the given labels, a value $f(L(v), L(u))$ such that $d_G(v, u) \leq f(L(v), L(u)) \leq s \cdot d_G(v, u) + r$. Note that the algorithm is not given any additional information, other that the two labels, regarding the graph from which the vertices were taken. Similarly, a family Γ of graphs is said (see [20]) to have an $l(n)$ *bit routing labeling scheme* if there exist a function L, labeling the vertices of each

n-vertex graph in Γ with distinct labels of up to $l(n)$ bits, and an efficient algorithm/function, called the *routing decision* or *routing protocol*, that given the label $L(v)$ of a current vertex v and the label $L(u)$ of the destination vertex u (the header of the packet), decides in time polynomial in the length of the given labels and using only those two labels, whether this packet has already reached its destination, and if not, to which neighbor of v to forward the packet.

Let $\mathcal{R}$ be a routing scheme and $R(x, y)$ be a route (path) produced by $\mathcal{R}$ for a pair of vertices x and y in a graph G. We say that $\mathcal{R}$ has: *hop* (α, β)*-route-stretch* if hop-count of $R(x, y)$ is at most $\alpha \cdot d_G(x, y) + \beta$, for any $x, y \in V$; *length* (α, β)*-route-stretch* if length of $R(x, y)$ is at most $\alpha \cdot l_G(x, y) + \beta$, for any $x, y \in V$.

Let $H = (V, E')$ be a spanning subgraph of a graph $G = (V, E)$. We say that H is: *hop* (α, β)*-spanner* of G if $d_H(x, y) \leq \alpha \cdot d_G(x, y) + \beta$, for any $x, y \in V$; *length* (α, β)*-spanner* of G if $l_H(x, y) \leq \alpha \cdot l_G(x, y) + \beta$, for any $x, y \in V$.

In Section 6, we will need also the notion of collective tree spanners from [6]. It is said that a graph G admits a system of μ collective tree (α, β)-spanners if there is a system $\mathcal{T}(G)$ of at most μ spanning trees of G such that for any two vertices x,y of G a spanning tree $T \in \mathcal{T}(G)$ exists such that $d_T(x, y) \leq \alpha \cdot d_G(x, y) + \beta$.

For a vertex v of G, the kth *neighborhood* of v in G is the set $N_G^k[v] = \{u \in V : d_G(v, u) \leq k\}$. For a vertex v of G, the sets $N_G[v] = N_G^1[v]$ and $N_G(v) = N_G[v] \setminus \{v\}$ are called the *neighborhood* and the *open neighborhood* of v, respectively. For a set $S \subseteq V$, by $N_G^k[S] = \bigcup_{v \in S} N_G^k[v]$ we denote the kth *neighborhood* of S in G.

3 Intersection Lemmas

In this section we present few auxiliary lemmas. From the definition of unit disk graphs, we immediately conclude the following (proofs of these lemmas and all other omitted proofs can be found in the journal version of the paper).

Lemma 1. *In an UDG $G = (V, E)$, if edges $(a, b), (c, d) \in E$ intersect, then G must have at least one of $(a, c), (b, d)$ and at least one of $(a, d), (c, b)$ in E.*

Let r be an arbitrary but fixed vertex of an UDG $G = (V, E)$, and $L_0, L_1, \ldots L_q$ be the *layering* of G with respect to r, where $L_i = \{u \in V : d_G(r, u) = i\}$. For G, using this layering, we construct a *layering tree* T_{orig} rooted at r as follows: each vertex $v \in L_i$ $(i \in \{1, \ldots, q\})$ chooses a neighbor u in L_{i-1} such that $|vu|$ is minimum (closest neighbor in L_{i-1}) to be its father in T_{orig} (breaking ties arbitrarily). Let $E(T_{orig})$ be the edge set of T_{orig}. This tree T_{orig} will help us to construct a balanced separator for G. It will be convenient, for each vertex $v \in V$, by $L(v)$ to denote the layer index of v, i.e., $L(v) = d_G(r, v)$. In what follows, we will also adopt the following agreements (unless otherwise is specified). When we refer to any edge (a, b) of T_{orig}, we assume $L(a) = L(b) - 1$. When we refer to any two intersecting edges (a, b) and (c, d) of T_{orig} (in that order), we assume that $L(a) \leq L(c)$.

Lemma 2. *In T_{orig}, no two edges (a, b) and (c, d) with $L(a) = L(c)$ and $L(b) = L(d)$ can cross.*

Lemma 3. *Let $(a,b),(c,d)$ be two edges in T_{orig} that intersect. If $L(a) = L(b) - 1$, $L(c) = L(d) - 1$ and $L(a) \leq L(c)$, then $L(a) = L(c) - 1$, $(a,d) \notin E$ and $(b,c) \in E$.*

For an UDG $G = (V, E)$, in what follows, by $G_p = (V_p, E_p)$ we denote the planar graph obtained from G by turning each edge intersection point in G into a vertex in G_p. The vertices of T_{orig} (i.e. vertices of G) will be called *real vertices*, to differentiate them from *imaginary* and *null* points that will be defined later. In the following, we will use the term "element" as a general name for real vertices, imaginary points and null points. For any graph $\mathcal{G}$, we will use $E(\mathcal{G})$ to denote the set of its edges and $V(\mathcal{G})$ to denote the set of its vertices (or elements, if $V(\mathcal{G})$ contains imaginary or null points). Below, we will create an imaginary point (details will be given later) at the point where two edges (a,b) and (c,d) from T_{orig} intersect. Recall that we agreed to assume that $L(a) = L(b) - 1$, $L(c) = L(d) - 1$ and $L(a) \leq L(c)$. By Lemma 3, we know that $L(a) = L(c) - 1$. Now, assuming that the imaginary point is m, we define $a(m) = a$, $b(m) = b$, $c(m) = c$ and $d(m) = d$.

4 Balanced Separator for Restricted UDGs

In this section, we consider a special unit disk graph, a simple-crossing UDG. On this simple case, we demonstrate our idea of construction of a balanced separator. It may help the reader to follow the much more complicated case, where we construct a balanced separator for an arbitrary UDG. We define a *simple-crossing UDG* to be an UDG $G = (V, E)$ with each edge crossing at most one other edge.

In what follows, we will transform tree T_{orig} into a special spanning tree T for the planar graph G_p. Let $T = T_{orig}$ initially. For each two intersecting edges (a,b) and (c,d) of T_{orig} (by Lemma 3, we know $L(a) = L(c) - 1$), we do the following. Create a vertex $m_{a,b,c,d}$ at the point where (a,b) and (c,d) intersect. We call $m_{a,b,c,d}$ an *imaginary point*. Remove edges (a,b), (c,d) from T and add vertex $m_{a,b,c,d}$ and edges $(m_{a,b,c,d}, d)$, $(a, m_{a,b,c,d})$ and $(b, m_{a,b,c,d})$ into T. One can see that all the descendants of b and d in T find their way to the root via a.

There are two other kinds of edge intersections in G: the intersection between a tree-edge and a non-tree-edge and the intersection between two non-tree-edges. We handle them separately. First, assume a tree-edge (u,w) intersects a non-tree-edge (s,t). We create a new vertex, called a *null point*, say o, at the point where (u,w) and (s,t) intersect. We remove edge (u,w) from T and add vertex o and edges (u,o), (o,w) into T. Now assume two non-tree-edges (a,b) and (c,d) intersect. We create a new vertex, called a *null point*, say o, at the point where (a,b) and (c,d) intersect. We add vertex o (as a pendant vertex) and edge (a,o) into T.

It is easy to see that T is a spanning tree for the planar graph G_p. We will need the Lipton and Tarjan's planar separator theorem [19] in the following form.

Theorem 1 (Planar Separator Theorem). [19] *Let G be any planar graph with non-negative vertex weights and W be the total weight of G (which is the sum of the weights of its vertices). Let T be any spanning tree of G rooted at a*

vertex r. Then, there exist two vertices x and y in G such that if one removes from G the tree-paths connecting in T r with x and r with y, then each connected component of the resulting graph has total weight at most $2/3W$. Vertices x and y can be found in linear time.

We can apply Theorem 1 to T and G_p by letting the weight of each real vertex be 1 and the weight of each imaginary or null point be 0 in G_p. Then, there must exist in T two paths $P_1 = P_T(r, x)$ and $P_2 = P_T(r, y)$ such that removal of them from G_p leaves no connected component with more than $2/3n$ real vertices.

Using paths $P_1 = (x_0 = r, x_1, \ldots, x_{k-1}, x_k = x)$ and $P_2 = (y_0 = r, y_1, \ldots, y_{l-1}, y_l = y)$ of G_p (of T), we can create a balanced separator for G as follows. (1) Skip all the null points in P_1 and P_2. (2) Skip every imaginary point in P_i which is collinear with its two neighbors in P_i $(i = 1, 2)$. (3) For any imaginary point $m_{a,b,c,d}$ in P_i $(i = 1, 2)$ which is not collinear with its two neighbors in P_i (the only possible case is where $L(a) = L(c) - 1$ and imaginary point $m_{a,b,c,d}$ connects a and d in P_i), replace the subpath $(a, m_{a,b,c,d}, d)$ by either (a, c, d) (if $(a, c) \in E$) or (a, b, d) (if $(b, d) \in E$). By Lemma 1, (a, c) or (b, d) is in E. Let P'_i be the resulting path obtained from P_i $(i = 1, 2)$. It is easy to check that P'_1 and P'_2 are shortest paths in G. Here and in what follows, by a shortest path we mean a hop-shortest path. We can also show that the union of $N^1_G[P'_1]$ and $N^1_G[P'_2]$ is a balanced separator for G, i.e., removal of $N^1_G[P'_1] \cup N^1_G[P'_2]$ from G leaves no connected component with more that $2/3n$ vertices. Assume that removal of P_1 and P_2 from $G_p = (V_p, E_p)$ results in removing a set of edges E'_p from E_p, and removal of $N^1_G[P'_1]$ and $N^1_G[P'_2]$ from $G = (V, E)$ results in removing a set of edges E' from E. It is easy to check that, for any edge $e'_p \in E'_p$ there exists an edge $e' \in E'$ that covers e'_p. The latter implies that the union of $N^1_G[P'_1]$ and $N^1_G[P'_2]$ is a balanced separator for G. A formal proof of this will be presented in the journal version of the paper.

5 Balanced Separator for Arbitrary UDGs

In an arbitrary unit disk graph $G = (V, E)$, an edge may cross any number of other edges. Our basic strategy for building a balanced separator for G is similar to one we used in the case of a simple-crossing UDG, but details are more complicated. Let $T = T_{orig}$ initially. We will revise T to create a special spanning tree for the planar graph G_p obtained from G. Then, we will apply the Planar Separator Theorem from [19] (Theorem 1 above) to G_p and T to get a balanced separator S for G_p. Finally, we will recover from S the required separator for G.

5.1 Building a Special Spanning Tree T of G_p

In what follows, the edges of the tree T_{orig} will be called *original tree-edges*. By Lemma 3, for any two intersecting original tree-edges (a, b) and (c, d) (for which we assumed that $L(a) = L(b) - 1$, $L(c) = L(d) - 1$ and $L(a) \leq L(c)$), we have $L(a) = L(c) - 1$, $(a, d) \notin E(G)$ and $(b, c) \in E(G)$. We handle this kind of intersections (between original tree-edges) using PROCEDURE 1.

PROCEDURE 1. Handle original tree-edge intersections
Input: A layering tree T_{orig} rooted at r.
Output: A tree T where all original tree-edge intersections resolved.
Method: /* Break ties arbitrarily */

(1) Let $L_i = \{v : L(v) = i\}$ and $T = T_{orig}$;
(2) Let q be the maximum layer number of T;
(3) **FOR** $i = 1$ to q **DO**
(4) **FOR** each vertex $v_j \in L_i$ **DO**
(5) **FOR** each vertex $v_k \in L_{i+1}$ adjacent to v_j in T **DO**
(6) **IF** there is an original tree-edge intersection on (v_j, v_k) such that $L(v_j)$ is the SECOND smallest layer index among the layer indices of all four end-vertices of the two edges giving the intersection **THEN DO**
(7) Choose such an original tree-edge intersection closest to v_k and assume it is the intersection between (v_j, v_k) and (x, y) in T and between (v_j, v_k) and (v_p, v_h) in T_{orig} (i.e., $(x, y) \subseteq (v_p, v_h)$);
(8) Create an imaginary point $m_{j,k,p,h}$ at the point where (v_j, v_k) and (x, y) intersect;
(9) Update T by removing edges (v_j, v_k) and (x, y), and adding vertex $m_{j,k,p,h}$ and edges $(m_{j,k,p,h}, x)$, $(m_{j,k,p,h}, y)$, $(m_{j,k,p,h}, v_k)$;
(10) **RETURN** T

Lemma 4. *PROCEDURE 1 returns a tree T with all original tree-edge intersections resolved (i.e., edges of T do not cross each other).*

In addition, there are two other kinds of intersections remaining: the intersection between an edge in $E(T)$ (T-edge) and an edge in $E(G) \setminus E(T)$ (non-T-edge), and intersection between two non-T-edges.

First we handle intersections between T-edges and non-T-edges. They are resolved the same way as in Section 4. Here, we rephrase the rule. Assume (u, w) is a T-edge, (s, t) is a non-T-edge. Add a null point, say o, at the point where (u, w) and (s, t) intersect. Remove edge (u, w) from T and add vertex o and edges (u, o), (o, w) into T. After resolving all intersections of this kind, T becomes a subgraph of G_p. Note that it is possible that T does not span yet all elements of $V(G_p)$. *Let name this T as T_{sub}.*

Now, we deal with intersections between two non-T_{sub}-edges. This is more complicated than it was in Section 4 for restricted UDGs. We will grow T_{sub} to a spanning tree T_{span} for G_p (extension T_{span} of T_{sub} will cover all elements of $V(G_p)$). We use a procedure similar to one of building a shortest path tree from a set of vertices. We assign to each vertex in T_{sub} a weight according to the following formula. In formula, if v is an imaginary point or a null point, we assume v is at the intersection between edges (a, b) and (c, d) of G.

$$weight(v) = \begin{cases} 0, & \text{if } v \text{ is a real vertex;} \\ min\{|av|, |bv|, |cv|, |dv|\}, & \text{if } v \text{ is an imaginary or a null point.} \end{cases}$$

To build our spanning tree for G_p, we use PROCEDURE 2. At the beginning, for any $v \in V(G_p) \setminus V(T_{sub})$, $distance[v] = \infty$ and father of v is undefined.

PROCEDURE 2. Build a spanning tree for G_p from T_{sub}
Input: A tree $T = T_{sub}$;
Output: A tree T_{span} as a spanning tree for G_p.
Method: /* Break ties arbitrarily */

(1) **FOR** each i in $V(T)$ **DO**
(2) **FOR** each neighbor $j \in V(G_p)\backslash V(T)$ of i **DO**
(3) $tmp := weight[i] + |ij|$;
(4) **IF** $tmp < distance[j]$ **DO**
(5) $distance[j] := tmp$;
(6) $father[j] := i$;
(7) $Q := V(G_p)\backslash V(T)$;
(8) **WHILE** Q is not empty **DO**
(9) $u :=$node in Q with smallest distance$[\cdot]$;
(10) remove u from Q and add u into T;
(11) **FOR** each neighbor $v \in Q$ of u **DO**
(12) $tmp := distance[u] + |uv|$;
(13) **IF** $tmp < distance[v]$ **DO**
(14) $distance[v] := tmp$;
(15) $father[v] := u$;
(16) **RETURN** $T_{span} := T$.

It is easy to check that T_{span} is a spanning tree of the planar graph G_p.

5.2 Finding a Balanced 2×Shortest-Paths—3-Hop-Neighborhood Separator for G

Now we can apply Theorem 1 to G_p and T_{span} by letting the weight of each real vertex be 1 and the weight of each imaginary or null point be 0, and get a balanced separator S of G_p. Assume that S is the union of paths $P_1 = P_{T_{span}}(r, x)$ and $P_2 = P_{T_{span}}(r, y)$. There are three kinds of elements on P_1 and P_2: real vertices, imaginary points and null points. Generally, each imaginary point or null point is adjacent to at most four elements in G_p, and each element in P_1 or P_2 has the previous element and the next element, except for the root r (it has only the next element) and elements x and y (they have only the previous element). Let u be the last real or imaginary point in P_1 (or P_2). We name all null points after u in P_1 (or P_2) as the *tail null points*. For any element in P_1 or P_2, there are two possible relations between itself, its previous element and its next element: the element, its previous element and its next element are on the same line, which means its previous element and its next element are on the same edge of G (according to our general assumption that no two edges of G are on the same line); the element, its previous element and its next element are not on the same line, which means its previous element and itself are on one edge of G, and its next element and itself are on another edge of G.

Using paths $P_1 = (x_0 = r, x_1, \ldots, x_{k-1}, x_k = x)$ and $P_2 = (y_0 = r, y_1, \ldots, y_{l-1}, y_l = y)$ of G_p (of T_{span}), We will find the corresponding balanced separator for G using the following steps. (1) We skip all null points in P_1 and P_2. Let

the resulting paths be P_1' and P_2', respectively. (2) We skip in P_1' and P_2' each imaginary point whose previous element and next element are on the same edge of T_{orig}. For example, let (x_f, x_i, x_j) be a fragment of path P_1' or P_2', where x_i is an imaginary point and $\{x_f, x_i, x_j\}$ are collinear, then (x_f, x_i, x_j) will be replaced with (x_f, x_j). Let the resulting paths be P_1'' and P_2'', respectively. (3) Replace each remaining imaginary point m in P_1'' and P_2'' with two vertices: $b(m)$ followed by $c(m)$ (see end of Section 3 for these notations). For example, let (x_f, x_i, x_j) be a fragment of path P_1'' or P_2'', where x_i is an imaginary point and x_f is closest to the root r among $\{x_f, x_i, x_j\}$. Then, (x_f, x_i, x_j) will be replaced with $(x_f, b(x_i), c(x_i), x_j)$. Let the resulting paths be P_1''' and P_2''', respectively. By Lemma 3, the edge $(b(x_i), c(x_i))$ exists in G. It is easy to check that P_1''' and P_2''' are valid paths in G.

A path P of G is called a 2×shortest path iff for any two vertices x,y in P, $d_P(x, y) \leq 2d_G(x, y)$.

Theorem 2. *P_1''' and P_2''' are 2×shortest paths in G.*

We can show also that the union of $N_G^3[P_1''']$ and $N_G^3[P_2''']$ is a balanced separator for G with 2/3-split, i.e., removal of $N_G^3[P_1'''] \cup N_G^3[P_2''']$ from G leaves no connected component with more than $2/3n$ vertices. Thus, there exist two paths P_1''' and P_2''' in G such that they are 2×shortest paths and the union of $N_G^3[P_1''']$ and $N_G^3[P_2''']$ is a balanced separator for G.

5.3 Finding a Balanced Shortest-Paths—3-Hop-Neighborhood Separator for *G*

In this section, we will improve the result of Section 5.2. We will show that any UDG G has two shortest paths P_1''' and P_2''' such that the union of $N_G^3[P_1''']$ and $N_G^3[P_2''']$ forms a balanced separator for G. Recall that, by a shortest path we mean a hop-shortest path.

Let P_1, P_2, P_1', P_2', P_1'' and P_2'' be the paths defined in Section 5.2. Analogs of paths P_1''' and P_2''' of Section 5.2 will be obtained from P_1'' and P_2'' in a more careful way (than in Section 5.2). We use PROCEDURE 3 for this.

PROCEDURE 3. Handle imaginary points

Input: Path $P \in \{P_1'', P_2''\}$ (containing still some imaginary points).
Output: Path P as a shortest path of G, with all imaginary points resolved.
Method: /* Break ties arbitrarily. The first vertex in P is the root r, a real vertex.*/

(1) Let $[v_1, \cdots, v_k]$ be the imaginary points in P in the order from r;
(2) **FOR** $i = 1$ to k **DO**
(3) **IF** vertex $c(v_i)$ is adjacent to $prev_P(v_i)$
($c(v_i)$ is always adjacent to $next_P(v_i)$, as it is shown later.)
(4) Replace v_i with $c(v_i)$ in P;
(5) **ELSE** (It implies that vertex $b(v_i)$ is adjacent to both $prev_P(v_i)$
and $next_P(v_i)$, as it is shown later.)
(6) Replace v_i with $b(v_i)$ in P;
(7) **RETURN** P

We call PROCEDURE 3 for both P_1'' and P_2''. Let the resulting paths be P_1''' and P_2''', respectively. We can show that P_1''' and P_2''' are shortest paths in G. Now, for these paths P_1''' and P_2''', we have.

Theorem 3. *The union of $N_G^3[P_1'''] $ and $N_G^3[P_2'''] $ is a balanced separator for G with $2/3$-split, i.e., removal of $N_G^3[P_1'''] \cup N_G^3[P_2'''] $ from G leaves no connected component with more than $2/3n$ vertices.*

6 Application of Balanced Separators for UDGs

In this section, we show how one can use the above balanced separator theorem for UDGs to construct for them collective tree spanners with low stretch and to develop a compact and low delay routing labeling scheme. For this, we combine strategies used in [5,6,13]. The details can be found in the full version of this paper. Here we list only the final results.

Theorem 4. *Any unit disk graph G with n vertices and m edges admits a system $\mathcal{T}(G)$ of at most $2\log_{3/2} n + 2$ collective tree $(3, 12)$-spanners, i.e., for any two vertices x and y in G, there exists a spanning tree $T \in \mathcal{T}(G)$ with $d_T(x, y) \leq 3d_G(x, y) + 12$. Moreover, such a system $\mathcal{T}(G)$ can be constructed in $O((C + m) \log n)$ time, where C is the number of crossings in G.*

Corollary 1. *Any unit disk graph G with n vertices admits a hop $(3, 12)$-spanner with at most $2(n - 1)(\log_{3/2} n + 1)$ edges.*

Theorem 5. *The family of n-vertex unit disk graphs admits an $O(\log^2 n)$ bit $(3, 12)$-approximate distance labeling scheme with $O(\log n)$ time distance decoder.*

Theorem 6. *The family of n-vertex unit disk graphs admits an $O(\log^2 n)$ bit routing labeling scheme. The scheme has hop $(3, 12)$-route-stretch. Once computed by the sender in $O(\log n)$ time, headers never change, and the routing decision is made in constant time per vertex.*

In the journal version of the paper, we show also how to extend this bounded hop route-stretch routing labeling scheme to a routing labeling scheme with bounded length route-stretch.

References

1. Alber, J., Fiala, J.: Geometric separation and exact solutions for the parameterized independent set problem on disk graphs. J. of Algorithms 52, 134–151 (2004)
2. Alzoubi, K., Li, X.-Y., Wang, Y., Wan, P.-J., Frieder, O.: Geometric spanners for wireless ad hoc networks. IEEE Trans. on Par. and Distr. Syst. 14, 408–421 (2003)
3. Clark, B.N., Colbourn, C.J.: Unit Disk Graphs. Discrete Math. 86, 165–177 (1990)
4. Bose, P., Morin, P., Stojmenovic, I., Urrutia, J.: Routing with guaranteed delivery in ad hoc wireless networks. In: 3rd Internat. workshop on discr. algor. and methods for mobile computing and communications, pp. 48–55. ACM Press, New York (1999)

5. Dragan, F.F., Yan, C., Corneil, D.G.: Collective Tree Spanners and Routing in AT-free Related Graphs. J. of Graph Algor. and Applic. 10(2), 97–122 (2006)
6. Dragan, F.F., Yan, C., Lomonosov, I.: Collective tree spanners of graphs. SIAM J. Discrete Math. 20, 241–260 (2006)
7. Fonseca, R., Ratnasamy, S., Zhao, J., Ee, C.T., Culler, D., Shenker, S., Stoica, I.: Beacon vector routing: Scalable point-to-point routing in wireless sensornets. In: 2nd USENIX/ACM Symp. on Netw. Syst. Design and Implement (NSDI 2005) (2005)
8. Fürer, M., Kasiviswanathan, S.P.: Spanners for geometric intersection graphs. In: Dehne, F., Sack, J.-R., Zeh, N. (eds.) WADS 2007. LNCS, vol. 4619, pp. 312–324. Springer, Heidelberg (2007)
9. Fraigniaud, P., Gavoille, C.: Routing in Trees. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 757–772. Springer, Heidelberg (2001)
10. Gao, J., Guibas, L.J., Hershberger, J., Zhang, L., Zhu, A.: Geometric spanner for routing in mobile networks. In: 2nd ACM international symposium on mobile ad hoc networking & computing, Long Beach, CA, USA, October 04-05 (2001)
11. Gao, J., Zhang, L.: Well-separated pair decomposition for the unit-disk graph metric and its applications. In: STOC 2003, pp. 483–492 (2003)
12. Giordano, S., Stojmenovic, I.: Position based routing algorithms for ad hoc networks: A taxonomy. In: Ad Hoc Wireless Networking, pp. 103–136. Kluwer, Dordrecht (2004)
13. Gupta, A., Kumar, A., Rastogi, R.: Traveling with a Pez Dispenser (Or, Routing Issues in MPLS). In: FOCS 2001, pp. 148–157 (2001)
14. Karp, B., Kung, H.T.: GPSR: greedy perimeter stateless routing for wireless networks. In: Proceedings of the 6th ACM/IEEE MobiCom, pp. 243–254. ACM Press, New York (2000)
15. Kleinberg, R.: Geographic routing using hyperbolic space. In: INFOCOM 2007, pp. 1902–1909 (2007)
16. Kuhn, F., Wattenhofer, R., Zhang, Y., Zollinger, A.: Geometric ad-hoc routing: of theory and practice. In: PODC 2003, pp. 63–72. ACM Press, New York (2003)
17. Li, X.-Y.: Ad Hoc Wireless Networking. In: Li, X.-Y. (ed.) Applications of Computational Geomety in Wireless Ad Hoc Networks, Kluwer, Dordrecht (2003)
18. Li, X.-Y., Wang, Y.: Geometrical Spanner for Wireless Ad Hoc Networks. In: Handbook of Approx. Algorithms and Metaheuristics. Chapman&Hall/Crc, Boca Raton (2006)
19. Lipton, R.J., Tarjan, R.E.: A Separator Theorem for Planar Graphs. SIAM Journal on Applied Mathematics 36, 177–189 (1979)
20. Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. SIAM Monographs on Discrete Math. Appl. SIAM, Philadelphia (2000)
21. Rao, A., Papadimitriou, C., Shenker, S., Stoica, I.: Geographical routing without location information. In: MobiCom 2003, pp. 96–108 (2003)
22. Thorup, M.: Compact Oracles for Reachability and Approximate Distances in Planar Digraphs. In: FOCS, pp. 242–251 (2001)
23. Thorup, M., Zwick, U.: Compact routing schemes. In: SPAA 2001, pp. 1–10 (2001)
24. Yan, C.: Approximating Distances in Complicated Graphs by Distances in Simple Graphs With Applications. PhD Dissertation, Kent State University (2007), http://www.ohiolink.edu/etd/send-pdf.cgi/Yan%20Chenyu.pdf?kent1184639623