Routing in Unit Disk Graphs*

Haim Kaplan[1], Wolfgang Mulzer[2], Liam Roditty[3], and Paul Seiferth[2]

[1] School of Computer Science, Tel Aviv University, Israel
haimk@post.tau.ac.il
[2] Institut für Informatik, Freie Universität Berlin, Berlin, Germany.
[mulzer, pseiferth]@inf.fu-berlin.de
[3] Department of Computer Science, Bar Ilan University, Israel
liamr@macs.biu.ac.il

Abstract. Let $S \subset \mathbb{R}^2$ be a set of n sites. The unit disk graph $\mathrm{UD}(S)$ on S has vertex set S and an edge between two distinct sites $s, t \in S$ if and only if s and t have Euclidean distance $|st| \leq 1$.
A routing scheme R for $\mathrm{UD}(S)$ assigns to each site $s \in S$ a *label* $\ell(s)$ and a *routing table* $\rho(s)$. For any two sites $s, t \in S$, the scheme R must be able to route a packet from s to t in the following way: given a *current site* r (initially, $r = s$), a *header* h (initially empty), and the *target label* $\ell(t)$, the scheme R may consult the current routing table $\rho(r)$ to compute a new site r' and a new header h', where r' is a neighbor of r. The packet is then routed to r', and the process is repeated until the packet reaches t. The resulting sequence of sites is called the *routing path*. The *stretch* of R is the maximum ratio of the (Euclidean) length of the routing path produced by R and the shortest path in $\mathrm{UD}(S)$, over all pairs of distinct sites in S.
For any given $\varepsilon > 0$, we show how to construct a routing scheme for $\mathrm{UD}(S)$ with stretch $1+\varepsilon$ using labels of $O(\log n)$ bits and routing tables of $O(\varepsilon^{-5} \log^2 n \log^2 D)$ bits, where D is the (Euclidean) diameter of $\mathrm{UD}(S)$. The header size is $O(\log n \log D)$ bits.

1 Introduction

Routing in graphs constitutes a fundamental problem in distributed graph algorithms [8,11]. Given a graph G, we would like to be able to route a packet from any node in G to any other node. The routing algorithm should be *local*, meaning that it uses only information stored with the packet and with the current node, and it should be *efficient*, meaning that the packet does not travel much longer than necessary. There is an obvious solution to this problem: with each node s of G, we store the shortest path tree for s. Then it is easy to route a packet along the shortest path to its destination. However, this solution is very inefficient: we need to store the complete topology of G with each node, leading to quadratic space usage. Thus, the goal of a routing scheme is to store as little information as possible with each node of the graph, while still guaranteeing a routing path that is not too far from optimal.

* This work is supported by GIF project 1161 & DFG project MU/3501/1.

For general graphs a plethora of results is available, reflecting the work of almost three decades (see, e.g., [3, 12] and the references therein). However, for general graphs, any efficient routing scheme needs to store $\Omega(n^\alpha)$ bits per node, for some $\alpha > 0$ [11]. Thus, it is natural to ask whether improved results are possible for specialized graph classes. For example, for trees it is known how to obtain a routing scheme that follows a shortest path and requires $O(\log n)$ bits of information at each node [6, 14]. In planar graphs, for any $\varepsilon > 0$ it is possible to store a polylogarithmic number of bits at each node in order to route a packet along a path of length at most $1 + \varepsilon$ times the length of the shortest path [13].

A graph class that is of particular interest for routing problems comes from the study of mobile and wireless networks. Such networks are traditionally modeled as *unit disk graphs* [4]. The nodes are modeled as points in the plane, and two nodes are connected if and only if the distance between the corresponding points is at most one. Even though unit disk graphs may be dense, they share many properties with planar graphs, in particular with respect to algorithmic problems. There exists a vast literature on routing in unit disk graphs (cf. [8]), but most known schemes cannot ensure a short routing path in the worst case. Yan, Xiang, and Dragan [15] present a scheme with provable worst case guarantees. They extend a scheme by Gupta et al. [9] for planar graphs to unit disk graphs by using a delicate planarization argument to obtain small-sized balanced separators. Even though the scheme by Yan et al. is conceptually simple, it requires a detailed analysis with an extensive case distinction.

We propose an alternative approach to routing in unit disk graphs. Our scheme is based on the well-separated pair decomposition for unit disk graphs [7]. It stores a polylogarithmic number of bits with each node of the graph, and it constructs a routing path that can be made arbitrarily close to a shortest path (see Section 2 for a precise statement of our results). This compares favorably with the scheme by Yan et al. [15] which achieves only a constant factor approximation. Moreover, our scheme is arguably simpler to analyze. However, unlike the algorithm by Yan et al., our scheme requires that the packet contain a modifiable *header* with a polylogarithmic number of bits. It is an interesting question whether this header can be removed.

2 The Model and Our Results

Let $S \subset \mathbb{R}^2$ be a set of n *sites* in the plane. We say that S has *density* δ if every unit disk contains at most δ points from S. The density δ of S is *bounded* if $\delta = O(1)$. The *unit disk graph* for S is the graph $\mathrm{UD}(S)$ with vertex set S and an edge st between two distinct sites $s, t \in S$ if and only if $|st| \leq 1$, where $|\cdot|$ denotes the Euclidean distance. We define the *weight* of the edge st to be its Euclidean length and use $d(\cdot, \cdot)$ to denote the shortest path distance in $\mathrm{UD}(S)$.

We would like to obtain a *routing scheme* for $\mathrm{UD}(S)$ with small *stretch* and compact *routing tables*. Formally, this is defined as follows: we can preprocess $\mathrm{UD}(S)$ to obtain for each site $s \in S$ (i) a *label* $\ell(s) \in \{0, 1\}^*$, and (ii) a *routing table* $\rho(s) \in \{0, 1\}^*$. Furthermore, we need to define a *routing function* f :

$S \times \{0,1\}^* \times \{0,1\}^* \to S \times \{0,1\}^* \times \{0,1\}^*$. The function f takes as input a *current site* s, the label $\ell(t)$ of a *target site* t, and a *header* $h \in \{0,1\}^*$. The routing function may use its input and the routing table $\rho(s)$ of s to compute a new site s', a modified header h', and the label of an *intermediate target* $\ell(t')$. The new site s' may be either s or a neighbor of s in $\mathrm{UD}(S)$. Even though the eventual goal of the packet is the target t, we introduce the intermediate target t' into the notation, since it allows for a more succinct presentation of the routing algorithm.

The routing scheme is *correct* if the following holds: let h_0 be the empty header. For any two sites $s, t \in S$, consider the sequence of triples given by $(s_0, \ell_0, h_0) = (s, \ell(t), h_0)$ and $(s_i, \ell_i, h_i) = f(s_{i-1}, \ell_{i-1}, h_{i-1})$ for $i \geq 1$. Then there exists a $k = k(s,t) \geq 0$ such that $s_k = t$ and $s_i \neq t$ for $i < k$, i.e., the routing scheme reaches t after k steps. We call $s_0, s_1, \dots, s_k$ the *routing path* between s and t, and we define the *routing distance* $d_\rho(s,t)$ between s and t as $d_\rho(s,t) = \sum_{i=1}^{k} |s_{i-1}s_i|$.

The quality of the routing scheme is measured by several parameters:

- the *label size* $L(n) = \max_{|S|=n} \max_{s \in S} |\ell(s)|$,
- the *table size* $T(n) = \max_{|S|=n} \max_{s \in S} |\rho(s)|$,
- the *header size* $H(n) = \max_{|S|=n} \max_{s \neq t \in S} \max_{i=1,\dots,k(s,t)} |h_i|$,
- and the *stretch* $\varphi(n) = \max_{|S|=n} \max_{s \neq t \in S} d_\rho(s,t)/d(s,t)$.

We show that for any $S \subset \mathbb{R}^2$, $|S| = n$, and any $\varepsilon > 0$ we can construct a routing scheme with $\varphi(n) = 1+\varepsilon$, $L(n) = O(\log n)$, $T(n) = O(\varepsilon^{-5} \log^2 n \log^2 D)$, and $H(n) = O(\log n \log D)$, where D is the weighted diameter of $\mathrm{UD}(S)$, i.e., the maximum length of a shortest path between two sites in $\mathrm{UD}(S)$.

3 The Well-Separated Pair Decomposition for UD(S)

Our routing scheme uses the well-separated pair decomposition (WSPD) for the unit disk graph metric given by Gao and Zhang [7]. WSPDs provide a compact way to efficiently encode the approximate pairwise distances in a metric space. Originally, WSPDs were introduced by Callahan and Kosaraju [2] in the context of the Euclidean metric, and they have found numerous applications since then (see e.g., [7, 10] and the references therein).

Since our routing scheme relies crucially on the specific structure of the WSPD described by Gao and Zhang, we remind the reader of the main steps of their algorithm and analysis.

First, Gao and Zhang assume that S has bounded density and that $\mathrm{UD}(S)$ is connected. They construct the Euclidean minimum spanning tree T for S. It is easy to see that T is a spanning tree for $\mathrm{UD}(S)$ with maximum degree 6. Furthermore, T can be constructed in $O(n \log n)$ time [1]. Since T has maximum degree 6, there exists an edge e in T such that $T \setminus e$ consists of two trees with at least $\lceil (n-1)/6 \rceil$ vertices each. By applying this observation recursively, we obtain a *hierarchical decomposition* H of T. The decomposition H is a binary tree. Each node v of H represents a subtree T_v of T with vertex set $S_v \subseteq S$ such that (i) the root of H corresponds to T; (ii) the leaves of H are in one-to-one correspondence with the sites in S; and (iii) let v be an inner node of H with children u and w. Then v has an *associated edge* $e_v \in T_v$ such that removing e_v from T_v yields the two subtrees T_u and T_w represented by u and w (see Figure 1). Furthermore, we have $|S_u|, |S_w| \geq \lceil (|S_v| - 1)/6 \rceil$.

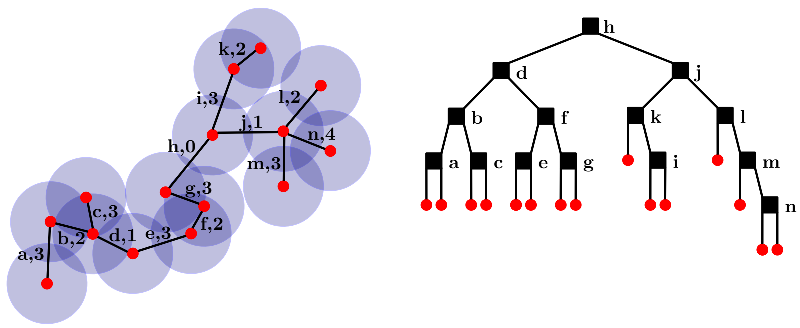

Fig. 1. An EMST of UD(S) (left) where the edges are annotated with their level in the hierarchical decomposition (right).

It follows that H has height $O(\log n)$. The *depth* $\delta(v)$ of a node $v \in H$ is defined as the number of edges on the path from v to the root of H. The *level* of the associated edge e_v of v is the depth of v in H. This uniquely defines a level for each edge in T. Now, for each node $v \in H$, the subtree T_v is a connected component in the forest that is induced in T by the edges of level at least $\delta(v)$.

After computing the hierarchical decomposition, the algorithm of Gao and Zhang essentially uses the greedy algorithm of Callahan and Kosaraju to construct a WSPD, with H in place of the quadtree (or the fair split tree). Let $c \geq 1$ be a separation parameter. The algorithm traverses H and produces a sequence $\Xi = (u_1, v_1), (u_2, v_2), \ldots, (u_m, v_m)$ of pairs of nodes of H, with the following properties:

1. The sets $S_{u_1} \times S_{v_1}, S_{u_2} \times S_{v_2}, \ldots, S_{u_m} \times S_{v_m}$ constitute a partition of $S \times S$. This means that for each ordered pair of sites $(s, t) \in S \times S$, there is exactly one pair $(u, v) \in \Xi$ with $(s, t) \in S_u \times S_v$. We say that (u, v) *represents* (s, t).
2. Each pair $(u, v) \in \Xi$ is *c-well-separated*, i.e., we have

$$(c + 2) \max\{|S_u| - 1, |S_v| - 1\} \leq |\sigma(u)\sigma(v)|, \tag{1}$$

where $\sigma(u), \sigma(v)$ are arbitrary sites in S_u and S_v chosen by the algorithm.

Since in the unit distance graph metric the diameter $\mathrm{diam}(S_u)$ is at most $|S_u| - 1$ and since $|\sigma(u)\sigma(v)| \leq d(\sigma(u), \sigma(v))$, (1) implies that

$$(c + 2) \max\{\mathrm{diam}(S_u), \mathrm{diam}(S_v)\} \leq d(\sigma(u), \sigma(v)), \tag{2}$$

which is the traditional well-separation condition. However, (1) is easier to check algorithmically and has additional advantages that we will exploit in our routing scheme below.

Gao and Zhang show that their algorithm produces a c-WSPD with $m = O(\delta c^2 n \log n)$ pairs, where δ is the density of S. More precisely, they prove the following lemma:

Lemma 3.1 (Lemma 4.3 and Corollary 4.6 in [7]). *For each node $u \in H$, the WSPD Ξ has $O(\delta c^2 |S_u|)$ pairs that contain u.* □

4 Preliminary Lemmas

We begin with two technical lemmas on WSPDs that will be useful later on. The first lemma shows that the choice of the sites $\sigma(u)$ for the nodes $u \in H$ is essentially arbitrary.

Lemma 4.1. *Let Ξ be a c-WSPD for S and let s, t be two sites such that the pair $(u, v) \in \Xi$ represents (s, t). Then $c \operatorname{diam}(S_u) \le c(|S_u| - 1) \le d(s, t)$.*

Proof. By triangle inequality and (1) we have

$$\begin{aligned} |st| &\ge |\sigma(u)\sigma(v)| - 2\max\{\operatorname{diam}(S_u), \operatorname{diam}(S_v)\} \\ &\ge (c+2)\max\{|S_u| - 1, |S_v| - 1\} - 2\max\{\operatorname{diam}(S_u), \operatorname{diam}(S_v)\}. \end{aligned}$$

Since $|S_u| - 1$ and $|S_v| - 1$ are upper bounds for $\operatorname{diam}(S_u)$ and $\operatorname{diam}(S_v)$, respectively, and since $d(s, t) \ge |st|$, the claim follows. □

The next lemma shows that short distances are represented by singletons in the WSPD.

Lemma 4.2. *Let Ξ be a c-WSPD for S and let $s, t \in S$ be two sites with $d(s, t) < c$. If $(u, v) \in \Xi$ represents (s, t), then $S_u = \{s\}$ and $S_v = \{t\}$.*

Proof. By triangle inequality and since $d(s, t) < c$, we have

$$\begin{aligned} d(\sigma(u), \sigma(v)) &\le d(s, t) + 2\max\{\operatorname{diam}(S_u), \operatorname{diam}(S_v)\} \\ &< c + 2\max\{|S_u| - 1, |S_v| - 1\}. \end{aligned}$$

Since (u, v) is c-well-separated,

$$d(\sigma(u), \sigma(v)) \ge (c+2)\max\{|S_u| - 1, |S_v| - 1\}.$$

This implies that $|S_u| = |S_v| = 1$, and the claim follows. □

5 The Routing Scheme

Let δ be the density of S. First we describe a routing scheme whose parameters depend on δ. Then we show how to remove this dependency and extend the scheme to work with arbitrary density. Our routing scheme uses the WSPD described in Section 3, and it is based on the following idea: let Ξ be the c-WSPD for UD(S) and let T be the EMST for S used to compute it. We distribute the information about the pairs in Ξ among the sites in S (in a way described later) such that each site stores $O(\delta c^2 \log n)$ pairs in its routing table. To route from s to t, we explore T, starting from s, until we find the site r with the pair (u, v) representing (s, t). Our scheme will guarantee that s and r are sites in S_u, and therefore it suffices to walk along T_u to find r (see Figure 2). This is called the *local routing*. With (u, v), we store in $\rho(r)$ the *middle site* m on the shortest path from r to $\sigma(v)$, i.e., the vertex "halfway" between r and $\sigma(v)$. We recursively route from r to m and when reaching m from m to t. To keep track of intermediate targets during the recursion, we store a stack in the header. This second step, the recursive routing through the middle site, we call the *global routing*.

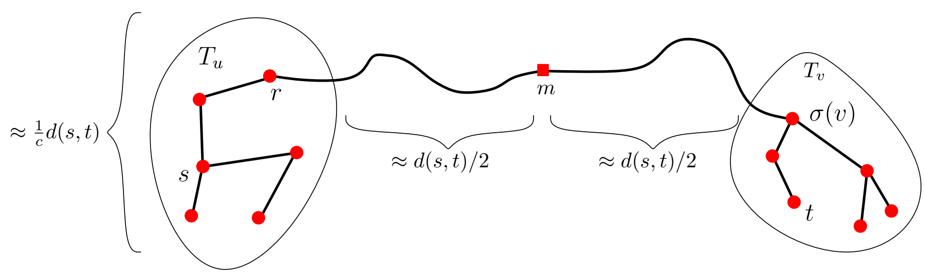

Fig. 2. To route a packet from s to t, we first walk along T_v until we find r. Then we recursively route from r to m and from m to t.

We now describe our routing scheme in detail. Let $1+\varepsilon$, $\varepsilon > 0$, be the desired stretch factor.

5.1 Preprocessing

The preprocessing phase works as follows. We set $c = (\alpha/\varepsilon) \log D$, where D is the Euclidean diameter of UD(S) and α is a sufficiently large constant we will fix later. Then we compute a c-WSPD for UD(S). As explained in Section 3, the WSPD consists of a bounded degree spanning tree T of UD(S), a hierarchical balanced decomposition H of T whose nodes $u \in H$ correspond to subtrees T_u of T, and a sequence $\Xi = (u_1, v_1), (u_2, v_2), \ldots, (u_m, v_m)$ of $m = O(\delta c^2 n \log n) = O(\delta\varepsilon^{-2} n \log n \log^2 D)$ well-separated pairs that represent a partition of $S \times S$.

First, we determine the *labeling* ℓ for the sites in S. For this, we perform a postorder traversal of H. Let l be a counter which is initialized to 1. Whenever we encounter a leaf of H, we set the label $\ell(s)$ of the corresponding site $s \in S$ to l, and we increment l by 1. Whenever we visit an internal node u of H for the last time, we annotate it with the interval I_u of the labels in T_u. Thus, a site $s \in S$ lies in a subtree T_u if and only if $\ell(s) \in I_u$. Each label has $O(\log n)$ bits.

Next, we describe the routing tables. Each routing table consists of two parts, the *local* routing table and the *global* routing table. The local routing table $\rho_L(s)$ of a site s stores the neighbors of s in T, in counterclockwise order, together with the levels in H of the corresponding edges (cf. Section 3). Since T has degree at most 6, each local routing table consists of $O(\log n)$ bits. The global routing table $\rho_G(s)$ of a site s is obtained as follows: we go through all $O(\log n)$ nodes u of H that contain s in their subtree T_u. By Lemma 3.1, Ξ contains at most $O(\delta c^2 |S_u|)$ well-separated pairs in which u represents one of the sets. We assign $O(\delta c^2) = O(\delta\varepsilon^{-2} \log^2 D)$ of these pairs to s, such that each pair is assigned to exactly one site in S_u. For each pair (u, v) assigned to s, we store the interval I_v corresponding to S_v. Furthermore, if $\sigma(v)$ is not a neighbor of s, we store the label $\ell(m)$ of the *middle site* m of a shortest path π from s to $\sigma(v)$. Here, m is a site on π that minimizes the maximum distance, $\max\{d(s, m), d(m, \sigma(v))\}$, to the endpoints of π. A site s lies in $O(\log n)$ different sets S_u, at most one for each level of H. For each such set, we store $O(\delta\varepsilon^{-2} \log^2 D)$ pairs in $\rho_G(s)$, each of which requires $O(\log n)$ bits. Thus, ρ_G has $O(\delta\varepsilon^{-2} \log^2 n \log^2 D)$ bits.

Finally, we argue that the routing scheme can be computed efficiently.

Lemma 5.1. *The preprocessing time for the routing scheme described above is* $O(n^2 \log n + \delta n^2 + \delta\varepsilon^{-2} n \log n \log^2 D)$.

Proof. The c-WSPD can be computed in $O(\delta c^2 n \log n) = O(\delta\varepsilon^{-2} n \log n \log^2 D)$ time [7]. Within the same time bound, we can distribute the WSPD-pairs to the sites in S and compute the labels for S.

It remains to compute the middle sites; we do this for all pairs $(s, t) \in S \times S$ as follows: we first compute $\mathrm{UD}(S)$ explicitly. Since S has density δ, we have $O(\delta n)$ edges in $\mathrm{UD}(S)$, and we can compute it naively in time $O(n^2)$. For each $s \in S$, we compute the shortest path tree $\mathcal{T}$ with root s. This takes total time $O(n^2 \log n + \delta n^2)$, using n invocations of Dijkstra's algorithm.

For each $s \in S$, we perform a post-order traversal of the shortest path tree $\mathcal{T}$ to find the middle sites for all s-t-paths. First, for each leaf t of $\mathcal{T}$, we create a max-heap that contains t with $d(s, t)$ as the key. We now describe how to process a site m during the traversal. First, we merge the heaps of all children of m into a new heap H and we insert m into H with $d(s, m)$ as key. During the traversal, we maintain the invariant that H contains all sites that are descendants of m in $\mathcal{T}$ for which we have not yet found a middle site. Furthermore, since $d(s, t)$ increases monotonically along every root-leaf path in $\mathcal{T}$, the sites for which m might be the middle site are a prefix of the decreasingly sorted distances $d(s, t)$ with $t \in H$. Thus, to find the sites in H for which m is the middle site, we repeatedly perform an extract-max operation on H to obtain the next candidate t. Then, we compare the value of $\max\{d(s, m), d(m, t)\}$ with $\max\{d(s, m'), d(m', t)\}$, where

m' is the parent of m in $\mathcal{T}$. That is, we check if m' is a "better" middle site than m. If not, m must be the middle site for s-t. Otherwise, m cannot be the middle site for any other site in H, and we proceed with our traversal. Using, e.g., Fibonacci Heaps, we can merge two heaps in $O(1)$ time and perform an extract-max operation in $O(\log n)$ amortized time [5]. Since each element of $\mathcal{T}$ is inserted and extracted at most once, we need $O(n \log n)$ time to find the middle sites for s. Thus, we can find all middle sites in time $O(n^2 \log n)$ and the total preprocessing time is $O(n^2 \log n + \delta n^2 + \delta\varepsilon^{-2} n \log n \log^2 D)$. □

5.2 Routing a Packet

Suppose we are given two sites s and t, and we would like to route a packet from s to t. Recall our overall strategy: we first perform a local exploration of $\mathrm{UD}(S)$ in order to discover a site r that stores a pair $(u, v) \in \Xi$ representing (s, t) in its global routing table $\rho_G(r)$. To find r, we consider the subtrees of T that contain s by increasing size, and we perform an Euler tour in each subtree until we find r. In $\rho_G(r)$ we have stored the middle site m of a shortest path from r to $\sigma(v)$. We put t into the header, and we recursively route the packet from r to m. Once we reach m, we retrieve the original target t from the header and recursively route from m to t, see Algorithm 1 for pseudo-code.

Local Routing: The Euler-Tour. We start at s, and we would like to find the site r that stores the pair (u, v) representing (s, t). By construction, both s and r are contained in S_u, and it suffices to perform an Euler tour on T_u to discover r. Since we do not know u in advance, we begin with the leaf in H that contains s, and we explore all nodes on the path to the root until we find u.

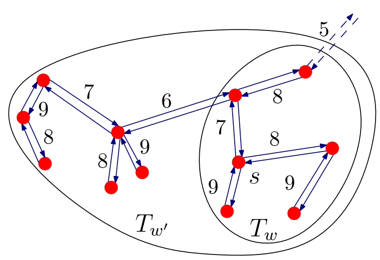

Fig. 3. To find r we do an Euler Tour on T_u, the subtree that contains s whose edges have level at least 7. Since we do not find r, we search the next larger subtree $T_{u'}$, where u' is the parent of u in H by decreasing the search level to 6.

We store s as the *start site* in the header h. Let $w \in H$ be the node to be explored, and let $l = \delta(w)$ be the depth of w in H. We store l in h. Recall that T_w is a connected component of the forest induced by all edges of level at least l. We perform an Euler tour on T_w using the local routing tables as follows: starting at s, we follow the first edge in $\rho_L(s)$ that has level at least l. Every time we visit a site r, we check for all WSPD-pairs (u, v) in $\rho_G(r)$ whether $\ell(t) \in I_v$, i.e., whether $t \in S_v$. If so, we clear the local routing information from h, and we proceed with the global routing. If not, we scan $\rho_L(r)$ for the next edge in $\rho_L(r)$ that has level at least l, going back to the beginning of $\rho_L(r)$ if necessary, and we follow this edge. For this, we must remember in h the edge through which we entered r (note that we must store only the last edge of the tour). Once we reach s for the last time (i.e., through the last edge in $\rho_L(s)$ with level at least l), we decrease l by one and restart the process. Decreasing l corresponds to proceeding with the parent of w in H.

Global Routing: The WSPD. Suppose we are at a site s such that $\rho_G(s)$ contains the pair (u, v) with the target t being in S_v. If t is not a neighbor of s, then $\rho_G(s)$ also contains the label of a middle site m for (u, v). We push (the label of) t onto the header stack, and we use $\ell(m)$ as the new target. Then we perform a local routing, starting at s, in order to find a pair (u', v') with $m \in S_{v'}$. If t is a neighbor of s, we go directly to t. Since t may be an intermediate target, we pop the next element from the header stack and set it as the new target label. If the header stack is empty, t is our final destination.

Input: currentSite s, targetLabel $\ell(t)$, header h
Output: nextSite, nextTargetLabel, header

if $\ell(s) = \ell(t)$ **then** `/* intermediate target reached? */`
    if h.stack $= \emptyset$ **then** `/* final target? */`
        return $(s, \perp, \perp)$
    else
        return $(s, h.\text{stack.pop}(), h)$
else if $\rho(s)$ stores a WSPD-pair (u, v) with $t \in S_v$ **then** `/* global routing */`
    h.startSite $\leftarrow \emptyset$
    if s and t are neighbors in $\mathrm{UD}(S)$ **then**
        return $(t, \ell(t), h)$
    else
        nextTargetLabel $\leftarrow$ label of middle site for (u, v)
        h.stack.push($\ell(t)$)
        return $(s, \text{nextTargetLabel}, h)$
else `/* local routing */`
    if h.startSite $= \emptyset$ **then**
        h.startSite $\leftarrow s$
        h.level $\leftarrow \delta(s)$
    $r \leftarrow$ next clockwise neighbor of s with level of edge $sr \geq h$.level
    if $r = \perp$ **then** `/* Euler tour is finished */`
        h.level $\leftarrow h$.level $- 1$
        return $(s, \ell(t), h)$
    else
        return $(r, \ell(t), h)$

Algorithm 1: The routing algorithm.

5.3 Analysis of the Routing Scheme

We now prove that the described routing scheme is correct and has low stretch, i.e., that for any two sites s and t, it produces a routing path $s = s_0, \dots, s_k = t$ of length at most $(1+\varepsilon)d(s,t)$.

Correctness. First, we consider only small distances and show that in this case our routing scheme produces an actual shortest path.

Lemma 5.2. *Let s, t be two sites in S with $d(s,t) < c$. Then, the routing scheme produces a routing path $s_0, s_1, \dots, s_k$ with the following properties*

(i) $s_0 = s$ *and* $s_k = t$,
(ii) $d_\rho(s,t) = d(s,t)$, *and*
(iii) the header stack is in the same state at the beginning and at the end of the routing path.

Proof. We prove that our routing scheme has properties (i)-(iii) by induction on the rank of $d(s,t)$ in the sorted list of the pairwise distances in $\mathrm{UD}(S)$.

For the base case, consider the edges st in $\mathrm{UD}(G)$, i.e., $d(s,t) = |st| \le 1$. By Lemma 4.2, there exists a pair (u,v) with $S_u = \{s\}$ and $S_v = \{t\}$. Thus, Algorithm 1 correctly routes to t in one step and does not manipulate the header stack. All properties are fulfilled.

Now, consider an arbitrary pair s, t with $1 < d(s,t) < c$. By Lemma 4.2, there is a pair (u,v) with $S_u = \{s\}$ and $S_v = \{t\}$. By construction, (u,v) is stored in $\rho_G(s)$ and the routing algorithm directly proceeds to the global routing phase. Since $d(s,t) > 1$, the routing table contains a middle site m and since S_u and S_v are singletons, m is a middle site on a shortest path from s to t. Algorithm 1 pushes $\ell(t)$ onto the stack and sets m as the new target. By induction, the routing scheme now routes the packet along a shortest path from s to m (i, ii), and when the packet arrives at m, the target label $\ell(t)$ is at the top of the stack (iii). Thus, Algorithm 1 executes line 5, and routes the packet from m to t. Again by induction, the packet now follows a shortest path from m to t (i, ii), and when the packet arrives at t, the stack is in the same state a before pushing $\ell(t)$ (iii). The claim follows. $\square$

Building on Lemma 5.2, we can now prove that our scheme is correct.

Lemma 5.3. *Let s, t be two sites in S. Then, the routing scheme produces a routing path $s_0, s_1, \dots, s_k$ with the following properties*

(i) $s_0 = s$ *and* $s_k = t$, *and*
(ii) the header stack is in the same state at the beginning and at the end of the routing path.

Proof. Again, we use induction on the rank of $d(s,t)$ in the sorted list of pairwise distances in $\mathrm{UD}(S)$. If $d(s,t) < c$, the claim is immediate by Lemma 5.2.

Now, consider an arbitrary pair $s, t \in S$. By construction, our routing scheme will eventually find a site $r \in S$ whose global routing table stores a WSPD-pair (u, v) that represents (r, t), together with a middle site m (m exists for $d(s, t) \geq c$ large enough). So far, the stack remains unchanged. Algorithm 1 pushes $\ell(t)$ onto the stack and sets m as the new target. By induction, the routing scheme routes the packet correctly from s to m (i), and when the packet arrives at m, the target label $\ell(t)$ is at the top of the stack (ii). Thus, Algorithm 1 executes line 5, and routes the packet from m to t. Again by induction, the packet arrives at t, with the stack in the same state as before pushing $\ell(t)$ (i, ii). $\square$

Stretch factor. The analysis of the stretch factor requires some more technical work. We begin with a lemma that justifies the term "middle site".

Lemma 5.4. *Let s, t be two sites in S with $d(s, t) \geq c \geq 14$ and let $(u, v) \in \Xi$ be the WSPD-pair that represents (s, t). If m is a middle site of a shortest path from s to $\sigma(v)$ in $\mathrm{UD}(S)$, then*

(i) $d(s, m) + d(m, t) \leq (1 + 2/c)d(s, t)$, *and*
(ii) $d(s, m), d(m, t) \leq (5/8)d(s, t)$.

Proof. For (i) we use that m is the middle site on a shortest s-$\sigma(v)$-path and we apply the triangle inequality twice to get

$$\begin{aligned} d(s, m) + d(m, t) &\leq d(s, m) + d(m, \sigma(v)) + d(\sigma(v), t) \\ &= d(s, \sigma(v)) + d(\sigma(v), t) \\ &\leq d(s, t) + 2d(\sigma(v), t) \leq (1 + 2/c)d(s, t), \end{aligned}$$

where the last inequality follows from Lemma 4.1 and the fact that $d(\sigma(v), t) \leq \mathrm{diam}(S_v)$.

For (ii) let π be a shortest path from s to $\sigma(v)$ that contains m, and let m' be the point on π with distance $d(s, \sigma(v))/2$ from s and from $\sigma(v)$ (m' may lie on an edge of π). Since the edges of π have length at most 1, there is a site m'' on π with $d(m', m'') = |m'm''| \leq 1/2$. Hence,

$$\begin{aligned} \max\{d(s, m''), d(m'', \sigma(v)\} &\leq d(s, \sigma(v))/2 + 1/2 \\ &\leq \big(1/2 + 1/(2c - 2)\big)d(s, \sigma(v)) \\ &= (c/(2c - 2))d(s, \sigma(v)), \end{aligned}$$

since the triangle inequality and Lemma 4.1 yield $d(s, \sigma(v)) \geq (1 - 1/c)d(s, t) \geq c - 1$. For $c \geq 14$, the site m'' is distinct from s and $\sigma(v)$, and the distances to and from the middle site m are at most

$$d(s, m), d(m, \sigma(v)) \leq \max\{d(s, m''), d(m'', \sigma(v))\} \leq (c/(2c - 2))d(s, \sigma(v)). \quad (3)$$

Using the triangle inequality and Lemma 4.1 again, we get $(1 - 1/c)d(m, t) \leq d(m, \sigma(v))$ and $d(s, \sigma(v)) \leq (1 + 1/c)d(s, t)$. Using both inequalities in (3) gives

$$\begin{aligned} d(s, m), d(m, t) &\leq (c/(2c - 2))(1 + 1/c)(1 + 1/(c - 1))d(s, t) \\ &= (c/(2c - 2))(1 + 2/(c - 1))d(s, t) = (c^2 + 1)/(2(c - 1)^2)d(s, t), \end{aligned}$$

and (ii) follows from $c \geq 14$. $\square$

In the next lemma, we bound the distance traveled during the local routing.

Lemma 5.5. *Let s, t be two sites in S with $d(s,t) \geq c$. Then, the total distance traveled by the packet during the local routing phase before the WSPD-pair representing (s,t) is discovered is at most $(48/c)d(s,t)$.*

Proof. Let (u, v) be the WSPD-pair representing (s,t), and let $u_0, u_1, \dots, u_k = u$ be the path in H from the leaf u_0 for s to u. Let $T_0, T_1, \dots, T_k$ and $S_0, S_1, \dots, S_k$ be the corresponding subtrees of T and sites of S . The local routing algorithm iteratively performs an Euler tour of $T_0, T_1, \dots, T_k$ (the tour of T_k may stop early). An Euler tour in T_i takes $2|S_i| - 2$ steps, and each edge has length at most 1. As described in Section 3, for $i = 0. \dots, k-1$, the WSPD ensures that

$$|S_i| \leq |S_{i+1}| - \lceil (|S_{i+1}| - 1)/6 \rceil \leq (5/6)|S_{i+1}| + 1/6 \leq (11/12)|S_{i+1}|,$$

since $|S_{i+1}| \geq 2$. It follows that the total distance for the local routing is at most

$$\sum_{i=0}^{k} (2|S_i| - 2) \leq 2|S_k| \sum_{i=0}^{k} (11/12)^i \leq 24|S_k|.$$

By Lemma 4.1, we have $d(s,t) \geq c(|S_u| - 1)$ and since $S_k = S_u$ the total distance is bounded by $24|S_u| \leq 24(d(s,t)/c + 1) \leq (48/c)d(s,t)$, where the last inequality is true for $d(s,t) \geq c$. $\square$

Finally, we can bound the stretch factor:

Lemma 5.6. *For any two sites s and t, we have $d_\rho(s,t) \leq \big(1 + \varepsilon\big)d(s,t)$.*

Proof. We show by induction on $d_\rho(s,t)$ that there is an $\alpha > 0$ with

$$d_\rho(s,t) \leq \big(1 + (\alpha/c)\log d(s,t)\big)d(s,t).$$

Since $d(s,t) \leq \operatorname{diam}(S) = D$, the lemma then follows from our choice of $c = (\alpha/\varepsilon)\log D$.

If $d(s,t) < c$, the claim follows by Lemma 5.2. If $d(s,t) \geq c$, Algorithm 1 performs a local routing to find the site r that has the WSPD-pair (u,v) representing (s,t) stored in $\rho_G(r)$. Then the packet is routed recursively from r to the middle site m and from m to t. By Lemma 5.5 the length of the routing path is $d_\rho(s,t) \leq (48/c)d(s,t) + d_\rho(r,m) + d_\rho(m,t)$, and by induction we get

$$\begin{aligned} d_\rho(s,t) \leq (48/c)d(s,t) &+ \big(1 + (\alpha/c)\log d(r,m)\big)d(r,m) \\ &+ \big(1 + (\alpha/c)\log d(m,t)\big)d(m,t). \end{aligned}$$

Since m is a middle site on a shortest r-$\sigma(v)$-path in $\mathrm{UD}(S)$, Lemma 5.4(i),(ii) and the fact that $\log(5/8) \leq -1/2$ imply

$$d_\rho(s,t) \leq 48d(s,t)/c + \Big(1 + (\alpha/c)\log(d(r,t)) - \alpha/2c\Big)(1 + 2/c)d(r,t).$$

By triangle inequality we have $d(r,t) \leq d(s,t) + \mathrm{diam}(S_u)$, so Lemma 4.1 gives

$$\begin{aligned} & d_\rho(s,t) \\ & \leq (48/c)d(s,t) \\ & \quad + \Big(1 + (\alpha/c)\log((1+1/c)d(s,t)) - \alpha/2c\Big)(1+2/c)(1+1/c)d(s,t) \\ & \leq (48/c)d(s,t) + \Big(1 + (\alpha/c)\log((1+1/c)d(s,t)) - \alpha/2c\Big)(1+4/c)d(s,t), \end{aligned}$$

for c large enough. For $\alpha > 192$, we can eliminate the first term to get

$$d_\rho(s,t) \leq \Big(1 + (\alpha/c)\log((1+1/c)d(s,t)) - \alpha/4c\Big)(1+4/c)d(s,t),$$

and since now $c \geq 192$ and hence $\log(1+1/c) \leq 1/8$, this is

$$\leq \Big(1+(\alpha/c)\log(d(s,t))-\alpha/8c\Big)(1+4/c)d(s,t) = (1+(\alpha/c)\log d(s,t))d(s,t)+\Delta,$$

with $\Delta = -(\alpha/8c)(1+4/c)d(s,t) + (4/c)d(s,t)\big(1+(\alpha/c)\log d(s,t)\big)$. It remains to show that $\Delta \leq 0$, i.e., that

$$(4/c)d(s,t)\big(1+(\alpha/c)\log d(s,t)\big) \leq (\alpha/8c)(1+4/c)d(s,t).$$

Now, since we picked $c = (\alpha/\varepsilon)\log D$ and $\alpha \geq 192$, we have

$$1 + (\alpha/c)\log(d(s,t)) \leq 2 \leq (\alpha/32)(1+4/c),$$

as desired. This finishes the proof. □

Combining Lemma 5.1 and 5.6 we obtain the following theorem.

Theorem 5.7. *Let S be a set of n sites in the plane with density δ. For any $\varepsilon > 0$, we can preprocess S into a routing scheme for* $\mathrm{UD}(S)$ *with labels of size $O(\log n)$ bits and routing tables of size $O(\delta\varepsilon^{-2}\log^2 n \log^2 D)$, where D is the diameter of* $\mathrm{UD}(S)$*. For any two sites s,t, the scheme produces a routing path with $d_\rho(s,t) \leq (1+\varepsilon)d(s,t)$ and during the routing the maximum header size is $O(\log n \log D)$. The preprocessing time is $O(n^2\log n + \delta n^2 + \delta\varepsilon^{-2} n \log n \log^2 D)$.*

5.4 Extension to Arbitrary Density

Let $1+\varepsilon$, $\varepsilon > 0$, be the desired stretch factor. To extend the routing scheme to point sets of unbounded density, we follow a strategy similar to Gao and Zhang [7, Section 4.2]: we first pick an appropriate $\varepsilon_1 > 0$, and we compute an ε_1-net $R \subseteq S$, i.e., a subset of sites such that each site in S has distance at most ε_1 to the closest site in R and such that any two sites in R have distance at least ε_1. It is easy to see that R has density $O(\varepsilon_1^{-2})$, and we would like to represent each site in S by the closest site in R. However, the connectivity in $\mathrm{UD}(R)$ might differ from $\mathrm{UD}(S)$. To rectify this, we add additional sites to R. This is done as follows: two sites $s,t \in R$ are called *neighbors* if $|st| > 1$, but there are $p, q \in S$ such that s, p, q, t is a path in $\text{UD}(S)$ and such that $|sp| \leq \varepsilon_1$ and $|qt| \leq \varepsilon_1$ (possibly, $s = p$ or $q = t$). In this case, p and q are called a *bridge* for s, t. Let R' be a point set that contains an arbitrary bridge for each pair of neighbors in R. Set $Z = R \cup R'$. A simple volume argument shows that Z has density $\delta = O(\varepsilon_1^{-3})$. Furthermore, Gao and Zhang show the following:

Lemma 5.8 (Lemma 4.8 and Lemma 4.9 in [7]). *We can compute Z in $O((n/\varepsilon_1^2)\log n)$ time, and if $d^Z(\cdot,\cdot)$ denotes the shortest path distance in $\text{UD}(Z)$, then, for any $s, t \in R$, we have $d^Z(s,t) \leq (1 + 12\varepsilon_1)d(s,t) + 12\varepsilon_1$.*

Now, our extended routing scheme proceeds as follows: first, we compute R and Z as described above, and we perform the preprocessing algorithm for Z with ε_1 as stretch parameter. We assign arbitrary new labels to the sites in $S \setminus Z$. Then, we extend the label $\ell(s)$ of each site $s \in S$, such that it also contains the label of a site in R closest to s. The label size remains $O(\log n)$.

To route between two sites $s, t \in S$, we first check whether we can go from s to t in one step (we assume that this can be checked locally in the routing function). If so, we route the packet directly. Otherwise, we have $d(s,t) > 1$. Let $s', t' \in R$ be the closest sites in R to s and to t. By construction, we can obtain s' and t' from $\ell(s)$ and $\ell(t)$. Now, we first go from s to s'. Then, we use the low-density algorithm to route from s' to t' in $\text{UD}(Z)$, and finally we go from t' to t in one step. Using the discussion above, the total routing distance is bounded by

$$d_\rho(s,t) \leq |ss'| + d_\rho^Z(s',t') + |t't|,$$

where $d_\rho^Z(\cdot,\cdot)$ is the routing distance in $\text{UD}(Z)$. By Lemma 5.6 and 5.8, this is

$$\begin{aligned} &\leq \varepsilon_1 + (1+\varepsilon_1)d^Z(s',t') + \varepsilon_1 \\ &\leq 2\varepsilon_1 + (1+\varepsilon_1)\big((1+12\varepsilon_1)d(s',t') + 12\varepsilon_1\big), \end{aligned}$$

and by using the triangle inequality twice this is

$$\leq 2\varepsilon_1 + (1+\varepsilon_1)\big((1+12\varepsilon_1)(d(s,t) + 2\varepsilon_1) + 12\varepsilon_1\big).$$

Rearranging and using $d(s,t) > 1$ yields

$$\leq (1 + 29\varepsilon_1 + 50\varepsilon_1^2 + 24\varepsilon_1^3)d(s,t) \leq (1+\varepsilon)d(s,t),$$

where the last inequality holds for $\varepsilon_1 \leq \varepsilon/103$. This establishes our main theorem:

Theorem 5.9. *Let S be a set of n sites in the plane. For any $\varepsilon > 0$, we can preprocess S into a routing scheme for $\text{UD}(S)$ with labels of $O(\log n)$ bits and routing tables of size $O(\varepsilon^{-5}\log^2 n \log^2 D)$, where D is the diameter of $\text{UD}(S)$. For any two sites s,t, the scheme produces a routing path with $d_\rho(s,t) \leq (1+\varepsilon)d(s,t)$ and during the routing the maximum header size is $O(\log n \log D)$. The preprocessing time is $O(n^2 \log n + \varepsilon^{-3}n^2 + \varepsilon^{-5} n \log n \log^2 D)$.*

Proof. The theorem follows from the above discussion and from the fact that the set Z has density $O(\varepsilon^{-3})$, by our choice of ε_1. $\square$

6 Conclusion

We have presented an efficient routing scheme for unit disk graphs that produces a routing path whose length can be made arbitrarily close to optimal. For this, we used the fact that the unit disk graph metric admits a small WSPD. Our techniques almost solely rely on properties of well-separated pairs and thus we expect our approach to generalize to other graph metrics for which WSPDs can be found. One such example is the *hop-distance* $d_h(\cdot,\cdot)$ in unit disk graphs, in which all edges have length 1. Let S be a set of sites and let $\mathrm{diam}_h(S)$ denote the diameter of S in terms of $d_h(\cdot,\cdot)$. Since $\mathrm{diam}_h(S) \le |S| - 1$ and $|st| \le d_h(s,t)$ for every two sites $s, t \in S$, the well-separation condition (1) implies also separation with respect to the hop-distance. Thus, we can also find a routing scheme that approximates the number of hops used in the routing path instead of its Euclidean length.

Various open questions remain. First of all, it would be interesting to improve the size of the routing tables. One way to achieve this might be to decrease the dependency on ε. The ε^{-5}-factor seems to be rather high. It is mostly owed to the ε^{-3}-factor we introduced in Section 5.4 when going from bounded to unbounded density. Further improvements might be on the side of the WSPD: traditional WSPDs have only $O(c^2 n)$ pairs, while the WSPD of Gao and Zhang has an additional logarithmic factor. Whether this factor can be avoided is still an open question and any improvement in the number of pairs would immediately decrease the size of our routing tables by the same amount.

Furthermore, our routing scheme makes extensive use of a modifiable header. While this is coherent with the usual model for routing schemes, the scheme of Yan et al. managed to avoid the need of a header completely. In order to be completely comparable to their result, we would need to have a routing scheme that only requires a small routing table to produce a routing path with stretch $1 + \varepsilon$.

References

1. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications. Springer, 3rd edn. (2008)
2. Callahan, P., Kosaraju, S.: A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. J. ACM 42(1), 67–90 (1995)
3. Chechik, S.: Compact routing schemes with improved stretch. In: Proc. 32nd ACM Symp. on Principles of Distributed Computing (PODC). pp. 33–41 (2013)
4. Clark, B.N., Colbourn, C.J., Johnson, D.S.: Unit disk graphs. Discrete Math. 86(1–3), 165–177 (1990)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press, third edn. (2009)
6. Fraigniaud, P., Gavoille, C.: Routing in trees. In: Proc. 28th Internat. Colloq. Automata Lang. Program. (ICALP). pp. 757–772 (2001)
7. Gao, J., Zhang, L.: Well-separated pair decomposition for the unit-disk graph metric and its applications. SIAM J. Comput. 35(1), 151–169 (2005)

8. Giordano, S., Stojmenovic, I.: Position based routing algorithms for ad hoc networks: A taxonomy. In: Cheng, X., Huang, X., Du, D.Z. (eds.) Ad Hoc Wireless Networking, Network Theory and Applications, vol. 14, pp. 103–136 (2004)
9. Gupta, A., Kumar, A., Rastogi, R.: Traveling with a Pez dispenser (or, routing issues in MPLS). SIAM J. Comput. 34(2), 453–474 (2004)
10. Narasimhan, G., Smid, M.H.M.: Geometric spanner networks. Cambridge (2007)
11. Peleg, D., Upfal, E.: A trade-off between space and efficiency for routing tables. J. ACM 36(3), 510–530 (1989)
12. Roditty, L., Tov, R.: New routing techniques and their applications. In: Proc. 34th ACM Symp. on Principles of Distributed Computing (PODC). pp. 23–32 (2015)
13. Thorup, M.: Compact oracles for reachability and approximate distances in planar digraphs. J. ACM 51(6), 993–1024 (2004)
14. Thorup, M., Zwick, U.: Compact routing schemes. In: Proc. 13th ACM Symposium on Parallel Algorithms and Architectures (SPAA). pp. 1–10 (2001)
15. Yan, C., Xiang, Y., Dragan, F.F.: Compact and low delay routing labeling scheme for unit disk graphs. Comput. Geom. 45(7), 305–325 (2012)