Greedy Routing with Bounded Stretch

Roland Flury
ETH Zurich, Switzerland
rflury@tik.ee.ethz.ch

Sriram V. Pemmaraju
The University of Iowa, USA
sriram@cs.uiowa.edu

Roger Wattenhofer
ETH Zurich, Switzerland
wattenhofer@tik.ee.ethz.ch

***Abstract*—Greedy routing is a novel routing paradigm where messages are always forwarded to the neighbor that is closest to the destination. Our main result is a polynomial-time algorithm that embeds combinatorial unit disk graphs (CUDGs – a CUDG is a UDG without any geometric information) into $O(\log^2 n)$-dimensional space, permitting greedy routing with constant stretch. To the best of our knowledge, this is the first greedy embedding with stretch guarantees for this class of networks. Our main technical contribution involves extracting, in polynomial time, a constant number of isometric and balanced tree separators from a given CUDG. We do this by extending the celebrated Lipton-Tarjan separator theorem for planar graphs to CUDGs. Our techniques extend to other classes of graphs; for example, for general graphs, we obtain an $O(\log n)$-stretch greedy embedding into $O(\log^2 n)$-dimensional space. The greedy embeddings constructed by our algorithm can also be viewed as a constant-stretch compact routing scheme in which each node is assigned an $O(\log^3 n)$-bit label. To the best of our knowledge, this result yields the best known stretch-space trade-off for compact routing on CUDGs. Extensive simulations on random wireless networks indicate that the average routing overhead is about 10%; only few routes have a stretch above 1.5.**

I. Introduction

Internet routing has been a success story. So far, emerging challenges such as address shortage have been taken care of in a timely manner, e.g. by classless inter-domain routing, or by proposing to increase the address space with IPv6. However, the networking research community fears that the success story is about to end. New threats are on the horizon, for instance mobility, security, or selfishness.

To address these upcoming threats, routing is continuously and controversially debated. The networking community is not shy to propose new architectural paradigms ("clean slate"), sometimes drastically deviating from classic link-state or distance-vector routing algorithms. This debate is at the fore-front in the community that studies infrastructure-less networks, e.g. wireless sensor networks, mesh networks, or mobile ad hoc networks, where many of these upcoming Internet challenges need to be addressed right away. We believe that a network architecture that works for wireless networks may be a viable candidate for a future Internet architecture.

One of the most promising and novel routing approach is geo-routing, also known as geometric, geographic, position-based, or location-based routing, or geo-casting. In geo-routing each node is addressed by its geographic location rather than a unique identifier. The idea is that messages are merely forwarded "in the right direction". The most basic form of geo-routing is greedy routing, where each node simply forwards a message to the neighbor closest to the destination. Greedy routing has many advantages, in particular it completely abstains from the concept of routing tables (and may be viewed as "routing without routers"), rendering routing ultimately scalable! However, there are three problems with greedy routing:

(i) The first problem is that the geographic location of a *mobile* destination may not be known. One possible solution to approach this problem may be through location service protocols [1,15], which are essentially geometric variants of a distributed hash table.

(ii) The second problem is due to the greediness itself: If the network is not dense enough it may happen that a message gets stuck in a local minimum, a node which unfortunately does not have a neighbor closer to the destination because of a node void. There is an ample body of research that shows how to deal with the problem, e.g. [6,22], however, there is also work that shows that in some models this problem is unsolvable [11]. Furthermore, forwarding a message greedily may be far from optimal in many cases, resulting in routes much longer than the optimal route.

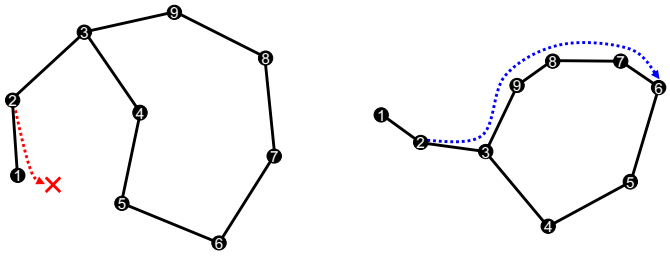

Fig. 1. Embeddings of networks need not be greedy by default. Here we have two embeddings of the same network in the Euclidean plane. For instance, there is no greedy path from vertex 2 to vertex 6 in the left embedding. As the greedy algorithm forwards the message to the neighbor which is closest (in the Euclidean L_2-norm) to the destination, it gets stuck at vertex 1. We solve this problem by constructing a greedy embedding for which the greedy forwarding scheme always succeeds. The right figure shows such a greedy embedding which ensures a greedy path between any two vertices. In our example, the greedy algorithm finds the path $2-3-9-8-7-6$ using 5 hops. The optimal path, however, is $2-3-4-5-6$ using only 4 hops. We capture this routing overhead with the ratio of the greedy path-length to the optimal path-length and call this value *stretch*. E.g. in our example, the path from 2 to 6 has stretch $5/4$. The fact that a greedy embedding may use long routes is a problem of critical importance that is solved in this work; we show that our greedy embedding introduces only a constant stretch.

(iii) The third problem is often a show stopper: If nodes are not equipped with a GPS, how can they work out their position? Several heuristics [28] and algorithms [24] have been proposed, however, their results are less than exciting. In general, inferring geometric information (e.g. coordinates) by indirect means, when technological solutions such as GPS are unavailable, is a fundamentally intractable problem [7,21].

In this paper we follow a radically different approach that essentially solves problems (ii) and (iii). We do not address mobility (i) – that needs to be subject of future work.

In a nutshell, our approach is as follows: Given the connectivity information of a wireless network, modeled as a combinatorial unit disk graph (CUDG) we assign each node a polylog-dimensional virtual coordinate such that greedy routing is always provably successful, between any source-destination pair, e.g. see Figure 1. In contrast to previous work our greedy routing provides a bounded stretch (see Figure 10 for experimental results supporting this claim). Previous work on greedy routing was done for *geometric* UDGs where an embedding of the network is known (i.e. each vertex knows its coordinates). In this work, we consider the more intricate problem of greedy routing on CUDGs, which are UDGs *without any position information*. We believe that our approach is valuable for real world networks where position information is not readily available. Our result can be adapted to a compact routing scheme with routing tables of size $\mathrm{O}\big(\log^3 n\big)$. Furthermore, the proposed approach extends to general graphs as well, with a slightly higher overhead. We support our results by extensive simulations and show that the constant overhead is on average only about 10%. For a detailed comparison of our contributions we refer to the related work section.

II. Related Work

There is a huge body of research on routing schemes of which we overview only the most relevant subset.

One of the most classical greedy routing protocols is *geographic routing*, where all vertices of a network know their own position and the positions of their neighbors. Given the position of the destination vertex, a message can be delivered by repeatedly forwarding it to the neighbor which is geographically closest to the destination. To circumvent voids in the network, several versions of *face routing* have been proposed [6,13,23]. Whereas a geographic routing scheme with guaranteed delivery is possible for 2-dimensional networks, Durocher et al. [11] have shown that there is no deterministic geographic routing algorithm for 3-dimensional networks. With this paper, we overcome this fundamental limit of geographic routing by embedding the network into a $\mathrm{O}\big(\log^2 n\big)$-dimensional space, enabling purely greedy forwarding.

Papadimitriou and Ratajczak conjectured that any planar 3-connected network has a greedy 2-dimensional embedding [28]. They also showed that any planar 3-connected newtork has a greedy embedding in 3 dimensions, for which an embedding algorithm was described in [10]. The Papadimitriou-Ratajczak conjecture was very recently, settled by Moitra and Leighton [27] who present a polynomial-time algorithm for constructing a greedy embedding of a given 3-connected planar graph. In this paper, we study the embedding of a *combinatorial* UDG where only the connectivity information but no position information of the vertices is given. The absence of such geometrical information eliminates the possibility of planarizing the network and using a triangulation technique. The embedding of a UDG in combination with routing has been studied widely [8,18,31,33], but none of these approaches guarantees a greedy embedding. In fact, finding an exact embedding is NP-hard [7] and cannot be approximated arbitrarily well [21]: If non-neighboring vertices must have a distance larger than 1, it can be shown that there may be neighbors with distance $\sqrt{3/2}$. The best known approximation algorithm for this problem is described in [30] and may induce distances in $\mathrm{O}\big(\log^{2.5} n\big)$. Even if each vertex knows the exact distance to any of its neighbors or the exact angles at which the neighbors are located, the problem of finding an exact embedding remains NP-hard [4,8].

As a way avoiding these difficulties and finding embeddings for CUDGs, anchor based routing was introduced, e.g. [14,16,35]. The main idea of these routing schemes is that a few vertices are elected as anchor nodes and each vertex stores the length of its shortest path to all anchor nodes. The set of these distances can be seen as a virtual coordinate on which greedy routing can be applied. To ensure a greedy embedding, however, $\Theta(n)$ anchors need to be chosen in the worst case [35], which renders this approach unattractive.

Kleinberg proposed a greedy embedding of arbitrary networks into the hyperbolic plane [20], which was improved by Eppstein et al. such that coordinates of the embedding only need $\mathrm{O}(\log n)$ bits [12]. As of this writing, such greedy embeddings into hyperbolic space are only known for trees, i.e. given a general network, only a spanning tree thereof is embedded. The reduced connectivity information of a spanning tree has severe consequences for the routing performance: Close-by vertices in the network may be far away on the tree on which the routing takes place, introducing a linear worst-case stretch. To the best of our knowledge, our paper proposes the first greedy embedding with guaranteed sublinear stretch. For CUDGs, a generalization of planar graphs, our embedding guarantees a constant stretch and for general graphs, the stretch is in $\mathrm{O}(\log n)$.

Compact routing is one of the most prominent routing paradigms. Compact routing studies the tradeoff between the efficiency of a routing algorithm (i.e. the stretch) and its space requirements for the routing tables. For general graphs, there is a stretch-k routing algorithm with an average routing table size of $\mathrm{O}\big(k^3 n^{1/k} \log n\big)$ and $\mathrm{O}(\log n)$ bit labels [29]. Renaming (labeling) of the vertices is a widely used technique to reduce the routing table size. In fact, any routing algorithm that does not rename the vertices and requires a stretch below 3 may need routing tables of $\Omega(n)$ bits [17]. Many impressive compact routing results have been developed for constant doubling metrics with doubling dimension α. These networks

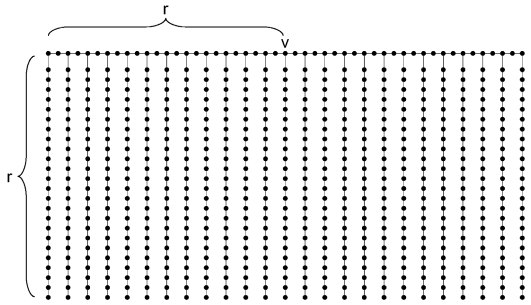

Fig. 2. Most compact routing results are analyzed for the general class of doubling metrics and provide excellent results if the metric has a small constant doubling dimension. Unfortunately, many reasonable metric spaces induced by even common network models do not have a constant doubling dimension. For example, consider the hop-metric induced by the UDG shown in this figure: A ball of radius $2r$ around v covers all vertices, whereas $\sqrt{n}$ balls of radius r are needed. Note that the if we take the nodes in the network to be embedded in the plane as shown, then the induced Euclidean metric has constant doubling dimension. But, for network routing the hop-metric may be more relevant.

have the property that the set of vertices in any ball of radius $2r$ can be covered by at most 2^α balls of radius r. We refer the interested reader to [2] for an overview of the results in this area. Note that many graph classes, including UDGs, do not have a constant doubling dimension (see Figure 2).

The work by Abraham et al. on growth bounded networks [3] is probably closest in spirit to our result. The authors obtain a compact routing scheme with stretch ϵ and $\mathrm{O}\Big(\frac{1}{\epsilon}^{\mathrm{O}(\Delta)} \log^5 n\Big)$ bit routing tables without renaming the vertices, where Δ is the growth factor of the network. Greedy routing schemes, such as the one presented in this paper, are close relatives to compact routing schemes: as the greedy routing decision is based solely on the coordinates of the neighbors, this neighborhood information could be stored locally, which is equivalent to the routing table in compact routing. Using this transformation, we show that our greedy routing for CUDGs can be transformed to a compact routing scheme with $\mathrm{O}\big(\log^3 n\big)$ bit routing tables, beating any known compact routing scheme for this class of networks.

III. Background, Results, and Approach

The focus of this paper is on *combinatorial unit disk graphs*. For points p and q in Euclidean space we use $|pq|$ to denote the Euclidean distance in L_2 norm between p and q. A graph $G = (V, E)$ is a *unit disk graph (UDG)* if there is an embedding $\phi : V \to \mathbb{R}^2$ of the vertices of G into the Euclidean plane such that $\{u, v\} \in E$ iff $|\phi(u)\phi(v)| \leq 1$. The embedding ϕ is called a *realization* of G. UDGs are widely used as models of wireless networks and this is what motivates our focus on this class of graphs. A UDG may be specified by its realization and in such a setting coordinates of all vertices are known. Alternately, a UDG may be specified as a combinatorial object, e.g., a collection of vertices and a collection of edges. In such a setting the neighbors of each node are known, but no geometric information such as node coordinates, pairwise Euclidean distances, etc. are known. These two specifications are fundamentally different from a computational point of view. Given a realization of a UDG, it is trivial to construct a combinatorial representation of it; on the other hand given a combinatorial representation of a UDG, it is impossible (unless P = NP) [7] to compute its realization. We work in the latter setting, in which we are given a UDG merely as a combinatorial object, with no recourse to any geometric information. To emphasize this we refer to these graphs as *combinatorial* UDGs. This makes our approach robust to situations in which geometric information is missing or is only partially available or is erroneous.

Let $f : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be a function that assigns to each pair of points in $\mathbb{R}^d$ a non-negative real. For any vertex v, let $N(v)$ denote the set of neighbors of v. A *greedy embedding* of an undirected graph $G = (V, E)$ into the space $(\mathbb{R}^d, f)$ is a mapping $\phi : V \to \mathbb{R}^d$ such that for any pair $s, t \in V$ of distinct vertices, there exists a vertex $u \in N(s)$ such that $f(u, t) < f(s, t)$. Greedy embeddings formally characterize embeddings for which the greedy routing algorithm will guarantee message delivery and were first defined in this manner by Papadimitriou and Ratajczak [28]. These authors were concerned with greedy embeddings into $\mathbb{R}^2$ and the function f was taken to be the Euclidean distance between pairs of points. In this paper, d will typically be poly-logarithmic in n, the number of vertices of G, and f will be the "min-max" function defined as follows. Let c be a factor of d and let $s = (s_1, s_2, \ldots, s_d)$ and $t = (t_1, t_2, \ldots, t_d)$ be points in $\mathbb{R}^d$. For each j, $0 \leq j < d/c$ let

$$D_j = \max_{c\cdot j+1 \leq i \leq c\cdot(j+1)} |s_i - t_i|.$$

Then the function $\texttt{min-max}_c$ is defined as

$$\texttt{min-max}_c(s, t) = \min_{0 \leq j < d/c} D_j.$$

The $\texttt{min-max}_c$ function essentially views the space $\mathbb{R}^d$ as composed of $\frac{d}{c}$ c-dimensional spaces, takes the L_∞ norm of the projections of s and t into those spaces, and finally takes the smallest of those L_∞ distances as the "distance" between s and t. Often c will either be irrelevant or be understood from the context and we will usually write $\texttt{min-max}_c$ as $\texttt{min-max}$. The $\texttt{min-max}$ function turns out to be a natural "distance" measure for us because the space $\mathbb{R}^d$ into which we embed G is obtained by "gluing" together a bunch of lower dimensional subspaces. Note that even though using the $\texttt{min-max}$ function is not as natural as using the L_1, L_2 or the L_∞ norm, it is computationally as easy to deal with as any of these norms.

Let ϕ be a greedy embedding of G into $(\mathbb{R}^d, f)$. An st-path $P = (s = v_1, v_2, \ldots, v_k = t)$ in G is called a *greedy st-path* (or just a *greedy path*, if s and t are clear from the context) if for each i, $1 \leq i < k$, $v_{i+1} = \text{argmin}_{u \in N(v_i)} f(u, t)$. In other words, among all neighbors of v_i, the vertex v_{i+1} is the neighbor that is closest to the destination t. A greedy embedding ϕ of G into $(\mathbb{R}^d, f)$ is said to have *stretch* ρ if for all distinct $s, t \in V$ and for all greedy st-paths P, $|P| \leq \rho \cdot d_G(s, t)$. Here $|P|$ denotes the number of hops in path

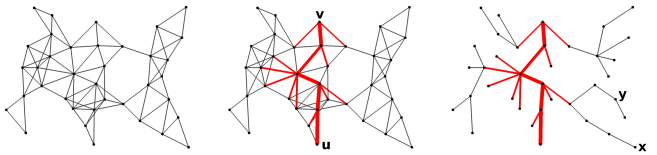

Fig. 3. We consider the network on the left. In a first step, we find a separator which ensures that any connected component after removing the separator has size at most $2/3$ of the total network size (middle figure). Note that this separator is a shortest path connecting u and v along with the 1-hop neighborhood of this path. From this separator, we grow a tree by repeatedly adding adjacent vertices not already contained in the tree, which finally gives the first tree of the desired tree cover (right figure). On this top level, we only ensure good routing paths from one side of the separator to the other. For instance, the vertices x and y in the right bottom corner are 5 hops apart on the tree even though their graph distance is 2. To fix this issue, additional trees are built on the components.

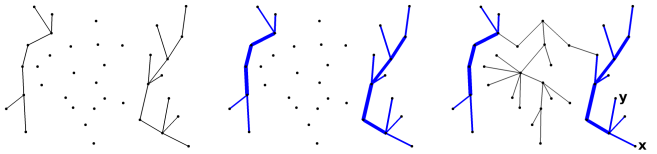

Fig. 4. After removing the first separator, we obtain several connected components (left figure), for each of which we find a separator (middle figure). In this case, the separator already covers all vertices of the components s.t. the tree of the components is equal to the separator. In the final step, all trees constructed in this step are connected by reinserting the removed separator and connecting the 3 trees. This gives the second tree of the tree cover. Note that for larger networks, the recursion would continue for the components created by the separators found in this step. Also notice that this tree connects the two vertices x and y much better.

P and $d_G(s,t)$ denotes the shortest path hop-distance between s and t in G. We will use the latter notation widely throughout the paper with different graphs serving as the subscript of "d" to denote the shortest hop-distance between pairs of vertices in that graph.

A. Results

The main result in this paper is a polynomial-time algorithm for constructing a low-dimensional greedy embedding with constant stretch of a given CUDG. Our algorithm constructs, for any given n-vertex CUDG, a greedy embedding into $(\mathbb{R}^d, \texttt{min-max})$, where $d = \mathrm{O}(\log^2 n)$. Furthermore, each coordinate of each vertex has size $\mathrm{O}(\log n)$ bits, implying a total of $\mathrm{O}(\log^3 n)$ bits for each vertex-label. Therefore our solution can be easily transformed into a constant-stretch, labeled, *compact routing scheme* for UDG hop-metrics that uses labels of size $\mathrm{O}(\log^3 n)$ per vertex[1].

The main ingredient of our approach is the construction of small-size constant-stretch *tree covers*. Given a graph $G = (V, E)$, a *tree cover* of size k and stretch ρ is a family $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of spanning subtrees of G such that for every $u, v \in V$, there is a tree T_i such that $d_G(u,v) \leq \rho \cdot d_{T_i}(u,v)$.

[1]To obtain the compact routing scheme, each node locally stores the labels of its neighbors in a routing table. By only embedding an MIS of the network and by routing on this virtual topology, only $\mathrm{O}(1)$ neighbors need to be stored.

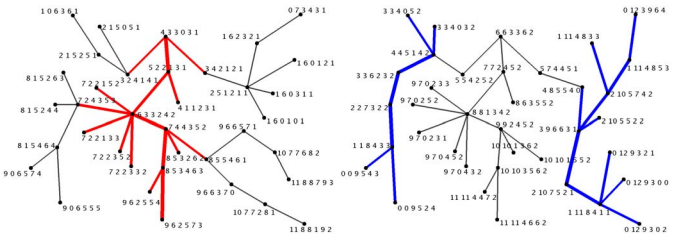

Fig. 5. Given the tree cover, each tree is embedded in a $\mathrm{O}(\log n)$-dimensional space. Note that each coordinate uses at most $\log n$ bits, resulting in coordinates of size $\mathrm{O}(\log^2 n)$ bits. In our example, 6 dimensions per tree are enough.

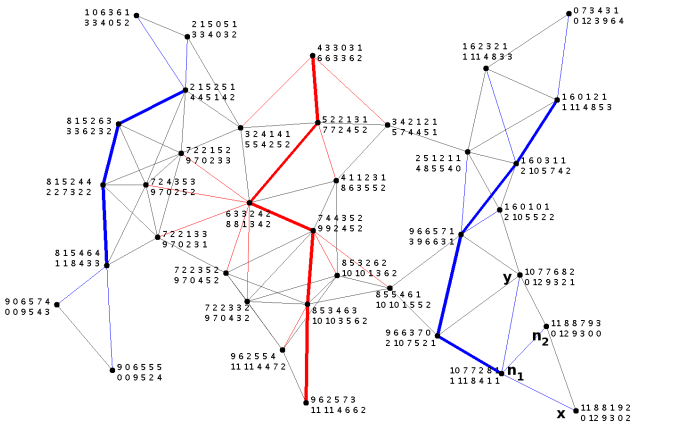

Fig. 6. Finally, the coordinates assigned for the trees are combined to a single coordinate for each vertex. Using at most $\mathrm{O}(\log n)$ trees, the size of the coordinates is bounded by $\mathrm{O}(\log^3 n)$ bits. To send a message from x to y, x compares the label of y to its neighbors n_1 and n_2 and forwards to the closer neighbor. The comparison is done tree by tree, e.g. for n_1 we have $L_\infty(1077281, 1077682) = 4$ for the first tree and $L_\infty(1118411, 0129321) = 1$ for the second tree, giving an estimated distance of 1 from n_1 to y. Similarly, the estimated distance from n_2 to y is 1 as well. Therefore, x forwards the message to either of its neighbors.

A tree cover $\mathcal{T}$ with stretch 1 is called an *exact* tree cover[2]. The main technical contributions of our paper are as follows.

- We show that every CUDG has an $\mathrm{O}(\log n)$-size tree cover with constant stretch and that such a tree cover can be computed in polynomial time even for CUDGs. This is comparable to the result of Gupta et al. [19] who show how to construct a constant-stretch $\mathrm{O}(\log n)$-size tree cover for planar graphs.
- The above tree cover result is obtained via an extension to CUDGs of the celebrated Lipton-Tarjan Separator theorem for planar graphs [26]. The algorithm implied by the Lipton-Tarjan Theorem makes explicit use of a planar embedding of the given planar graph. Similarly, recent work by Chen et al. [9] constructs separators for UDGs, but again with explicit use of a given UDG realization. As mentioned in Section II, recovering a realization of a CUDG is intractable and our result is the first to show that Lipton-Tarjan type separators can be constructed for UDGs even without any geometric information.

[2]In literature on tree covers, see e.g., Awerbuch and Peleg [5], the trees in the tree cover are not required to be spanning. Furthermore, the size of a tree cover is defined as the maximum number of trees that a vertex participates in. For the purposes of greedy routing, it is more convenient to require all trees to be spanning and therefore the size of a tree cover can simply be defined as the number of trees in the collection.

Algorithm 1: GREEDY-EMBED(G)

Step 1
Construct a *tree cover* $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of G with stretch ρ (Figures 3 and 4).

Step 2
Embed each tree T_i *isometrically* into $\mathbb{R}^b$, where $b = c \cdot \log n$ for some constant c. For this we use a simple and well-known algorithm due to Linial et al. [25], Theorem 5.3. We note that using this algorithm guarantees that every coordinate of every vertex uses at most $\log n$ bits. Let $\phi_i(v)$ denote the coordinates of vertex v obtained by constructing the embedding of T_i (Figure 5).

Step 3
Output $\phi := \phi_1 \cdot \phi_2 \cdots \phi_k$, the "concatenation" of the mappings ϕ_i. In other words, $\phi(v)$ consists of $k \cdot c \log n$ coordinates, obtained by writing the coordinates in $\phi_1(v)$, followed by the coordinates in $\phi_2(v)$, followed by the coordinates in $\phi_3(v)$, and so on (Figure 6).

Our approach works for any class of graphs for which small-size, small-stretch tree covers can be constructed. For example, via the result of Awerbuch and Peleg (Lemma 6.8 in [5]) we can compute an $\mathrm{O}(\log n)$-stretch greedy embedding of arbitrary graphs into $(\mathbb{R}^d, \texttt{min-max})$, where $d = \mathrm{O}\big(\log^3 n\big)$.

B. Overall Approach

At a high level, our algorithm, which we call GREEDY-EMBED, consist of the three steps of Algorithm 1, which are depicted in the Figures 3,4,5, and 6. The following theorem about algorithm GREEDY-EMBED drives the rest of the paper.

Theorem 1: Algorithm GREEDY-EMBED takes as input an n-vertex graph G and returns a greedy embedding with stretch ρ of G into $(\mathbb{R}^d, \texttt{min-max}_b)$, where $d = k \cdot c \log n$ and each coordinate of each vertex uses $\log n$ bits.

Proof: Let s and t be an arbitrary pair of vertices in G and let $T_i \in \mathcal{T}$ be a tree containing a shortest st-path, among all trees in $\mathcal{T}$. Then $d_{T_i}(s,t) \leq \rho \cdot d_G(s,t)$. Let u be the neighbor of s along the unique, simple st-path in T_i. Then the L_∞ distance between $\phi_i(u)$ and $\phi_i(t)$ is smallest over all neighbors of s and over all trees. The $\texttt{min-max}_b$ function will therefore lead the message to vertex u. Now the message is at a node that is at most $\rho \cdot d_G(s,t) - 1$ hops from the destination t. Continuing this argument we see that the message will be greedily routed to t in at most $\rho \cdot d_G(s,t)$ hops. □

For arbitrary graphs we utilize the above theorem as follows. Awerbuch and Peleg [5] have shown that every graph has a tree cover of stretch $\mathrm{O}(\log n)$ and size $\mathrm{O}\big(\log^2 n\big)$ and such a tree cover can be computed in polynomial time. This yields $\rho = \mathrm{O}(\log n)$ and $k = \mathrm{O}\big(\log^2 n\big)$ and leads to the following corollary.

Corollary 2: There is a polynomial time algorithm that can compute for any n-vertex graph an $\mathrm{O}(\log n)$-stretch greedy embedding into $(\mathbb{R}^d, \texttt{min-max})$ where $d = \mathrm{O}\big(\log^3 n\big)$. Each coordinate of each vertex uses $\mathrm{O}(\log n)$ bits.

In the next section we show our main technical result: for any CUDG we can compute in polynomial time a constant-stretch $\mathrm{O}(\log n)$-size tree cover.

IV. Greedy Embeddings of CUDGs

We now show how to construct a constant-stretch tree cover $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ with $k = \mathrm{O}(\log n)$ of any given CUDG $G = (V, E)$. Before we describe the algorithm, we point out that it is not possible to reduce the stretch down to $(1 + \varepsilon)$ for arbitrarily small ε, while keeping the number of trees in the tree cover polylogarithmic in n. To see this suppose that G is an n-vertex clique. If we want the tree cover $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ to have stretch smaller than 2 then every edge $\{u, v\}$ in G needs to appear in some tree T_i. Since G has $n(n-1)/2$ edges and each tree T_i has $n - 1$ edges, the tree cover has size $k \geq n/2$.

To obtain a tree cover of G we use a recursive algorithm inspired by the approach of Gupta et al. [19], based on *isometric separators*; Gupta's algorithm yields a constant-stretch $\mathrm{O}(\log n)$-size tree cover for any planar graph. For any graph $G = (V, E)$ a vertex-subset $V' \subseteq V$ is called a *1/3-2/3 separator* of G if a largest connected component in $G \setminus V'$ has size at most $\frac{2}{3} \cdot |V|$. Such "balanced" separators have played a fundamental role in algorithm design for a variety of problems [32]. Given a graph $G = (V, E)$, a *k-part, isometric separator* of G is a family $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ of subtrees (not necessarily spanning) of G such that

1) $S = \cup_i V(S_i)$ is a 1/3-2/3 separator for G. Here $V(S_i)$ is the vertex set of the tree S_i.
2) For each i and each pair of vertices $u, v \in V(S_i)$, $d_{S_i}(u,v) = d_G(u,v)$. In other words, each of the subtrees S_i contain the shortest paths between their constituent vertices and hence are isometric to the restriction of G to $V(S_i)$.

This definition is due to Gupta et al. [19]. For a CUDG G we obtain a 2-part tree separator that is not quite isometric, but preserves distances approximately. Specifically, we show that for some constant $\rho \geq 1$, $d_{S_i}(u,v) \leq \rho \cdot d_G(u,v)$ for all $u, v \in V$. We will call such a family of separators a *2-part stretch-ρ separator*. In the following subsection we extend the Lipton-Tarjan Separator Theorem for planar graphs to prove the following result. This is the main technical contribution of the paper.

Theorem 3: Every CUDG has a 2-part, stretch-3 separator and such a separator can be computed in polynomial time.

The proof of the above theorem implies Algorithm 2, which we call CUDG-SEPARATOR, for computing a 2-part, stretch-3 separator of G. In a sense the proof of the above theorem should be thought of as a "proof of correctness" for Algorithm 2 and it shows that Step 2 of CUDG-SEPARATOR will always terminate successfully.

The reason Algorithm 2 returns $T'(u)$ and $T'(v)$ rather than just $T(u)$ and $T(v)$, is worth mentioning here. For planar graphs, the edge $\{u, v\}$ (assuming it exists) along with paths $T(u)$ and $T(v)$ forms a cycle C that separates its interior from its exterior. Since a UDG is not planar, it is possible

Algorithm 2: CUDG-SEPARATOR(G)

Step 1
Pick an arbitrary vertex r of G and construct a breadth-first search tree T rooted at r.

Step 2
For any $u \in V$, let $T(u)$ denote the (unique) path in T from r to u and let $\mathtt{N}(T(u))$ denote the set of vertices in the closed neighborhood of $T(u)$. In other words, $\mathtt{N}(T(u))$ contains all vertices in $T(u)$ along with vertices that have a neighbor on $T(u)$. For every pair of vertices $u, v \in V$ construct G_{uv} by deleting from G the vertices in $\mathtt{N}(T(u)) \cup \mathtt{N}(T(v))$. Terminate this step successfully on finding a G_{uv} such that every connected component in G_{uv} has size at most $2|V|/3$.

Step 3
From $T(u)$ construct a tree $T'(u)$ by taking each vertex v in $\mathtt{N}(T(u))$ that is not in $T(u)$ and connecting v to an arbitrary neighbor in $T(u)$. Similarly construct $T'(v)$ and return $T'(u)$ and $T'(v)$.

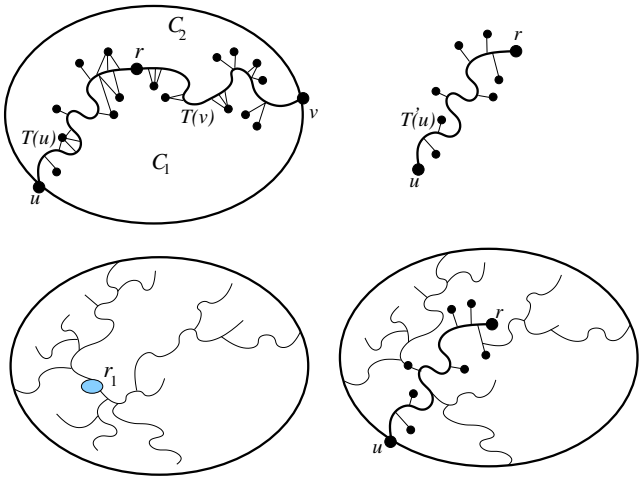

Fig. 7. The top left figure shows shortest paths $T(u)$ and $T(v)$. The removal of the vertex set $\mathtt{N}(T(u)) \cup \mathtt{N}(T(v))$ separates the graph into components C_1 and C_2. The top right figure shows the "caterpillar" tree $T'(u)$ constructed from $T(u)$ and its neighborhood. This tree has stretch 3. The bottom left figure shows a BFS tree T_1^* rooted at the "super vertex" r_1 obtained by contracting tree $T'(u)$ into a single vertex. Finally, the bottom right figure shows the tree S_1 obtained from T_1^* be expanding r_1 back to $T'(u)$.

to "escape" the cycle C by just crossing it. However, UDGs have the nice and well-known property (stated precisely in the following lemma) that if a pair of edges cross then at least one pair of end points of the crossing edges are neighbors.

Lemma 4.1: Consider an arbitrary realization of a UDG $G = (V, E)$. Suppose that $\{u, v\}$ and $\{x, y\}$ cross in the realization. Then at least one of the edges $\{u, x\}$, $\{x, v\}$, $\{v, y\}$ or $\{u, y\}$ is an edge in G.

This lemma implies that if the cycle C is crossed by an edge $\{u, v\}$ then either u or v is a neighbor of a vertex in C.

Algorithm CUDG-SEPARATOR for finding a 2-part, stretch-3 separator of G can be recursively applied to obtain a tree cover of G as follows. Let $T_1 := T'(u)$ and $T_2 := T'(v)$ be the trees returned by the above algorithm. For each $i = 1, 2$, contract the vertices of T_i into a "super" vertex r_i and construct a BFS tree T_i^* rooted at r_i. Expand r_i back to T_i in T_i^* to obtain a tree called S_i. S_1 and S_2 are the first two trees in the tree cover being constructed. For an illustration of this step see Figure 7. Now recurse on the connected components in G_{uv}. More specifically, suppose that the connected components in G_{uv} are $C_1, C_2, \ldots, C_t$ and suppose that for each C_i we find $\{T_1^i, T_2^i\}$, a 2-part, stretch-3 separator promised by Theorem 3 and obtain the corresponding trees $\{S_1^i, S_2^i\}$. The collection $\{S_1^1, S_1^2, \ldots, S_1^t\}$ is a forest and we arbitrarily add vertices and edges to this forest to obtain a spanning tree of G. We similarly extend the forest $\{S_2^1, S_2^2, \ldots, S_2^t\}$ to a spanning tree of G, thus obtaining two more spanning trees at the second level of recursion.

Assuming Theorem 3, the above recursive algorithm yields a tree cover whose properties are proved in the following theorem.

Theorem 4: Given an n-vertex CUDG G, there is a polynomial-time algorithm that computes a constant-stretch tree cover $\{T_1, T_2, \ldots, T_k\}$ of G with $k = \mathrm{O}(\log n)$.

Proof: Consider an arbitrary non-neighboring pair s, t of vertices in G. There are two cases depending on whether s and t are separated by any of the trees constructed by Algorithm CUDG-SEPARATOR.

1) Vertices s and t are not separated by any tree (in any level of the recursion). In this case s and t together lie on some tree T. Since T is a shortest path along with nodes in the neighborhood of this path, it is easy to see that $d_T(s, t) \leq 3 \cdot d_G(s, t)$.
2) Suppose that s and t are separated by some tree (in some level of recursion). The shortest path in G between s and t intersects some tree T. In this case also, using arguments such as those used by Gupta et al. (Theorem 5.1, [19]) we obtain that there is a tree T in the tree cover such that $d_T(s, t) \leq 3 \cdot d_G(s, t)$.

□

Theorem 5: There is a polynomial time algorithm that can compute for any n-vertex CUDG an $\mathrm{O}(1)$-stretch greedy embedding into $(\mathbb{R}^d, \mathtt{min\text{-}max})$ where $d = \mathrm{O}\big(\log^2 n\big)$. Each coordinate of each vertex uses $\mathrm{O}(\log n)$ bits.

A. Constructing Isometric Separators

This subsection is devoted to the proof of Theorem 3. Start by fixing a realization of G; note that this is only for the purposes of the proof and in general the problem of obtaining a realization of a given CUDG is NP-hard [7]. From this realization we extract a graph H as follows. Replace each edge crossing in the realization of G by a vertex, to obtain a planar embedding of a graph. Add edges to this embedding so that every face becomes a 3-cycle. We call this graph H (see Figure 8 for an illustration). In the rest of this section, when we refer to H, we will sometimes refer to the combinatorial object H and sometimes to the particular planar embedding of H extracted from the realization of G. The vertex set $V(H)$ of H can be partitioned in two sets:

(i) *virtual* vertices: which are vertices that replaced edge crossings in the realization of G.
(ii) *G-vertices*: which are vertices that H inherited from G.

There is a natural connection between edges in H and edges in G, which we now make precise. Let $\{u, v\}$ be an edge in G. Walk along the straight line segment from u to v and suppose that we encounter edge crossings $c_1, c_2, \ldots, c_t$, in that order. Note that the c_i's are all virtual vertices in H. The edges $\{u, c_1\}, \{c_1, c_2\}, \ldots, \{c_t, v\}$ of H are called *pieces* of edge $\{u, v\}$. If an edge $\{x, y\}$ (of H) is a piece of edge $\{u, v\}$ (of G) then edge $\{u, v\}$ is said to be the *parent* of $\{x, y\}$. Note that edge $\{u, v\}$ may not be crossed by any edge in the realization of G; in this case $\{u, v\}$ appears as itself in H. Of course, there are edges in H, namely the ones that were added in order to triangulate the embedding, that do not have any parents in G. Refer to Figure 8 for an illustration.

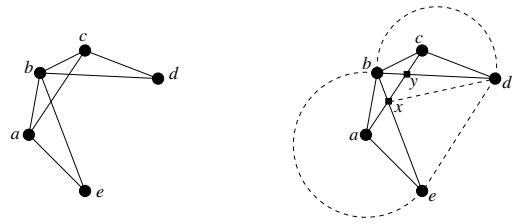

Fig. 8. A realization of a UDG G is shown on the left. The planar, triangulated graph H, obtained from G, is shown on the right. Vertices x and y in H are virtual vertices; the remaining vertices are G-vertices. Edges $\{a, x\}$, $\{x, y\}$, and $\{y, c\}$ are pieces of edge $\{a, c\}$ and edge $\{a, c\}$ is the parent of these.

Let r be an arbitrary vertex in G and let T be a breadth first search (BFS) tree of G rooted at r. Let u be a vertex in G. Recall the notation $T(u)$ (i.e., the unique path from r to u in T) and $\mathtt{N}(T(u))$ (i.e., closed neighborhood of $T(u)$) from the previous section. Any edge in T is called a *tree edge*; the remaining edges in G are called *non-tree* edges. Furthermore, if an edge $\{x, y\}$ in H is a piece of a tree edge in G, then $\{x, y\}$ is called a *tree edge* of H. Any edge of H that is not a tree edge is called a *non-tree edge*.

The overall approach of the Lipton-Tarjan construction [26] is to consider a non-tree edge $\{u, v\}$ of G, add it to T, and consider the *separating cycle* C induced by $\{u, v\}$ and the tree edges in $T(u)$ and $T(v)$. Since G is planar, the cycle C separates the vertex set of G into two sets: the set of vertices that lie in the interior of C and the set of vertices that lie in the exterior of C. The removal of C disconnects these two sets and Lipton and Tarjan [26] show that for some non-tree edge $\{u, v\}$ the cardinalities of the interior and the exterior are "balanced," i.e., neither set exceeds $\frac{2}{3} \cdot |V|$ in size. We follow this overall approach, but our construction and proof are technically more intricate for two reasons: (i) the graph G is not planar and (ii) we do not have access to a realization of G and therefore we do not have access to H.

With some amount of work, we extend the Lipton-Tarjan notion of separating cycles as follows. Consider a non-tree edge $\{x, y\}$ of H. Depending on the "types" of the vertices x and y there are three cases to consider:

1) Both x and y are G-vertices. In this case, the *separating cycle* induced by edge $\{x, y\}$ is simply the concatenation of $T(x)$, $T(y)$, and $\{x, y\}$. The corresponding separator is the set of vertices $\mathtt{N}(T(x)) \cup \mathtt{N}(T(y))$.

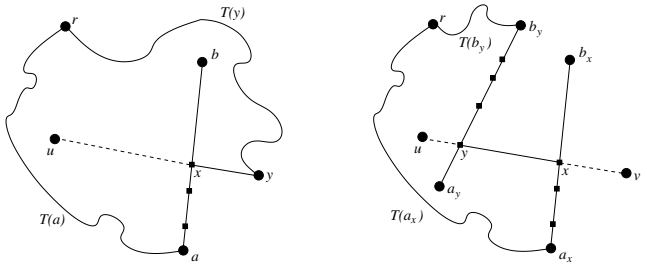

Fig. 9. This figure shows the separating cycle induced by a non-tree edge $\{x, y\}$ of H. On the left, we show the case in which x is a virtual vertex, but y is not. On the right we show the case in which both x and y are virtual vertices. In this example, $x^* = a_x$ and $y^* = b_y$

2) Exactly one of x or y, say x, is a virtual vertex of H. Let $\{u, y\}$ be the parent of edge $\{x, y\}$. Let $\{a, b\}$ be the edge in G that crosses $\{u, y\}$ to result in the virtual vertex x. Refer to Figure 9 (left) for an illustration. Pick a vertex from among a and b, whichever is farther from the root r, breaking ties arbitrarily. Suppose this vertex is a. The *separating cycle* induced by edge $\{x, y\}$ is obtained by concatenating $T(a)$, $T(y)$, $a \rightsquigarrow x$, and $\{x, y\}$. Here we use $a \rightsquigarrow x$ to denote the simple path in H from a to x composed of pieces of edge $\{a, b\}$. The corresponding separator we pick is the set of vertices $\mathtt{N}(T(a)) \cup \mathtt{N}(T(y))$.
3) Both vertices x and y are virtual vertices of H. Let $\{u, v\}$ be the parent edge of $\{x, y\}$ and let $\{a_x, b_x\}$ and $\{a_y, b_y\}$ be the edges in G that cross $\{u, v\}$ respectively to yield virtual vertices x and y. See Figure 9 (right) for an illustration. Let x^* be the one of the two vertices, a_x and b_x, whichever is farther from r. Similarly, let y^* be one of the two vertices, a_y and b_y, whichever is farther from r. The *separating cycle* induced by $\{x, y\}$ is obtained by concatenating $T(x^*)$, $T(y^*)$, $x^* \rightsquigarrow x$, $y^* \rightsquigarrow y$, and $\{x, y\}$. The corresponding separator is the set of vertices $\mathtt{N}(T(x^*)) \cup \mathtt{N}(T(y^*))$.

As in the case of the Lipton-Tarjan approach [26], we would like to identify an "interior" and an "exterior" of the separating cycle induced by each non-tree edge. However, the notion of interior and exterior of the separating cycle are not as clear as in the planar graph case since a path $T(u)$ may itself be self-intersecting in the fixed realization of G. We now make these notions precise. Consider a separating cycle C induced by a non-tree edge $\{x, y\}$ of H. A point p in the plane is in the *interior* of C if every ray originating at p intersects some point in C; otherwise p is said to be in the exterior of C. A vertex v of G is said to be in the *interior* (respectively, *exterior*) of C if in the realization of G, v lies in the interior (respectively, exterior) of C. Based on this definition of interior and exterior of C, we prove the following lemma.

Lemma 4.2: Let $\{x, y\}$ be a non-tree edge of H, let C

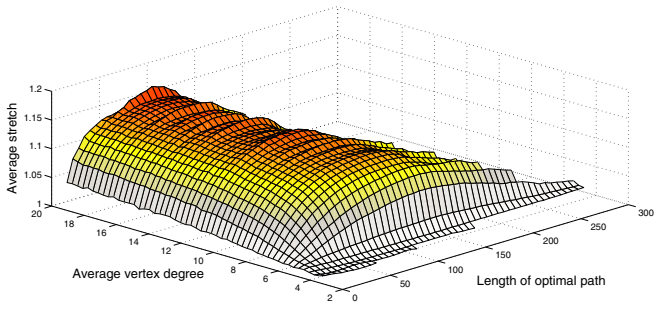

Fig. 10. Average stretch as a function of the average vertex degree and the length of the optimal path. Note that our greedy embedding guarantees an upper bound on the stretch for any UDG. In practice, the stretch is always below this bound and depends not only on the network topology, but also on the length of the optimal route.

be the separating cycle induced by $\{x, y\}$ and let S be the separator corresponding to $\{x, y\}$. Then there is no path in $G \setminus S$ between any vertex in $V \setminus S$ in the interior of C and any vertex in $V \setminus S$ in the exterior of C.

Proof: Let $u \in V \setminus S$ be a vertex in the interior of C and let $v \in V \setminus S$ be a vertex in the exterior of C. Now suppose there is a path P in $G \setminus S$ between u and v. Then some edge $\{a, b\}$ of P crosses some edge $\{c, d\}$ of C. Lemma 4.1 tells us that at least one of $\{a, c\}$, $\{c, b\}$, $\{b, d\}$, or $\{a, d\}$ is an edge in G, implying that either a or b is in the neighborhood of C. Since the separator S contains the neighborhood of C, either a or b does not exist in $G \setminus S$. □

Now we would like to show that for some non-tree edge $\{x, y\}$, the separating cycle induced by $\{x, y\}$ partitions the vertex set in a "balanced" manner. It is worth emphasizing here that showing this does not mean that our algorithm needs to know x and y in order to find a "balanced" cycle separator. In each of the three cases enumerated above, the separator has the form $\mathtt{N}(T(u)) \cup \mathtt{N}(T(v))$ for some vertices u and v in G. Thus considering $\mathtt{N}(T(u)) \cup \mathtt{N}(T(v))$ for all pairs of vertices u and v of G, will lead to a "balanced" separator and this is what is done in Algorithm `CUDG-SEPARATOR`. We conclude with the following lemma whose proof (skipped due to space constraints) is similar to the proof of Lemma 2 in Lipton and Tarjan [26].

Lemma 4.3: For a non-tree edge $\{x, y\}$ of H let $C(x, y)$ and $S(x, y)$ respectively denote the separating cycle induced by $\{x, y\}$ and the corresponding separator. There exists a non-tree edge $\{x, y\}$ of H such that a largest component in $G \setminus S(x, y)$ has size at most $\frac{2}{3}|V|$.

V. Simulation

Extensive simulations on large, randomly generated networks show that on average our embedding algorithms provide extremely low stretch routes, the average stretch being much smaller than the worst case guarantees proved in the previous sections. We considered a range from very sparse to dense networks for each of which we sampled random source and destination vertex pairs. The average routing overhead is around 10% and the worst stretch we ever encountered is 3. Of course, such a simulation is much weaker than our formal proof (Theorem 5) on the maximum stretch. However, it provides a good approximation for the average case and also helps to validate the overall approach.

We used the sinalgo simulator [34] to construct the desired network topologies which all live in an area of size 100×100 units. For each network, we deployed n vertices at random positions, constructed the UDG with a transmission radius of 1 unit, and kept the giant connected component for the embedding. Varying n from 10,000 to 60,000 yields networks in the desired density and size range. In the following, we characterize the resulting networks by their average degree of the vertices, which grows linearly with n. With over 2,000 networks and 50,000 random source and destination pairs, the simulation analyzed over 10^8 routing paths with an average length of 74 hops.

Figure 10 shows the average stretch as a function of the network density (the average degree of the vertices) and the length of the optimal path for which the samples were taken. It is interesting to note that the average stretch partially depends on the length of the chosen route: close vertex pairs tend to have at least one tree connecting them nearly optimally, yielding a low stretch for short routes. Longer routes, on the other hand, have a higher chance that the greedy algorithm first needs to travel along one or several trees which do not connect optimally (or close to optimal) to the destination, resulting in an increased stretch. While Figure 10 seems to indicate that the average stretch falls as the length of optimal paths increases, this decrease may be due to a sampling bias: in our experiments distant s-t pairs show up relatively infrequently and as a result our experiments may be encountering costly situations rarely, as well.

The density of the network is another key parameter for the average stretch of our greedy routing algorithm. For sparse networks, the tree cover tends to include many optimal paths resulting in a nearly optimal stretch. With increasing density, the trees miss more and more shortcuts, the average stretch grows. This growth, however, is stopped when an increased connectivity only adds additional edges but no additional shortcuts. Figure 10 shows that this critical density is reached with an average vertex degree of approximately 10.

Figure 11 shows the maximum stretch we have encountered for any of the 10^8 sampled routing paths. The highest peaks stem from the shortest routes, for which even a short detour may result in a high stretch factor. However, we know from Figure 10 that on average, only very few routes suffer such a high stretch.

A comparison of our simulation results with related work is rather difficult. For instance in [20], Kleinberg illustrated his work with a set of very small networks of 50 vertices, not covering the challenging networks where the routing stretch may be linear in the network size. Kuhn et al. [22] compared several greedy routing algorithms for UDGs *with* position information. Their simulation for varying network sizes indicates that the average stretch for sparse networks

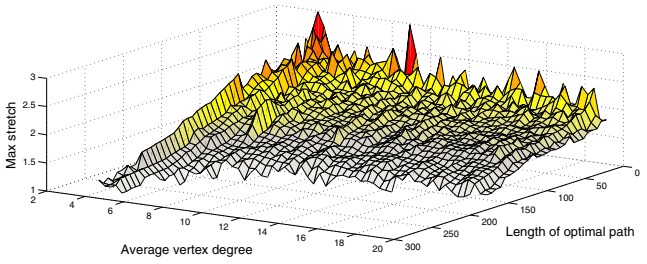

Fig. 11. Maximum stretch found over all sampled routing paths.

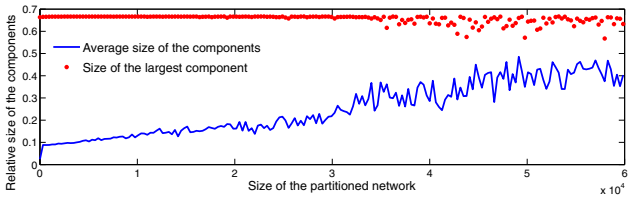

Fig. 12. When removing a separator, one or several connected components remain in the network. This plot shows the relative size of these components, compared to the network which was separated.

is well above 3 for small networks and increases linearly with the size of the network.

For completeness, we also include some statistics on the size of the connected components that we obtained during the simulation. Figure 12 shows that indeed, after removing a separator, none of the connected components had a size larger than $2/3$ of the original network. It is interesting to note that the average size of the components increases with the density, which shows that in sparse networks, we tend to obtain several components and in the dense networks only 2 or 3.

VI. Future Work

Even though our embedding algorithm is fast, it is centralized and seems inherently "global" in nature (e.g., the use of breadth-first search trees is quite important). Whether there are local variants of this embedding algorithm that lend themselves to lightweight distributed implementations, is a question that interests us. Some aspects of our algorithm, for example, the construction of isometric separators, depends explicitly on the fact that UDGs have a 2-dimensional realization. As a result, it seems that extending our approach to unit ball graphs in 3-dimensions might be a significant challenge.

References

[1] I. Abraham, D. Dolev, and D. Malkhi. LLS: a Locality Aware Location Service for Mobile Ad Hoc Networks. In *DIALM-POMC*, pages 75–84, 2004.

[2] I. Abraham, C. Gavoille, A. Goldberg, and D. Malkhi. Routing in Networks with Low Doubling Dimension. In *ICDCS*, 2006.

[3] I. Abraham and D. Malkhi. Name Independent Routing for Growth Bounded Networks. In *SPAA*, 2005.

[4] J. Aspnes, D. K. Goldenberg, and Y. R. Yang. On the Computational Complexity of Sensor Network Localization. In *ALGOSENSORS*, pages 32–44, 2004.

[5] B. Awerbuch and D. Peleg. Sparse partitions (extended abstract). In *FOCS*, pages 503–513, 1990.

[6] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing With Guaranteed Delivery in Ad Hoc Wireless Networks. In *DIAL-M*, pages 48–55, 1999.

[7] H. Breu and D. G. Kirkpatrick. Unit disk graph recognition is NP-hard. *Comput. Geom. Theory Appl.*, 9(1-2):3–24, 1998.

[8] J. Bruck, J. Gao, and A. Jiang. Localization and routing in sensor networks by local angle information. In *MobiHoc*, pages 181–192, 2005.

[9] J. Chen, A. Jiang, I. A. Kanj, G. Xia, and F. Zhang. Separability and topology control of quasi unit disk graphs. In *INFOCOM*, pages 2225–2233, 2007.

[10] M. B. Chen, C. Gotsman, and C. Wormser. Distributed Computation of Virtual Coordinates. In *SCG*, pages 210–219, 2007.

[11] S. Durocher, D. Kirkpatrick, and L. Narayanan. On Routing with Guaranteed Delivery in Three-Dimensional Ad Hoc Wireless Networks. In *ICDCN*, pages 546–557, 2008.

[12] D. Eppstein and M. T. Goodrich. Succinct greedy graph drawing in the hyperbolic plane. In *Proc. 16th Int. Symp. Graph Drawing*, 2008.

[13] Q. Fang, J. Gao, and L. J. Guibas. Locating and Bypassing Routing Holes in Sensor Networks. In *INFOCOM*, 2004.

[14] Q. Fang, J. Gao, L. J. Guibas, V. Silva, and L. Zhang. GLIDER: Gradient landmark-based distributed routing for sensor networks. In *INFOCOM*, pages 339–350, 2005.

[15] R. Flury and R. Wattenhofer. MLS: An Efficient Location Service for Mobile Ad Hoc Networks. In *MobiHoc*, 2006.

[16] R. Fonseca, R. Fonseca, S. Ratnasamy, S. Ratnasamy, D. Culler, D. Culler, S. Shenker, S. Shenker, I. Stoica, and I. Stoica. Beacon vector routing: Scalable point-to-point routing in wireless sensornets. In *NSDI*, 2005.

[17] C. Gavoille and M. Gengler. Space-Efficiency for Routing Schemes of Stretch Factor Three. *J. Parallel Distrib. Comput.*, 61(5):679–687, 2001.

[18] C. Gotsman and Y. Koren. Distributed graph layout for sensor networks. In *GD*, 2004.

[19] A. Gupta, A. Kumar, and R. Rastogi. Traveling with a pez dispenser (or, routing issues in mpls). *SIAM J. Comput.*, 34(2):453–474, 2004.

[20] R. Kleinberg. Geographic routing using hyperbolic space. In *INFOCOM*, pages 1902–1909, 2007.

[21] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Unit Disk Graph Approximation. In *DIALM-POMC*, 2004.

[22] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger. Geometric ad-hoc routing: of theory and practice. In *PODC*, pages 63–72, 2003.

[23] F. Kuhn, R. Wattenhofer, and A. Zollinger. Asymptotically Optimal Geometric Mobile Ad Hoc Routing. In *DIAL-M*, 2002.

[24] K. M. Lillis and S. V. Pemmaraju. On the Efficiency of a Local Iterative Algorithm to Compute Delaunay Realizations. In *WEA*, pages 69–86, 2008.

[25] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995.

[26] R. J. Lipton and R. E. Tarjan. A Separator Theorem for Planar Graphs. *SIAM J. Comput.*, 36(2):177–189, 1979.

[27] A. Moitra and T. Leighton. Some results on greedy embeddings in metric spaces. In *Proceedings of the 49th Annual Symposium on Foundations of Computer Science (FOCS)*, 2008.

[28] C. H. Papadimitriou and D. Ratajczak. On a conjecture related to geometric routing. *Theor. Comput. Sci.*, 344(1):3–14, 2005.

[29] D. Peleg and E. Upfal. A Trade-Off between Space and Efficiency for Routing Tables. *J. ACM*, 36(3):510–530, 1989.

[30] S. V. Pemmaraju and I. A. Pirwani. Good quality virtual realization of unit ball graphs. In *ESA*, pages 311–322, 2007.

[31] A. Rao, S. Ratnasamy, C. H. Papadimitriou, S. Shenker, and I. Stoica. Geographic Routing without Location Information. In *MOBICOM*, 2003.

[32] A. L. Rosenberg and L. S. Heath. *Graph Separators, with Applications*. Springer: Frontiers in Computer Science, 2000.

[33] Y. Shang and W. Ruml. Improved MDS-based localization. In *INFOCOM*, 2004.

[34] Sinalgo. *Simulator for Network Algorithms*. http://dcg.ethz.ch/projects/sinalgo, 2007.

[35] M. Wattenhofer, R. Wattenhofer, and P. Widmayer. Geometric routing without geometry. In *SIROCCO*, pages 307–322, 2005.