Succinct Greedy Geometric Routing Using Hyperbolic Geometry

David Eppstein and Michael T. Goodrich, *Fellow*, *IEEE*

Abstract—We describe a method for performing greedy geometric routing for any n-vertex simple connected graph G in the hyperbolic plane, so that a message M between any pair of vertices may be routed by having each vertex that receives M pass it to a neighbor that is closer to M's destination. Our algorithm produces *succinct* embeddings, where vertex positions are represented using $O(\log n)$ bits and distance comparisons may be performed efficiently using these representations. These properties are useful, for example, for routing in sensor networks, where storage and bandwidth are limited.

Index Terms—Greedy routing, hyperbolic geometry, autocratic weight-balanced trees, dyadic tree metric space.

1 Introduction

Viewing network routing as an algorithmic problem, we are given an n-vertex simple connected graph G representing a communication network, where each vertex in G is a computational agent, such as a sensor, smart phone, base station, PC, or workstation, and the edges in G represent communication channels. The routing problem is to set up an efficient means to support message passing between the vertices in G.

The traditional way to do routing is via protocols, like link-state/OSPF or distance-vector/RIP (e.g., see [1], [2]), that set up routing tables for each vertex v in G. Each such routing table has size n (represented using $\Theta(n \log n)$ bits) for each vertex v in G, which allows v to determine to which of its neighbors it should send a message destined for another node w in G. Such a solution allows for a simple message-forwarding policy, but it is space inefficient and it requires considerable setup overhead.

There is a recent nontraditional approach to solving the routing problem, however. In this new approach, called *geometric routing* [3], [4], [5], [6], [7] or *geographic routing* [8], the graph G is embedded in a geometric metric space $\mathcal{S}$ in the standard way, so that vertices are associated with points in $\mathcal{S}$ and each edge is the locus of points along the shortest path between its two endpoints. For example, if $\mathcal{S}$ is the euclidean plane $\mathbf{R}^2$, then edges would be straight line segments in this approach; edges are allowed to cross each other, in contrast to other applications of geometric graph embedding such as graph drawing and VLSI routing. Routing is then performed by having any vertex v holding a message destined for a node w use a simple policy involving only the coordinates of v and w and the coordinates and topology of v's neighbors to determine the neighbor of v to which v should forward the message. It is important to note that even in applications where the vertices of G come with predefined geometric coordinates (e.g., GPS coordinates of smart sensors), the embedding of G need not take these coordinates into consideration, and, in fact, many known geometric routing schemes ignore preexisting coordinates and create a new embedding using only the graph structure.

Perhaps the simplest geometric routing policy imaginable is the *greedy* one:

- If a vertex v receives a message M with destination w, v should forward M to any neighbor of v in G that is closer than v to w.

We are interested in this paper in *greedy* geometric routing in an arbitrary network. That is, given a network, we would like to determine an embedding of the network that causes greedy routing to be guaranteed to succeed no matter which two vertices are selected as the source and destination of a routing problem.

Following Papadimitriou and Ratajczak [9], we say that a *distance decreasing path* from v to w in a geometric embedding of G is a path $(v_1, v_2, \ldots, v_k)$ such that $v = v_1$, $w = v_k$, and

$$d(v_i, w) > d(v_{i+1}, w),$$

for $i = 1, 2, \ldots, k-1$. A *greedy embedding*[1] of a graph G in a geometric metric space $\mathcal{S}$ is a drawing of G in $\mathcal{S}$ such that a distance decreasing path exists between every pair of vertices in G.

Unfortunately, not every embedding is greedy. It is not uncommon for geometric graph embeddings in euclidean spaces, and even for noncrossing embeddings of graphs in the plane, to have "lakes" and "voids" that make greedy routing impossible in some cases [9]. See Fig. 1 for an example; in this figure, the only network path from c to a passes through b, a node that is farther away from the eventual destination.

- *The authors are with the Department of Computer Science, University of California, Irvine, CA 92697-3435.*
  E-mail : {eppstein, goodrich}@ics.uci.edu.

Manuscript received 2 Oct. 2009; revised 27 Feb. 2010; accepted 17 July 2010; published online 8 Dec. 2010.
Recommended for acceptance by J. Chen.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2009-10-0499.
Digital Object Identifier no. 10.1109/TC.2010.257.

1. Note that this formalism is equivalent to the informal notion that defines a greedy embedding as one in which greedy routing always works. The formalism based on distance decreasing paths is a little easier to work with than this informal notion, however, so it is the one we use in this paper.

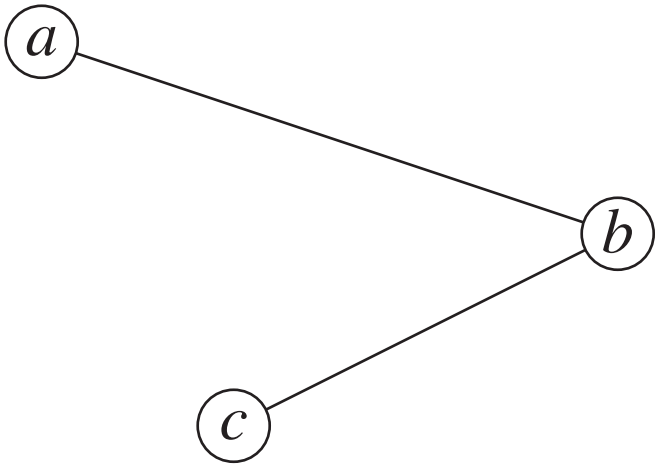

Fig. 1. An embedding of a graph with three vertices and two edges into the euclidean plane that is not greedy. To route a message from c to a along graph edges, it must pass through the vertex b, which is farther from the eventual destination.

Indeed, in any fixed-dimensional euclidean space, a star (a tree with one nonleaf node) with sufficiently many leaves cannot be embedded so that all paths are greedy. The *kissing number* of the space is the maximum number of nonoverlapping unit spheres that can be tangent to a common unit sphere; kissing numbers in two and three dimensions are 6 and 12, respectively, and in general the kissing number is upper bounded by an exponential function of the dimension [10]. In a star with a number of leaves that exceeds the kissing number, some two leaves would form an angle less than $\pi/3$ at the star's center node, for otherwise an arrangement of spheres with the spheres' centers at distance two from the center node in the direction of each leaf would violate the kissing number bound. But, when such a tight angle is formed, the embedding cannot be greedy: if, as in the figure, abc is a tight angle, with $|bc| \leq |ab|$, then the route from c to a via b cannot be greedy. Thus, in order to find greedy embedding schemes for arbitrary connected graphs, it will not suffice to use euclidean spaces of any bounded dimension; we must instead consider noneuclidean spaces.

1.1 Prior Related Work

Early papers on geometric routing include work by Bose et al. [3], who extract a planar subgraph of G, embed it, and then route a message from v to w by a more complicated greedy algorithm than the one considered here: their routing algorithm marches around the faces intersected by the line segment vw using a subdivision traversal algorithm of Kranakis et al. [11]. Karp and Kung [4] introduce a hybrid scheme, which combines a greedy routing strategy with face routing. Similar hybrid schemes were subsequently studied by several other researchers [5], [6], [7], [12]. An alternative hybrid augmented greedy scheme is introduced by Carlsson and Eager [13].

Rao et al. [14] introduce the idea of embedding a graph using virtual coordinates and doing a pure greedy routing strategy with that embedding, although they provide no theoretical guarantee. Papadimitriou and Ratajczak [9] continue this line of work on greedy routing, studying greedy schemes that are guaranteed to work, and they conjecture that euclidean greedy embeddings exist for any graph containing a 3-connected planar spanning subgraph. They present a greedy algorithm for embedding 3-connected planar graphs in $\mathbf{R}^3$ based on a specialization of Steinitz's Theorem for circle packings, albeit with a nonstandard notion of distance. Dhandapani [15] provides an existence proof that two-dimensional euclidean greedy embeddings of 3-connected triangulations are always possible, but he does not provide a polynomial-time algorithm to find them. Chen et al. [16] study methods for producing two-dimensional euclidean greedy embeddings for graphs containing power diagrams, Lillis and Pemmaraju [17] provide similar methods for graphs containing Delaunay triangulations, and Ghosh and Sinha [18] study relationships of greedy drawings and the weights of maximum and minimum spanning trees. It is not clear whether either of these greedy geometric routing methods in euclidean spaces runs in polynomial time, however. Nevertheless, Leighton and Moitra [19] give a polynomial-time algorithm for producing two-dimensional euclidean greedy embeddings of 3-connected planar graphs, resolving the conjecture of Papadimitriou and Ratajczak, and a similar result was independently discovered by Angelini et al. [20] for 3-connected triangulations. The corresponding two-dimensional problem for greedy embeddings of arbitrary graphs in noneuclidean geometries also has a solution, in that Kleinberg [8] provides a polynomial-time algorithm for embedding any graph in the hyperbolic plane so as to allow for greedy routing using the standard metric for hyperbolic space. It is Kleinberg's work on two-dimensional hyperbolic greedy routing that most closely forms the basis for our own research.

1.2 The Importance of Succinctness

Unfortunately, all of the algorithms mentioned above for producing greedy embeddings, including the hyperbolic-space solution of Kleinberg [8] and the euclidean-space solutions of Leighton and Moitra [19] and Angelini et al. [20], contain a hidden drawback that makes them ill suited for the motivating application of geometric routing. Namely, each of the greedy embeddings mentioned above uses vertex coordinates with representations requiring $\Omega(n \log n)$ bits in the worst case. Thus, these greedy approaches to geometric routing have the same space usage as traditional routing table approaches. Worse, the above greedy embedding schemes have larger bandwidth requirements, since they use message headers of length $\Omega(n \log n)$ bits in the worst case, whereas traditional routing table approaches use message headers of size $\Theta(\log n)$ bits. Since the *raison d'tre* for greedy embeddings is to improve and simplify traditional routing schemes, if embeddings are to be useful for geometric routing purposes, they should be *succinct*, that is, they should use vertices with representations having a number of bits that is polylogarithmic in n and they should allow for efficient distance comparisons using these representations.

We are, in fact, not the first to make this observation. Muhammad [21] specifically addresses succinctness, observing that a method based on extracting a planar subgraph of the routing network G and performing a hybrid greedy/face-routing algorithm in this embedding can be implemented using only $O(\log n)$ bits for each vertex coordinate, since planar graphs can be drawn in $O(n) \times O(n)$ grids [22], [23].

For euclidean spaces, Goodrich and Strash [24] give a succinct embedding strategy in the plane for 3-connected planar graphs, which uses $O(\log n)$ bits per vertex coordinate. For noneuclidean spaces, Maymounkov [25] provides a greedy embedding method for three-dimensional hyperbolic space using vertices that can be represented with $O(\log^2 n)$ bits. His work leaves open the existence of succinct greedy embeddings for two-dimensional noneuclidean spaces, however, as well as whether there are succinct noneuclidean greedy embeddings that use only $O(\log n)$ bits per vertex.

1.3 Our Results

In this paper, we show that succinct greedy embeddings in two-dimensional noneuclidean spaces are possible and that such embeddings can be represented using an asymptotically optimal number of bits. In particular, we show that any n-vertex simple connected graph can be embedded in the hyperbolic plane with coordinates that can be represented using $O(\log n)$ bits so as to support greedy geometric routing between any pair of vertices, using a standard distance metric for hyperbolic space. Our scheme is constructive, runs in polynomial time, and allows the distance between any two vertices to be calculated simply and efficiently from our representation of their coordinates. In addition, our greedy embedding scheme is based on the combination of a number of interesting graph drawing and data structuring techniques.

The remainder of the paper is organized as follows: In Section 2, we describe the *autocratic weight-balanced tree*, a new binary tree data structure that forms a key component of our constructions. Using a new algorithm for median split tree construction, we show how to construct autocratic weight-balanced trees in linear time. In Section 3, we review a previously used scheme from the data structures literature, *heavy path decomposition* which allows an arbitrary tree to be decomposed into a set of paths that connect to each other with the structure of a binary tree. In Section 4, we combine these two techniques to form $O(\log n)$-bit labels for the vertices of an arbitrary graph, and a simple distance function on these labels, that is guaranteed to support greedy routing; the space of possible labels forms a novel and interesting but nongeometric metric space that we call the *dyadic tree metric space*. As we show, an arbitrary graph may be labeled in this way in linear time. In Section 5, we show how to interpret these labels as coordinates of points in the hyperbolic plane in such a way as to preserve the greedy routing properties of the embedding; we then finish with a conclusions section.

2 Autocratic Weight-Balanced Trees

One of the new data structuring techniques, we use in our greedy embedding scheme is a data structure that we call *autocratic weight-balanced binary trees*. These are first and foremost weight-balanced binary trees, which store weighted items at their leaves so that the depth of each item of weight w_i is $O(\log W/w_i)$, where W is the sum of all weights. Just as important, however, is that they are *autocratic*, by which we mean that the distance from any leaf v to any other leaf w is strictly greater than the distance from the root to w, where tree distance is measured by simple path length. Of course, this autocratic property implies that such binary trees are not proper, in that we allow for some internal nodes in such trees to have only one child. The advantage of autocratic trees, in the context of greedy embeddings, is that in going from any leaf to any other leaf, the root is always closer to the destination than is the source. The challenge, of course, is to have a structure that is both autocratic and weight balanced.

2.1 Weight-Balanced Trees

It turns out that there is a fairly simple method for turning any weight-balanced binary tree into an autocratic weight-balanced tree. So suppose we are given an ordered collection of k items with weights $\{w_1, w_2, \ldots, w_k\}$, such that each $w_i \geq 1$. As implied above, if we store these items at the leaves of a binary tree T, we say that T is *weight balanced* if the depth of each item i is $O(\log W/w_i)$, where $W = \sum_i w_i$.

There are several existing schemes for producing a weight-balanced binary tree so that an in order listing of the items stored at its leaves preserves the given order (e.g., see [26], [27]). For our purposes, it is more important to find a good weight-balanced tree efficiently than to find the best possible weight-balanced tree. Therefore, we now describe a simple technique for finding weight-balanced trees in linear time, based on the *median split tree* of Sheil [28].

In a median split tree, for a given sequence of weighted items, the partition into left and right subtrees is performed in such a way as to make the balance of weights of these two subtrees as even as possible: if L and R represent the total weights of the two subtrees, then the partition is chosen so that $|L - R|$ is less than or equal to the corresponding quantity for any other partition of the input sequence into two contiguous subsequences. The left and right subtrees are then constructed in the same way recursively. A single split in a median split tree may be arbitrarily unbalanced, but only in the case that there is an item in the middle of the sequence with high weight. For any item x from the input sequence, with weight w_i, each split either causes x to become part of a subsequence with at most $2/3$ of the weight of the subsequence it was previously part of, or has a high weight element that will be removed in the very next split causing x to become part of a subsequence with at most $1/3$ of the weight of the subsequence it was previously part of. Thus, x may participate in at most $O(\log W/w_i)$ splits until it has been placed at a leaf in the tree, from which it follows that the median split tree is weight balanced. Sheil [28] described an $O(n \log n)$ time algorithm for finding median split trees; as we now show, this can be improved to linear.

We represent the input sequence of weights in an array, and compute a second array of the prefix sums of the input weights; using these prefix sums, we may test in constant time how evenly any particular partition splits the weight sequence. A subsequence of the input is represented by the pair of indices of its start and end values; any prefix sum of the values within that subsequence may be computed in constant time as the prefix sum at the corresponding position of the input sequence minus the prefix sum stored one position prior to the start of the subsequence. To find the position of the most even split, we perform a doubling search from both ends of the sequence: that is, for $i = 1, 2, 3, \ldots$ we consider the split in which the left subsequence has 2^i elements and the split in which the right

subsequence has 2^i elements, until finding the smallest i such that a subsequence of 2^i elements starting at one end of the sequence contains at least half of the total weight. After finding i in this way, we switch to a binary search within the subsequence of 2^i elements found by the doubling search in order to find the optimal split point.

We analyze this algorithm using a recurrence in which $T(n)$ denotes its running time. The algorithm above will partition the sequence into two subsequences, of sizes n_1 and n_2, where we may assume without loss of generality that $n_1 \geq n_2$; the doubling search will find an i such that $2^i \geq n_2 > 2^{i-1}$, in i steps, and the subsequent binary search will also take i steps. After finding the best split, the algorithm continues recursively within each of the two split subsequences. Therefore, its total time is governed by a recurrence of the form

$$T(n) = T(n_1) + T(n_2) + k_1 \log n_2 + k_2.$$

(Here, $\log$ represents the base-2 logarithm.) As we now show, the solution to this recurrence (with the assumptions $n_1 \geq n_2$ and $n_1 + n_2 = n$) is $O(n)$. More specifically, we show by induction that there exists a constant c such that $T(n) \leq cn - k_1(1 + \log n) - k_2$. The base case, when $n = 1$, is trivial for sufficiently large c. Otherwise,

$$\begin{aligned} T(n) &\leq cn_1 - k_1(1 + \log n_1) - k_2 \\ &\quad + cn_2 - k_1(1 + \log n_2) - k_2 \\ &\quad + k_1 \log n_2 + k_2 \\ &= cn - k_1(2 + \log n_1) - k_2 \\ &\leq cn - k_1(1 + \log n) - k_2. \end{aligned}$$

Thus, the overall median split tree construction runs in linear time, as claimed.

Although we do not need it for our greedy embedding algorithm, we observe that the same technique can be used for linear time construction of the variant of median split trees in which the input items are placed in the internal nodes of the tree as well as at its leaves.

2.2 Making a Tree Autocratic

Suppose, then, that T is an ordered weight-balanced tree, and let r denote the root of T. To convert T into an autocratic weight-balanced tree, T', we replace the edge connecting each leaf v to its parent with a path of length

$$1 + d_T(r, \text{parent}(v)),$$

where $d_T(v, w)$ denotes the length of the path from v to w in the tree T. That is, we insert a number of "dummy" nodes between each leaf and its parent that is equal to the depth of its parent. (See Fig. 2.)

This transformation increases the depth of each leaf in T by less than a factor of two and it keeps the depth of all other nodes in T unchanged. Thus, if the depth of a leaf storing item i in T was previously at most $c \log W/w_i$, for some constant c, then the depth of the corresponding leaf in T' is less than $2c \log W/w_i$, which is still $O(\log W/w_i)$. Given that T was weight-balanced, this implies that T' is a weight-balanced tree. More importantly, we have the following lemma.

Lemma 1. *The above transformation of a weight-balanced tree T produces an autocratic weight-balanced tree T'.*

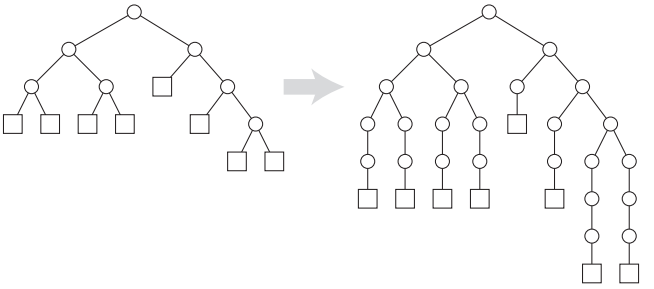

Fig. 2. Converting a weight-balanced binary tree into an autocratic weight-balanced binary tree.

Proof. We have already observed that the tree T' is weight-balanced. So we have yet to show that T' is autocratic. First, observe that, by the construction of T', if u is an ancestor in T of a leaf v, then in T' we have the following:

$$d_{T'}(u, v) = d_T(r, v) + d_T(u, v) - 1.$$

In particular, we have the following:

$$d_{T'}(r, v) = 2d_T(r, v) - 1.$$

Let v and w be two leaves in T'. Furthermore, let u be the least common ancestor of v and w in T'. Then

$$\begin{aligned} d_{T'}(v, w) &= d_{T'}(u, v) + d_{T'}(u, w) \\ &= d_T(r, v) + d_T(u, v) - 1 \\ &\quad + d_T(r, w) + d_T(u, w) - 1 \\ &= d_T(r, u) + d_T(u, v) + d_T(u, v) - 1 \\ &\quad + d_T(r, w) + d_T(u, w) - 1 \\ &= (d_T(r, u) + d_T(u, w)) + d_T(r, w) \\ &\quad + 2d_T(u, v) - 2 \\ &= 2d_T(r, w) + 2d_T(u, v) - 2 \\ &\geq 2d_T(r, w) \\ &> d_{T'}(r, w). \end{aligned}$$

Thus, T' is an autocratic weight-balanced tree. □

Therefore, we have a way of constructing for any ordered set of weighted items an autocratic weight-balanced tree for that set. The expanded tree T' may have a greater-than-linear number of nodes (for instance, when T is a complete binary tree, T' has $\Theta(n \log n)$ nodes), but our greedy embedding algorithm will not need to construct T' explicitly. We will use such data structures as auxiliary components in the structures we discuss next.

3 Heavy Path Decompositions

Let T be a rooted ordered tree of arbitrary degree and depth having n nodes. Sleator and Tarjan [29] describe a scheme, which we call the *heavy path decomposition*, for decomposing T into a hierarchical collection of paths (see also [30] for an alternative path decomposition scheme with similar properties). Their scheme works as follows: For each node v in T, let $n(v)$ denote the number of descendants in the subtree rooted at v, including v itself. For each child-to-parent edge, $e = (v, w)$ in T, label e as a *heavy* edge if $n(v) > n(w)/2$. Otherwise, label e as a *light* edge. Connected components of heavy edges form paths,

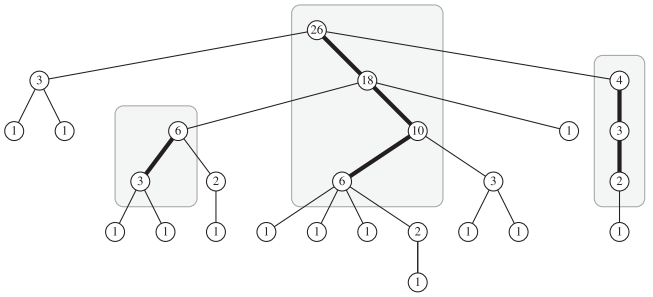

Fig. 3. The heavy path decomposition of a tree. Three heavy paths are shown; the remaining 17 nodes form degenerate length 0 heavy paths. The numbers shown at each node v are the sizes $n(v)$ of the subtree rooted at that node.

called *heavy paths*, which may in turn have many incident light edges. As a degenerate case, we also consider the zero length path consisting of a single node in T incident only to light edges as a heavy path.

Note that the size of a subtree at least doubles every time we traverse a light edge from a child to a parent. (See Fig. 3.) Thus, if we compress every heavy path in T to a single "super" node, preserving the relative order of the nodes, then we define a tree, Z, of depth $O(\log n)$. Of course, the nodes in Z can have arbitrary degree. Nevertheless, for data structuring purposes, following Alstrup et al. [31], we may replace each vertex v in Z having d children $v_1, v_2, \ldots, v_d$ with a weight-balanced binary tree that uses the $n(v_i)$ values as weights. The useful property of this substitution is that any leaf-to-root path P in the resulting binary tree, Z'', will have length $O(\log n)$, since the lengths of the subpaths of P in the weight-balanced binary trees traversed in P form a telescoping sum that adds up to $O(\log n)$. That is, such a sum can be written as a value proportional to something of the form

$$\begin{aligned}&\log n_0 + \log n_1/n_0 + \log n_2/n_1 + \cdots + \log n/n_k\\ &\quad = \log n_0 + \log n_1 - \log n_0 + \cdots + \log n - \log n_k\\ &\quad = \log n,\end{aligned}$$

where each n_i is an $n(v_i)$ value.

In our case, we use autocratic weight-balanced binary trees for the substitutions of high-degree super nodes in Z, so as to define a binary tree of depth $O(\log n)$. This construction will prove essential for our greedy embedding scheme. Before we present this geometric embedding, however, we first present a combinatorial greedy embedding in a completely contrived metric space, which we will subsequently show how to turn into a greedy embedding in the hyperbolic plane using the standard hyperbolic metric.

4 Greedy Embeddings in the Dyadic Tree Metric Space

Let G be a graph with n vertices and m edges for which we wish to construct a succinct greedy embedding. We show in this section how to produce a combinatorial greedy embedding in a contrived space we call the *dyadic tree metric space*.

We may consider the infinite binary tree $\mathcal{B}$, to be an abstract metric space, in which the distance between any two tree nodes is just the number of edges on the shortest path between them. But there is another natural metric that can be

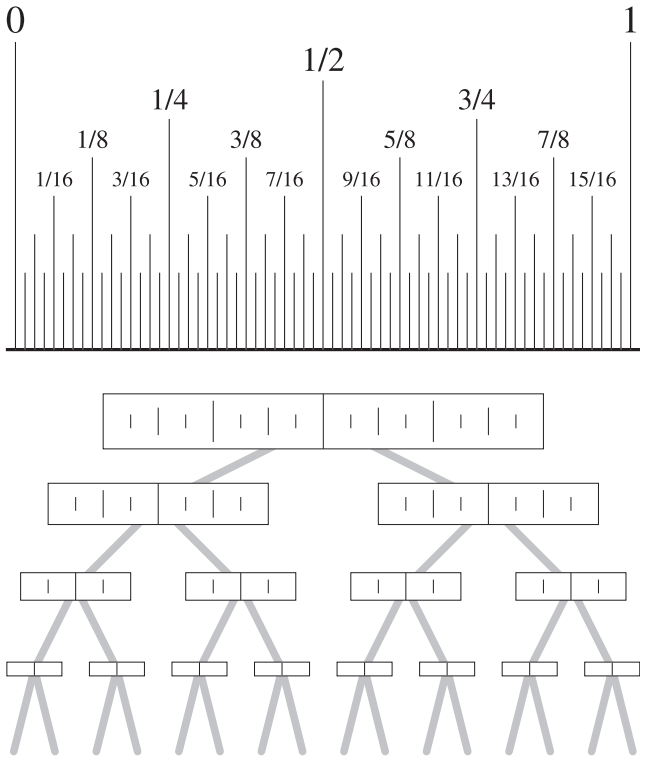

Fig. 4. The dyadic rational numbers (top) and a schematic of the dyadic tree metric space (bottom).

formed on the same tree by embedding it into the *dyadic rational numbers* (Fig. 4, top), that is, rational numbers with denominators that are powers of two. Let f be the map from $\mathcal{B}$ to the open interval $(0, 1)$ that maps the root of the tree to $1/2$, and that maps the children of a node x at level i of the tree to $f(x) \pm 2^{-i-2}$; thus, the children of the root map to the dyadic rational numbers $1/4$ and $3/4$, the grandchildren of the root map to $1/8, 3/8, 5/8, 7/8$, and so on. We define the *dyadic metric* on $\mathcal{B}$ as the metric in which the distance between two tree nodes x and y is $|f(x) - f(y)|$. Note that all distances in the dyadic metric are less than one.

We will show that any graph may be greedily embedded into an ad-hoc metric space that combines these two tree metrics; we call it the *dyadic tree metric space*. A point in this space is represented by a pair (x, y), where x and y are nodes in the infinite binary tree $\mathcal{B}$, and where x must be an ancestor of y (possibly equal to y itself). We define the distance between two points (x, y) and (x', y') in the dyadic tree metric space to be the sum of the tree distance between x and x' and of the dyadic distance $|f(y) - f(y')|$. The dyadic tree metric space can be represented as an infinite binary tree representing the x coordinates of each of its points, in which each tree node contains an interval of dyadic rational numbers; this interval of numbers is split into two halves at the two children of each node. This representation is depicted in Fig. 4, bottom.

Our embedding begins with us finding a spanning tree T of G, choosing a root arbitrarily, and producing a heavy path decomposition of T. For technical reasons, we require that each node in a nontrivial heavy path of the decomposition has at least one child that is not in the path; we add dummy nodes to T if necessary, after forming the path

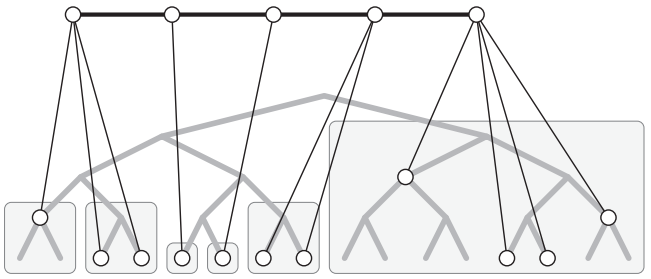

Fig. 5. Our two-level weight-balanced strategy for placing the children of the nodes on a heavy path. The groups of children for each heavy path node are assigned to subtrees in a weight-balanced way (gray shaded areas), and then within each subtree the individual children are placed using a second level of weight balancing. The third step of child placement, in which we make the subtree between the root (representing the heavy path) and its children autocratic, is not shown.

decomposition, to ensure that this is true. There will be fewer than n dummy nodes added, so they will not significantly increase the number of bits needed to represent each vertex in our greedy embedding.

We fix a combinatorial planar embedding of T by specifying a cyclic ordering of the edges incident to each vertex. At any internal vertex v of a heavy path P, we order the two edges of P consecutively, with the edge into v from the root of the tree counterclockwise and the edge out of v clockwise in the ordering; otherwise, the ordering may be chosen arbitrarily. In this way, all the light edges incident to P are placed on the same side of P in the combinatorial embedding, as they are depicted in Fig. 5.

We then compress each heavy path P into a super node. In the resulting compressed tree Z we may still determine a cyclic ordering of the light edges incident to each super node, in the same order that they appeared around P in T, by concatenating together the consecutive sequences of light edges from the cyclic orderings of the vertices in P.

Next, we form groups of the nodes in Z, where each group contains a maximal set of nodes that have the same parent in T. We form a weight-balanced binary tree that has one node for each group, in which the tree order of the nodes is the same as the order of the parents along the path P. Furthermore, within each group, we form a weight-balanced binary tree of the nodes of Z that belong to the group, where the order of these nodes is the same as the order of the light edges connecting them to their parent in the combinatorial planar embedding of T. Concatenating these two levels of weight-balanced trees forms a single weight-balanced tree connecting the node for P in Z to each of its children, in the order given by the combinatorial planar embedding of P; we apply the transformation described earlier to make this tree autocratic. The first three steps, in which we form a weight-balanced tree of the groups and a weight-balanced tree within each group, and then concatenate these two levels of weight-balanced trees to form a single binary tree for all children of the node in Z, are depicted in Fig. 5.

This construction of an autocratic weight-balanced tree for each node in Z can be used to embed Z into the infinite binary tree, $\mathcal{B}$. The root of Z may be placed at the root of $\mathcal{B}$, and the children of each node v in Z are placed under that node in the positions of $\mathcal{B}$ corresponding to their positions in the autocratic weight-balanced tree constructed for v. We observe that, in this way, all nodes of Z are placed at most $O(\log n)$ levels deep; for, as noted above, due to the weight balancing, the distance in $\mathcal{B}$ between any node w and its parent v is proportional to the difference in the logarithms of the weights of the subtrees rooted at v and w, and along any path of Z these differences add in a telescoping series to $O(\log n)$.

We have embedded Z into the infinite binary tree $\mathcal{B}$; hence, we are now ready to embed T itself into the dyadic tree metric. To do so, we must determine a pair (x, y) of coordinates for any node v of T; both x and y must be nodes of $\mathcal{B}$, and x must be an ancestor of y. The x coordinate of v is simply the node of $\mathcal{B}$ at which the heavy path of v is placed. The y coordinate of v is the least common ancestor in $\mathcal{B}$ of the placements of all the children of v. This calculation is the reason we required v to have at least one child; for leaf nodes of T, we instead set $y = x$. Due to our two-level weight balancing strategy, two nodes of T that belong to the same heavy path (and that, therefore, share the same x coordinate) will have different y coordinates, for their children will be placed within disjoint subtrees of the infinite binary tree $\mathcal{B}$.

Lemma 2. *The above embedding of T into the dyadic tree metric space is greedy.*

Proof. Any directed path in T consists of edges that, when translated into the dyadic tree metric space, have three types: edges from a node to the parent heavy path in Z, edges within a heavy path, and edges from a node to a child heavy path in Z. We must show that edges of each type lead to a node that is closer to the terminus of the path.

For the edges that go from a node to the parent heavy path or to a child heavy path, this is straightforward: the contribution of the x-coordinates to the distance to the terminus decreases by one at each step, due to the autocratic property of our weight-balanced trees, more than offsetting any possible increase in the contribution of the y-coordinates.

For the edges that are in a heavy path, the end-vertices of such edges have the same the x coordinates and do not lead to any increase or decrease of the distance to the terminus. The y coordinates are linearly ordered by the map f from infinite binary tree nodes to dyadic rationals, and our weight-balanced trees were chosen to be consistent with this linear ordering; therefore, any step along the heavy path, either toward a node of the path that is the ancestor of the terminus or toward the topmost node of the path and the edge leading to the parent node in Z, decreases the distance to the terminus. □

As in previous works (e.g., see [8]), we note that a greedy embedding for the spanning tree T is automatically greedy for the overall graph G from which it was drawn.

So far we have described our embedding into the dyadic tree metric space mathematically, but to use it we need an algorithm for constructing the embedding. This algorithm computes a spanning tree T for the graph, forms a heavy path decomposition of T, constructs the embedding of the heavy paths into the infinite binary tree $\mathcal{B}$ in a top-down order, and performs a bottom-up calculation to find least common ancestors in $\mathcal{B}$ of the children of each node in T to determine that node's precise placement in the dyadic tree metric space. The embedding of heavy paths into $\mathcal{B}$ involves the construction of a collection of weight-balanced trees with linear total size, and otherwise (including the step in

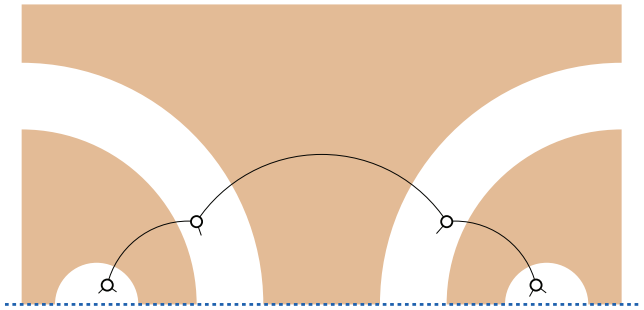

Fig. 6. Disjoint buffer zones of width D are crossed by each edge of an embedding of $\mathcal{B}$ into the hyperbolic plane, so that tree distance and hyperbolic distance closely approximate each other.

which these trees are made autocratic) can be performed using a linear number of operations in which labels representing positions in $\mathcal{B}$ are padded and concatenated; each of these padding and concatenation operations can be performed in constant time using standard bitwise Boolean operations. The least common ancestor operations used in the final step of the embedding algorithm are, in terms of the binary numbers used to represent positions in $\mathcal{B}$, simply a calculation of the most significant bit at which two numbers differ; these operations can again be performed in constant time, by a bitwise exclusive or followed by a constant number of lookups in precomputed tables of the most significant bit of a $\frac{1}{2}\log n$-bit subsequence of the number's binary representation. Therefore, the embedding can be constructed in linear total time.

5 Succinct Greedy Embedding in the Hyperbolic Plane

We have shown that any tree T (and any graph G by choosing a spanning tree of G) may be greedily, succinctly, and efficiently embedded into a dyadic tree metric space. To complete our greedy embedding, it remains to show that this space may be embedded, independently of our original graph (but depending on a parameter D determined by the number of vertices of the graph), into the hyperbolic plane in such a way that the greedy property of the embedding of T is preserved. That is, although the distances themselves in the hyperbolic plane may differ from those in the dyadic tree metric space, composing our embedding of T into the dyadic tree metric space with our embedding of the dyadic tree metric space into the hyperbolic plane should yield a greedy embedding of T into the hyperbolic plane.

Due to the existence of this embedding, we may reinterpret the succinct coordinates computed for the embedding of a graph into the dyadic tree metric space as coordinates for a subset of points in the hyperbolic plane. Not every hyperbolic plane point will be representable with such coordinates, but this is no different in principle from using pairs of integers to represent grid points in the euclidean plane: not every euclidean point is representable as an integer grid point. The parameter D is analogous to the scale of a grid embedding.

Our overall strategy will be to embed the infinite binary tree $\mathcal{B}$, into the hyperbolic plane in such a way that any edge has length $D + O(1)$ and crosses a *buffer zone* of width D, bounded by two hyperbolic lines (Fig. 6). The buffer zones for different edges will be disjoint from each other. Thus,

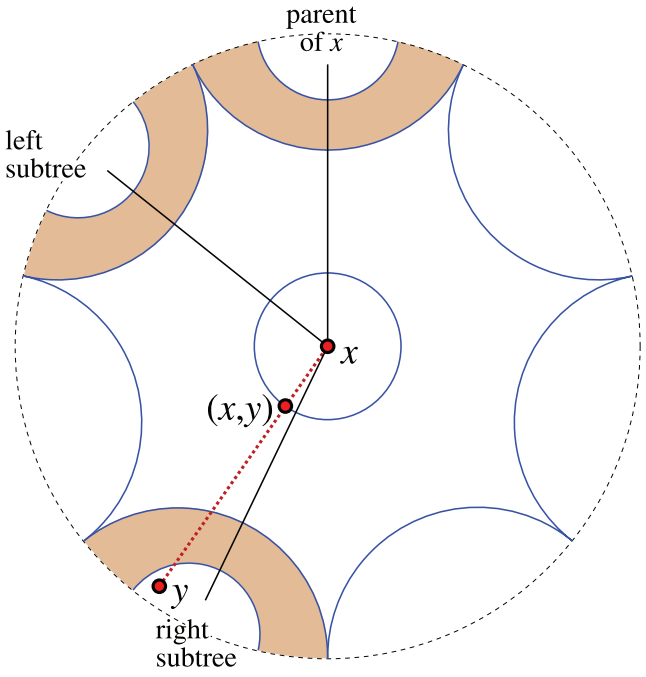

Fig. 7. Top-down placement of node x of $\mathcal{B}$ and point (x, y) of the dyadic tree metric space into the hyperbolic plane, shown in a Poincaré disk model centered at x.

any two nodes of the tree that have tree distance k units apart will have hyperbolic distance at least Dk (because any path between the two nodes must cross k buffer zones) and at most $(D + O(1))k$ (there exists a path following tree edges with that length). In our application, all tree paths will have $O(\log n)$ edges; thus, by choosing $D \geq \alpha \log n$, for a large enough constant α, we may guarantee that the order relation between any two distinct tree distances remains unchanged by this hyperbolic embedding. Any point (x, y) of the dyadic tree metric will be placed near the embedding of tree node x, and this placement will ensure the greediness of any edge whose endpoints belong to different paths of our heavy path decomposition.

Next, we place nodes of the infinite binary tree $\mathcal{B}$, into the hyperbolic plane, with buffer zones as described above. Although this placement is conceptual rather than algorithmic, we may view it as being performed in a top down traversal of the tree, so that when node x is placed we will already know the location of its parent, the buffer zone separating x from its parent, and a line connecting it to its parent and on which it must be placed. We place x itself on this line in such a way that the boundary of the parental buffer zone forms one of the seven sides of an ideal regular heptagon—a figure in the hyperbolic plane formed by seven lines that are asymptotic to each other but never intersect, such that the angle subtended by each line as viewed from x is equal. Fig. 7 shows this placement, in a Poincaré disk model of the hyperbolic plane centered at x; the parental buffer zone is the topmost shaded region in the figure and the vertical line through x is the one connecting it to its parent node. The large arcs depict hyperbolic lines forming the heptagon described above.

In the case where x is the right child of its parent, so that the upper nodes of the heavy path represented by x have children in its left subtree and the lower nodes of the heavy

path have children in the right subtree, shown in the figure, we place the left subtree within the halfplane bounded by the heptagon side one step counterclockwise from the parent, and the right subtree within the halfplane bounded by the heptagon side three steps counterclockwise from the parent, as shown in the figure. In the case where x is its parent's left child, we reverse the figure, placing the right subtree within the halfplane one step clockwise from the parent and the left subtree within the halfplane three steps clockwise from the parent. In either case, we draw lines connecting x to its child nodes, at angles of $2\pi/7$ and $6\pi/7$ from the angle of the line connecting x to its parent (the solid straight lines of the figure). We use the heptagon edges as the outer boundaries of buffer zones between x and its children, and we set the inner boundaries of the buffer zones to be hyperbolic lines perpendicular to the lines connecting x to its children, at distance D from the outer boundaries of the buffer zones. With this information determined, we may continue to place the children of x in the same way.

We are finally ready to describe the mapping of the dyadic tree metric space into the hyperbolic plane. Recall that each point of the dyadic tree metric space consists of a pair (x, y) where x and y are nodes of $\mathcal{B}, x$ a parent of y. We draw small circles of equal radius centered at each point where we have placed a node of $\mathcal{B}$—the precise radius is unimportant as long as it is small enough that the circles are disjoint from the buffer zones. Then, given a point (x, y) of the dyadic tree metric space, we draw a hyperbolic line segment from x to y (the dotted straight line in the figure), and place (x, y) at the point where this line segment intersects the circle centered at x. In the case $x = y$, which happens in our construction only for leaves, we instead place (x, x) at the point where the line segment from x to its parent intersects the circle centered at x.

Theorem 3. *For sufficiently large values of D, the embedding of G formed by composing the embedding from G into the dyadic tree metric space and the embedding of the dyadic tree metric space into the hyperbolic plane is greedy.*

Proof. We show that, for every edge e of the chosen spanning tree, and every possible terminus v of a path using e, traveling along e reduces the distance to the terminus. We assume that the starting endpoint of e is placed at point (x, y) of the dyadic tree metric, the ending endpoint is placed at point (x', y'), and that these points are mapped as described above to the hyperbolic plane. We distinguish several cases.

First, if $x \neq x'$, let $k = O(\log n)$ be the tree distance from x' to the destination. Then, due to the autocratic property of our weight-balanced placement of heavy paths into the dyadic tree metric, x is at tree distance at least $k + 1$ from the destination. As discussed above, due to the buffer zones of our construction, (x, y) is at hyperbolic distance at least $(k + 1)D$ from the destination, while (x', y') is at hyperbolic distance at most $k(D + O(1))$. By choosing D sufficiently large (a constant times $\log n$), we can guarantee that the former distance is larger than the latter and that this step is greedy.

Second, if $x = x'$ and the eventual destination also has the same value of x, the result follows from the fact that our embedding places the nodes of any heavy path consecutively over an arc of less than half of a circle. Such an embedding is greedy for any path, no matter how the nodes are distributed within the arc.

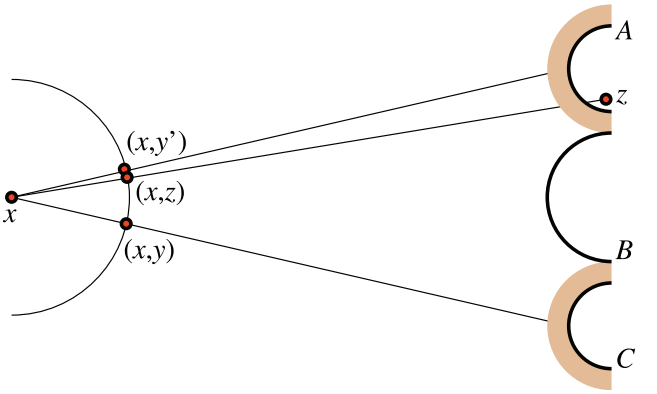

Fig. 8. Illustration for proof of greediness of our embedding (not to scale).

Third, if $x = x'$ and the eventual destination is reached via the parent of x, the step is greedy for the same reason as in the second case: the nodes that are mapped to x form a heavy path placed in order along an arc of less than half the circle, with the node of the arc closest to the parent being the apex of the heavy path.

The most complicated case is the fourth: $x = x'$ and the eventual destination z has x' as a proper descendant of x. The closest point to z on the circle surrounding x onto which (x, y) and (x', y') are both mapped is the hyperbolic point represented by the coordinates (x, z); the distance to z from other points on the circle can be calculated as a monotonic function of the arc length between those other points and (x, z). Thus, moving around the circle toward (x, z) is a greedy step. Unfortunately, the point (x, z) may not be a node of the heavy path; rather, the node of the heavy path from which z descends may be some other nearby point (x, y''). We must show that any step along the heavy path toward this point is greedy.

In most cases, it is straightforward to show that this step is greedy: a step around the circle toward (x, z) is also a step toward (x, y''), which as we have argued immediately above is greedy. The only possible exception occurs when $y' = y''$ and when the true closest point on the circle to z, that is, (x, z), lies on the arc of the circle between y and y'. In this case, we must show that (x, y') and (x, z) are closer in arc length than (x, y) and (x, z), for then the greediness of the step will follow from the monotonicity of the distance to z as a function of arc length.

Let $\hat{y}$ be the least common ancestor in the binary tree of the two disjoint subtrees containing y and y'. Let A be the inner boundary of the buffer zone adjacent to $\hat{y}$ that contains y', let C be the inner boundary of the buffer zone adjacent to $\hat{y}$ that contains y, and let B be the edge of the regular ideal heptagon adjacent to $\hat{y}$ that separates A from C. Fig. 8 illustrates this notation. These three hyperbolic lines may not be symmetrically placed relative to x, due to the asymmetry of the placement of the two subtrees relative to the parent at each node x. However, the distances from x to A and to C are within $O(1)$ of each other, and B is closer to x by a distance of $D - O(1)$. It is a basic property of hyperbolic geometry that the angle that

an object subtends, as viewed from a fixed point of view x, is inversely proportional to an exponential function of the distance of the object from x. Thus, B will subtend an angle, as viewed from x, that is larger than the angles subtended by A and C by a factor exponential in $D - O(1)$. In particular, for sufficiently large D (larger than some fixed constant, a weaker requirement than the one above that $D = \Omega(\log n)$), both A and C will subtend smaller angles than the angle subtended by B. Then, any point behind line A, and in particular the point z, will form an arc from (x, y') to (x, z) that is shorter than the arc from (x, y) to (x, z). The greediness of the step from (x, y) to (x, y') follows from the monotonicity of the distance to z as a function of arc length. □

In another paper [32], we provide an isometric embedding of the dyadic tree metric space into a different two-dimensional geometric metric space, a manifold with the local geometry of the L_1 metric on the plane. Thus, our greedy embedding technique can be interpreted as applying as well to the spaces described in that paper.

To pull all the details together, then, we have shown how to embed any graph into a metric space such that the coordinates can be represented using $O(\log n)$ bits, and we have also shown how a tree in that metric space can be embedded in the hyperbolic plane so that greedy paths are preserved. Thus, we have a greedy embedding in the hyperbolic plane that can be represented using $O(\log n)$ bits for the coordinates. We should mention that representing the coordinates of embedded vertices using standard representations, such as the Poincaré disk, would result in much larger sizes, which is one motivation for why we developed this alternate approach.

6 Conclusion and Future Directions

We have given an succinct method for embedding a connected graph in the hyperbolic plane so as to support greedy routing. Future directions include the following:

1. Design an efficient distributed algorithm for succinct greedy routing for its own network.
2. Maintain a succinct greedy routing scheme subject to vertex and/or edge updates.

Acknowledgments

This research was supported by NSF grants 0713046 and 0830403, and by ONR grant N00014-08-1-1015. This paper appeared in preliminary form as [33].

References

[1] D. Comer, *Internetworking with TCP/IP, Volume 1: Principles, Protocols, and Architecture.* Prentice-Hall, Inc., 2006.

[2] A.S. Tanenbaum, *Computer Networks,* fourth ed. Prentice-Hall, Inc., 2003.

[3] P. Bose, P. Morin, I. Stojmenović, and J. Urrutia, "Routing with Guaranteed Delivery in Ad Hoc Wireless Networks," *Wireless Networks,* vol. 6, no. 7, pp. 609-616, 2001.

[4] B. Karp and H.T. Kung, "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks," *Proc. Sixth ACM MobiCom,* pp. 243-254, 2000.

[5] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger, "Geometric Ad-Hoc Routing: Of Theory and Practice," *Proc. 22nd ACM Symp. Principles of Distributed Computing (PODC),* pp. 63-72, 2003.

[6] F. Kuhn, R. Wattenhofer, and A. Zollinger, "Asymptotically Optimal Geometric Mobile Ad-Hoc Routing," *Proc. Sixth ACM Discrete Algorithms and Methods for Mobile Computing and Comm. (DIALM),* pp. 24-33, 2002.

[7] F. Kuhn, R. Wattenhofer, and A. Zollinger, "Worst-Case Optimal and Average-Case Efficient Geometric Ad-Hoc Routing," *Proc. Fourth ACM Symp. Mobile Ad Hoc Networking and Computing (MobiHoc),* pp. 267-278, 2003.

[8] R. Kleinberg, "Geographic Routing Using Hyperbolic Space," *Proc. 26th IEEE INFOCOM '07,* pp. 1902-1909, 2007.

[9] C.H. Papadimitriou and D. Ratajczak, "On a Conjecture Related to Geometric Routing," *Theoretical Computer Sciences,* vol. 344, no. 1, pp. 3-14, 2005.

[10] K. Bezdek, "Sphere Packings Revisited," *European J. Combinatorics,* vol. 27, no. 6, pp. 864-883, 2006.

[11] E. Kranakis, H. Singh, and J. Urrutia, "Compass Routing on Geometric Networks," *Proc. 11th Canadian Conf. Computational Geometry (CCCG),* pp. 51-54, 1999.

[12] H. Frey and I. Stojmenović, "On Delivery Guarantees of Face and Combined Greedy-Face Routing in Ad Hoc and Sensor Networks," *Proc. 12th Int'l Conf. MobiCom '06,* pp. 390-401, 2006.

[13] N. Carlsson and D.L. Eager, "Non-Euclidean Geographic Routing in Wireless Networks," *Ad Hoc Networks,* vol. 5, no. 7, pp. 1173-1193, 2007.

[14] A. Rao, S. Ratnasamy, C.H. Papadimitriou, S. Shenker, and I. Stoica, "Geographic Routing without Location Information," *Proc. Ninth Int'l Conf. MobiCom '03,* pp. 96-108, 2003.

[15] R. Dhandapani, "Greedy Drawings of Triangulations," *Proc. 19th ACM-SIAM Symp. Discrete Algorithms (SODA),* pp. 102-111, 2008.

[16] M.B. Chen, C. Gotsman, and C. Wormser, "Distributed Computation of Virtual Coordinates," *Proc. 23rd Symp. Computational Geometry (SoCG '97),* pp. 210-219, 2007.

[17] K.M. Lillis and S.V. Pemmaraju, "On the Efficiency of a Local Iterative Algorithm to Compute Delaunay Realizations," *Proc. Workshop Experimental Algorithms (WEA),* 2008.

[18] S.K. Ghosh and K. Sinha, "On Convex Greedy Embedding Conjecture for 3-Connected Planar Graphs," *FCT,* M. Kutylowski, W. Charatonik, and M. Gebala, eds., pp. 145-156, Springer, 2009.

[19] T. Leighton and A. Moitra, "Some Results on Greedy Embeddings in Metric Spaces," *Proc. 49th IEEE Symp. Foundations of Computer Science (FOCS),* 2008.

[20] P. Angelini, F. Frati, and L. Grilli, "An Algorithm to Construct Greedy Drawings of Triangulations," *Proc. 16th Int'l Graph Drawing Conf.,* pp. 26-37, 2008.

[21] R.B. Muhammad, "A Distributed Geometric Routing Algorithm for Ad Hoc Wireless Networks," *Proc. IEEE Conf. Information Technology (ITNG),* pp. 961-963, 2007.

[22] H. de Fraysseix, J. Pach, and R. Pollack, "How to Draw a Planar Graph on a Grid," *Combinatorics,* vol. 10, no. 1, pp. 41-51, 1990.

[23] W. Schnyder, "Embedding Planar Graphs on the Grid," *Proc. First ACM-SIAM Symp. Discrete Algorithms,* pp. 138-148, 1990.

[24] M.T. Goodrich and D. Strash, "Succinct Greedy Geometric Routing in the Euclidean Plane," *Proc. 20th Int'l Symp. Algorithms and Computation (ISAAC),* pp. 781-791, 2009.

[25] P. Maymounkov, "Greedy Embeddings, Trees, and Euclidean vs. Lobachevsky Geometry," M.I.T., http://pdos.csail.mit.edu/petar/papers/maymounkov-greedy-prelim.pdf, 2006.

[26] E.N. Gilbert and E.F. Moore, "Variable-Length Binary Encodings," *Bell System Technical J.,* vol. 38, pp. 933-968, 1959.

[27] D.E. Knuth, "Optimum Binary Search Trees," *Acta Informatica,* vol. 1, pp. 14-25, 1971.

[28] B.A. Sheil, "Median Split Trees: A Fast Lookup Technique for Frequently Occurring Keys," *Comm. ACM,* vol. 21, no. 11, pp. 947-958, 1978.

[29] D.D. Sleator and R.E. Tarjan, "A Data Structure for Dynamic Trees," *J. Computer and System Sciences,* vol. 26, no. 3, pp. 362-391, 1983.

[30] B. Schieber and U. Vishkin, "On Finding Lowest Common Ancestors: Simplification and Parallelization," *SIAM J. Computing,* vol. 17, no. 6, pp. 1253-1262, 1988.

[31] S. Alstrup, P.W. Lauridsen, P. Sommerlund, and M. Thorup, "Finding Cores of Limited Length," *Proc. Int'l Workshop Algorithms and Data Structures (WADS '97),* pp. 45-54, 1997.

[32] D. Eppstein, "Manhattan Orbifolds," Electronic preprint math.MG/0612109, 2009.
[33] D. Eppstein and M.T. Goodrich, "Succinct Greedy Graph Drawing in the Hyperbolic Plane," *Proc. 16th Int'l Graph Drawing Conf.*, pp. 14-25, 2008.

David Eppstein received the Phd degree in computer science from Columbia University in 1989. Currently he is working as a professor in the Department of Computer Science at the University of California, Irvine. After majoring in mathematics at Stanford University, he worked as a postdoctoral researcher at the Xerox Palo Alto Research Center from 1989 to 1990. His research specialties include computational geometry, graph algorithms, and graph drawing.

Michael T. Goodrich received the Phd degree in computer sciences from Purdue University in 1987. Currently he is working as a chancellor's professor at the University of California, Irvine, in the Department of Computer Science and he was a faculty member at Johns Hopkins University from 1987 to 2001. His research is directed at algorithms for problems motivated from information security, the Internet, information visualization, and geometric computing. He is a fellow of the IEEE.