NEW PROXIMITY-PRESERVING ORDERINGS FOR SPATIAL DATA

Alan Saalfeld
Bureau of the Census[1]
Washington, DC 20233

ABSTRACT

This paper presents new methods for ordering vertices or edges in a tree (connected acyclic graph). The new orderings are called *tree-orders*; they can be constructed in linear time; and they are fully characterized by a useful proximity-preserving property called *branch-recursion.* The paper describes how *tree-ordering* techniques can be applied to find orderings for other types of spatial entities:

- ordering points in the plane,
- ordering points in higher dimensional spaces,
- ordering vertices of any graph,
- ordering edges of any graph,
- ordering line segments in two-dimensional networks,
- ordering line segments of networks in higher dimensions,
- ordering regions in the plane,
- ordering $(n-1)$-cells in n-dimensional polytopal regions,
- ordering n-cells in n-dimensional cell decompositions.

For each of the spatial entities listed above, the orderings produced by extending the tree-ordering methods exhibit important proximity-preserving properties. The paper includes a description of several potential applications of the new orderings of the diverse spatial objects.

PRELIMINARIES

At its most elementary level, database management is the art of organizing or ordering data so that they may be accessed and utilized most efficiently for some particular set of operations of interest. This paper presents a new way of ordering data that will permit a collection of important operations related to clustering and to systematic sampling to be carried out efficiently and effectively. In this

[1]The views expressed herein are those of the author and do not necessarily represent the views of the Bureau of the Census.

section we review and summarize some definitions and basic concepts needed to describe our ordering techniques.

Orderings and Lists

Throughout this paper, *ordering* or *order*, without any qualifying adjectives, will refer to a total order or linear order of a finite set of n elements. Such an order is nothing more than a sequencing of the n elements, a one-to-one association of the elements of the set with the integers 1 through n, or a listing of the n elements. A set of n elements that have been ordered will be called a *list* or an *ordered list.*

In a list of n elements, the $(i+1)$st element is the *successor* to the ith element; and every element except the nth or last element has a unique successor. Similarly, every element except the first element has a unique *predecessor.* We may build a *cyclic list* or a *cyclic order* from a list by naming the first element to be the successor to the last element (and the last element to be the predecessor to the first). Cyclic lists are often useful because they have no distinguished elements that require special case handling. For example, with a cyclic list, one may begin *anywhere* in the list and exhaustively enumerate elements by taking successors until one returns to the chosen starting element.

Spatial Queries

Points in two-dimensional and higher dimensional space are often assigned an order or primary key to facilitate their storage in and retrieval from databases. Space-filling curves, such as the Peano key and Hilbert curve ([FAL1] and [FAL2]), have proved quite useful for range queries and nearest neighbor queries. These curves are instances of a large class of orderings called *quadrant-recursive* orderings ([MARK]). The defining property of quadrant-recursive orderings is that, in any recursive decomposition of a rectangular region into subquadrants, the points of any subquadrant always appear consecutively in the quadrant-recursive ordering. Points within any subquadrant are enumerated exhaustively before exiting the subquadrant. We will see in the section on branch-recursion that quadrant-recursive orderings are a special case of a more general class of orderings called *branch-recursive.*

Systematic Sampling

Systematic sampling traditionally refers to selection of a subset from a list, where the subset is formed by selecting elements at regular intervals (called the *skip interval*) [KISH]. Elements may be *weighted* to adjust their probability of selection (see figure 1).

If all weights are 1, then a skip interval of k produces a $1/k$ sample. We may think of the sampled elements as having the induced order of their sequenced systematic selection achieved by skipping through the list.

If points in the plane are assigned any quadrant-recursive order, then a systematic selection procedure will sample every subquadrant, no matter what its size, to within one unit of the overall sampling fraction. This representative coverage property was noted and utilized for Peano key ordering by Wolter and Harter [WOLT].

If we regard systematic sampling as a means of ordering subsets, then trivially we may recover the original order of an unweighted list by sampling with skip interval equal to 1. This seemingly trivial means of recovering an ordering will be exploited in the section on tree-ordering vertices to build an ordering when

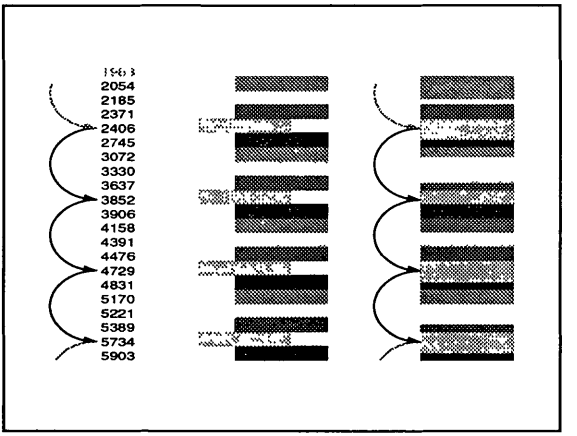

Figure 1: Systematic Sampling from Lists

we sample a set systematically *after* breaking each element into weighted pieces having total weight 1.

Graphs and Maps

The linework of any map has an underlying structure of a *graph*[2]. We will use the usual combinatorial definitions of graph theory found in the standard text by Harary [HARA]. A *graph* $G = (V, E)$ consists of a finite non-empty set V of *vertices* together with a set E of unordered pairs of vertices called *edges*. A vertex v and an edge $\{u, w\}$ are *incident* if and only if $v = u$ or $v = w$. The *degree* of a vertex is the number of edges incident to the vertex. A *walk* of the graph G consists of a sequence $(v_1 v_2 v_3 \cdots v_k)$ of vertices v_i, not necessarily all distinct, such that for each $j = 1, 2, \ldots, (k-1)$, $\{v_j, v_{j+1}\}$ is an edge of G. A *tour* is a walk $(v_1 v_2 v_3 \cdots v_k)$ such that $v_1 = v_k$. A *path* is a walk with no edges repeated. A *cycle* is a path $(v_1 v_2 v_3 \cdots v_k)$ with $k \geq 3$ such that $v_1 = v_k$. A *tree* is a graph with no cycles. A tree as we have defined it is sometimes called a *free tree* to differentiate it from a *rooted tree*, which possesses a distinguished vertex called the *root*.

PROPERTIES OF TREES

We describe some properties of trees that make them easier to work with than graphs in general. We will show in the sections on ordering vertices and edges in a graph how problems of ordering graph components can be converted to problems of ordering tree components for a derived tree. Computer scientists have developed a number of ways of ordering the vertices of rooted trees embedded in the plane [AHO2]. We will be looking at new orderings for free trees.

Combinatorial Properties

We list some important properties of trees.

[2]For some applications it may be preferable and even necessary to regard the linework of a map as a *pseudo-graph*, a structure which allows multiple edges between two vertices. For the applications which we are examining here, the distinction is unimportant.

Property 1 *Every tree with n vertices has exactly $(n-1)$ edges.*

Property 2 *A connected graph having n vertices and $(n-1)$ edges is a tree.*

Property 3 *Adding a new edge to a tree (between existing vertices) always creates a cycle.*

Property 4 *Removing an edge always disconnects a tree.*

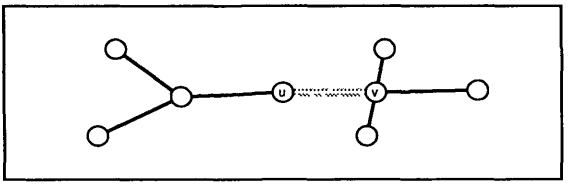

Figure 2: Edge Removal Creates Two Branches

We say that each edge determines two *branches* which are the disconnected component subtrees resulting from that edge's removal. Always one of the branches determined by the edge $\{u, v\}$ contains u and the other branch always contains v.

Planar Embeddings

Not every graph can be drawn in the plane with non-intersecting line-segment edges, but a tree can always be represented or realized as a straight-line drawing in the plane. Moreover, suppose that for each vertex in a tree, we arbitrarily assign a cyclic order to the edges incident to that vertex. Then there is always a drawing in the plane of that tree with straight-line-segment edges such that the clockwise order of the edges incident to any vertex is the arbitrarily assigned cyclic order of the edges about the vertex.

EULERIAN TOURS

An Eulerian tour of a tree is a special well-balanced tour that traverses every edge exactly twice, once in each direction. We give two equivalent descriptions of an Eulerian tour of a tree. *Each description depends on our having assigned a cyclic order to the incident edges of each vertex.*

Geometric Version

Draw the tree so that the assigned cyclic order of edges at each vertex is the clockwise order. Start a tour at any vertex x. Depart along any incident edge $\{x, u\}$ toward u. Upon arriving at u, depart along the edge $\{u, v\}$ that is next to $\{x, u\}$ in the clockwise order around u. Upon arriving at v, depart along the edge $\{v, z\}$ that is next to $\{u, v\}$ in the clockwise order around v, etc., until finally you return to x along the edge that precedes $\{x, u\}$ in the clockwise order (see figure 3).

The tour that we have described will traverse each edge twice, once in each direction and visit every vertex a number of times equal to its degree.

This tour can also be visualized as follows: imagine that the tree itself is the top view of a wall. Walk next to the wall with your right hand continuously

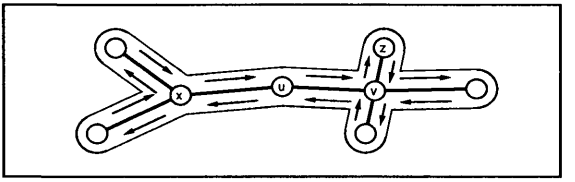

Figure 3: Geometric Depiction of Eulerian Tour

touching the wall. You will eventually return to your starting point, at which time you will have touched both sides of every wall. Thus, had someone else been walking on top of the wall and keeping up with you, that person would have walked every edge of the tree exactly twice.

Combinatorial Version

An Eulerian tour of a tree on n vertices is a walk $(v_1v_2v_3\cdots v_{2n-1})$, where the v_i's are vertices, clearly not all distinct, satisfying

1. $v_1 = v_{2n-1}$.
2. For $i = 1, 2, \ldots, 2n-3$, $\{v_{i+1}, v_{i+2}\}$ is the successor to $\{v_i, v_{i+1}\}$ in the cyclic ordering of edges about the vertex v_{i+1}.
3. $\{v_1, v_2\}$ is the successor to $\{v_{2n-2}, v_1\}$ in the cyclic ordering of edges about the vertex v_1.
4. For every edge $\{u, v\}$ of the tree, the sequence uv and the sequence vu each appear exactly once in the walk $(v_1v_2\cdots v_{2n-1})$.
5. Each vertex except v_1 appears as often as its degree.
6. The vertex v_1 appears one more time than its degree.

Notice further that if uv appears before vu in the Eulerian tour, then the subwalk between uv and vu that starts and ends at v completely consumes every edge and vertex in the v-branch determined by edge $\{u, v\}$. Moreover, this subwalk touches nothing but the v-branch of the tree (see figure 4).

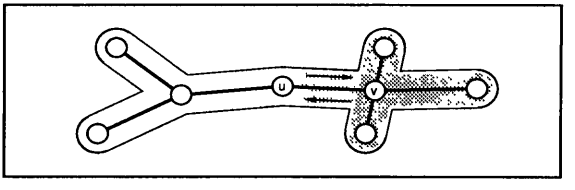

Figure 4: A Subwalk Consumes an Entire Branch

Similarly, the remainder of the tour (the part before uv and after vu) completely consumes every edge and vertex of the u-branch resulting from the removal of the edge $\{u, v\}$. Clearly, no vertex or edge that appears in the v-branch can ever appear in the u-branch.

We summarize this partitioning of the tour into two non-intersecting walks in the following lemmas.

Lemma 1 *Let* $(\cdots x \cdots uv \cdots y \cdots vu \cdots z \cdots)$ *be an Eulerian tour of some tree, with* x, y, z, u, v *vertices on the tour. Then* $u \neq y$, $x \neq y$, *and* $z \neq y$.

Lemma 2 *Let* $(\cdots u_1 v \cdots x u_2 \cdots)$ *be an Eulerian tour of some tree, with* $u_1 = u_2$ *and no other occurrences of* u_1 *between* u_1 *and* u_2. *Then* $v = x$.

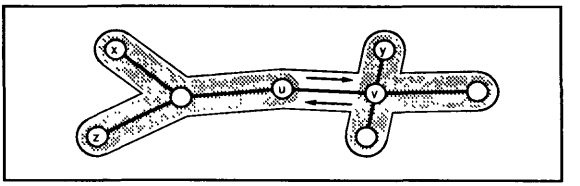

Figure 5: Edge Removal Splits the Tour

Figure 5 illustrates the proof of both lemmas. The subwalks uv and vu are the only means of getting from one branch to the other.

Eulerian Tree Orderings

During the course of an Eulerian tour, all of the vertices and edges are visited at least once. Suppose we wish to assign the integers 1 through n to our n vertices. A procedure that visits the vertices in Eulerian tour order, assigning either the next available number or no number to every *visit* of each vertex, in such a way that exactly one of the visits of each vertex receives an order number, will be called an *Eulerian tree-ordering* (ETO) of vertices.

A procedure that visits the edges in Eulerian tour order, assigning either the next available number or no number to every *visit* of each edge, in such a way that exactly one of the two visits of each edge receives an order number, will be called an *Eulerian tree-ordering* (ETO) of edges.

Garey and Johnson [GARE] and others [PREP], [EDE1] describe one such Eulerian tree-ordering of vertices of a Euclidean minimum spanning tree (EMST) obtained by starting anywhere on the Eulerian tour and assigning the next available number to the *first* visit to each and every vertex (see figure 6).

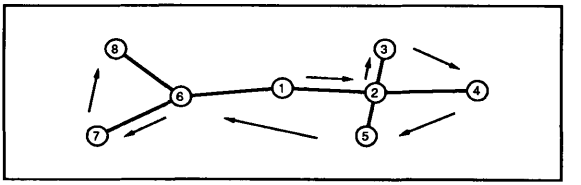

Figure 6: First-Visit Eulerian Tree-Ordering

Their ordering, or for that matter, any other Eulerian tree-ordering of a EMST will always approximate a Euclidean Travelling Salesman Tour to within a factor of 2, since the Eulerian tour itself is never more than twice the length of the Euclidean Travelling Salesman Tour.

We now describe a simple procedure for generating all other Eulerian tree-orderings of the vertices of a tree.

TREE-ORDERING VERTICES

Suppose that we are given a tree *and an Eulerian tour* $(v_1, v_2, \cdots, v_{2n-1})$ for that tree (equivalently we are given an embedding of the tree in the plane and a starting vertex and edge). Then to order the vertices we will proceed as follows.

Setup: Weighting Vertex Visits

For $i = 1, 2, 3, \ldots, (2n-1)$, regard each v_i that appears in the tour $(v_1, v_2, \cdots, v_{2n-1})$ as a vertex visit.

For $i = 1, 2, 3, \ldots, (2n-1)$, assign a non-negative weight w_i to the ith visit so that the sum of weights for all visits to any fixed vertex v is one:

$$\text{For every } v \in V, \sum_{\{i|v_i=v\}} w_i = 1.$$

We call any such weight assignment a ***unit-sum weight assignment.***

An important instance of a unit-sum weight assignment assigns the same weight to all visits to the same vertex. Because each vertex v_i is visited $\deg(v_i)$ times[3], that uniform weight is exactly given by:

$$\begin{aligned} w_i &= \frac{1}{\deg(v_i)}, \text{ for } i = 1, 2, \ldots, (2n-2); \text{ and} \\ w_{2n-1} &= 0. \end{aligned}$$

Building the Sampling Interval

As we walk the Eulerian tour, we begin accumulating weights, (exactly as is done to build a weighted list for systematic sampling).

$$\begin{aligned} \text{Let } W_0 &= 0, \text{ and} \\ W_j &= W_{j-1} + w_j. \end{aligned}$$

An Illustration: Uniform Weighting. We illustrate the accumulating of weights for the uniform weighting scheme for the walk (*defegehedbabcbd*) drawn in figure 7.

The total accumulated weights are exactly n and the total weight corresponding to each vertex of the tree is exactly one. We can assign numbers to the vertices by skipping through the weighted interval with skip interval equal to 1. This is the same as assigning an order number to a vertex each time a vertex visit takes us up to or past the next whole integer:

If $\lfloor W_{j-1} \rfloor < \lfloor W_j \rfloor$, then assign vertex v_j the number $\lfloor W_j \rfloor$.

Because some vertices appear in several places in our accumulated weighted interval, one may suspect that our numbering scheme may assign more than one order number to a vertex. This cannot happen.

Proofs of Correctness

The proof that the selection procedure outlined above actually produces an ordering of the vertices follows immediately from the following lemma and its first corollary.

[3]Because the Eulerian tour is cyclic, we like to count v_1 and v_{2n-1} as the same visit. We should only assign the appropriate weight to one or the other. We have chosen to assign the uniform weight to v_1.

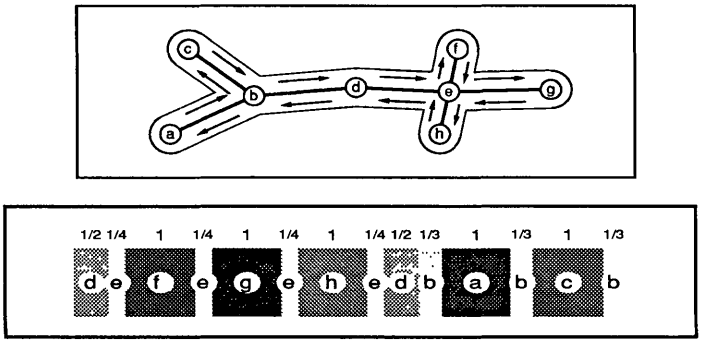

Figure 7: A Tour and its Accumulated Weights

Lemma 3 (Integral-Branch-Weights) *The fractional vertex weights accumulated between any two consecutive visits of the Eulerian tour to a multivisited vertex always add up to an integer.*

Proof: The proof of this lemma rests entirely on the observation that between two consecutive visits to any vertex v, one must depart and enter along the same edge, and an entire branch emanating from that vertex v is completely consumed by the subwalk of the Eulerian tour, as seen in lemmas 1 and 2.

In consuming an entire branch, one must visit every vertex in that branch as many times as possible, *i.e.* as many times as the degree of that vertex. Thus each vertex in the branch gets fully counted. In other words, the sum of weights for all the visits for any individual vertex during the walk of the branch is 1. And the sum of weights for *all* the visits of *all* vertices during the walk of the branch is an *integer*, equal to the number of distinct vertices in the branch. □

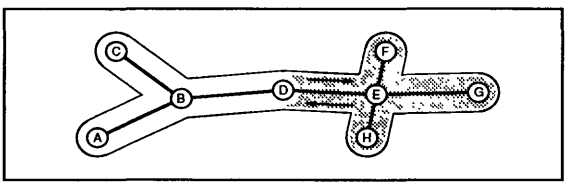

Figure 8: All Vertices of a Branch are Consumed

This lemma has two useful corollaries. To prove the first corollary we will want to talk about the fractional part of a number or an interval of numbers. Our meaning is the usual one: the fractional part of 5.35 is 0.35. The fractional part of an interval such as [17.32, 17.84) is just the set of all possible fractional values: [0.32, 0.84).

Corollary to Lemma 3.1 *Every vertex gets hit exactly once by skipping one unit at a time through the n-interval.*

Proof: Consider any vertex v of degree $= k$. Let the visits to the vertex v occur at $i_1, i_2, \cdots, i_k$. Then the visits to the vertex v will result in intervals of length

$w_{i_1}, w_{i_2}, \cdots, w_{i_k}$ being added to the cumulative interval. We want to prove that the fractional parts of the accumulated subintervals corresponding to v, namely,

$$(W_{i_1-1}, W_{i_1}], (W_{i_2-1}, W_{i_2}], \cdots, \text{ and } (W_{i_k-1}, W_{i_k}],$$

in the total interval of length n have no overlap. From lemma 3, it is clear that each successive interval, $(W_{i_j-1}, W_{i_j}]$, corresponding to a visit to v has its fractional part begin (at W_{i_j-1}) where the fractional part of the previous interval $(W_{i_{j-1}-1}, W_{i_{j-1}}]$, corresponding to a visit to v left off (at $W_{i_{j-1}}$), since an interval of integer length (*i.e.* having *no* fractional part) corresponding to all of the vertex visits of the branch consumed, has intervened.

Since the intervals $(W_{i_1-1}, W_{i_1}]$, $(W_{i_2-1}, W_{i_2}]$, $\cdots$, $(W_{i_k-1}, W_{i_k}]$, have no fractional parts overlapping and have total length equal to one, the fractional parts of values assumed in the accumulated intervals corresponding to any individual vertex must span all of the values between 0 and 1.

This last observation tells us that we can take a random start r in $[0, 1)$, take skip interval 1 once again, and we will again produce an ordering of the tree vertices. Any real number r or integral augmentation $r + m$ of r can hit at most one of the k intervals of determined by visits to v; and there is exactly one integer m_0 such that $r + m_0$ will hit one of the k intervals. □

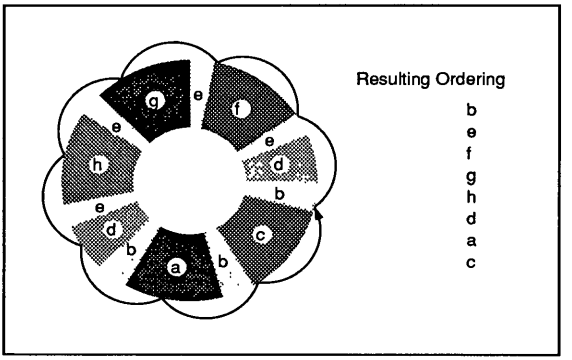

Figure 9: Cyclic Ordering with Uniform Weighting

Because the Eulerian tour is cyclic, we can make our cumulative interval cyclic and our resulting ordering cyclic as well by removing the dependence on the starting point of the tour when building our cumulative vertex-visit weight interval, as shown in figure 9. Putting all of the lemmas and corollaries together, we have the following theorem:

Theorem 1 *While making an Eulerian tour of a tree, build a separate (cyclic) interval of total length n units by assigning a non-negative weight to each vertex visit in any way so that the total weight for all visits to any individual vertex is one. Then every vertex gets hit exactly once by skipping one unit at a time through the cyclic n-interval.*

Branch-Recursion

Throughout this section we will regard the orderings generated by our ordering procedure as cyclic by making the first vertex successor to final vertex.

The next corollary follows immediately from lemma 1 and the proof of lemma 3.

Corollary to Lemma 3.2 *The collection of vertices of* **any branch** *of the tree always constitute a complete interval (i.e. appear consecutively) for any cyclic Eulerian tree-ordering.*

We will say that any cyclic vertex ordering that keeps vertices of a branch together for all branches is a *branch-recursive ordering.* Corollary 3.2 states that every Eulerian tree-ordering is branch recursive. It is not difficult to prove the converse using induction on branch size. We leave the proof of that theorem as an exercise.

Theorem 2 *Every branch-recursive cyclic ordering of the vertices of a tree is an Eulerian tree-ordering for some Eulerian tour of the tree and some unit-sum weight assignment to the vertex visits of that Eulerian tour.*

Branch-recursion constitutes a very strong proximity preservation property, where proximity is measured by the link-distance in the tree or graph. Branches of a tree may correspond to data clusters in cases where we have built minimum spanning trees. All quadrant-recursive orderings of a point set in the plane may be realized as orderings induced on the leaf subsets of branch-recursive orderings (*i.e.* Eulerian tree-orderings) of the quad-tree of those points.

Analysis of Complexity

An analysis of the time complexity of our tree-ordering algorithms depends on the choice of data structure with which we represent the tree. If we have a topological data structure which allows us to find the adjacent edge to any edge at any vertex in constant time, then we can order the vertices in linear time. If we need to build topology from an elementary list of vertices and edges, we can do so in time $O(n \log n)$, then proceed in linear time to complete the ordering procedure.

Space complexity is even easier to analyze. The Eulerian tour is always linear in the size of the tree. It is exactly of size $(2n - 1)$. The cumulative interval of weights that we must build is also of that size.

Enumerating Orderings

If we allow arbitrary unit-sum weighting schemes and arbitrary Eulerian tours, then we can generate all possible branch-recursive orderings. The number of branch-recursive cyclic orderings can be shown to be:

$$\prod_{v \in V} (\deg(v))!$$

as follows. Since there are $(\deg(v) - 1)!$ ways of cyclically ordering the edges incident to the vertex v, there are:

$$\prod_{v \in V} (\deg(v) - 1)!$$

possible distinct cyclic Eulerian tours. In each Eulerian tour, a vertex v may be enumerated immediately prior to any of its $\deg(v)$ branches. This results in

$$\prod_{v \in V} (\deg(v))$$

distinct cyclic orderings for each Eulerian tour. If, however, we only consider uniform unit-sum weighting schemes for fixed Eulerian tours, then we have proved [SAAL] that there are no more than:

$$LCM\{\deg(v) \mid v \in V\} \text{ distinct cyclic orderings;}$$

where LCM is the least common multiple. This translates into the following: If the maximum degree in the tree is 3, then there are at most 6 distinct cyclic orderings (independent of the number of vertices) for a fixed Eulerian tour. Maximum degree 4 translates into at most 12 distinct orderings; and maximum degree 5 or 6 results in at most 60 distinct cyclic orderings.

Some important trees have small maximum degree. A Euclidean Minimum Spanning Tree (EMST) of points in the plane, for example, has maximum degree 6. The EMST for points in general position has a canonical Eulerian tour as well; so the unique EMST generates at most 60 distinct cyclic orderings of points in general position in the plane, no matter how many points are in the point set!

TREE-ORDERING EDGES

Many of our results and methods for ordering vertices are equally valid for edge ordering. Theorem 1 for vertices has an exact counterpart for edges:

Theorem 3 *While making an Eulerian tour of a tree, build a separate (cyclic) interval of total length* $(n-1)$ *units by assigning a non-negative weight to each edge visit in any way so that the total weight for all visits to any individual edge is one. Then every edge gets hit exactly once by skipping one unit at a time through the cyclic* $(n-1)$*-interval.*

The proof the theorem 3 is identical to the proof of theorem 1: between consecutive vertex visits to some vertex, all (both) edge visits to any particular edge within a branch are exhausted.

Uniform Edge Weighting

A *uniform* weighting scheme for edges instead of vertices would have each edge getting weight exactly $1/2$ (since every edge is visited twice in the Eulerian Tour). But giving every edge weight $1/2$ amounts to nothing more than skipping every other edge in our selection procedure. So we have the following corollary to theorem 3:

Corollary 3.1 *While making an Eulerian tour of a tree, number every other edge visited. Then every edge gets exactly one number assigned to it.*

We also see immediately that:

Corollary 3.2 *Edges which are consecutively numbered using a uniform weighting scheme are never more than link distance 2 apart.*

Analysis of Complexity

As before, an analysis of the time complexity of our tree-ordering algorithms depends on the choice of data structure with which we represent the tree. With a topological data structure we can order the edges in linear time. If we need to build topology from an elementary list of vertices and edges, we can do so in

time $O(n \log n)$, then proceed in linear time to complete the ordering procedure. Space complexity is once again linear for edge ordering.

Branch-Recursion

As with vertices, every Eulerian tree-order of edges in branch-recursive in the same sense:

Corollary 3.3 *The collection of edges of* **any branch** *of the tree always constitute a complete interval (i.e. appear consecutively) for any cyclic Eulerian tree-ordering of edges.*

And, conversely,

Theorem 4 *Every branch-recursive cyclic ordering of the edges of a tree is an Eulerian tree-ordering of edges for some Eulerian tour of the tree and some unit-sum weight assignment to the edge visits of that Eulerian tour.*

Enumerating Edge Orders

As with vertex orders, the number of branch-recursive edge orders is equal to the number of Eulerian tours times the number of distinct edge orders for every fixed Eulerian tour. Since each edge may be weighted so that it gets enumerated either on its first visit in the Eulerian tour or on its second visit, then as long as these two visits are not adjacent in the Eulerian tour, they will produce different orderings. Two edge visits to the same edge are adjacent in an Eulerian tour if and only if the edge is incident to a leaf vertex. A tree with more than two edges and ℓ leaf vertices has exactly $n - \ell - 1$ non-leaf edges. Thus for a fixed Eulerian tour and an arbitrary unit-sum edge weighting scheme there are $2^{n-\ell-1}$ possible orderings. The number of branch-recursive edge orderings is, therefore:

$$2^{n-\ell-1} \cdot \prod_{v \in V} (\deg(v) - 1)!$$

Enumerating uniform-weight edge orders for a fixed Eulerian tour is even more trivial than enumerating uniform vertex orders. There are exactly two uniform-weight edge orders if the tree has 3 or more edges.

ORDERING SPATIAL OBJECTS

In this section we will adapt our tree-ordering techniques to order spatial objects. Our approach in every case will be to convert the ordering problem to a tree-ordering problem, then solve the tree-ordering problem by a uniform-weighting of vertices or edges, as appropriate.

Ordering Points in the Plane

Suppose that we want to assign an ordering to a set of points in the plane. We know how to order vertices of a tree. So we may convert the points into vertices by building a tree (adding edges); and one natural tree to build is a Euclidean minimum spanning tree (EMST). The EMST is unique if the points are in general position or if no two interpoint distances are equal. So the steps needed to convert the problem of ordering points in space to one of ordering tree vertices are:

1. Build Euclidean minimum spanning tree.

2. Walk Eulerian tour, tree-ordering vertices.

We can build a Euclidean minimum spanning tree in time $O(n \log n)$ [AHO2], sorting the edges at each vertex in clockwise order as they are inserted. The planar embedding of the tree gives us the geometric version of the Eulerian tour for free (*i.e.* the usual clockwise ordering of edges around a vertex). We can then walk the Eulerian tour and order the vertices in $O(n)$ additional time.

A Cluster Sampling Application. Cluster sampling is a survey sampling strategy of selecting small groups (clusters) of neighboring points instead of selecting individual points randomly distributed. Within-cluster correlation may reduce the efficiency of such a strategy from a pure sampling viewpoint, but that consideration is often outweighed by the economic impact of reduced travel costs for interviewers.

A serious limitation to successfully selecting clusters from lists, however, is the fact that proximity in the list does not guarantee proximity on the ground. Selection of points from a list that has been ordered by performing our uniform-weight tree-ordering algorithm on a EMST of the points will guarantee very strong proximity correspondence. The following theorem holds:

Theorem 5 *Order points in the plane by building their EMST and applying the uniform-weight vertex tree-ordering algorithm. Then two consecutive points in the order have a maximum link distance of six and an average link distance of less than two.*

Proof: The degree of any vertex in a EMST is less than or equal to six. Thus the uniform-weight tree-ordering algorithm accumulates a weight of at least $1/6$ with each vertex visit. Moreover, in any tree, the average degree is $\frac{2n-2}{n}$.

Ordering Points in Higher Dimensional Spaces

To apply the methods of the section on points in the plane to points in higher dimensions, we must first address two issues: (1) building a EMST in higher dimensions, and (2) defining an Eulerian tour in higher dimensions.

There are straightforward $O(n^2)$ time algorithms for building a EMST in higher dimensions [AHO1]. Some exact algorithms are known with complexity slightly sub-quadratic [YAO].

Building an Eulerian tour in higher dimensions is not so straightforward. It requires establishing a cyclic order of edges about every vertex. One possibility is to project the edges onto some two-dimensional subspace, then order the projection of the edges clockwise on that plane. Another more canonical approach, suggested by Herbert Edelsbrunner [EDE2], is to map the edge configuration about a vertex onto points on the surface of a sphere of dimension one less than the space of the EMST, then apply the ordering scheme to those points on the sphere recursively (*i.e.* build their EMST and order them in a space of smaller dimension).

In any case, if all we require is some ordering of the edges around each vertex, we can find one in $O(n \log n)$ time. We summarize the steps needed to convert the problem to a tree-ordering problem.

1. Build EMST.

2. Cyclically order edges at each vertex.

3. Walk Eulerian tour, tree-ordering vertices.

A Sample Stratification Application. Sample stratification is a partitioning of the universe into groups which are similar across several characteristics. The characteristics should be in some sense comparable (dealing with relative incomparability is sometimes known as the Scaling Problem). Stratification is often accomplished by treating the observations as n-tuples of the n characteristics (*i.e.* as points in n-space) and finding a hyperplane or collection of hyperplanes that optimize separation of the points across the half-spaces or n-cells created. A more straightforward approach to stratification (and one that would be computationally much simpler) might be to partition a EMST of the points into branches of greatest separation. With branch-recursive ordering methods, this operation boils down to list splitting! We at the Bureau of the Census will be comparing results of using tree-ordering methods to the standard more complex stratification algorithms.

Ordering Vertices of any Graph

If we are only concerned with ordering the vertices of a graph, we may think of the graph as a tree with too many edges. So we throw away the least useful edges until we have whittled the graph down to a tree. If the edges have costs associated with them, we may wish to minimize the cost of the resulting tree, for example. We know exactly how many edges to throw away. We will discard an edge as long as it does not disconnect the graph and we still have $(n-1)$ edges left. We summarize the steps needed to convert the problem to a tree-ordering problem.

1. Build a (minimum) spanning tree.
2. Cyclically order edges at each vertex.
3. Walk Eulerian tour, tree-ordering vertices.

Ordering Edges of any Graph

In the section on ordering vertices in a graph, we regarded our graph as having too many edges; and we threw some away. To order the edges of our graphs, we regard our graph as having too few vertices to be a tree; and we add vertices by splitting the vertices of the graph and creating more vertices with the same number of edges. Once again we use our knowledge of the edge/vertex relationship in a tree to know when to stop splitting vertices. We summarize the steps needed to convert the problem to a tree-ordering problem.

1. Split vertices.
2. Cyclically order edges at each vertex.
3. Walk Eulerian tour, tree-ordering edges.

We must next order the tree edges about each split vertex. Then the tree edges may be assigned a cyclic order based on selecting alternate hits from an Eulerian tour of the corresponding edges of the derived tree.

Since we can certainly split vertices in $O(n \log n)$ time using sorting and a plane sweep operation, and also order edges about each vertex in some arbitrary fashion in the same time complexity, we can accomplish the following ordering for the edges of any connected graph efficiently:

Corollary 5.1 *One may find a cyclic ordering for the edges of any connected graph in $O(n \log n)$ time so that any two edges which are consecutive in the cyclic ordering never have link distance greater than two in the graph.*

Ordering Line Segments in Two-Dimensional Networks
This is just the graph-edge ordering problem, but with fewer decisions to make because the Eulerian tour is given by the geometry. The word *network* will also imply that the topological information of the graph permits linear-time generation of the ordering. The steps for converting a connected-network edge-ordering problem to a tree-edge-ordering problem are:

1. Split vertices.
2. Walk Eulerian tour, tree-ordering edges.

Ordering Line Segments Of Networks in Higher Dimensions
The difference between this section and the section on 2-D networks lies in establishing a cyclic ordering of edges about each vertex. There may be such a structure implicitly or explicitly embedded in the topological structure of the network. We summarize the steps needed to convert the problem to a tree-ordering problem.

1. Split vertices.
2. Cyclically order edges at each vertex.
3. Walk Eulerian tour, tree-ordering edges.

Ordering Regions in the Plane
There is planar graph dual to every graph or pseudograph in the plane that is itself a pseudograph. Every region of the plane corresponds to a vertex in the new pseudograph; and two vertices in the new pseudograph are adjacent (share an edge) if and only if the regions shared a face or common side. This dual is called the *adjacency pseudograph*; and to reduce a pseudograph to a tree on the same vertex set, the procedure is the same as with a graph—you throw away edges.

We summarize the steps needed to convert the problem to a tree-ordering problem.

1. Build adjacency pseudograph.
2. Find MST.
3. Walk Eulerian tour, tree-ordering vertices.

Application to Block Numbering. Consider the problem of numbering regions of a map in such a way that consecutively numbered regions are adjacent. It is well known that not every arrangement of blocks can be so numbered. In fact, when formulated as a problem in the adjacency graph, block numbering is nothing more or less than the problem of finding a Hamiltonian path for the adjacency graph (*i.e.* a path that passes through each vertex exactly once). Even

the problem of merely deciding whether such a path exists for an arbitrary planar graph is NP-complete.

By throwing away edges so as to minimize the maximum degree of vertices in the resulting pruned tree, one may guarantee that the link distance between blocks numbered consecutively is no greater than the maximum degree of the resulting pruned tree by the same argument used to prove theorem 5.

Multistage Sampling. Sampling is often done in stages. Regions may be selected; and then individual households within selected regions may be subsampled. Region clustering, the capability of selecting groups of nearby regions, is important to reduce travel and other operational costs of surveys. ***Non-compact region clustering*** involves the selection of nearby, but non-adjacent regions. Non-compact clustering is an attempt to gain the benefits of reduced travel costs without the negative impact of high correlation.

Ordering regions by tree-ordering a pruned version of their adjacency graph will provide a reliable means of forming non-compact region clusters.

Ordering $(n-1)$-Cells in n-Dimensional Polytopal Regions

The adjacency dual pseudograph can be constructed for higher-dimensional cell decompositions. We may split vertices to realize the edges of the adjacency pseudograph as edges of a tree, as we do in this section; or we may prune edges and keep the vertices of the adjacency graph, as we do in the next section. We summarize the steps needed to convert the problem to a tree-ordering problem.

1. Build adjacency pseudograph.
2. Split vertices.
3. Cyclically order edges at each vertex.
4. Walk Eulerian tour, tree-ordering edges.

Ordering n-Cells in n-Dimensional Cell Decompositions

We summarize the steps needed to convert the n-cell ordering problem to a tree-ordering problem.

1. Build adjacency graph.
2. Find MST.
3. Cyclically order edges at each vertex.
4. Walk Eulerian tour, tree-ordering vertices.

CONCLUSIONS AND FOLLOW-UP

This introduction to branch-recursive orderings does not include empirical evaluations of the performance of those orderings. There was neither time to conduct those evaluations nor space to include them in this restricted paper. However, the principal reason for not assessing the performance empirically is that it is evident that these orderings will not do very well for the usual tasks of image analysis, range search, and nearest-neighbor-finding as studied in the recent comparative paper by Abel and Mark [ABEL]. Objects which are adjacent in the

branch-recursive ordering are fairly close in space; however, objects which are adjacent in space may be rather distant in the branch-recursive ordering. And there is no easy way to predict how distant or when discontinuities will occur with general branch-recursive orderings, as is the case with the more common quadrant-recursive orderings. The somewhat unorthodox nearness properties that are described in this paper should, nevertheless, prove very useful for sampling activities and analysis related to those activities.

The fact that many spatial entities can be realized as or identified with vertices or edges of trees or graphs makes our results widely applicable. The following example illustrates both strengths and weaknesses of our methods. Consider the two tasks of (1) finding a cyclic ordering for n points all lying on a straight line, and (2) finding a cyclic ordering for n points all lying on a circle. The reason for considering the two tasks simultaneously is that their Euclidean Minimum Spanning Trees are topologically the same: they are both linear trees, as illustrated in figure 10.

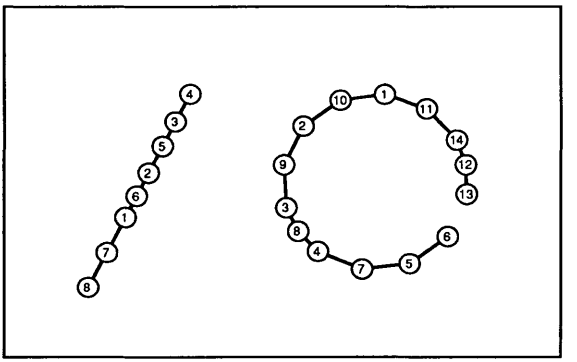

Figure 10: Cyclic Ordering of Collinear and Co-circular Points

The uniform weighting strategy will cause us to skip every other point in our numbering scheme (except at the ends of our linear tree). For the collinear points, this is clearly optimal in the following sense: This strategy minimizes the maximum distance between neighbors (*i.e.* adjacent elements in the *cyclic* numbering scheme). On the other hand, for the co-circular points, the uniform weighting strategy may produce a distance nearly double that of the optimal numbering strategy in terms of minimizing the maximum distance between neighbors.

What this example illustrates is that we necessarily lose some shape information when we embed our data in a tree and use only the topological structure of the tree from that point on. What the example also illustrates is that we may in some cases get optimal performance for cyclic orderings.

REFERENCES

[**ABEL**] Abel, David J.,and David M. Mark, 1990, *A Comparative Analysis of Some Two-Dimensional Orderings*, **International Journal of Geographical Information Systems**, 4(1), 21-31.

[**AHO1**] Aho, Alfred, John Hopcroft, and Jeffrey Ullman, 1974, **The Design and Analysis of Computer Algorithms**, Addison-Wesley, Reading, MA.

[**AHO2**] Aho, Alfred, John Hopcroft, and Jeffrey Ullman, 1985, **Data Structures and Algorithms**, Addison-Wesley, Reading, MA.

[**EDE1**] Edelsbrunner, Herbert, 1987, **Algorithms in Combinatorial Geometry**, Springer-Verlag, New York.

[**EDE2**] Edelsbrunner, Herbert, 1990, personal communication.

[**FAL1**] Faloutsos, Christos and Yi Rong, 1989, *Spatial Access Methods Using Fractals: Algorithms and Performance Evaluation*, University of Maryland Computer Science Technical Report Series, UMIACS-TR-89-31, CS-TR-2214.

[**FAL2**] Faloutsos, Christos and Shari Roseman, 1989, *Fractals for Secondary Key Retrieval*, University of Maryland Computer Science Technical Report Series, UMIACS-TR-89-47, CS-TR-2242.

[**GARE**] Garey, Michael R., and David S. Johnson, 1979, **Computers and Intractability, A Guide to the Theory of NP-Completeness**, W. H. Freeman, New York.

[**HARA**] Harary, Frank, 1969, **Graph Theory**, Addison-Wesley, Reading, MA.

[**KISH**] Kish, Leslie, 1965, **Survey Sampling**, John Wiley, New York.

[**MARK**] Mark, David M., 1990, *Neighbor-based Properties of Some Orderings of Two-Dimensional Space*, **Geographical Analysis**, April, 22(2), 145-157.

[**PREP**] Preparata, Franco, and Michael Shamos, 1985, **Computational Geometry, An Introduction**, Springer-Verlag, New York.

[**SAAL**] Saalfeld, Alan, 1990, *Canonical Cyclic Orders for Points in the Plane*, submitted to **Journal of Computational Geometry: Theory and Applications**, Elsevier.

[**WOLT**] Wolter, Kirk, and Rachel Harter, 1989, *Sample Maintenance Based on Peano Keys*, presented at Statistics Canada Symposium on Analysis of Data in Time, Ottawa, Canada.

[**YAO**] Yao, Andrew Chi-Chih, 1982, *On Constructing Minimum Spanning Trees in k-Dimensional Spaces and Related Problems*, **SIAM Journal of Computing**, November, 11(4), 721-736.