Information Processing Letters 40 (1991) 175–179
North-Holland

25 November 1991

Satisfying general proximity/similarity queries with metric trees

Jeffrey K. Uhlmann

Information Technology Division, Naval Research Laboratory, Washington, DC 20375-5000, USA

Communicated by D. Gries
Received 23 April 1991
Revised 19 September 1991

Abstract

Uhlmann, J.K., Satisfying general proximity/similarity queries with metric trees, Information Processing Letters 40 (1991) 175–179.

Divide-and-conquer search strategies are described for satisfying proximity queries involving arbitrary distance metrics.

Keywords: Computational geometry, search trees, pattern recognition, metric spaces

Introduction

A great variety of practical problems [5,6] require the efficient identification of elements from a finite set of points that are in some defined proximity to a given query point, where an efficient algorithm here is loosely defined as one that avoids the examination of every point in the set. The precise meaning of "proximity" varies from problem to problem. Problems involving spatial points, for example, often measure proximity in terms of Euclidean distance. Many correlation and assignment problems define proximity in terms of a distance measure in permutation space [3], and numerous metrics arise in areas such as pattern recognition and network optimization. The extensive literature on data structures for representing multidimensional points generally has been concerned only with problems involving proximity queries in which the query regions are approximately convex, or can be decomposed into a relatively small number of approximately convex regions. In this paper approaches are described for efficiently satisfying a large class of proximity queries involving arbitrary distance metrics.

Many data structures have been developed to satisfy certain classes of proximity queries for vectors having d arbitrary real-valued elements. Most of these data structures are based on the paradigm of recursive hyperplane decomposition. The kd tree, for example, is constructed by recursively selecting one of the coordinates and partitioning the dataset into the subset of vectors whose values for the chosen coordinate are less than the median value and the subset whose values for the coordinate are not less than the median [1]. The methods for searching the tree are then closely analogous to those used to search ordinary single-dimension binary trees. This data structure requires storage linear in the size of the dataset and usually displays good query-time performance when the number of dimensions is small. When the number of dimensions exceeds $O(\log n)$, assuming the tree is balanced, no sequence of partitions can discriminate on every coordinate. In other words, a search of the tree can only determine proximity based on a subset

of the coordinates. This problem can be partially alleviated by judicious selection of the partitioning planes. Unfortunately, techniques for selecting (and even defining) partitioning surfaces in general metric spaces have not been widely studied.

Ball decompositions

Recall the definition of a metric distance function $d(x, y)$:

(i) $d(x, y) = d(y, x)$,
(ii) $0 < d(x, y) < \infty$, $x \neq y$,
(iii) $d(x, x) = 0$,
(iv) $d(x, y) \leqslant d(x, z) + d(z, y)$ (triangle inequality).

Most commonly used distance and similarity measures between objects satisfy the above conditions [1]. Consider for example $d(x, y)$ giving the minimum time required to travel between points x and y. Assuming there are no one-way streets, condition (i) is satisfied. As long as it is possible to travel from x to y, condition (ii) is satisfied. Condition (iii) is trivially satisfied. And condition (iv) is always satisfied since at least equality is assured by traveling from x to z and then from z to y. A query requiring the identification of the set of points within some travel time of a given point defines a query region that is not generally convex. Specifically, long and winding fingers may follow major thoroughfares while shorter ones extend along other avenues. Thus, data structures that efficiently support proximity queries only when the query region is approximately convex may fail to achieve any appreciable performance advantage over brute force. A data structure that exploits local features of the metric is required.

Given a finite set S of n objects with metric $d(S_i, S_j)$, $S_i, S_j \in S$, a *ball decomposition* [7] $\mathscr{B}$ is a metric tree constructed from S, using $O(n \log n)$ distance calculations, by the following:

1. If $|S| = 0$, then create the empty tree $\mathscr{B} = nil$.
2. Otherwise:
   Let $\mathscr{B}_x$ be an arbitrary object from S,
   $\mathscr{B}_m \leftarrow$ median of $\{d(\mathscr{B}_x, y)\ \forall y \in S\}$,
   $\mathscr{B}_{left} \leftarrow$ metric tree of $\{S_i \mid d(\mathscr{B}_x, S_i) \leqslant \mathscr{B}_m,\ S_i \neq \mathscr{B}_x\}$,
   $\mathscr{B}_{right} \leftarrow$ metric tree of $\{S_i \mid d(\mathscr{B}_x, S_i) \geqslant \mathscr{B}_m\}$.

Basically, the construction process consists of selecting one of the objects, $\mathscr{B}_x$, and finding the median distance $\mathscr{B}_m$ such that half of the objects are within the metric ball of radius $\mathscr{B}_m$ about $\mathscr{B}_x$. The set is then partitioned by $\mathscr{B}_m$ and the process is recursively applied. (The redundant conditions, "$\leqslant$" and "$\geqslant$," assure balance when the median value is not unique. This is accomplished by assigning each element on the partition to one of the subtrees in an arbitrary, but balanced, fashion. A routine for searching the tree must then use the same redundant inequalities.)

This decomposition is interesting because it requires nothing to be known about the objects other than their pairwise distances. Thus, it is appropriate for applications dealing with sets of objects whose topological relationships are not characterized analytically. For example, many practical routing problems involve points whose pairwise distances are determined by a function evaluating the minimum cost to traverse from one point to another (e.g., over various types of terrain). Having established that the distances satisfy the metric conditions, the set of points within distance r of a point v can be identified from the metric tree $\mathscr{B}$ as follows:

1. If $d(\mathscr{B}_x, v) \leqslant r$, then $\mathscr{B}_x$ is an element of the desired set.
2. If $d(\mathscr{B}_x, v) + r \geqslant \mathscr{B}_m$, then recursively search $\mathscr{B}_{right}$.
   If $d(\mathscr{B}_x, v) + \mathscr{B}_m \leqslant r$, then every point in $\mathscr{B}_{left}$ is an element of the desired set.

[1] Many other distance-like measures can be transformed to metrics. For example, the pseudodistance measure $d(x, y) = (x - y)^2$ fails to satisfy the triangle inequality, but $d'(x, y) = \sqrt{d}(x, y)$ transforms it to the Euclidean metric. Thus, most proximity queries for d can be transformed to queries for metric d' that yield the same solution set. Notable exceptions include many distance measures between *sized* objects (e.g., spatial objects, sets, probability distributions, etc.). The data structures described in this paper can be enhanced to process sized objects using techniques applied in [9].

Else, if $d(\mathscr{B}_x, v) - r \leqslant \mathscr{B}_m$, then recursively search $\mathscr{B}_{left}$.

More generally, the set of objects in an arbitrary region χ can be identified from $\mathscr{B}$ as follows:

1. If $\mathscr{B}_x$ is in χ, then $\mathscr{B}_x$ is an element of the desired set.
2. If χ intersects the region associated with $\mathscr{B}_{right}$, then recursively search $\mathscr{B}_{right}$.
   If the ball associated with $\mathscr{B}_{left}$ is entirely within χ, then every point in $\mathscr{B}_{left}$ is an element of the desired set.
   Otherwise, if χ intersects the ball associated with $\mathscr{B}_{left}$, then recursively search $\mathscr{B}_{left}$.

Ball decompositions often provide good query-time performance because their use of dataset elements in defining partitions tends to allow them to exploit distribution features. Unfortunately, for uniformly distributed points in common metric spaces such as Euclidean, Manhattan, etc., the surface area associated with ball partitions tends to be greater within the convex hull of the points in the dataset than the amount of area associated with the intersection of partitioning planes and the interior of the hull. In other words, under these condititions searches of ball decompositions will be more expensive because query regions are more likely to intersect the partitions. [2] This suggest that a generalization of the hyperplane decomposition approach to other metric spaces might also provide superior performance.

Generalized hyperplane decompositions

One of the most attractive features of the ball decomposition is that it can be computed given only that the distance measure satisfies four relatively modest conditions (and even these conditions can be further relaxed with minor enhancements to the algorithm). How to define hyperplane partitions, unlike ball partitions, is not obvious from such limited information. A definition that coincides with intuitive, as well as most analytic notions of hyperplanes is the following:

Definition. A *generalized hyperplane* (GH) is defined by two points p_1 and p_2, $p_1 \neq p_2$, and consists of the set of points q satisfying $d(q, p_1) = d(q, p_2)$. A point x is said to lie on the p_1-side of the plane if $d(x, p_1) < d(x, p_2)$.

Like ball partitions, GH partitions tend to exploit distribution features when defined by randomly selected points from the dataset. For example, a set of points distributed on a line in d-dimensional Euclidean space will generate GH partition planes orthogonal to that line. An advantage of GH over ball decompositions is that the latter are strongly static data structures while the former are not. In particular, a ball partition consists of a center point and a radius that is determined by examining each point in the dataset and identifying the median distance from that center point. Without knowledge about the distribution extent of the points to be processed, there can be no strategy for dynamically selecting radii that can be expected to produce an approximately balanced data structure. [3] A heuristic sampling argument, however, suggests that GH partitions usually can be expected to produce approximately balanced decompositions because they are determined by randomly sampled points (the pairs of defining points) that they partition into equally sized subsets (each partition separates its two defining points). Calculating the expected deviation from perfect balance, of course, requires additional distribution and metric information.

[2] In the case of binary data using the Manhattan metric, however, every point represents a vertex of a multidimensional hypercube. Every ball partition is therefore centered at a vertex and will intersect the space identically to a hyperplane. Tests reveal that only 10–100 distance calculations are required to satisfy small-radius queries (having solution sets of approximately 5 vectors) on datasets of size $1K$ to $16K$ randomly selected binary vectors of lengths $1K$ to $16K$, $K = 1024$ [8].

[3] However, dynamic techniques have been developed that yield good *amortized* performance for insertions and deletions in general tree structures. These approaches perform no balancing until a subtree becomes sufficiently unbalanced, whereupon it is completely reconstructed by using the static construction algorithm [4].

The dynamic insertion of a point p into a GH-based tree $\mathcal{H}$ can be effected by the function Insert($\mathcal{H}$, p) defined as follows:

1. If $\mathcal{H}$ is empty, then create and return a node $\mathcal{H}$ with $\mathcal{H}_x = p$ and $\mathcal{H}_y$, $\mathcal{H}_{\text{left}}$, and $\mathcal{H}_{\text{right}} = nil$, where $\mathcal{H}_x$ and $\mathcal{H}_y$ are the two points defining the partition and $\mathcal{H}_{\text{left}}$ and $\mathcal{H}_{\text{right}}$ are the two subtrees.
2. If $\mathcal{H}_y = nil$, then return $\mathcal{H}$ with $\mathcal{H}_y = p$.
3. If $d(p, \mathcal{H}_x) \leq d(p, \mathcal{H}_y)$, then return $\mathcal{H}$ with $\mathcal{H}_{\text{left}} = \text{Insert}(\mathcal{H}_{\text{left}}, p)$.
4. Return $\mathcal{H}$ with $\mathcal{H}_{\text{left}} = \text{Insert}(\mathcal{H}_{\text{right}}, p)$.

The advantages of GH over ball decompositions in the dynamic case are gained at a cost. Without knowing that the distance measure defines a linear space, for example, it is not generally possible to translate generalized hyperplanes to assure balanced decompositions even in the static case.

Assume a set of points characterizes a region such that a point in the region lies on a given side of a GH partition if and only if one of the characterizing points lies on that side of the partition. (In Euclidean space, for example, a polyhedral region would be fully characterized by its vertex points.) The set of points in a GH decomposition $\mathcal{H}$ within a region characterized by a set of points ϕ can be found simply by comparing each element of ϕ to the partition associated with each visited node of $\mathcal{H}$. A branch of the tree is then examined if and only if an element of ϕ lies on the side of the partition, associated with that branch. In many cases, an analytic characterization of the search region can avoid the need for a set ϕ and can permit the comparison step to be computed more efficiently. In general, however, the set of queries that can be satisfied by GH-decompositions is severely limited by the fact that, without additional information, only points can be compared to GH partitions. For example, given only a blackbox distance function, two points defining a partition plane, and a ball of radius r about a point x, it is not possible to determine whether the ball about x intersects the partition plane. Thus GH decompositions are less general than ball decompositions. In practice, however, it is rare for so little information to be known as to preclude the use of GH partitions.

Conclusions

In summary, metric trees extend the divide-and-conquer paradigm to search problems in arbitrary metric spaces. Problem areas in which such general techniques can be valuable include pattern recognition and a variety of network optimization applications. Although these data structures usually provide query-time performance that is significantly better than brute force, they cannot guarantee such performance for all queries in all metric spaces. For example, worst-case performance for ball decompositions can be demonstrated by a set of points distributed on the surface of the unit ball and a query asking for the set of points strictly within the unit ball. In this case both the query region and its complement will be intersected by every partition and every point therefore will require examination. (This performance is analagous to that of kd trees for deviously selected half-space queries [2].) In practice, however, the potential for improved average-case performance usually justifies the application of metric trees instead of brute force. In special cases, of course, it is possible to establish superior worst-case bounds for metric trees. In the case of range queries in the L_∞ metric, for example, virtually all of the analyses conducted on kd trees can also be applied to GH decompositions.

References

[1] J. Bentley, Multidimensional binary search trees for associative searching, *Comm. ACM* **18** (1975) 509–515.
[2] H. Edelsbrunner, *Algorithms in Combinatorial Geometry* (Springer, Berlin, 1987) Chapter 14.
[3] D. Golenko-Ginzburg, Metrics in the permutation space, *Appl. Math. Lett.* 4 (2) (1991) 5–7.
[4] K. Mehlhorn, *Multidimensional Searching and Computational Geometry* (Springer, Berlin, 1984) 37–38.
[5] F. Preparata and M. Shamos, *Computational Geometry: An Introduction* (Springer, New York, 1985).
[6] H. Samet, *The Design and Analysis of Spatial Data Structures* (Addison-Wesley, Reading, MA, 1990) Preface.

[7] J. Uhlmann, Metric trees, *Appl. Math. Lett.* **4** (5) (1991).
[8] J. Uhlmann, Real-time decision support with metric trees, in: *Proc. Command and Control Decision Aids Conf.*, Washington, DC, July, 1991.
[9] J. Uhlmann, Adaptive partitioning strategies for ternary tree structures, *Pattern Recognition Lett.* **12** (9) (1991) 537–541.