Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE @ DIRECT®

Pattern Recognition Letters 24 (2003) 2785–2795

Pattern Recognition Letters

www.elsevier.com/locate/patrec

Improved search heuristics for the sa-tree

Gísli R. Hjaltason [a,*], Hanan Samet [b]

[a] *School of Computer Science, University of Waterloo, 200 University Ave W, Waterloo, Ont., Canada N2L 3G1*
[b] *Computer Science Department, Center for Automation Research, Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, USA*

Received 31 December 2002; received in revised form 22 April 2003

Abstract

The *sa-tree* is an interesting metric space indexing structure that is inspired by the Voronoi diagram. In essence, the sa-tree records a portion of the Delaunay graph of the data set, a graph whose vertices are the Voronoi cells, with edges between adjacent cells. An improvement is presented on the original search strategy for the sa-tree. This consists of details on the intuition behind the improvement as well as the original search strategy and a proof of their correctness. Furthermore, it is shown how to adapt an incremental nearest neighbor algorithm to the sa-tree, which allows computing nearest neighbor in a progressive manner. Unlike other adaptations, the resulting algorithm does not take the unnecessary steps to ensure that keys of "node" elements are monotonically non-decreasing.
© 2003 Elsevier B.V. All rights reserved.

Keywords: Metric spaces; Distance-based indexing; Nearest neighbor algorithms

1. Introduction

Similarity searching is an important task when trying to find patterns in applications involving mining different types of data such as images, video, time series, text documents, DNA sequences, etc. (e.g., see Han and Kamber, 2000). Similarity searching often reduces to a question of finding the nearest neighbors to a query object. Usually, this data is not drawn from a vector space. Instead, we are given the data and a distance metric or function that enables the computation of interobject distances. In other words, we are given an underlying space $\mathbb{U}$ and a set of objects $S \subset \mathbb{U}$ such that for each query object q in $\mathbb{U}$, we wish to find the nearest object in s to q (or, more generally, the k nearest objects to q or the objects within ε of q). A primary challenge in performing similarity search for such data is that the evaluation of the distance function d is typically quite expensive. One approach in addressing this challenge is to construct index structures that are based solely on distances between objects. Examples of such structures are the vp-tree (Uhlmann, 1991; Yianilos, 1993), GNAT (Brin, 1995), and M-tree (Ciaccia et al., 1997), see Chávez et al. (2001) and Hjaltason and Samet (2003) for surveys of these and other distance-based indexing structures. The sa-tree (Navarro, 2002) is an example of another distance-based index that is

[*] Corresponding author.
E-mail addresses: gisli@db.uwaterloo.ca (G.R. Hjaltason), hjs@cs.umd.edu (H. Samet).

0167-8655/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S0167-8655(03)00122-3

based on an analogy to the use of the Voronoi diagram in geometric space. Although other distance-based indexing methods, such as GNAT, were also inspired by the Voronoi diagram, the sa-tree is a novel departure in the way it is constructed and searched.

To understand the sa-tree, it is important to look at how Voronoi diagrams (Voronoi, 1909) can be used for performing search. In a Voronoi diagram for point data, for each "site" p, the Voronoi cell of p identifies the area closer to p than to any other site. Thus, given a query point q, nearest neighbor search simply involves identifying the Voronoi cell that contains q. Another, somewhat indirect, way of constructing a search structure for nearest neighbor search based on the Voronoi diagram is to build a graph termed a *Delaunay graph*, defined as the graph where each object is a node and two nodes have an edge between them if their Voronoi cells have a common boundary (in an earlier publication, Navarro, 1999 used the term "Voronoi graph"). In other words, the Delaunay graph is simply an explicit representation of neighbor relations that are implicitly represented in the Voronoi diagram; clearly, Delaunay graphs are closely related to Delaunay triangulations, the difference being that in the latter, the edges have an associated geometric shape.

Searching a Delaunay graph for the nearest neighbor in S of a query point q in $\mathbb{U}$ starts with an arbitrary point in S, and proceeds to a neighboring point in S that is closer to q as long as this is possible. Once we reach a point o in S where the points in its neighbor set $N(o)$ in S (i.e., the points connected to o by an edge) are all farther away from q than o, we know that o is the nearest neighbor of q. The reason this search process works on the Delaunay graph of a set of points is that the Delaunay graph has the property that if q is closer to a point p than to any of the neighbors of p in the Delaunay graph, then p is the point in S closest to q. The same search process can be used on any graph that satisfies this *Voronoi property*. In fact, for an arbitrary metric space $(\mathbb{U}, d)$, a Delaunay graph for a set $S \subset \mathbb{U}$ is a minimal graph that satisfies the Voronoi property (i.e., removing any edge would cause violation of the property). Thus, any graph that satisfies the Voronoi property must include a Delaunay graph as a subgraph. Note, however, that the Delaunay graph is not necessarily unique as there can be several such minimal graphs (possibly even with a different number of edges).

The Voronoi diagram serves as the inspiration for the sa-tree (Navarro, 1999; Navarro, 2002), in that the sa-tree attempts to approximate the structure of the Delaunay graph (its name is an abbreviation for *Spatial Approximation Tree*). Unfortunately, Voronoi cells (or, perhaps more accurately, Dirichlet domains (Brin, 1995)) for data objects cannot be constructed explicitly (i.e., their boundaries specified) if only interobject distances are available. Moreover, it is possible to show (Navarro, 2002) that without more information about the structure of the underlying metric space $(\mathbb{U}, d)$, just knowing the set of interobject distances for a finite metric space (S, d), $S \subset \mathbb{U}$, is not enough to enable the construction of a valid Delaunay graph for S based on d—that is, we also need information about the distances between the elements of S and the elements of $\mathbb{U}$. In other words, for the two sets $S \subset \mathbb{U}$ and $S' \subset \mathbb{U}'$ with identical interobject distances (i.e., (S, d) and (S', d') are isometric), possibly drawn from different underlying spaces $\mathbb{U}$ and $\mathbb{U}'$, (S, d) may have a Delaunay graph D that is not a Delaunay graph for (S', d'), or vice versa. [1]

[1] For example, suppose that $\mathbb{U} = \mathbb{U}' = \{a, b, c, x\}$, $d(a, b) = d(a, c) = d(b, c) = 2$ and $d'(a, b) = d'(a, c) = d'(b, c) = 2$. Furthermore, assume that $d(a, x) = 1$, $d(b, x) = 2$, and $d(c, x) = 3$ while $d'(a, x) = 3$, $d'(b, x) = 2$, and $d'(c, x) = 1$. If $S = S' = \{a, b, c\}$, the distance matrices for the two sets are the same. The graph with edges (a, b) and (a, c) (i.e., $N(a) = \{b, c\}$ and $N(b) = N(c) = \{a\}$) satisfies the Voronoi property for (S, d), since the nearest neighbor of any query object drawn from $\mathbb{U}$ can be arrived at starting at any object in S by only transitioning to neighbors that are closer to or at the same distance from the query object. Thus this graph is a Delaunay graph for (S, d). However, it does not satisfy the Voronoi property for (S', d'), since starting at b with $q = x$, b's only neighbor a is farther away from x than b is, so we cannot transition to the nearest neighbor c of x. Thus it is not a Delaunay graph for (S', d'). It is interesting to note that the graph with edges (a, b) and (b, c) (i.e., $N(b) = \{a, c\}$ and $N(a) = N(c) = \{b\}$) satisfies the Voronoi property for both (S, d) and (S', d') and thus it is a Delaunay graph for both (S, d) and (S', d'). Of course, this example does not invalidate our observation that knowledge of (S, d) is insufficient to determine the Delaunay graph.

Moreover, for any two objects a and b, a finite metric space (S, d) exists whose Delaunay graph contains the edge between a and b. Hence, given only the interobject distances for a set S, the only way to construct a graph G such that G satisfies the Voronoi property for all potential query objects in $\mathbb{U}$ (i.e., contains all the edges in the Delaunay graph) is for G to be the complete graph—that is, the graph containing an edge between all pairs of nodes (each of which represents an object in S). However, such a graph is useless for search, as deciding on what edge to traverse from the initial object in S requires computing the distances from the query object to all the remaining objects in S (i.e., it is as expensive, $O(N)$, as brute-force search). The idea behind the sa-tree is to approximate the proper Delaunay graph with a tree structure that retains enough edges to be useful for guiding search, but not so many that an excessive number of distance computations are required when deciding on what node to visit next.

In this paper, we describe the sa-tree in more detail. Section 2 defines the sa-tree. Section 3 is the main part of the paper and discusses how to search in an sa-tree. In particular, we introduce an improvement over the search strategy originally proposed by Navarro (1999). This improved strategy was later adopted by Navarro (2002) (based on our suggestion). Here, we also provide more details on the intuition behind the improvement, prove its correctness, and discuss its limitations. Section 4 presents an algorithm for finding nearest neighbors incrementally. One of its novel features is that unlike the algorithm of Navarro (2002), our algorithm does not take the unnecessary steps to ensure that keys of "node" elements are monotonically non-decreasing. Section 5 contains concluding remarks and some suggestions for future research.

2. Definition of the sa-tree

The sa-tree is an indexing method on a finite metric space (S, d), where $S \subset \mathbb{U}$ is a set of objects and d is a distance metric. This means that d satisfies the following three properties, where o_1, o_2, $o_3 \in S$:

1. $d(o_1, o_2) = d(o_2, o_1)$ (symmetry)
2. $d(o_1, o_2) \geqslant 0, d(o_1, o_2) = 0$ iff $o_1 = o_2$ (non-negativity)
3. $d(o_1, o_3) \leqslant d(o_1, o_2) + d(o_2, o_3)$ (triangle inequality)

The sa-tree is defined as follows (see the example in Fig. 1 to clarify some of the questions that may arise). An arbitrary object a is chosen as the root node of the tree (since each object is associated with exactly one node, we use the terms object and node interchangeably in this discussion). Next, a smallest possible set $N(a) \subset S \setminus \{a\}$ is identified, such that x is in $N(a)$ iff for all $y \in N(a) \setminus \{x\}$, $d(x, a) < d(x, y)$. The set $N(a)$ is termed the neighbor set of a, by analogy with the Delaunay graph, and the objects in $N(a)$ are said to be the neighbors of a. Intuitively, for a legal neighbor set $N(a)$ (i.e., not necessarily the smallest such set), each object in $N(a)$ is closer to a than to the other objects in $N(a)$, and all the objects in $S \setminus N(a)$ are closer to one of the objects in $N(a)$ than to a. The objects in $N(a)$ then become children of a. The remaining objects in S are associated with the closest child of a (i.e., the closest object in $N(a)$), and the subtrees are defined recursively in the same way for each child of a. As we shall see below, it is useful for search to store in each node b the distance $d_{\max}(b)$ to the farthest object in the subtree rooted at b. More precisely, $d_{\max}(b) \equiv \max_{o \in S_b} d(o, b)$, where S_b denotes the set of objects in the subtree rooted at b. Fig. 1b shows a sample sa-tree for

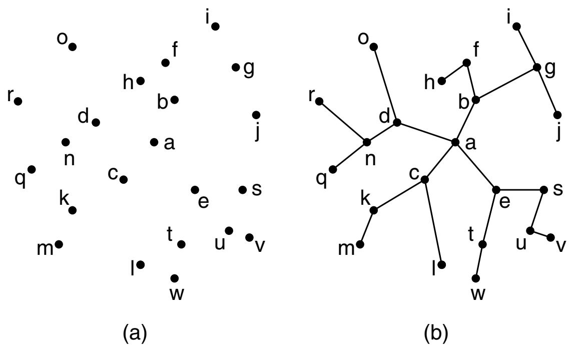

Fig. 1. (a) A set of points in a two-dimensional Euclidean space, and (b) its corresponding sa-tree constructed using the algorithm of Navarro (2002) when a is chosen as the root.

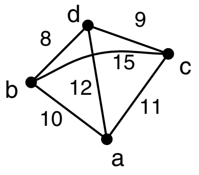

Fig. 2. An example of four points a, b, c, d where the sa-tree construction algorithm does not find the minimal neighbor set $N(\mathsf{a})$.

the two-dimensional points a–w given in Fig. 1a, with a chosen as the root. In this example, $N(\mathsf{a}) = \{\mathsf{b,c,d,e}\}$. Note that h is not in $N(\mathsf{a})$ as h is closer to b than to a.

The fact that the neighbor set $N(a)$ is used in its definition (i.e., in a sense, the definition is circular) makes constructing a minimal set $N(a)$ expensive. In fact, Navarro (2002) argues that its construction is an NP-complete problem. Thus, Navarro (2002) resorts to a heuristic for identifying the neighbor set. This heuristic considers the objects in $S \setminus \{a\}$ in the order of their distance from a, and adds an object o to $N(a)$ if o is closer to a than to the existing objects in $N(a)$. In fact, the sa-tree in Fig. 1b has been constructed using this heuristic with a chosen as the root. An example of a situation where the heuristic would not find the minimal neighbor set is shown in Fig. 2, where approximate distances between four two-dimensional points a through d are labeled. The minimum neighbor set of a in this case is $N(\mathsf{a}) = \{\mathsf{d}\}$ (and $N(\mathsf{d}) = \{\mathsf{b,c}\}$) whereas use of the heuristic would lead to $N(\mathsf{a}) = \{\mathsf{b,c}\}$ (and $N(\mathsf{b}) = \{\mathsf{d}\}$). Although the heuristic does not necessarily find the minimal neighbor set, it is deterministic in the sense that for a given set of distance values, the same neighbor set is found (except for possible ties in distance values). Thus, using the heuristic, the structure of the sa-tree is uniquely determined once the root has been chosen. However, different choices of the root lead to different tree structures.

Using the sa-tree, it is easy to perform exact match queries (i.e., to search for an object in S) with the same procedure as in the Delaunay graph as described in Section 1. Of course, this is not very useful, as the query object is typically not in S in most actual queries. In the next section, we show how to perform more general queries.

3. Search in the sa-tree

When searching the sa-tree with respect to query object q, we exploit the relationship between the objects in the nodes. In particular, given a subtree T having an object p_1 in its root node, let p_2 be an object that is known to be farther away from all objects o in T than is p_1 (i.e., $d(p_1, o) \leqslant d(p_2, o)$ for all o in T). Section 3.1 shows how to derive a lower bound on $d(q, o)$ for all objects in T based on this information. Section 3.2 sketches Navarro's (1999) original proposal for how to choose p_2. Section 3.3 gives the intuition and a proof of correctness of our method of choosing p_2 from a larger set (later adopted by Navarro (2002)), thereby enabling us to obtain a tighter lower bound.

3.1. Lower bound on distances

The following lemma provides the desired lower bound in the situation outlined above:

Lemma 1. *Let $o \in \mathbb{U}$ be an object that is closer to p_1 than to p_2, or equidistant from both (i.e., $d(p_1, o) \leqslant d(p_2, o)$). Given $d(q, p_1)$ and $d(q, p_2)$, we can establish a lower bound on $d(q, o)$:*

$$\max\left\{\frac{d(q,p_1) - d(q,p_2)}{2}, 0\right\} \leqslant d(q,o). \tag{1}$$

Proof. From the triangle inequality, we have $d(q,p_1) \leqslant d(q,o) + d(p_1,o)$, which yields $d(q,p_1) - d(q,o) \leqslant d(p_1,o)$. When combined with $d(p_2,o) \leqslant d(q,p_2) + d(q,o)$ (from the triangle inequality) and $d(p_1,o) \leqslant d(p_2,o)$, we obtain $d(q,p_1) - d(q,o) \leqslant d(q,p_2) + d(q,o)$. Rearranging yields $d(q,p_1) - d(q,p_2) \leqslant 2d(q,o)$, which yields the first component of the lower bound in Eq. (1), the second component being furnished by non-negativity. □

One way to get some intuition about this result is to consider the situation shown in Fig. 3a where q lies on the line between p_1 and p_2 in a two-dimensional Euclidean space, closer to p_2. If o is closer to p_1, then o is to the left of the horizontal line midway between p_1 and p_2 which separates the regions in which objects are closer to p_1 or to p_2. Thus, $d(\mathsf{q,o})$ is lower-bounded by the distance

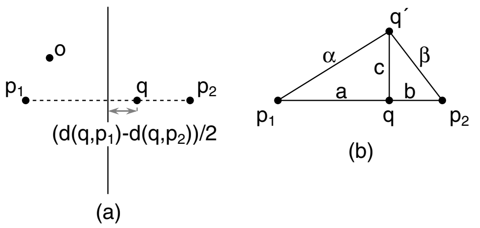

Fig. 3. (a) Lower bound on $d(\mathrm{q},\mathrm{o})$, illustrated in a two-dimensional Euclidean space when q is on the line between p_1 and p_2, closer to p_2, while o is closer to p_1. (b) The lower bound can be shown to decrease when q is moved off the line (e.g., to q′).

from q to the dividing line, which equals $(d(\mathrm{q},\mathrm{p}_1) - d(\mathrm{q},\mathrm{p}_2))/2$ for the particular position of q in the figure. If we move q up or down parallel to the dividing line (e.g., up in Fig. 3b to position q′), the distance from q′ to the line is clearly unchanged (i.e., it is still $(d(\mathrm{q},\mathrm{p}_1) - d(\mathrm{q},\mathrm{p}_2))/2$). However, the difference between $d(\mathrm{q}',\mathrm{p}_1)$ and $d(\mathrm{q}',\mathrm{p}_2)$ can be shown to decrease as both increase, [2] so the value of $(d(\mathrm{q}',\mathrm{p}_1) - d(\mathrm{q}',\mathrm{p}_2))/2$ will also decrease. In other words, we see that the distance from q to the dividing line in the figure is exactly $(d(\mathrm{q},\mathrm{p}_1) - d(\mathrm{q},\mathrm{p}_2))/2$, while $(d(\mathrm{q}',\mathrm{p}_1) - d(\mathrm{q}',\mathrm{p}_2))/2$ decreases as q′ is moved while keeping the distance from q′ to the dividing line constant. Therefore, the value $(d(\mathrm{q},\mathrm{p}_1) - d(\mathrm{q},\mathrm{p}_2))/2$ is indeed a lower bound on the distance from q to any point on the dividing line or from any point q′ on the line parallel to the dividing line that passes through q, and thus also a lower bound on the distance between q and o. Note that this argument holds for all positions of q that are closer to p_2 than to p_1, as the initial position of q can be anywhere on the line between p_1 and p_2. Observe that without additional information, an upper bound on $d(\mathrm{q},\mathrm{o})$ cannot be established, as o may be arbitrarily far away from p_1 or p_2.

[2] Fig. 3b depicts the relative distances for a query point q′ that is above q. From $\alpha^2 = a^2 + c^2$ we obtain $\alpha^2 - a^2 = (\alpha - a)(\alpha + a) = c^2$ or $\alpha - a = c^2/(\alpha + a)$. In the same manner, we can show that $\beta - b = c^2/(\beta + b)$. Since q is closer to p^2, we have $a > b$ and $\alpha > \beta$, and therefore $\alpha + a > \beta + b$. Thus, $\alpha - a = c^2/(\alpha + a) < c^2/(\beta + b) = \beta - b$, implying that $\alpha - \beta < a - b$, and thus $(d(\mathrm{q}',\mathrm{p}_1) - d(\mathrm{q}',\mathrm{p}_2))/2 < (d(\mathrm{q},\mathrm{p}_1) - d(\mathrm{q},\mathrm{p}_2))/2$.

3.2. Original search strategy

Nearest neighbor and range search can be performed in the sa-tree for arbitrary query objects q by using the observation in Lemma 1. In particular, if a is the object corresponding to a root node, let c be some object in $\{a\} \cup N(a)$. Letting b be an arbitrary object in $N(a)$ and o be an object in the subtree associated with b (i.e., rooted at b), we know that o is closer to b than to c (or equidistant, e.g., if $c = b$). Thus, we can apply Lemma 1 to yield the lower bound $(d(q,b) - d(q,c))/2$ on $d(q,o)$—that is, o is at a distance of at least $(d(q,b) - d(q,c))/2$ from q. Since o does not depend on c, we can select c in such a way that the lower bound on $d(q,o)$ is maximized, which occurs when $d(q,c)$ is as small as possible—that is, c is the object in $\{a\} \cup N(a)$ that is closest to q.

When performing range search with query radius ε, we can use the lower bound on the distances derived above to prune the search. In particular, the search is realized with a depth-first traversal of the tree, starting at the root. When at some node a, we first determine the object $c \in \{a\} \cup N(a)$ for which $d(q,c)$ is minimized. Next, the search traversal visits each child $b \in N(a)$, except those for which $(d(q,b) - d(q,c))/2 > \varepsilon$ (or, equivalently, $d(q,b) > d(q,c) + 2\varepsilon$, as used in Navarro, 2002), since in this case we know that $d(q,o) > \varepsilon$ for any object o in the subtree associated with b.

Note that this strategy is the same as the one that we would use in determining which Voronoi regions to examine when performing a range search in a Voronoi diagram (i.e., find all objects within ε of query object q). In essence, we compute the distance from q to each site s_i of Voronoi region v_i and then choose the closest site s to q and eliminate every Voronoi region whose site s_i satisfies $(d(q,s_i) - d(q,s)) > 2 \cdot \varepsilon$ as the intersection of the Voronoi region v_i of s_i with the query range of radius ε centered at q is empty. In this case, the set $N(a)$ plays the role of the sites of the Voronoi regions and c plays the role of the site that is closest to q.

In addition, the search traversal can also make use of $d_{\max}(b)$ (the maximum distance from b of the objects in the subtree rooted at b) for pruning, thus discarding subtrees for which $d(q,b) - d_{\max}(b) > \varepsilon$. To see why this can be done, we observe that from the triangle inequality we know that $d(q,o) \geqslant d(q,b) - d(b,o)$ for any object o including any object o in the subtree rooted at b. We also know that for every object o in the subtree rooted at b we have that $d_{\max}(b) \geqslant d(b,o)$. Substituting this inequality into $d(q,o) \geqslant d(q,b) - d(b,o)$ shows that

$$d(q,o) \geqslant d(q,b) - d_{\max}(b) \quad \text{for all objects } o \text{ in the subtree rooted at } b. \tag{2}$$

Therefore, whenever $d(q,b) - d_{\max}(b) > \varepsilon$, we have that $d(q,o) > \varepsilon$ as well, and thus any object o in the subtree rooted at b can be pruned in this case.

It should be clear that the search process that we have described may have to descend several of the subtrees of a in the process of determining the objects that satisfy the range query, and thus it may require backtracking. This is because, unlike exact match search in an sa-tree where the fact that we know in advance the identity of the object q that we are seeking means that we only pursue one path to find it, in the range query we do not know in advance the identity of the objects that will satisfy the range and thus more paths in the tree must be pursued.

3.3. Improved search strategy

In the search algorithm sketched above, instead of basing the selection of c on the set $\{a\} \cup N(a)$ (i.e., based on their distances from q), we can use the larger set $\bigcup_{a' \in A(b)} (\{a'\} \cup N(a'))$, where $A(b)$ is the set of ancestors of b and b is in $N(a)$. This strategy makes it more likely that c is close to q (since a larger set is used to select it), thus possibly providing a greater value for the lower bound $(d(q,b) - d(q,c))/2$ on $d(q,o)$. The correctness of choosing c in this manner is shown in the following lemma.

Lemma 2. *Let a be a node in an sa-tree and let b' be an object in the subtree rooted at b, where $b \in N(a)$ (i.e., b is a child of a). Then, b' is closer to b (or equidistant) than to any of the ancestors of b or their immediate children—that is,*

$$\forall c \in A_N(b) \equiv \bigcup_{a' \in A(b)} \{a'\} \cup N(a'), d(b',b) \leqslant d(b',c)$$

We call the objects in $A_N(b)$ the "ancestor neighbors" of b where we define the ancestor neighbor of the root to be the root itself.

Proof. We divide the proof into two parts. First, we show that b' is closer to b than to all of the ancestors of b, and then we do the same for all of the children of the ancestors.

Let c be an ancestor of b (or b itself), and let c' be the parent of c. We claim that b' is closer to c than to c', or, more accurately, $d(b',c) \leqslant d(b',c')$. To show this, we will use contradiction. Assume that $d(b',c) > d(b',c')$. Since $c \in N(c')$, the definition of the neighbor set means then demands that b' should also be in $N(c')$, but this contradicts the original assumption that c is an ancestor of b (and, by extension, of b'). Thus, our claim holds, implying that the ancestors of b are progressively farther away from b' (or, more accurately, "farther away or equidistant"). Clearly, this implication subsumes the statement that we wished to prove, that b' is closer to b than to ancestors of b.

Now, let a' be an ancestor of b (including, again, b itself), and let c be a member of $N(a')$ that is not an ancestor of b; the case of c being an ancestor was treated above. Suppose that b' is closer to c than to b, i.e., $d(b',c) < d(b',b)$. Letting $c' \in N(a')$ be the ancestor of b among the siblings of c, we saw above that $d(b',b) \leqslant d(b',c')$. This, coupled with our assumption about c, further implies that $d(b',c) < d(b',c')$. However, in this case, b' should have been associated with the subtree rooted at c instead of the one rooted at c', thereby contradicting the assumption that c is not an ancestor of b. Hence, b' cannot be closer to c than to b, and the proof is complete. □

The proof of Lemma 2 demonstrates that the object c that maximizes the lower bound $d(q,o)$ can be chosen from any of b's ancestors or siblings. Recalling that the search in the sa-tree proceeds in a top–down manner starting at its root, it may appear that Lemma 2 means that we must examine the ancestors and all their children each time that a node is visited. However, this is not necessary, as we can simply keep track of the ancestor neighbor c that is closest to q as the tree is descended. Thus, when visiting a new node a, we evaluate $d(q,b)$ for

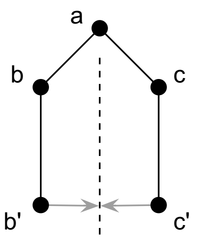

Fig. 4. A simple sa-tree for five two-dimensional points, with a as root. The points b′ and c′ can move arbitrarily close to the broken line, as indicated with the gray arrows, without giving rise to a different tree structure.

all $b \in N(a)$ and replace c as the closest to q if $d(q,b) < d(q,c)$ for some b. Actually, rather than c itself, what we keep track of is the distance value $d(q,c)$.

Having succeeded in enlarging the set of objects from which c is selected during the search, it is tempting to speculate whether an even larger set can be used. Unfortunately, this hope is unlikely to be fulfilled, as demonstrated by the example in Fig. 4. In the figure, the two-dimensional points b′ and c′ can be moved arbitrarily close to each other while maintaining the same sa-tree structure. Hence, an object b′ can easily be closer to some other object c′ than to its parent b if c′ is not among the ancestor neighbors of b. Therefore, objects that are not among the ancestor neighbors of b can be arbitrarily close to b′ thereby precluding their consideration as candidates for improving the pruning power.

4. Incremental nearest neighbor search

In Section 3, we sketched how to perform range search with the sa-tree. Another common way of realizing similarity search for metric space data is with nearest neighbor search, namely finding the k objects in S that are closest to the query object q, where $k \geqslant 1$. In many applications, the number k of desired nearest neighbors is not known a priori, but is instead controlled by some stopping condition (Carey and Kossmann, 1997), which can be based on arbitrary criteria. In this case, the most useful algorithms are ones that compute the result progressively, such that results are reported as early as possible. This allows making use of pipelined query execution in complex queries involving nearest neighbor queries as subqueries (Carey and Kossmann, 1997; Fagin et al., 2001).

4.1. Algorithm

The nearest neighbor algorithm that we introduced in (Hjaltason and Samet, 1999) is incremental, in the sense of computing its results progressively as described above. Furthermore, the algorithm is general in the sense that it can be applied to a variety of domains and search structures, including the sa-tree. The basic idea behind the algorithm is to traverse the hierarchical structure in a "best-first" manner, which essentially means that at each step, the next node to visit is chosen from a global list of nodes based on their "distance" from the query object q. The distance of a node from q must be some value that lower-bounds the distances from q to all objects in the subtree rooted at the node. The traversal is facilitated by organizing the list of nodes with a priority queue, where the key is the distance from q of their entries. A second important idea in the algorithm is to also insert the objects onto the priority queue, with their actual distances as their keys. Thus, when an object reaches the front of the queue, we know that all other objects that have not already been removed from the queue must be farther from the query object.

Fig. 5 presents this incremental nearest neighbor algorithm in a form adapted to the sa-tree. The process is initialized in lines 1–3, after which the newly created priority queue, *queue*, contains as its only element the root of the sa-tree T. In each iteration of the **while**-loop, starting at line 4, the element e with the smallest key is removed from the priority queue (with DEQUEUE) and processed appropriately. In particular, if e represents an object a (i.e., if e is labeled "object"), a is reported as the next nearest neighbor; at this point, the algorithm can be terminated if desired. Otherwise, if e represents a node in the sa-tree, corresponding to an object a, where e is of the form $[a, d(q,a), d(q,c)]$, where c is closest to q among the ancestor neighbors of a. In this case, an "object" element is inserted into the queue for a (line 11),

```
INCNEAREST($q$, $T$)
 1 $queue \leftarrow$ NEWPRIORITYQUEUE()
 2 $a \leftarrow$ root of the sa-tree $T$
 3 ENQUEUE($queue$, $[a, d(q,a), d(q,a)]$ : "node", 0)
 4 while not ISEMPTY($queue$) do
 5   $e \leftarrow$ DEQUEUE($queue$)
 6   if $e$ is an object element then
 7     $[a] \leftarrow e$
 8     Report $a$ as the next nearest neighbor
 9   else /* $e$ is a node element */
10     $[a, d_a, minDist] \leftarrow e$
11     ENQUEUE($queue$, $[a]$ : "object", $d_a$)
12     $minDist \leftarrow \min\{minDist, \min_{b \in N(a)}\{d(q,b)\}\}$
13     for each $b \in N(a)$ do
14       $d_{lo} = \max\{d(q,b) - d_{\max}(b), (d(q,b) - minDist)/2\}$
15       ENQUEUE($queue$, $[b, d(q,b), minDist]$ : "node", $d_{lo}$)
16     enddo
17   endif
18 enddo
```

Fig. 5. Incremental nearest neighbor algorithm on an sa-tree T given a query object q.

and "node" elements are inserted for all elements b of the neighbor set of a (line 15). The second argument of ENQUEUE specifies the content of the element to be inserted into the queue, while the third argument specifies the key used for ordering the element. Note that although the presentation suggests that some distances must be computed many times by the algorithm, it is easy to reuse distance values such that $d(q,a)$ is computed only once for each object a that has a corresponding element on the priority queue; that is, $d(q,a)$ for the root a need be computed only once in line 3, and the distances $d(q,b)$ for the neighbors b of a computed in line 12 can be reused in lines 14 and 15.

As intimated above, the key for an element e should be a lower bound on the distance from q of all objects that are in the subset of S represented by e. Furthermore, to facilitate good performance of the algorithm, this lower bound should be as tight as possible (i.e., as close to the actual minimum distance from q to the objects in the subset). For an "object" element $e = [a]$, this subset consists of the single object a, so the key then equals $d(q,a)$. For a "node" element $e = [a, d(q,a), d(q,c)]$, the subset represented by e equals a and all objects that are descendants of the node corresponding to a. The lower bound for this subset is computed in the same manner as when performing range search, and makes use of both $d_{\max}(a)$ (the maximum distance between a and the objects in its subtree) and $d(q,c)$ (the minimum distance between q and "ancestor siblings" of a). Thus, to obtain the greatest possible lower bound, we use the maximum of both $d(q,a) - d_{\max}(a)$ and $(d(q,a) - d(q,c))/2$ (line 14). The lower-bound property of $d(q,a) - d_{\max}(a)$, stated in Eq. (2), follows from the triangle inequality in a straightforward manner, while the lower-bound property of $(d(q,a) - d(q,c))/2$ is guaranteed by Lemmas 1 and 2 (where Lemma 2 shows that $d(q,c)$ can be used in Lemma 1).

In the algorithm (Fig. 5), the value of $d(q,c)$ is propagated down the tree as the search progresses by storing it in priority queue elements (i.e., as the third component). It is initialized to $d(q,a)$ for the root a in line 3 (since the root a is the only ancestor neighbor of itself), while it is computed "incrementally" when processing a node element e; namely, for the neighbors b of the object a associated with e. In particular, the variable *minDist* is in line 10 set to $d(q,c)$ for c among the ancestor neighbors of a (as defined in Lemma 2). Then, in line 12 *minDist* is decreased if one or more of the neighbors of a is found to be closer to q, thus ensuring that *minDist* takes into account the ancestor neighbors of each $b \in N(a)$ even though we pointed out that they are all the same.

Observe that an "object" element for an object a is inserted into the queue when the associated "node" element e is processed (line 11) instead of when e is originally enqueued (i.e., line 15). The rationale for postponing the enqueuing of "object" elements in this manner is that this results in a smaller priority queue. In particular, if the corresponding "node" and "object" were enqueued at the same time (i.e., in line 15), the priority queue size can at worst be nearly twice as large as when postponing the enqueuing of "object" elements, as many objects may have both types of corresponding elements on the queue. Having a smaller priority queue is an advantage, in turn, due to the fact that the cost of priority queue operations is related to the queue size s_q; at best, the cost of ENQUEUE, DEQUEUE, or both is at least proportional to $\log s_q$.

Incidentally, there is a close relationship between the algorithm of Fig. 5 and the k-nearest neighbor algorithm described by Navarro, in Fig. 7 of Navarro (2002). [3] Both algorithms use a best-first traversal of the hierarchical sa-tree structure and a priority queue (Q in Navarro's algorithm and *queue* in ours) to manage the traversal. However, the fixed-size list A in Navarro's algorithm, representing the candidate k nearest neighbors, is absent in our algorithm. Essentially, the role of A is assumed by the priority queue *queue* in our algorithm, so our algorithm can be viewed as the result of merging the two priority queues Q and A in Navarro's algorithm.

Finally, as discussed in Section 4.2 below, unlike the algorithm described by Navarro, the algorithm in Fig. 5 does not take the unnecessary steps to ensure that the keys of the "node" elements are monotonically non-decreasing.

4.2. Monotonicity

As pointed out at the end of Section 4.1, the algorithm in Fig. 5 has the property that the keys of "node" elements are not necessarily monotonically non-decreasing as the tree is traversed downward, which may appear to be a flaw. In particular, for $b \in N(a)$, the lower-bound d_{lo} value for b, as computed in line 14, may be smaller than that for its parent a. Simply speaking, the fact that b is in the neighbor set of a does not represent a sufficient criteria to ensure that the lower-bound properties used in line 14 (that is, the ones stated by Eqs. (2) and (1), respectively) provide a value for b that is greater than or equal value to that for a. [4] An example that demonstrates this for the lower bound of Eq. (2), which makes use of the bound on ancestor distances (i.e., $d_{\max}(a) \geqslant d(a,b)$ for all descendants b of a) is shown in Fig. 6. Given a Euclidean space as in the Figure, the lower bound for an object p (e.g. points a and b in Fig. 6) equals the distance to the "ball" (i.e., solid circle in two dimensions) around p of radius equal to $d_{\max}(p)$. Thus, this lower bound is clearly smaller for b than for a in the figure. In fact, if the query point q were moved inside the dark shaded area, the value of $d(\mathsf{q},\mathsf{b}) - d_{\max}(\mathsf{b})$ would be negative.

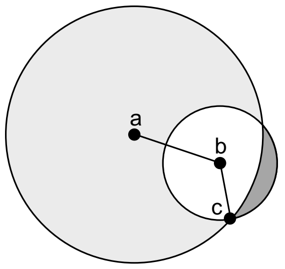

Fig. 6. An sa-tree for three points a, b, and c, rooted at a, assuming the Euclidean distance metric. The circles around a and b denote their $d_{\max}$ "balls", beyond which we know there to be no descendants.

It is easy to modify the incremental nearest neighbor algorithm above such that monotonicity is guaranteed on the keys of "node" elements, as defined above. In particular, in line 14 of Fig. 5, we would simply include d_a inside the $\max\{\cdots\}$ computation. Nevertheless, the algorithm is actually correct even without this modification, in the sense of traversing the exact same portion of the sa-tree T (i.e., dequeuing the same set of "node" elements, albeit possibly in different order) as the modified algorithm for producing the same number of neighbors, [5] thus guaranteeing that the neighbors of q are produced in proper non-decreasing order of distance. To see why, let $o_k \in S$ be the kth neighbor of q and let $D_k := d(q, o_k)$. Furthermore, for an sa-tree node n, let $d_l'o(n)$ denote the value of d_lo computed in line 14 for n in the original algorithm, and $d_l'o(n)$ the corresponding value in the monotonic version of the algorithm. Clearly, we have $d_lo(n) \leqslant d_l'o(n)$. Thus, if n is visited by the monotonic version, then $D_k \geqslant d_l'o(n) \geqslant d_lo(n)$, so n is also visited by the

[3] An unfortunate error crept into Navarro's algorithm listing, in that $m/2$ in line 14 should have been $(d(q,v) - m)/2$.

[4] The same situation would arise for other metric indexing methods that also use these lower-bound properties, such as the vp-tree (Uhlmann, 1991; Yianilos, 1993) and gh-tree (Uhlmann, 1991).

[5] This statement is somewhat imprecise, as it disregards the possibility of ties in the key values of priority queue elements. However, by giving "node" elements priority over "object" elements, the claims made in this paragraph can all be shown to be true even in the presence of ties.

original algorithm. Suppose, conversely, that n is not visited by the monotonic version, implying that $D_k < d'_lo(n)$. If $d_lo(n) = d'_lo(n)$, then the original algorithm also does not visit n. If $d_lo(n) < d'_lo(n)$, on the other hand, then it may appear that the original algorithm could visit n. However, in this case, the value for $d'_lo(n)$ is based on the lower bound for an ancestor n' of n, such that $d'_lo(n) = d_lo(n')$. Therefore, n is also not visited by the original algorithm since its ancestor n' is not visited (due to $D_k < d_lo(n')$), which is implied by $D_k < d'_lo(n)$ and $d'_lo(n) = d_lo(n')$, and we have shown that the two versions visit exactly the same set of nodes.

5. Concluding remarks

In this paper, we have presented an improvement on the original search strategy for the sa-tree, a recently proposed distance-based indexing structure. In the process, we detailed the intuition behind the sa-tree and how it was inspired by the Voronoi diagram. We also presented an adaptation of our incremental nearest neighbor algorithm for the sa-tree. Finally, we showed that it is not necessary to maintain a particular monotonicity property (see Section 4.2) on subtrees as the tree is traversed, which other authors have emphasized; abandoning monotonicity has the advantage that it leads to slightly simplified search algorithms.

The sa-tree represents a highly novel development in indexing support for search in metric spaces. We intend to continue to seek improvements on this idea and to investigate other new directions of research. We are currently developing a prototype system with the goal of permitting realistic experimental comparison between various distance-based indexing methods on meaningful data sets. With this system in place, we will be able to empirically evaluate the effectiveness of our proposed improvement to the sa-tree. One interesting extension to the sa-tree is a dynamic version (Navarro and Reyes, 2002) that supports both insertions and deletions. Whether our improved search strategy for the sa-tree can be applied to this dynamic variant is still an open question, and we plan to investigate the issue.

Among other future work on similarity search that we wish to pursue is the design of incremental nearest neighbor (INN) algorithms for distance-matrix (or *pivot-based*; Chávez et al., 2001) methods such as AESA (Vidal, 1994) and LAESA (Micó et al., 1994). Based on preliminary work, we believe that it is feasible to devise efficient INN algorithms for them, even though such methods are not based on hierarchical partitioning, which is the basis of our earlier INN algorithm (Hjaltason and Samet, 1999).

Acknowledgements

This work was supported in part by the National Science Foundation under grants EIA-99-00268, EIA-99-01636, IIS-00-86162, and EIA-00-91474.

References

Brin, S., 1995. Near neighbor search in large metric spaces. In: Dayal, U., Gray, P.M.D., Nishio, S., (Eds.), Proceedings of the 21st International Conference on Very Large Data Bases (VLDB), Zurich, Switzerland, September 1995, pp. 574–584.

Carey, M.J., Kossmann, D., 1997. On saying "enough already!" in SQL. In: Peckham, J., (Ed.), Proceedings of the ACM SIGMOD Conference, Tucson, AZ, May 1997, pp. 219–230.

Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J., 2001. Searching in metric spaces. ACM Computing Surveys. 33(3), 273–322, September 2001.

Ciaccia, P., Patella, M., Zezula, P., 1997. M-tree: An efficient access method for similarity search in metric spaces. In: Jarke, M., Carey, M.J., Dittrich, K.R., Lochovsky, F.H., Loucopoulos, P., Jeusfeld, M.A., (Eds.), Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB), Athens, Greece, August 1997, pp. 426–435.

Fagin, R., Lotem, A., Naor, M., 2001. Optimal aggregation algorithms for middleware. In: Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), Santa Barbara, CA, May 2001, pp. 102–113.

Han, J., Kamber, M., 2000. Data Mining: Concepts and Techniques. Morgan Kaufmann, San Francisco.

Hjaltason, G.R., Samet, H., 1999. Distance browsing in spatial databases. ACM Transactions on Database Systems 24(2), 265–318, June 1999. Also Computer Science TR-3919, University of Maryland, College Park, MD.

Hjaltason, G.R., Samet, H., 2003. Index-driven similarity search in metric spaces. ACM Transactions on Database System, to appear. Also Computer Science TR-3919, University of Maryland, College Park, MD.

Micó, L., Oncina, J., Vidal, E., 1994. A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing-time and memory requirements. Pattern Recognition Letters 15 (1), 9–17.

Navarro, G., 1999. Searching in metric spaces by spatial approximation. In: Proceedings String Processing and Information Retrieval and International Workshop on Groupware (SPIRE/CRIWG 1999), Cancun, Mexico, September 1999, pp. 141–148.

Navarro, G., 2002. Searching in metric spaces by spatial approximation. VLDB Journal 11 (1), 28–46.

Navarro, G., Reyes, N., 2002. Fully dynamic spatial approximation trees. In: Laender, A.H.F., Oliveira, A.L., (Eds.), String Processing and Information Retrieval–Ninth International Symposium (SPIRE 2002), Lisbon, Portugal, September 2002, pp. 254–270; Also Springer-Verlag Lecture Notes in Computer Science 2476.

Uhlmann, J.K., 1991. Metric trees. Applied Mathematics Letters 4 (5), 61–62.

Uhlmann, J.K., 1991. Satisfying general proximity/similarity queries with metric trees. Information Processing Letters 40 (4), 175–179.

Vidal, E., 1994. New formulation and improvements of the nearest-neighbour approximating and eliminating search algorithm (AESA). Pattern Recognition Letters 15 (1), 1–7.

Voronoi, G., 1909. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Deuxiême mémoire: Recherches sur les parallèlloèdres primitifs. seconde partie. Journal für die Reine und Angewandte Mathematik 136 (2), 67–181.

Yianilos, P.Y., 1993. Data structures and algorithms for nearest neighbor search in general metric spaces. In: Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, Austin, TX, January 1993, pp. 311–321.