In *GIS'11: Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, Agrawal, I. Cruz, C. S. Jensen, E. Ofek, and E. Tanin, eds., pages 132–141, Chicago, November 2011.

# Similarity Search on a Large Collection of Point Sets

Marco D. Adelfio Sarana Nutanong Hanan Samet
Center for Automation Research, Institute for Advanced Computer Studies,
Department of Computer Science, University of Maryland
College Park, MD 20742, USA
{marco, nutanong, hjs}@cs.umd.edu

## ABSTRACT

Spatial applications often require the ability to perform similarity search over a collection of point sets. For example, given a geographical distribution of a disease outbreak, find $k$ historical outbreaks with similar spatial distributions from a data collection $\mathcal{D}$. In this paper, we study the problem of similarity search over a collection of point sets using the Hausdorff distance, which is a measure commonly used to determine the maximum discrepancy between two point sets. To avoid computing the Hausdorff distance for all point sets $S$ in $\mathcal{D}$, one may compute an *optimistic estimate* (i.e., lower bound value) of the actual Hausdorff distance $\textsc{HausDist}(Q, S)$ for each $S$ to rule out sets that are obviously dissimilar to $Q$. In our investigation, we observed that a commonly used method (called BscLB) to compute an estimate may not produce a result which is indicative of the actual Hausdorff distance. Consequently, we propose a method (called EnhLB) which produces a tighter estimate than the existing one. We then formulate a similarity search algorithm which uses a combination of BscLB and EnhLB to find similar point sets efficiently. In addition, we also extend our method to support an outlier-resistant variant of the Hausdorff distance called the *modified Hausdorff distance*. We compare our proposed algorithm with an algorithm using only BscLB. The results of our experiments show a reduction in computation time of 72% for searches using the Hausdorff distance and a reduction of 53% using the modified Hausdorff distance.

## Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

## General Terms

Algorithms, Design

## Keywords

Similarity search, Spatial databases, Query processing.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
*ACM SIGSPATIAL GIS* '11, November 1-4, 2011. Chicago, IL, USA
Copyright © 2011 ACM ISBN 978-1-4503-1031-4/11/11 ...$10.00.

## 1. INTRODUCTION

Existing spatial search systems (e.g., Google Maps, Yelp, Zillow) allow users to find points of interest by using a proximity query like "find the 3 nearest gas stations with respect to one's location $\boldsymbol{q}$". This problem can be formalized as the nearest neighbor (NN) query problem. That is, given a set $\mathcal{D}$ of gas stations and a query location $\boldsymbol{q}$, the nearest neighbor (NN) query [11, 17] identifies a point $\boldsymbol{p}$ in $\mathcal{D}$ which minimizes the distance $\textsc{Dist}(\boldsymbol{q}, \boldsymbol{p})$.

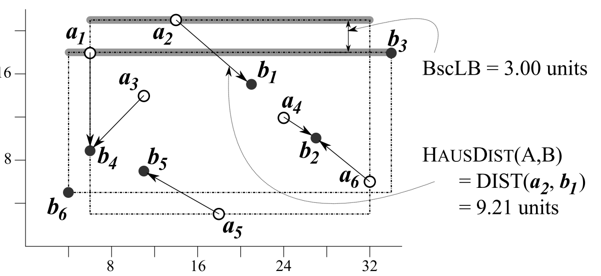
The NN query can also be generalized to the *aggregate NN query* [16] and the *distance join query* [6, 10, 19, 22]. These queries involve identifying a closest point $\boldsymbol{p}$ in $\mathcal{D}$ that minimizes the distance to a query object $Q$ which is a set of locations rather than a single location. For example, given a set $Q$ of stations of a train line and a set $\mathcal{D}$ of bus stops of a bus route, find a stop $\boldsymbol{p}$ in $\mathcal{D}$ which minimizes the distance to any train station in $Q$ for transfer purposes.

In this paper, the concept of NN search is further extended to a case where (i) the dataset $\mathcal{D}$ comprises sets of locations, (ii) the query object $Q$ is also a set of locations, and (iii) we want to find the $k$ most similar sets in $\mathcal{D}$ with respect to $Q$ using measures described later in this section. Example applications that may benefit from our work include:

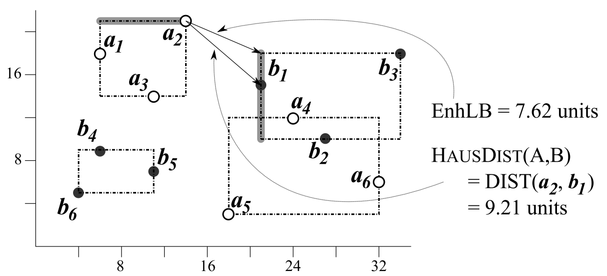
- Given a geographical distribution of a current disease outbreak represented as a location set $Q$, an epidemiologist may wish to find $k$ occurrences of outbreaks (from a set $\mathcal{D}$ of historical outbreak distributions) that are most similar to $Q$. These results can then be used to help identify correlations between the outbreak in question and other outbreaks.
- Let $Q$ denote a location set of warehouses of one logistics company and $\mathcal{D}$ denote a collection of location sets of petrol stations where each location set contains locations of petrol stations within a single petrol company. To form a partnership with a petrol company, the logistics company may wish to find the petrol company whose location set $S$ minimizes the average distance from each warehouse in $Q$ to the nearest station in $S$.

We use the Hausdorff distance $\textsc{HausDist}$ as a dissimilarity measure of a set $A$ with respect to another set $B$. The Hausdorff distance $\textsc{HausDist}(A, B)$ can be regarded as the worst-case discrepancy of $A$ with respect to $B$. Specifically, the distance $\textsc{HausDist}(A, B)$ is defined as the maximum value of the distance from each point $\boldsymbol{a}$ in $A$ to its nearest point in $B$, i.e.,

$$\textsc{HausDist}(A, B) = \\ \textsc{max}\{\textsc{min}\{\textsc{Dist}(\boldsymbol{a}, \boldsymbol{b}) : \boldsymbol{b} \in B\} : \boldsymbol{a} \in A\}.$$



(a) BscLB is 32.6% of the corresponding HausDist



(b) EnhLB is 82.7% of the corresponding HausDist

**Figure 1: Illustrations of BscLB, EnhLB and HausDist from $A$ to $B$ ($\{a_1, ..., a_6\}$ to $\{b_1, ..., b_6\}$)**

Intuitively, using this measure, a set $A$ is considered similar to $B$ iff each point in $A$ is close to at least one point in $B$. Since HausDist is asymmetric, it is not technically a distance function. However, throughout this paper we refer to it as a distance function in the interest of brevity. To form a symmetric relation, SymHausDist$(A, B)$ is defined as $\max\{$HausDist$(A, B)$, HausDist$(B, A)\}$. That is, two sets $A$ and $B$ are considered similar to each other if $A$ is similar to $B$ and $B$ is similar to $A$.

A naive method to identify a point set in a collection $\mathcal{D}$ of point sets which minimizes the HausDist from $Q$ is to compute HausDist$(Q, S)$ for each point set $S$ in $\mathcal{D}$ and identify a point set that yields the smallest HausDist. One may reduce the number of HausDist computations by computing a lower bound value (an optimistic estimate) of the actual distance HausDist$(Q, S)$ for each $S$ to rule out sets that are obviously dissimilar to $Q$. Ideally, we want this lower bound to provide an estimate that is close to the actual distance HausDist$(Q, S)$, while keeping the computation cost low with respect to that of HausDist$(Q, S)$. Hence, the challenge of formulating a Hausdorff lower bound lies in the tradeoff between the quality of the estimate and its computation cost.

Traditionally, a lower bound on HausDist$(A, B)$ can be computed as the MaxMin distance [18] from the minimum bounding rectangle (MBR) of $A$ to the MBR of $B$. Our research is based on the observation that this *basic lower bound* (BscLB) may be inaccurate when the MBR that covers $A$ significantly overlaps the MBR that covers $B$. In such a case, the inaccuracy of BscLB may result in a large number of HausDist computations which can be highly undesirable when the number of point sets is large. Based on this observation, we propose a novel method which decomposes the two MBRs into sub-MBRs and computes a lower bound using sub-MBRs of the two sets. We call this new method the *enhanced lower bound* (EnhLB). Our experimental results show that EnhLB provides an estimate that is significantly closer to the actual HausDist than BscLB, which results in a greater pruning capability. Although EnhLB incurs a greater computation cost than BscLB, a significant overall performance improvement is obtained.

Figure 1 provides a comparison between BscLB and EnhLB (computed using the methods described in Section 4). In Figure 1(a), the value of BscLB from $A$ to $B$ is calculated as the MaxMin distance from the MBR that encloses $A$ to the MBR that encloses $B$. An edge pair that yields the BscLB value of 3 is highlighted in grey. In Figure 1(b), the value of EnhLB from $A$ to $B$ is calculated using the sub-MBRs of those in Figure 1(a). An edge pair that yields the EnhLB value of 7.62 is highlighted in grey. In this example, EnhLB is 2.54 times closer to the actual HausDist than BscLB.

It can be seen that the accuracy of EnhLB$(A, B)$ depends on how the MBRs of $A$ and $B$ are decomposed into sub-MBRs as well as the number $n$ of sub-MBRs. Hence, we formulate an algorithm to find appropriate MBRs for each point set $S$. Specifically, we propose an algorithm which utilizes an R-Tree index $R$ to hierarchically organize the data points in $S$. The algorithm traverses $R$ starting from the root and decomposes larger MBRs into sub-MBRs (MBRs of their children) until a desired number $n$ of sub-MBRs is reached. We also present an empirical study to choose an appropriate value of $n$ in Section 6.

In addition to HausDist, which is a measure of maximum discrepancy, we extend the concept of EnhLB to support a measure of average discrepancy called the *modified Hausdorff distance* (MHD). We then formulate an incremental search algorithm which can be applied to both HausDist and MHD. We also use this search algorithm to demonstrate the effectiveness of EnhLB in comparison to BscLB.

The contributions of our work are summarized as follows.

- An improved method (EnhLB) of calculating a HausDist lower bound that provides a greater pruning capability than the basic method (BscLB).
- An incremental search algorithm that utilizes BscLB and EnhLB and can be applied to both HausDist and MHD.
- Performance evaluations of our proposed search algorithm in terms of the (i) total response time, (ii) I/O cost, and (iii) processing cost.

The rest of this paper is organized as follows. Section 2 provides background knowledge of HausDist algorithms and branch-and-bound search algorithms. Section 3 contains a definition of our research problem. Our proposed lower bound computation method and the proposed search algorithm are given in Section 4. In Section 5, we show how our proposed method can be extended to support the *modified Hausdorff distance*, an outlier-resistant variant of HausDist. In Section 6, we report our experimental results, while Section 7 provides conclusions and directions of future research.

## 2. RELATED WORK

### 2.1 Hausdorff Distance Computation

The Hausdorff distance is frequently used in spatial and geometric matching problems in a variety of contexts, such as shape- or image-matching, geometric modeling, model rendering, and image recognition [1, 7, 15, 20]. In these contexts, it is used to measure how well two shapes, images, or polygonal meshes resemble each other (i.e., A matches B within a maximum discrepancy of $\delta$).

The problem of computing Hausdorff distances between pairs of point sets, polygons, or meshes is a well-studied problem. For two point sets with total cardinality $O(n)$, the naive approach computes all pairwise distances to find the MaxMin distance of the two sets, which has a running time of $O(n^2)$. Some approaches to achieve more efficient performance involve calculating the Voronoi diagram of one point set, and then performing plane sweep [2]. These approaches become computationally intractable in high dimensions [20]. Some methods attempt to improve efficiency by introducing randomization [2] or providing approximate solutions [20].

Computing the Hausdorff distance is naturally related to executing a nearest neighbor query, as the Hausdorff distance from point set $A$ to point set $B$ is determined by the maximum distance of a point in $A$ to its nearest neighbor in $B$. Thus it is not surprising that an approach similar to the one used for solving the k-nearest neighbors (kNN) problem can also be useful in calculating the Hausdorff distance between two point sets. Nutanong et al. proposed an algorithm for computing the Hausdorff distance in the context of trajectory matching, which is based on the branch-and-bound approach used for solving the k-nearest neighbors (kNN) problem [15]. The branch-and-bound search method involves incrementally proceeding through a search tree, and re-ranking and pruning the candidate solutions as the process continues. Tang et al. discuss the difficulties of computing the exact Hausdorff distance between polygons efficiently in $\mathbb{R}^3$, and present an approximation algorithm that uses a similar branch-and-bound technique that stops when the bounds are within the specified approximation factor [20].

In this paper our focus is on using the Hausdorff distance as a similarity measure between point sets. In particular our goal is not one of finding the most efficient method of computing the Hausdorff distance, but instead one of reducing the number of times that the Hausdorff distance is computed. This is done by improving Hausdorff distance estimates used in a branch-and-bound search to provide a greater pruning capability. We elaborate further the concept of distance estimators and branch-and-bound search in the next subsection.

### 2.2 Branch-and-Bound Search

The branch-and-bound principle is widely adopted for similarity search problems [9]. Classic examples of branch-and-bound search in spatial databases are the depth-first [17] and best-first [11] algorithms to search for nearest neighbors (NNs) over a point set indexed in a hierarchical index, such as the R-Tree [4, 8]. These algorithms use an optimistic estimator to provide the order in which index nodes are visited and to disregard index nodes containing points that clearly cannot be resultant NNs.

For example, the best-first NN algorithm uses a priority queue to sort index nodes $N$ according to the minimum distance MinDist from the query point $\boldsymbol{q}$, which serves as an optimistic estimate of the distance from $\boldsymbol{q}$ to any object in $N$. In this way, index nodes with large MinDists are scheduled to be visited later than those with smaller MinDists. When a data point is retrieved from the priority queue, the MinDist estimator guarantees that none of the nodes currently in the priority queue can produce an object closer to $\boldsymbol{q}$. As a result, the search can be used to incrementally find NNs and terminate when a desired number $k$ of data points are retrieved from the priority queue.

The best-first search principle can also be used to process aggregate NN queries [16], which are multiple query point generalizations of the NN query. Specifically, given a dataset $\mathcal{D}$, the aggregate NN of a query point set $Q$ is the data object $\boldsymbol{p}$ in $\mathcal{D}$, which minimizes the distance to $Q$ according to an aggregate function: MIN, MAX, or SUM. One can calculate an optimistic estimate as the smallest possible aggregate distance of any data point in the node $N$ to $Q$. For example, an optimistic estimate of MAX-aggregate from $Q$ to objects in a node $N$ is given as $\textsc{max}\{\textsc{MinDist}(\boldsymbol{q}, N) : \boldsymbol{q} \in Q\}$. A best-first search can then be conducted by visiting nodes $N$ in ascending order of the optimistic estimates. In the next subsection, we show how the same concept can be applied to similarity search over a collection of point sets.

## 3. PROBLEM DEFINITION

We model the problem of similarity search over a collection of point sets as a HausDist minimization problem. Formally, we define our similarity search function as follows.

Definition 1 (Similar Point Set Query). *The query accepts a point set $Q$, a collection $\mathcal{D}$ of point sets and the number $k$ of resultant point sets. As output, the function returns a list $\mathcal{A}$ of point sets such that*

*(i) each element of $\mathcal{A}$ is a member of $\mathcal{D}$;*
*(ii) $|\mathcal{A}|$ is equal to $\textsc{min}\{k, |\mathcal{D}|\}$;*
*(iii) for each $S$ in $\mathcal{A}$ and each $T$ in $\mathcal{D} \setminus \mathcal{A}$,*

$$\textsc{HausDist}(Q, S) \leq \textsc{HausDist}(Q, T);$$

*(iv) for each $S_i$ and $S_j$ in $\mathcal{A}$ where $i$ is less than $j$,*

$$\textsc{HausDist}(Q, S_i) \leq \textsc{HausDist}(Q, S_j).$$

This query can be processed by separating the resultant list $\mathcal{A}$ of point sets from the rest $(\mathcal{D} \setminus \mathcal{A})$.

To avoid computing HausDist for every point set in $\mathcal{D}$, we can compute an optimistic estimate for each point set $S$ in $\mathcal{D}$. Specifically, an optimistic estimator of $\textsc{HausDist}(Q, S)$ is a function which returns a distance guaranteed to be less than or equal to $\textsc{HausDist}(Q, S)$. We use this optimistic estimate to provide the search order and to rule out entries that clearly cannot be in the result $\mathcal{A}$. Ideally, we want this estimator to produce a value as close to $\textsc{HausDist}(Q, S)$ as possible. At the same time, we also want to keep the computation cost low with respect to that of $\textsc{HausDist}(Q, S)$.

To further avoid computing an optimistic estimate for every point set, we can index the point sets in $\mathcal{D}$ as rectangular objects (using their respective MBRs) in a hierarchical structure like the R-Tree. In this case, an optimistic estimate of the Hausdorff distance from $Q$ to an R-Tree node $N$ is a value guaranteed to be smaller than the Hausdorff distance from $Q$ to any point set in $N$.

The objectives of our investigation are given as follows: (i) to improve the accuracy of the existing Hausdorff estimator without introducing an excessive computation cost; and (ii) to formulate a search algorithm which utilizes this estimator. This estimator improvement and the search algorithm are described in the next section.

## 4. PROPOSED METHOD

In this section, we propose a method which improves the accuracy in computing an optimistic estimate (a lower bound) of HausDist$(A, B)$ using the MBRs of $A$ and $B$. We observe that real-world geographic point sets have a tendency to cluster around key locations like big cities or industrialized coastal areas. Our solution is formulated based on a hypothesis that the accuracy in estimating HausDist$(A, B)$ can be improved by using the sub-MBRs, MBRs of such clusters.

We use the R-Tree index to store all point sets in the collection $\mathcal{D}$ where each point set $S$ is represented as a rectangular object using its MBR. We use two types of R-Trees. The first type, *primary R-Tree*, is used to store a collection of point sets where each point set is represented by its MBR. The second type, *secondary R-Tree*, is used to store points in each point set. Note that the root node of a secondary R-Tree is equivalent to its representative MBR in the primary R-Tree.

In the rest of this section, we present our search algorithm which uses the basic lower bound BscLB to provide the search order for nodes in the primary R-Tree and uses the enhanced lower bound EnhLB to refine the search order for each point set. Subsequently, we describe how BscLB and EnhLB are computed.

### 4.1 Incremental Search Algorithm

In this section, we describe our proposed search algorithm (Algorithm 1) which uses the two optimistic estimators BscLB and EnhLB to help search for $k$ similar point sets with respect to a query point set $Q$. Specifically, we use BscLB for preliminary search ordering and EnhLB to refine the search order. Our rationale behind this practice is that BscLB, which is cheaper to compute than EnhLB, can provide a reasonable estimate of HausDist$(Q, S)$ when $Q$ and $S$ are far from each other. Hence, BscLB can be used as a preliminary pruning criterion to rule out point sets with large BscLB values.

We conducted a simple experiment to support our argument. Given a query point set $Q$ and a collection $\mathcal{D}$ of point sets, we rank each point set $S$ in $\mathcal{D}$ according to HausDist$(Q, S)$. We then show the accuracy of the two estimators as the rank $k$ of $S$ increases. The description of the dataset is given in Section 6 and the results of our experiment are reported in Appendix A.

We now consider the algorithm description (Algorithm 1). The algorithm finds $k$ point sets in $\mathcal{D}$ which minimize the HausDists from a query point set $Q$. An environment variable $n$ specifies the resolution in which EnhLB is computed and is shared throughout the algorithm descriptions in this section.

The initialization steps are given by Lines 1 to 8. We create two levels of R-Trees (as described in Section 3) to index all point sets. The primary R-Tree is used to index all point sets and data points in each point set are in turn indexed in a secondary R-Tree. The remaining steps of initialization include (i) creating an R-Tree *QueryRT* for the query point set; (ii) initializing a priority queue *PQ*; and (iii) creating an empty list $\mathcal{A}$ to store resultant point sets.

The control loop is given by Lines 9 to 25. At the beginning of each iteration (Line 10), we retrieve the head entry ($N$, $d$, *LB-Stage*) from *PQ*, where $N$ is the node which has the smallest estimated HausDist $d$. The value of *LB-Stage* identifies the nature in which the current value of $d$ has been calculated: "0" denotes BscLB, "1" denotes EnhLB, and "2" denotes an actual HausDist. The rest of the control loop is organized into the two following cases:

- Node $N$ contains only one secondary R-Tree *SecRT*. In this case, we check the value of *LB-Stage*. If *LB-Stage* is "0", then $d$ is currently a BscLB value. Hence, we reset $d$ to a EnhLB value and insert the entry back into *PQ* with an *LB-Stage* of "1". If *LB-Stage* is "1", then $d$ is currently a EnhLB value. Hence, we compute the actual HausDist and assign it to $d$. Then, we insert the entry back into *PQ* with an *LB-Stage* of "2". Otherwise *LB-Stage* is "2", which means that $d$ is final and *SecRT* can be included as a query result in $\mathcal{A}$.
- Node $N$ contains multiple children $C$. In this case, for each child node $C$, we compute an estimate $d$ using BscLB and then we insert a priority queue entry ($C$, $d$, *LB-Stage*) into *PQ* where *LB-Stage* is set to "0".

The control loop terminates when $\mathcal{A}$ contains $k$ objects or when *PQ* is exhausted.

**Algorithm 1**: SimSearch($Q$, $\mathcal{D}$, $k$)

```
input       : Query point set $Q$, Collection $\mathcal{D}$ of point
              sets, and Number $k$ of results
output      : $k$ point sets with the smallest HausDists
              with respect to $Q$
environment : Number $n$ of MBRs used to compute
              EnhLB
1  PrimRT ← Create an empty R-Tree;
2  for each Point Set $S$ in $\mathcal{D}$ do
3      SecRT ← Create an R-Tree of $S$;
4      Insert SecRT into PrimRT;
5  QueryRT ← Create an R-Tree of $Q$;
6  Priority Queue PQ ← Create an "ascending order" PQ;
7  Insert (RootOf(PrimRT), 0, 0) into PQ;
8  List $\mathcal{A}$ ← Create an empty list;
9  while PQ is not empty do
10     PQ-Entry ($N$, $d$, LB-Stage) ← Dequeue(PQ);
11     if $N$ contains one secondary R-Tree SecRT then
12         if LB-Stage is 0 then
13             $d$ ← EnhLB(QueryRT, SecRT);
14             Insert ($N$, $d$, LB-Stage=1) into PQ;
15         else if LB-Stage is 1 then
16             $d$ ← HausDist(QueryRT, SecRT);
17             Insert ($N$, $d$, LB-Stage=2) into PQ;
18         else
19             Insert the point set from SecRT into $\mathcal{A}$;
20             if $\mathcal{A}$ contains $k$ point sets then
21                 return $\mathcal{A}$;
22     else
23         for each Child $C$ of $N$ do
24             Distance $d$ ← BscLB(RootOf(QueryRT), $C$);
25             Insert ($C$, $d$, 0) into PQ;
26 return $\mathcal{A}$;
```

### 4.2 Lower Bound Computation

In this subsection, we describe how BscLB and EnhLB used by Algorithm 1 are computed. Traditionally, a lower bound of HausDist$(A, B)$ can be computed as the MaxMin distance [18] from the MBR which encloses $A$ to the MBR which encloses $B$. This is because HausDist$(A, B)$ can be considered as the MaxMin distance from $A$ to $B$. We formally define this lower bound function as follows.

Definition 2 (Basic Hausdorff Lower bound). *Let $M_A$ and $M_B$ denote the MBRs of $A$ and $B$, respectively.*

$$\text{BscLB}(M_A, M_B) = \max\{\text{MinDist}(f_a, M_B) : f_a \in \text{FacesOf}(M_A)\}.$$

That is, we exploit the minimum enclosing property of MBRs and assume that each MBR face touches at least one object. We then compute a lower bound value of each face using MinDist. The maximum of these lower bound values becomes the resultant estimate and is guaranteed to be smaller than or equal to the actual HausDist.

We now present our proposed method to compute an optimistic estimate of HausDist$(A, B)$ using MBRs of subsets of $A$ and $B$. First, we describe our algorithm Algorithm 2 to find $n$ MBRs which cover the point set $S$ indexed in an R-Tree $R$. Our algorithm utilizes the R-Tree index which organizes objects in a hierarchy of MBRs and accepts an R-Tree $R$ of a point set and the number of MBRs to be selected from $R$. The objective here is to find a list of MBRs which cover the point set $S$. We formulate an algorithm which traverses the R-Tree $R$ according to the areas of MBRs in the hierarchy. Specifically, we decompose the largest MBRs because smaller MBRs are likely to provide tighter estimates than large ones.

In the initialization steps, we first create an empty list $L$ to store the resultant MBRs (Line 1). Second, we initialize a priority queue $PQ$ to arrange MBR entries according to their areas in descending order (Line 2) and insert the root of $R$ as the first entry (Line 3).

We now consider the control loop (Line 4 to 11). At the beginning of each iteration (Line 5), the R-Tree node $N$ with the largest area $a$ is retrieved from the priority queue. If $N$ contains points, then $N$ is inserted into the resultant list $L$ since there are no R-Tree nodes beneath $N$. Otherwise, child entries of $N$ are inserted into $PQ$. The control loop terminates when $PQ$ is exhausted or there are at least $n$ entries in $PQ$ and $L$. Finally, all entries in $PQ$ are inserted into $L$, and $L$ is returned as output.

**Algorithm 2**: GetCovMBRs$(R)$

```
input       : R-Tree $R$ of data points
output      : List $L$ of MBRs
environment : Requested number $n$ of MBRs in $L$
1  List $L \leftarrow$ Create an empty list of nodes (MBRs);
2  Priority Queue $PQ \leftarrow$ Create a "descending order" PQ;
3  Insert (RootOf($R$), 0) into $PQ$;
4  while SizeOf($PQ$) + SizeOf($L$) $< n$ and PQ is not empty do
5      PQ-Entry (Node $N$, Area $a$) $\leftarrow$ Dequeue($PQ$);
6      if $N$ contains points then
7          Insert $N$ into $L$;
8      else
9          for each Child $C$ of $N$ do
10             Area $a \leftarrow$ AreaOf($C$);
11             Insert($C$, $a$) into $PQ$;
12 for each (Node $N$, Area $a$) in $PQ$ do
13     Insert $N$ into $L$;
14 return $L$;
```

Figure 2 provides an example run of Algorithm 2. The figure contains a three-level R-Tree $R$, where the top level (Level 3) corresponds to the root node and the bottom level (Level 1) comprises nodes whose immediate children are data points. Assume that the $n$ value is 4. At the initialization, the resultant list $L$ is initialized to an empty list and the root node is inserted into the priority queue $PQ$ which arranges MBRs in descending order according to their areas. In the first iteration of the control loop, the root node is retrieved from the head of $PQ$. Then, we expand the root node by inserting its immediate children $M_1$, $M_2$, and $M_3$ into $PQ$. Since the size of $PQ$ is 3 and $L$ is still empty, we need to further explore $R$ to meet the minimum requirement of $n$ MBRs. In the second iteration, $M_3$, which is the largest MBR, is retrieved from $PQ$. Then, we expand $M_3$ by inserting the children $M_{10}$, $M_{11}$ and $M_{12}$ into $PQ$. At this point there are 5 MBRs in $PQ$ which is greater than the $n$ value of 4. Hence, all these MBRs are inserted into $\mathcal{A}$ and returned as query results.

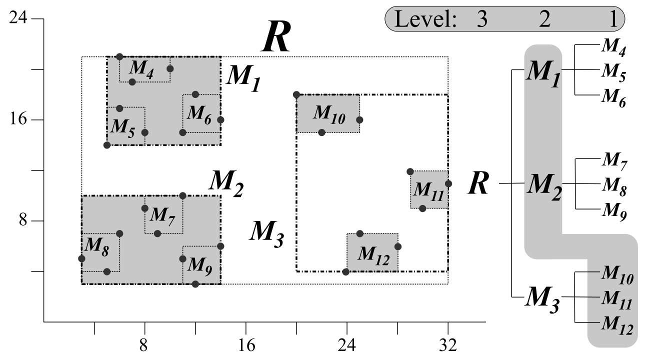


**Figure 2: An example of GetCovMBRs($R$) with the requested number $n$ of MBRs of 4, where selected MBRs are highlighted in gray.**

Note that the actual length $l$ of the resultant list $L$ may not exactly match the requested number $n$ of MBRs. Specifically, the length $l$ of $L$ depends on how the conditions in Line 4 are broken, i.e., whether $n$ MBRs are obtained or $PQ$ is exhausted first. If $n$ MBRs are obtained first, $l$ must be greater than or equal to $n$ but smaller than $(n + b)$ where $b$ is the branching factor of $R$. This is because, at each iteration we can add at most $b$ MBRs into $PQ$. In the case where $PQ$ is exhausted first, i.e., we do not have enough MBRs to satisfy the request, $l$ is less than $n$.

We now present our algorithm (Algorithm 3) to compute an optimistic estimate of HausDist$(A, B)$ using MBRs of subsets of $A$ and $B$. The algorithm accepts R-Trees $R_A$ and $R_B$ of two point sets and the number $n$ of MBRs from each R-Tree that will be used to calculate a lower bound. Specifically, we use GetCovMBRs (Algorithm 2) to select $n$ MBRs from $R_A$ and another $n$ MBRs from $R_B$, and store them in lists $L_A$ and $L_B$ respectively (Lines 1 and 2). The resultant distance is computed as the MaxMin distance from faces of MBRs in $L_A$ to the MBRs in $L_B$ (Lines 4 to 12). Specifically, for each face $F_A$ of MBRs in $L_A$ we compute the minimum distance from $F_A$ to all MBRs $B$ in $L_B$. The resultant distance is calculated as the maximum of these minimum distances computed in the while loop (Lines 8 to 10). Note that the while loop may not need to iterate through the entire $L_B$ if it is found that the minimum distance of the current $F_A$ to $L_B$ cannot affect $d_{max}$. That is, the current $d_{min}$ is less than or equal to the current $d_{max}$.

**Algorithm 3**: EnhLB($R_A$, $R_B$)

```
input       : R-Trees $R_A$ and $R_B$ of two point sets, and
              the number $n$ of MBRs used to compute
              the result
output      : Optimistic Estimate of the HausDist from
              points in $R_A$ to points in $R_B$
environment : Number $n$ of MBRs used to compute
              EnhLB
1  MBR-List $L_A \leftarrow$ GetCovMBRs($R_A$);
2  MBR-List $L_B \leftarrow$ GetCovMBRs($R_B$);
3  Distance $d_{\max} \leftarrow 0$;
4  for each MBR $A$ in $L_A$ do
5      for each Face $F_A$ in $A$ do
6          Distance $d_{\min} \leftarrow \infty$;
7          $B \leftarrow$ First MBR in $L_B$;
8          while $d_{\min} \leq d_{\max}$ and $B$ is not null do
9              $d_{\min} \leftarrow \text{MIN}\{d_{\min}, \text{MinDist}(F_A, B)\}$;
10             $B \leftarrow$ Next MBR in $L_B$;
11         $d_{\max} \leftarrow \text{MAX}\{d_{\max}, d_{\min}\}$;
12 return $d_{max}$;
```

## 4.3 Discussion

As can be seen, the proposed similarity search algorithm (Algorithm 1) uses BscLB for preliminary sorting and uses EnhLB to refine the estimate initially given by BscLB. Consequently, we only compute EnhLB values for those point sets whose BscLB values are insufficient to rule them out from the search. However, by introducing EnhLB as an intermediate step, each of the resultant point sets has to be considered three times, i.e., once for each of BscLB, EnhLB and the actual HausDist. This incurs an overhead in terms of priority queue operations. To provide a better insight into performance evaluation, we compared this method to methods that use BscLB or EnhLB alone in our experiments (Section 6).

## 5. EXTENSION: HANDLING OUTLIERS

Since HausDist$(A, B)$ is a measure of maximum discrepancy of $A$ with respect to $B$, the measure can be sensitive to outliers. Specifically, if there is only one object $\boldsymbol{a}$ in $A$ that is far away from $B$, then distance from that object $\boldsymbol{a}$ to $B$ will be used as the resultant distance. That is, the measure disregards the majority of points in $A$ which are much closer to $B$. To mitigate this problem, a variant of the Hausdorff distance called the *modified Hausdorff distance (MHD)* [14] can be used to spread out the effect of outliers over the entire point set $A$. A formal definition of MHD can be given as

$$\text{MHD}(A, B) = \frac{\sum\{\text{MIN}\{\text{Dist}(\boldsymbol{a}, \boldsymbol{b}) : \boldsymbol{b} \in B\} : \boldsymbol{a} \in A\}}{|A|}.$$

In this section, we extend our concept of lower bound calculations to support the MHD measure.

Since MHD$(A, B)$ is the average of the distances from points in $A$ to their nearest point in $B$, the HausDist lower bound computed as the MaxMin distance from the MBRs of $A$ and $B$ is not guaranteed to be smaller than or equal to MHD$(A, B)$. As a result, we have to use MinDist as our *MHD basic lower bound (*MHD-BscLB*)* in this case.

In a similar manner as the enhanced lower bound for the Hausdorff distance, an MHD enhanced lower bound can be computed from MBRs of subsets inside the point sets $A$ and $B$. Algorithm 4 displays our MHD modification of Algorithm 3. Specifically, we can represent the point sets $A$ in $B$ as lists $L_A$ and $L_B$ of sub-MBRs, respectively (Lines 1 and 2). We can then compute a weighted sum of MinDist of MBRs in $L_A$ to $L_B$ based on the point count of the node corresponding to each MBRs in $L_A$ (Lines 3 to 11). The resultant distance is the sum divided by the total number of points (Line 12).

**Algorithm 4**: MHD-EnhLB($R_A$, $R_B$)

```
input       : R-Trees $R_A$ and $R_B$ of two point sets
output      : Optimistic Estimate of the HausDist from
              points in $R_A$ to points in $R_B$
environment : Number $n$ of MBRs used to compute
              EnhLB
1  MBR-List $L_A \leftarrow$ GetCovMBRs($R_A$);
2  MBR-List $L_B \leftarrow$ GetCovMBRs($R_B$);
3  Distance $c_{\text{total}} \leftarrow 0$;
4  Distance $d_{\text{sum}} \leftarrow 0$;
5  for each MBR $A$ in $L_A$ do
6      Distance $d_{\min} \leftarrow \infty$;
7      Count $c \leftarrow$ Number of points in $A$;
8      for each MBR $B$ in $L_B$ do
9          $d_{\min} \leftarrow \text{MIN}\{d_{\min}, \text{MinDist}(A, B)\}$;
10     $d_{\text{sum}} \leftarrow d_{\text{sum}} + d_{\min} \cdot c$ ;
11     $c_{\text{total}} \leftarrow c_{\text{total}} + c$;
12 return $d_{sum}/c_{total}$;
```

Figure 3 provides a comparison between MHD-BscLB, MHD-EnhLB and the actual HausDist from one point set to another. It can be seen that MHD-BscLB which is computed as MinDist yields an estimate of 0 units due to the overlap. On the other hand, we can decompose the root MBRs of $A$ and $B$ into sub MBRs where $M_{A1}$ corresponds to $\{\boldsymbol{a_1}, \boldsymbol{a_2}, \boldsymbol{a_3}\}$, $M_{A2}$ corresponds to $\{\boldsymbol{a_4}, \boldsymbol{a_5}, \boldsymbol{a_6}\}$, $M_{B1}$ corresponds to $\{\boldsymbol{b_1}, \boldsymbol{b_2}, \boldsymbol{b_3}\}$, and $M_{B2}$ corresponds to $\{\boldsymbol{b_4}, \boldsymbol{b_5}, \boldsymbol{b_6}\}$. MHD-EnhLB can be computed using Algorithm 4, where $L_A$ and $L_B$ are $\{M_{A1}, M_{A2}\}$ and $\{M_{B1}, M_{B2}\}$, respectively. In this case, MHD-EnhLB yields an estimate of 3 units which is much closer to the actual MHD of 7.23 units than MHD-BscLB.

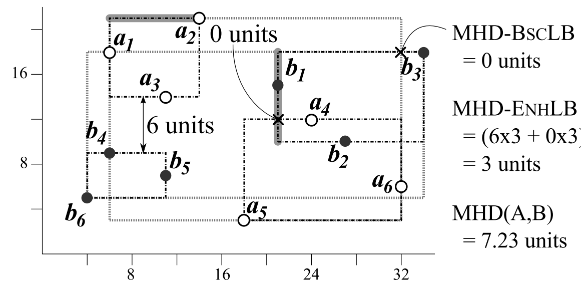


**Figure 3: Comparison between** MHD-BscLB, MHD-EnhLB **and** MHD **from** $A$ **to** $B$ **(**$\{\boldsymbol{a_1}, ..., \boldsymbol{a_6}\}$ **to** $\{\boldsymbol{b_1}, ..., \boldsymbol{b_6}\}$**)**

To form a similarity search algorithm, we can modify Algorithm 1 by replacing BscLB with MHD-BscLB, EnhLB with MHD-EnhLB, and HausDist with MHD. In our experimental studies, we compare a method which uses both MHD-BscLB and MHD-EnhLB to ones which use MHD-BscLB or MHD-EnhLB alone. The difference between the similarity measures HausDist and MHD are shown as example query results in Appendix B.

In *GIS'11: Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, Agrawal, I. Cruz, C. S. Jensen, E. Ofek, and E. Tanin, eds., pages 132–141, Chicago, November 2011.

## 6. EXPERIMENTAL STUDIES

In this section, we evaluate the effectiveness of our method. As shown in Table 1, we compare methods which use only BscLB or EnhLB to Algorithm 1 which uses a hybrid of BscLB and EnhLB. To emphasize this contrast, this method is referred to as Hyb in this section.

The rest of this section is organized as follows. We first describe the dataset used for testing. Next, we study the impact of adjusting the number $n$ of sub-MBRs selected for each point set on the performance of the three methods. Finally, we show the performance improvements achieved using Hyb for our sample datasets.

**Table 1: Similarity Search Methods**

| Search Method | Preliminary Sort | Refinement |
|---|---|---|
| Bsc | BscLB | - |
| Enh | EnhLB | - |
| Hyb | BscLB | EnhLB |

### 6.1 Setup

Evaluating our Hausdorff search algorithm requires running the algorithm on a collection of point sets. This subsection describes the creation of our testing dataset. Specifically, our collection was generated by geotagging [13] a large number of spreadsheets found on the Web. We first located a large number of spreadsheets that were likely to contain location data. To do this, we identified spreadsheets indexed by large search engines (`google.com` and `yahoo.com`) using their APIs to execute search queries of the following form.

```
filetype:xls <place_name_1> <place_name_2>
```

The parameters `place_name_1` and `place_name_2` were randomly selected from lists of country names, U.S. state names, and U.S. city names. The choice to use one or two search terms was also randomly determined. This technique yielded a large collection of 26,150 distinct URIs ending in "`.xls`" from a variety of sources on the Web.

After downloading from the resultant Web locations and excluding invalid Excel spreadsheet files, we used a method similar to that of WebTables [5] to locate sections of each spreadsheet that contained data rows (i.e., not column headers, table titles, or notes). Finally, we attempted to geotag each data row by identifying columns that contain values of a consistent geographical type (such as city, state, ZIP code), and using a gazetteer to combine geographic values from multiple columns to associate the row with a single geographic location (or no geographic location, if we cannot identify one) [12, 13, 21].

This procedure resulted in a collection of spreadsheets, each containing multiple data rows, many of which are annotated with a latitude and longitude. Due to the search query format we used to locate these spreadsheets, the references to locations have a variety of geographical distributions and scope. We should note that, although the accuracy and coverage of the spreadsheet extraction and geotagging algorithms is not the focus of this paper, the observed results appear to be highly accurate, producing many successfully geotagged documents.

The output of the spreadsheet selection, extraction, and geotagging process is a collection of 16,724 point sets with latitude and longitude coordinates, representing real-world data from the Web. From this collection, we selected a subset of the point sets in which there was a large degree of overlap between the point set MBRs and each set contained a large number of points. In particular, we selected only point sets containing over 300 points in North America to form a test dataset that we call NA-Test. Note that the Hyb search method exhibits fast performance on the full data set as well—however, small point sets were excluded because they cause the Hausdorff distance computation to become less expensive, so differences between the methods are negligible. As shown in Table 2, the resulting collection contains 923 point sets, with an average point set size of 955 points.

**Table 2: NA-Test Point Sets**

| | |
|---|---|
| Number of Point Sets | 923 |
| Minimum Point Count | 300 |
| Maximum Point Count | 5,796 |
| Total Point Count | 881,713 |

In our implementation, HausDist$(A, B)$ and MHD$(A, B)$ are computed by iterating through the pairwise distances $\{\text{Dist}(\boldsymbol{a}, \boldsymbol{b}) : \boldsymbol{a} \in A, \boldsymbol{b} \in B\}$. Although more sophisticated methods can be applied here, we chose the pairwise approach due to (i) its small memory footprint; and (ii) its reasonable performance in our setting where the average point set size is less than 1000 points.

### 6.2 Selecting the number of MBRs

The motivating hypothesis behind the Enh and Hyb search methods is that using multiple sub-MBRs to calculate a lower bound of the Hausdorff distance is significantly more accurate than using a single MBR (i.e., the root MBR). To test this hypothesis, we first look at the performance of the SimSearch algorithm using varying numbers of sub-MBRs.

Figure 4 shows the performance of the SimSearch algorithm on the NA-Test dataset, using different values of $n$. For this test, we randomly selected a sample of 100 point sets to serve as our collection of query point sets *QuerySets*. For each $Q \in$ *QuerySets*, we perform SimSearch with the number $k$ of results set to 1 and the number of MBRs set to $n$. The average running time for each value of $n$ is displayed. The tests were performed using the Enh and Hyb methods for $20 \leq n \leq 240$, and using the Bsc method. Since the Bsc method is equivalent to using either of the other methods with $n = 1$, the Bsc result is displayed as *num MBRs* = 1 in the figures. The running time for each value of $n$ is broken into three components: (i) *LB Time*, (ii) *HausDist Time*, and (iii) *PQ Time*. These correspond to (i) time spent generating lower bound estimates, (ii) time spent computing exact Hausdorff distances, and (iii) time spent maintaining the priority queue of the collection of point sets, NA-Test.

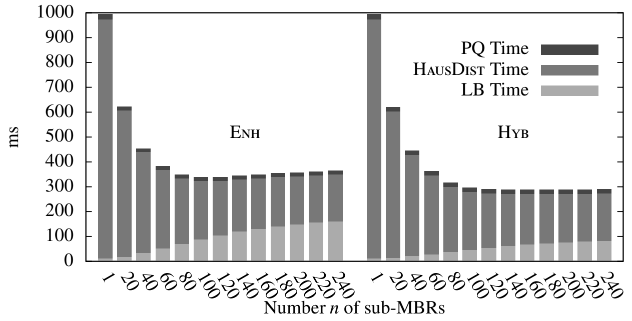


**Figure 4: Average performance of Enh and Hyb search methods on the NA-Test dataset, for different numbers of sub-MBRs, and $k = 1$.**

We can see that the worst performance, in terms of total running time, occurs when $n = 1$ (the BSC case). The majority of this time is spent computing exact Hausdorff distances between the query set $Q$ and other point sets in NA-TEST. *PQ Time* is relatively small in this case, as it is for all other values of $n$, so we focus our discussion on the *LB Time* and HAUSDIST *Time* factors. When $n = 1$, *LB Time* is also small, since only the BSCLB value is being computed for candidate point sets. However, as expected, BSCLB alone does not provide a particularly accurate ranking of point sets in NA-TEST, so identifying the set with the smallest Hausdorff distance to $Q$ requires performing a large number of exact Hausdorff distance calculations.

The value of $n$ has a positive correlation with *LB Time*. This is because calculating ENHLB requires visiting each sub-MBR in the query and candidate point sets. However, *HausDist Time* has a negative correlation with $n$ due to the accuracy improvement. This dominates the effect of $n$ on *LB Time*. As a result, we observe an overall decrease in the total running time as $n$ increases.

The same experiment was performed using MHD instead of HAUSDIST, as shown in Figure 5. The total computation time is generally one order-of-magnitude greater than for the HAUSDIST search experiment on the same data set. The slower behavior is caused by the inherent differences in the distance measures, which we observed in Section 5. In particular, since the lower bound computations are based on MINDIST instead of MAXMIN, the bounds are generally much smaller than their HAUSDIST counterparts. This means that many more candidate point sets must be considered before we identify point sets that have a MHD value that is less than the minimum remaining BSCLB or ENHLB in the priority queue of SIMSEARCH.

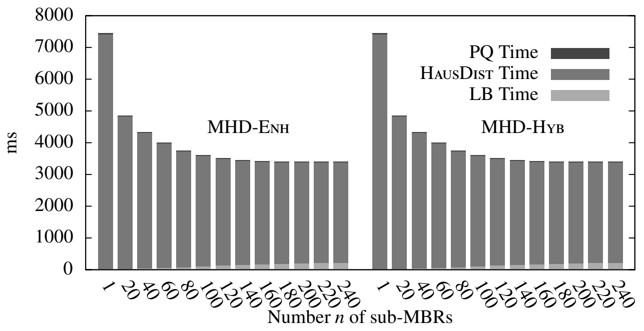


**Figure 5: Average performance of MHD-ENH and MHD-HYB search methods on the NA-TEST dataset, for different numbers $n$ of sub-MBRs and $k = 1$.**

Despite the overall increase in running time for performing SIMSEARCH with MHD, there is still a large increase in performance as $n$ increases from small values. The performance of the ENH and HYB search methods is nearly identical, which shows that using BSCLB provides very little benefit in the SIMSEARCH procedure.

In summary, the most significant outcomes of this experiment are (i) the performance of SIMSEARCH benefits greatly from using ENHLB, and (ii) choosing an appropriate value of $n$ does not require extreme precision. The first outcome is clear from the reduced running time for any $n$ value greater than 1, whereas the second follows from the nearly flat behavior of the HYB graphs for sufficiently large values of $n$. In this case, the flat behavior starts at the $n$ value of 140. Hence, we choose 140 as the default value of $n$ hereafter.

## 6.3 Performance Studies

Next, we focus on the performance improvements achieved by the HYB method under various queries and query parameters. The performance improvements can be evaluated using multiple measures, as displayed in Figures 6-8, which show how the performance of SIMSEARCH changes for different values of the $k$ parameter, while fixing $n$ to the default value of 140 MBRs. The experiment involved running SIMSEARCH once against the NA-TEST dataset for each combination of the following parameters:

- every $Q \in$ *QuerySets* (cardinality: 100)
- every search method (BSC, ENH, and HYB)
- every odd value of $k$ from 1 to 19
- both HAUSDIST and MHD

For each combination, we recorded the following measures:

- number of point set to point set Hausdorff distance computations performed
- number of point-to-point and MBR-to-MBR distance calculations performed
- total search time

Each measure was averaged over all $Q$ in *QuerySets*.

Figure 6 plots the increase in the total number of Hausdorff distance computations performed using each lower bound method, for increasing values of $k$. The results show that the BSC method consistently requires the greatest number of Hausdorff distance computations, while ENH and HYB both require significantly fewer calculations. In fact, ENH and HYB require exactly the same number of Hausdorff distance computations in each case. This result is due to the fact that both methods use the ENHLB value to order the search priority queue before computing the Hausdorff distance between point sets. The results also show that for every lower bound method, the number of distance calculations increases as $k$ increases, since the results of searching with a larger $k$ value will be a superset of the results with a smaller $k$ value. An interesting observation from this experiment is that the relative performance improvement from BSC to both ENH and HYB decreases as $k$ increases, which means that the largest reduction in Hausdorff distance computations occurs when $k$ is equal to 1.

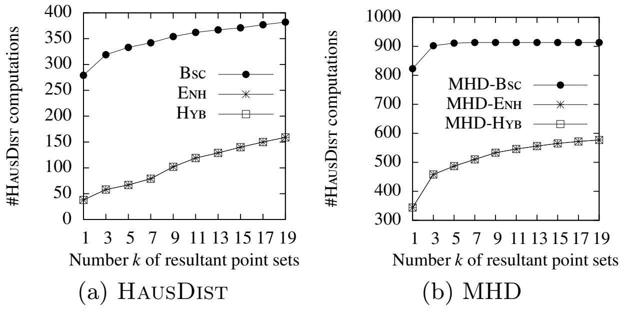


**Figure 6: Average number of full Hausdorff distance computations performed during SIMSEARCH queries using BSC, ENH and HYB search methods on the NA-TEST dataset, for different values of $k$.**

Figure 7 shows the total number of distance calculations that occur during search for increasing values of $k$. The number of distance calculations includes both point-to-point distance calculations performed during Hausdorff distance computations, and MBR-to-MBR distance calculations performed during lower bound computations. In the chart, we see a similar result to Figure 6, except that Enh and Hyb are slightly separated, representing the fact that measuring the total distance calculations also takes the lower bound computations into account. Hence, this is a more complete measure of the total computation costs than Hausdorff distance computations alone.

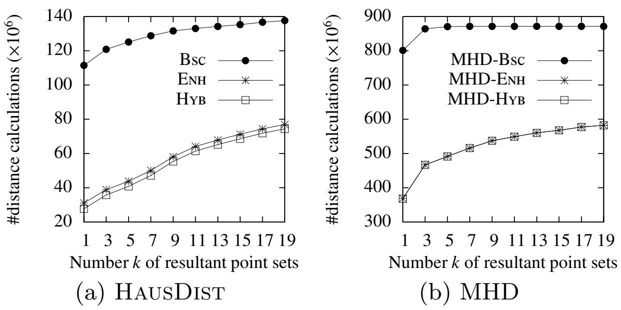


**Figure 7: Average number of distance calculations performed during SimSearch queries using Bsc, Enh and Hyb search methods on the NA-Test dataset, for different values of $k$.**

Figure 8 plots the average time required to return the $k$ point sets from *QuerySets* with the lowest Hausdorff distance from each $Q$. The elapsed time has a strong correlation to the number of distance calculations performed, as shown in Figure 7. However, here we see the average time required for each search, which ranges between 275 ms and 684 ms for the Hyb method using the Hausdorff distance, and between 3468 ms and 5459 ms for the Hyb method using the Modified HausDist (MHD). These times were recorded on an Intel i7-2720QM @ 2.20 GHz with 8GB RAM.

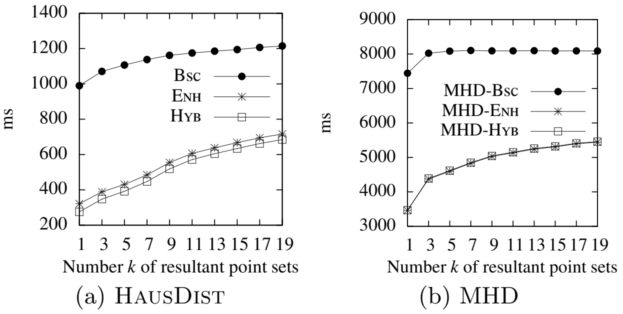


**Figure 8: Average performance of SimSearch queries using Bsc, Enh and Hyb search methods on the NA-Test dataset, for different values of $k$.**

## 6.4 Performance Distribution

To gain a better insight into the performance of Hyb relative to Bsc, we show distributions of performance improvements using histograms in addition to the average performance presented previously in Sections 6.2 and 6.3. The histogram in Figure 9(a) presents the relative performance of Hyb with respect to Bsc for 923 query point sets. The $x$-axis of the histogram represents the relative search time, i.e., the total search time of Hyb divided by that of Bsc for each query point set. For example, a relative performance value of 0.2 means that Bsc takes 5 times as long as Hyb to process the same query. The $x$ values are organized into 11 bins, where the leftmost bin represents a relative performance range of $[0.0, 0.1)$ and the rightmost bin represents a relative performance range of $[1.0, 1.1)$. The $y$-axis represents the count for each bin. We set $k$ and $n$ to the default values of 1 point set and 140 MBRs, respectively. The same setup also applies to Figure 9(b) which presents the relative performance of MHD-Hyb with respect to MHD-Bsc.

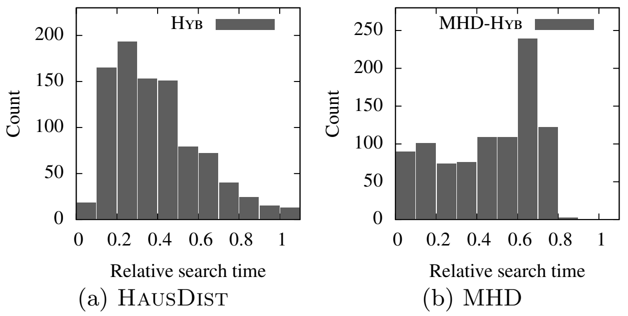


**Figure 9: Histogram of performance improvements for different query point sets. Performance improvement is measured as elapsed search time using the Hyb method, as a fraction of elapsed search time using Bsc, so smaller values represent larger speedups. For all tests, $n = 140$ and $k = 1$.**

The distribution of relative search times for Hausdorff distance searches is shown in Figure 9(a). The vast majority of Hyb searches take only 0.1 to 0.5 of the original Bsc time, corresponding to a 50% to 90% reduction in search time, which is a significant improvement. 73.7% of queries result in Hyb search times that are at least 50% less than the corresponding Bsc search time. A small fraction of query point sets (1.4%) suffer an increased search time, which occurs when using the EnhLB does not result in a better ordering of point sets in the priority queue.

Using the Hyb method for modified Hausdorff distance searching also achieves significant speedup factors, although the distribution is different, as shown in Figure 9(b). In particular, all queries are sped up using Hyb, and a larger fraction achieve speedup factors higher than 90%. However, only 48.8% of queries result in Hyb search times that are at least 50% less than the corresponding Bsc search time.

## 7. CONCLUSIONS AND FUTURE WORK

This paper presents a new approach for similarity search over a large collection of point sets, where similarity is measured using the Hausdorff distance. Our method constructs an ordering of the collection of point sets using a new lower bound estimation technique called EnhLB, which allows us to rule out dissimilar point sets without computing the full Hausdorff distance between them and our query set. We also applied this technique to searches using the modified Hausdorff distance, an outlier-resistant variant. On a dataset of geotagged spreadsheets from the Web, similarity search times improved significantly using our method, for both distance measures.

One avenue for future work is to formulate an algorithm that incrementally refines the lower bound until it becomes an exact Hausdorff distance, so that no computation is wasted. Such a method would allow for early termination, if the range between the lower and upper bounds is sufficiently small to guarantee a specified error threshold (i.e. an approximation algorithm). This work could also be used to support feature-based queries as in spatial data mining [3].

**Acknowledgements.** This work was supported in part by the National Science Foundation under Grants IIS-10-18475, IIS-09-48548, IIS-08-12377, CCF-08-30618, and IIS-07-13501.

## 8. REFERENCES

[1] H. Alt, B. Behrends, and J. Blömer. Approximate matching of polygonal shapes. In *SCG*, pages 186–193, 1991.

[2] H. Alt, P. Brass, M. Godau, C. Knauer, and C. Wenk. Computing the Hausdorff distance of geometric patterns and shapes. *Discrete and Comput. Geometry*, 25:65–76, 2003.

[3] W. G. Aref and H. Samet. Efficient processing of window queries in the pyramid data structure. In *PODS*, pages 265–272, 1990.

[4] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *SIGMOD*, pages 322–331, 1990.

[5] M. J. Cafarella, A. Y. Halevy, Y. Zhang, D. Z. Wang, and E. Wu. Uncovering the relational web. In *WebDB*, 2008.

[6] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos. Closest pair queries in spatial databases. In *SIGMOD*, pages 189–200, 2000.

[7] M. Guthe, P. Borodin, and R. Klein. Fast and accurate Hausdorff distance calculation between meshes. In *WSCG*, pages 41–48, 2005.

[8] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[9] G. R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. *ACM Trans. Database Syst.*, 28(4):517–580, 2003.

[10] G. R. Hjaltason and H. Samet. Incremental distance join algorithms for spatial databases. In *SIGMOD*, pages 237–248, 1998.

[11] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Trans. Database Syst.*, 24(2):265–318, 1999.

[12] M. D. Lieberman, H. Samet, J. Sankaranarayanan, and J. Sperling. STEWARD: architecture of a spatio-textual search engine. In *GIS*, pages 186–193, 2007.

[13] M. D. Lieberman, H. Samet, J. Sankaranaryanan, and J. Sperling. Spatio-textual spreadsheets: Geotagging via spatial coherence. In *GIS*, pages 524–527, 2009.

[14] R. Lipikorn, A. Shimizu, and H. Kobatake. A modified exoskeleton and a Hausdorff distance matching algorithm for shape-based object recognition. In *CISST*, pages 507–511, 2003.

[15] S. Nutanong, E. H. Jacox, and H. Samet. An incremental Hausdorff distance calculation algorithm. *PVLDB*, 4(8):506–517, 2011.

[16] D. Papadias, Y. Tao, K. Mouratidis, and C. K. Hui. Aggregate nearest neighbor queries in spatial databases. *ACM Trans. Database Syst.*, 30(2):529–576, 2005.

[17] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *SIGMOD*, pages 71–79, 1995.

[18] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan-Kaufmann, San Francisco, 2006.

[19] H. Shin, B. Moon, and S. Lee. Adaptive multi-stage distance join processing. In *SIGMOD*, pages 343–354, 2000.

[20] M. Tang, M. Lee, and Y. J. Kim. Interactive Hausdorff distance computation for general polygonal models. In *SIGGRAPH*, pages 74:1–74:9, 2009.

[21] B. Teitler, M. D. Lieberman, D. Panozzo, J. Sankaranarayanan, H. Samet, and J. Sperling. NewsStand: A new view on news. In *GIS*, pages 144–153, 2008.

[22] C. Xia, H. Lu, B. C. Ooi, and J. Hu. Gorder: An efficient method for KNN join processing. In *VLDB*, pages 756–767, 2004.

## APPENDIX

## A. ACCURACY OF ESTIMATORS

Figure 10 shows the accuracy of BscLB and EnhLB as we vary the number $k$ of resultant point sets. For smaller $k$ values, EnhLB produces estimates which are much closer to the actual HausDist (normalized as 1). As $k$ increases (which means that HausDist$(Q, S)$ increases), the difference between EnhLB and BscLB diminishes. As a result, when sorting point sets $S$ with respect to $Q$, we can use BscLB to rule out point sets that are obviously far away from $Q$ and use EnhLB for point sets that require further examinations before calculating the HausDist. The description of this dataset, NA-Test, is provided in Section 6.

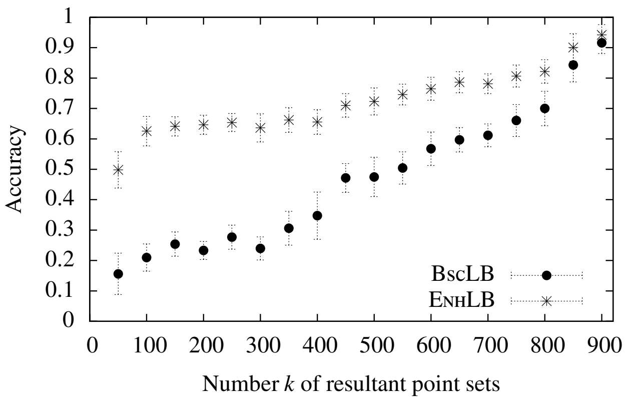


**Figure 10: Accuracy of the two estimators BscLB and EnhLB given as the estimated distance divided by HausDist where (i) the measured value is the average $\mu$ of 100 runs, and (ii) each error bar represents one standard deviation in either direction from $\mu$.**

## B. EXAMPLE SEARCH RESULTS

Figure 11 shows a query point set $Q$, comprising locations in Illinois, and two of the top results (point sets $A$ and $B$) when searching with the symmetric measures SymHausDist and SymMHD in the NA-Test dataset (described in Section 6). SymHausDist$(Q, A)$ of 1.01 units is lower than SymHausDist$(Q, B)$ of 1.32 units. For the SymMHD, the results are reversed. That is, SymMHD$(Q, A)$ is 0.26 units which is greater than SymMHD$(Q, B)$ of 0.22 units. As can be seen, $B$ has a large collection of points near query points in the top right corner but has a few outliers near the bottom which are far away from the query points. Using MHD means that the effects of these outliers are reduced by the averaging nature of the distance function. Hence, $B$ is considered nearer to $Q$ than $A$ according to MHD.

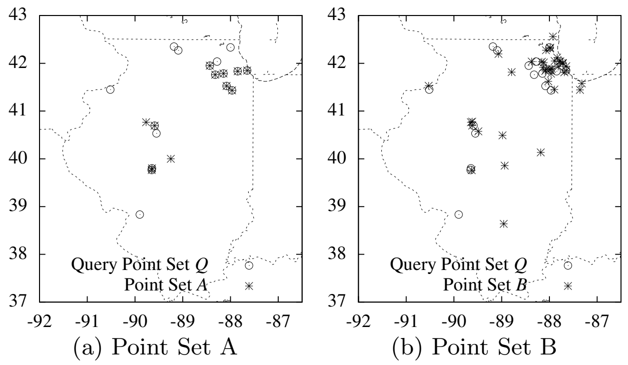


**Figure 11: Example SimSearch results comparing HausDist and MHD.**