New Techniques for Best-Match Retrieval

DENNIS SHASHA and TSONG-LI WANG
New York University

A scheme to answer best-match queries from a file containing a collection of objects is described. A best-match query is to find the objects in the file that are closest (according to some (dis)similarity measure) to a given target.

Previous work [5, 33] suggests that one can reduce the number of comparisons required to achieve the desired results using the triangle inequality, starting with a data structure for the file that reflects some precomputed intrafile distances. We generalize the technique to allow the optimum use of any given set of precomputed intrafile distances. Some empirical results are presented which illustrate the effectiveness of our scheme, and its performance relative to previous algorithms.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*sorting and searching*; H.2.4 [**Database Management**]: Systems—*query processing*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*search process*; I.2.m [**Artificial Intelligence**]: Miscellaneous

General Terms: Algorithms, Performance, Theory

Additional Key Words and Phrases: Best match, distance metrics, file searching, heuristics, lower bounds, matching, topology, upper bounds

1. INTRODUCTION

In this paper we are concerned with the best-match problem (also known as the "nearest neighbor problem" [35], or the "closest point problem" [31]). Given a file of objects, the best-match problem is to find the ones which are most similar or closest to a given target (or query) according to some (dis)similarity measure. This type of retrieval arises in many applications.[1] In information systems, documents in a file are often ranked in order of decreasing similarity with a given query. One way of computing the similarity between a document and the query is to count the number of terms in common between them; the documents presented to the user first are those that contain the greatest number of terms specified in the query. The best-match searching procedure in such ranked output

[1] Depending on the application, the objects could refer to documents [30, 38, 39], records [42], patterns [15], points [31], strings [17–19], trees [32], graphs [6, 12], and so forth.

This work was supported in part by the NSF under grant IRI-8901699 and by the Office of Naval Research under grants N00014-85-K-0046 and N00014-90-J-1110.

Authors' address: Courant Institute of Mathematical Sciences, New York University, New York, NY 10012.

retrieval systems identifies the terms in common between the query and each of the documents in the file [36]. In molecular biology, to gain information about a newly sequenced protein, biologists compare the protein's amino acid sequence against those of many known proteins, searching for ones with very similar sequences. From such sequence similarities it is often possible to infer similarities in the structures or functions of the related proteins [18, 19]. In pattern classification, a nearest neighbor assignment strategy is widely used: an unlabeled sample is assigned to the category to which a majority of its nearest neighbors belong [8, 25]. In many other areas, the best-match type algorithm has been found effective for estimating multivariate density [4], minimizing head movement on direct access I/O devices [25], or sequencing pens for plotting devices [3].

1.1 Previous Techniques for Best-Match Retrieval

One straightforward way of solving the best-match problem is to compute the (dis)similarity value between each object of the file and the target, and then to search for the objects with maximum similarity (or minimum dissimilarity). The major problem with this approach is its computational expense, particularly when there are many targets to be identified and the file is large. To reduce computational effort, many techniques have been presented in the literature.

Smeaton and van Rijsbergen [35], Murtagh [22], and Perry and Willett [27] employ inverted files for best-match searching in document retrieval systems. The search procedure starts with the shortest inverted file lists and calculates an upper bound during (or before) the processing of each query list. The bound represents the maximum possible similarity value for those documents that have not yet been inspected; if it is less than the similarity for the current nearest neighbor, no further documents need to be processed because none of them can possibly be a better match. Various upper bounds have been derived by the authors; the similarity measures applied include Dice, Overlap, Cosine coefficients, etc. (see [30, 39] for definitions of these measures). Other strategies that use bounding procedures to eliminate (dis)similarity computation in various contexts have been suggested by Mohan and Willett [21], Fukunaga and Narendra [15], Feustel and Shapiro [12], to name a few. Lucarella, in [20], describes a document retrieval system based on inverted file organizations and nearest neighbor search techniques.

Special data structures other than inverted files have also been proposed. Shamos and Hoey [31], for example, employ the Voronoi diagram for the best-match problem for points on a plane. Ito and Kizawa [17] devised *HL* files (*H*ierarchically organized files based on a *L*inear ordering) for spelling correction applications. They give a recursive procedure to search the file for retrieving similar strings. Friedman et al. [14] partition a d-dimensional feature space using the $k - d$ tree method and search by descending the tree. Eastman, Weiss and Zemankova [9–11] generalize $k - d$ tree methods to high dimensional space, which are particularly useful for document retrieval.

Another strategy for the best-match problem is to use hashing. Du and Lee [7] use Gray code as a multikey hashing function for finding closest symbolic records. The dissimilarity measure they use is the Hamming distance. Bentley et al. [4],

Rohlf [29], and Murtagh [23] give algorithms for points in Euclidean space. Their algorithms work by hashing points onto directly addressable cells and then searching for best matches in the same or closely adjacent cells. An excellent survey on using tree structures and hashing techniques for best-match retrieval can be found in [24, 25].

In contrast to the above approaches, many of which need assumptions such as that the (dis)similarity measures have finite dimensionality (e.g., Euclidean), or that the objects can be ordered linearly, Burkhard and Keller [5] give an algorithm based on the weakest possible assumption on dissimilarity measures, namely, they only satisfy the fundamental properties of a distance metric. The authors precompute the distances of all objects in a file to a randomly chosen object, called the *reference point*, and develop cut-off procedures to eliminate certain distance calculations by simply using the triangle inequality. Shapiro [33] later improved their method by having more reference points and by deriving stricter cut-off criteria for eliminating objects. He concluded that large improvements can be achieved by the proper choice and location of reference points.

1.2 Motivation and Assumptions

In this paper we introduce new techniques for best-match retrieval, assuming with [5, 33] that only distance metric information is available. We generalize Burkhard and Keller's methods to make use of an *arbitrary* set of precomputed distances. Our motivation for considering arbitrary sets is that at times one may be given a set of distances which have been calculated, rather than being able to choose which ones to calculate. Also, in practice, files are *dynamic* rather than *static*—objects can be inserted or deleted from a file or may be updated in such a way that distances change. Hence, as time goes on, certain precomputed distances may become absent or obsolete, and thus our techniques can apply.

Our cost assumption is that distance computation is the dominant cost, so our goal is to minimize such computation during searching. This assumption is reasonable when object comparison is an extremely time-consuming job (for example, when retrieving best matches from a sequence database, where it can take seconds or minutes to compare even one protein or RNA structure against another on a current model VAX [18, 32].)

The paper is organized as follows. Section 2 reviews Burkhard and Keller's methods. Section 3 describes our approach, where we use a Floyd–Warshall [13, 41] style algorithm to approximate absent intrafile distances, and develop a search algorithm that best uses the given distance information. Section 4 reports some experimental results. We conclude the paper in Section 5.

2. BURKHARD AND KELLER'S METHOD REVIEWED

Let $\mathcal{F}$ be a file of n objects $O_1, O_2, \ldots, O_n$. The best-match problem is defined as follows: Given a target T, find the pairs (T, O), $\forall O \in \mathcal{F}$, with minimum distance. The distance between two objects O and O' is given by the value of a metric $d(O, O')$. Specifically, a metric is a function d that takes pairs of objects into nonnegative numbers, satisfying the following three properties: for any objects O_1, O_2, O_3, $d(O_1, O_2) \geq 0$, and $d(O_1, O_2) = 0$ iff $O_1 = O_2$ (nonnegative

```
1.    precompute d(O, O^0), ∀ O ∈ F, for a randomly chosen reference point O^0 ∈ F;
2.    compute d(T, O^0); ξ := d(T, O^0); B := {O^0}; I := F - {O^0};
      while I ≠ ∅
3.        use a heuristic (described below) to pick an object O in I;
4.        update (B, O, T, ξ);
5.        I := {O | (d(T, O) is not computed) ∧ (|d(O, O^0) - d(T, O^0)| ≤ ξ)}
      end;
```

Fig. 1. Basic search algorithm of Burkhard and Keller.

definiteness); $d(O_1, O_2) = d(O_2, O_1)$ (symmetry); $d(O_1, O_2) \leq d(O_1, O_3) + d(O_3, O_2)$ (triangle inequality).

Using the terminology of [5, 33], ξ = the current minimum distance to T, B = the set of current best matches, and the function update (B, O, T, ξ) tests whether $d(T, O) \leq \xi$ and, if so, updates B and ξ. Let I = the set of candidates (i.e., objects that haven't been eliminated, nor been compared). Burkhard and Keller's algorithm proceeds in stages and is paraphrased in Figure 1.

Step 3 picks objects according to the criterion: $|d(X_k, O^0) - d(T, O^0)| \leq |d(X_{k+1}, O^0) - d(T, O^0)|$, where X_k, $k = 1, 2, 3, \ldots$, is the object picked at stage k. Observe that $|d(O, O^0) - d(T, O^0)|$ is a lower bound for $d(T, O)$. The heuristic used picks the object with the smallest lower bound first. Each stage eliminates objects whose lower bounds are already greater than ξ, the current minimum distance (step 5, cut-off criterion). The algorithm stops with the closest objects being in B and the minimum distance being ξ.

Shapiro improves step 1 above by precomputing $d(O, O^i)$, $\forall O \in \mathcal{F}$, $i = 1, 2, \ldots, s$, for s reference points $O^i \in \mathcal{F}$. Step 2 is then refined by computing the distances between T and the s reference points, and ξ is set to the minimum of these distances. In step 3, objects are chosen such that $|d(X_k, O^1) - d(T, O^1)| \leq |d(X_{k+1}, O^1) - d(T, O^1)|$, and the I obtained at each stage is $\{O \mid (d(T, O)$ is not computed) $\wedge$ $(\forall_{i=1}^{s}(|d(O, O^i) - d(T, O^i)| \leq \xi))\}$ (cut-off criterion).

Let us construct a weighted graph on $\mathcal{F}$, which reflects precomputed information, such that there is an edge e between O_i and O_j iff $d(O_i, O_j)$ has been computed, and the weight of e, denoted $w(e)$, is $d(O_i, O_j)$. It can be seen that Burkhard's algorithm starts with a star, whereas there are s stars for Shapiro's, each centered with a different reference point. In the next section we relax this requirement by accepting any topology on the weighted graph and present an algorithm that achieves the optimal approximation for those unknown intrafile distances.

3. OUR APPROACH

Our data structure must handle an arbitrary set of precomputed distances and allow us to approximate, with the greatest possible accuracy, other distances. We call our structure an *approximate distance map* (*ADM*) for $\mathcal{F}$, which is an $n \times n$ matrix with each entry ADM$[i, j]$ being either the exact distance between objects O_i and O_j, or (if that is not computed) being a lower bound for $d(O_i, O_j)$. This bound may be rather crude (i.e., too low) initially, and it will be gradually refined after new distances between T and objects in $\mathcal{F}$ are computed.

To compute the ADM, consider the weighted graph constructed on $\mathcal{F}$. Define a path from $O_i = O_{i_1}$ to $O_j = O_{i_n}$ as a sequence of distinct objects $O_{i_1}, O_{i_2}, \ldots, O_{i_n}$ such that $\{O_{i_1}, O_{i_2}\}, \{O_{i_2}, O_{i_3}\}, \ldots, \{O_{i_{n-1}}, O_{i_n}\}$ are edges in the graph, and the weight of the path is the sum of the weights of its constituent edges.

LEMMA 1. (Generalized Triangle Inequality). *Suppose there is a path P from O_i to O_j. Let $\hat{e}$ be the edge of maximum weight in P. Then*

$$d(O_i, O_j) \geq w(\hat{e}) - \sum_{e \in P - \{\hat{e}\}} w(e).$$

PROOF. By induction on the number of objects in P and repeated application of the triangle inequality. □

Lemma 1 states that one can obtain a lower bound for $d(O_i, O_j)$ by applying the triangle inequality to a path from O_i to O_j. Of course, such a bound is useless if the term on the right-hand side of the inequality is less than or equal to 0. Generally, we want this bound to be as high as possible. Let $P(i, j)$ be the set containing all paths from O_i to O_j. We define ADM$[i, j]$ to be the maximum bound obtained from all paths in $P(i, j)$. So, ADM$[i, j] \leq d(O_i, O_j)$. By the triangle inequality, ADM$[i, j] = d(O_i, O_j)$ if edge $\{O_i, O_j\} \in P(i, j)$.

It is impractical, in general, to enumerate all paths in $P(i, j)$ to get ADM$[i, j]$, because there may be an exponential number of them. Instead, we use a dynamic programming technique similar to the transitive closure algorithm [41] to compute the ADM. To facilitate the computation, we also maintain an additional matrix MIN, where MIN$[i, j]$ is the minimum weight of any path from O_i to O_j. Thus, MIN$[i, j]$ gives the least upper bound of the distance between O_i and O_j, given the current distance information. So, MIN$[i, j] \geq d(O_i, O_j)$. Clearly, MIN$[i, j] = d(O_i, O_j)$ if $\{O_i, O_j\} \in P(i, j)$.

Following [2], let ADM$_k[i, j]$ (respectively, MIN$_k[i, j]$), $0 \leq k \leq n$, be the greatest lower bound (respectively, least upper bound) of any path from O_i to O_j that does not pass through an object numbered higher than k.

LEMMA 2. *Let $S_k(i, j)$, $1 \leq k \leq n$, denote the set of paths going from O_i to O_k and then from O_k to O_j, without passing through an object numbered higher than k. Suppose $S_k(i, j) \neq 0$. Let $B_k(i, j)$ be the greatest lower bound obtained by applying the generalized triangle inequality to all the paths in $S_k(i, j)$. Then*

$$B_k(i, j) = \max \begin{cases} \text{ADM}_{k-1}[i, k] - \text{MIN}_{k-1}[k, j] \\ \text{ADM}_{k-1}[j, k] - \text{MIN}_{k-1}[k, i] \end{cases} \quad \text{for} \quad 1 \leq k \leq n$$

PROOF. Let $P \in S_k(i, j)$ be a path yielding $B_k(i, j)$. Let P_1 be the segment of P between O_i and O_k and P_2 be the segment of P between O_k and O_j. Suppose first that the edge $\hat{e}$ of maximum weight is in P_1. By Lemma 1, we get

$$B_k(i, j) = w(\hat{e}) - \sum_{e \in P_1 - \{\hat{e}\}} w(e) - \sum_{e \in P_2} w(e).$$

Claim that

$$\text{ADM}_{k-1}[i, k] = w(\hat{e}) - \sum_{e \in P_1 - \{\hat{e}\}} w(e).$$

Proof of Claim. By induction,

$$\mathrm{ADM}_{k-1}[i, k] \geq w(\hat{e}) - \sum_{e \in P_1 - \{\hat{e}\}} w(e).$$

If inequality held, we could construct a path P' in $S_k(i, j)$ by concatenating a path P_1', which yields $\mathrm{ADM}_{k-1}[i, k]$, and P_2. The bound achieved by P' would be greater than $B_k(i, j)$, contradicting the definition of $B_k(i, j)$. □

By an analogous argument,

$$\mathrm{MIN}_{k-1}[k, j] = \sum_{e \in P_2} w(e).$$

Thus, $B_k(i, j) = \mathrm{ADM}_{k-1}[i, k] - \mathrm{MIN}_{k-1}[k, j]$.

If $\hat{e}$ is in P_2, symmetric arguments yield $B_k(i, j) = \mathrm{ADM}_{k-1}[j, k] - \mathrm{MIN}_{k-1}[k, i]$. □

From the lemma above, we have, for each k,

$$\mathrm{ADM}_k[i, j] = \max \begin{cases} \mathrm{ADM}_{k-1}[i, j] \\ \mathrm{ADM}_{k-1}[i, k] - \mathrm{MIN}_{k-1}[k, j] \\ \mathrm{ADM}_{k-1}[j, k] - \mathrm{MIN}_{k-1}[k, i] \end{cases}$$

Moreover [2],

$$\mathrm{MIN}_k[i, j] = \min \begin{cases} \mathrm{MIN}_{k-1}[i, j] \\ \mathrm{MIN}_{k-1}[i, k] + \mathrm{MIN}_{k-1}[k, j] \end{cases}$$

These formulas give rise to a Floyd–Warshall style algorithm for computing the approximate distance map. The procedure is given in Figure 2.[2]

Using induction on k, we obtain

THEOREM 1. *Algorithm APPROXIMATE correctly computes matrices ADM and MIN; that is, the lower (respectively, upper) bound of any path going from O_i to O_j is less (respectively, greater) than or equal to ADM[i, j] (respectively, MIN[i, j]), given the distances that have been computed.*

Thus, given a weighted graph G of arbitrary topology (i.e., an arbitrary set of precomputed distances), we can apply algorithm APPROXIMATE to it and obtain two matrices ADM and MIN. Theorem 1 guarantees that each entry (i, j) in the matrix ADM (respectively, MIN) represents the greatest lower bound (respectively, least upper bound) of all paths between O_i and O_j in G. Using the bound information, we are able to eliminate the largest possible number of objects that could not be a best match, given the computed distance information.

3.1 Searching Using an ADM

We first augment the ADM with an additional row, row $n + 1$, for object T (i.e., treating T as O_{n+1}) with $\mathrm{ADM}[n + 1, i]$ being the current greatest lower

[2] Due to the symmetry, one may improve the running time of the presented algorithm by only computing the lower triangular part of the matrices.

```
for i := 1 to n do
    for j := 1 to n do
        if d(O_i, O_j) is known then begin
            ADM[i, j] := d(O_i, O_j); MIN[i, j] := d(O_i, O_j)
        end
        else begin
            ADM[i, j] := 0; MIN[i, j] := ∞
        end;
for k := 1 to n do
    for i := 1 to n do
        for j := 1 to n do begin
            ADM[i, j] := max (ADM[i, j], ADM[i, k] − MIN[k, j], ADM[j, k] - MIN[k, i]);
            MIN[i, j] := min (MIN[i, j], MIN[i, k] + MIN[k, j]);
        end;
```

Fig. 2. Algorithm APPROXIMATE.

bound for $d(T, O_i)$.[3] After comparing an object with T, we update I so that it contains only the objects O_i's whose ADM[$n + 1, i$] is still less than or equal to ξ. Figure 3 gives our search algorithm.

The algorithm picks candidates (i.e., objects that are still in I) according to the following heuristic (step 3). It picks the first object randomly, and in subsequent stages, it selects an object O_i such that the lower bound of the distance between O_i and the given target T is minimized based on all previous candidates (i.e., ADM[$n + 1, i$] $\leq$ ADM[$n + 1, j$], $\forall O_j \in I$). The object having the least lower bound is expected to be the closest object to T. If several candidates have the same lower bound, the algorithm selects one that has the least upper bound (i.e., the one with the smallest MIN value). The reason for doing so is that the smaller the difference between the lower and upper bounds, the more precise the estimated distance is. Ties on the difference are broken arbitrarily.[4]

It is worth noting that, starting with an optimal approximate distance map (Theorem 1), the algorithm developed here is the best possible for the best-match problem, in the sense that given an object at stage i, it throws out *all* the objects that can be inferred to be irrelevant to the solution at that stage. What may influence the performance of the algorithm is the heuristic utilized in selecting objects at each stage—the better the heuristic (or the better our luck), the better performance the algorithm achieves.

3.2 Updating Augmented ADM and MIN

Each computation of the distance between T and some object O_k may lead to modifications of the augmented ADM and MIN. Observe that the value of $d(T, O_k)$ affects only the paths going through $\{T, O_k\}$. Let L (respectively, U) be the new lower (respectively, upper) bound of the paths from O_i to O_j via $\{T, O_k\}$;

[3] We discuss how to update such an augmented ADM in Section 3.2. For now, let us assume that this map can somehow be maintained.

[4] We have tested several other heuristics for picking candidates, such as picking objects with the greatest lower bound, picking objects with the least (or greatest) upper bound, or picking objects at random. It is shown [40] that the heuristic presented here (i.e., picking objects with the least lower bound) achieves the best performance over all other heuristics.

```
1.    ξ := ∞; B := ∅; I := F;
2.    initialize and approximate ADM and MIN as done in Figure 2, and
      augment ADM and MIN with an additional row for object T;
      while I ≠ ∅
3.        use a heuristic (described below) to pick an object O in I;
4.        update (B, O, T, ξ);
5.        update the augmented ADM and MIN;
6.        I := {O_i | (d(T, O_i) is not computed) ∧ (ADM[n + 1, i] ≤ ξ)}
      end;
```

Fig. 3. Our search algorithm.

as in Lemma 2, we obtain

$$L = \max\begin{cases} d(T, O_k) - \text{MIN}[i, n+1] - \text{MIN}[k, j] \\ d(T, O_k) - \text{MIN}[i, k] - \text{MIN}[j, n+1] \\ \text{ADM}[i, n+1] - d(T, O_k) - \text{MIN}[k, j] \\ \text{ADM}[i, k] - d(T, O_k) - \text{MIN}[n+1, j] \\ \text{ADM}]j, k] - d(T, O_k) - \text{MIN}[i, n+1] \\ \text{ADM}[n+1, j] - d(T, O_k) - \text{MIN}[i, k] \end{cases}$$

and

$$U = \min\begin{cases} \text{MIN}[i, n+1] + d(T, O_k) + \text{MIN}[k, j] \\ \text{MIN}[i, k] + d(T, O_k) + \text{MIN}[n+1, j] \end{cases}.$$

Thus, after computing $d(T, O_k)$, to find the new (tighter) bounds for the distances between objects O_i, $O_j \in \{T\} \cup \mathcal{F}$, it suffices to compare ADM$[i, j]$ (respectively, MIN$[i, j]$) with L (respectively, U) (recall that ADM$[n+1, i]$ always gives the current greatest lower bound for $d(T, O_i)$).

Note that we update only the pairs whose distances are still unknown. For those pairs of objects whose distances have been calculated, the distance values already represent both the best lower bounds and upper bounds, and hence they need not be modified. Calculating L and U takes only constant time. Thus the overhead incurred by updating a map is negligible when most intrafile distances are present.

If, however, there exist a large portion of object pairs in the file whose distances are absent, the recomputation would be quite expensive. In such a situation we could update the bounds for pairs (T, O_i), $O_i \in \mathcal{F}$, while keeping the initial bounds for (O_i, O_j), $O_i, O_j \in \mathcal{F}$ (this strategy is similar to the one suggested in [1] for maintaining shortest paths in a sizable graph), or could only update the bounds for pairs (T, O), where O is still a candidate. In [34], both the updating policies have been shown empirically to be very competitive to the one that globally updates the bounds for all object pairs (including the target), yet saving a significant amount of computation time.

4. PERFORMANCE ANALYSIS

A series of experiments were performed to evaluate the effectiveness of our search algorithm, as well as its performance relative to those proposed previously. Table I shows the basic parameters used in the experiments.

Table I. Experimental Parameters

Parameter	Meaning
Size	Number of objects in the file
Density	Portion of known distances in the map
MinDistance	Minimum distance between objects
MaxDistance	Maximum distance between objects

[*MinDistance*, *MaxDistance*] specifies the range over which distances between objects (including the target) are distributed. The *Density* parameter represents the portion of known distances in a map, and is computed by dividing the number of object pairs with known distances by the total number of object pairs in the corresponding file. To compare different algorithms for the best-match query, the following metric was used:

$$PERFO = \frac{NumCompared}{Size} \times 100\%$$

where *NumCompared* is the number of objects actually compared. *PERFO* stands for *PER*centage of brute *FO*rce cost (i.e., the cost of comparing the target with every object in the file). One would like this percentage to be as low as possible.

4.1 Uniformly Distributed Distances

In the first set of experiments we showed how varying densities and file sizes impact the performance of our search algorithm. The sample maps used in the experiments were synthesized as follows. We used a random-number generator to produce interobject distance values for each pair of objects (including the target), where the values were distributed uniformly over some positive interval. Each such value was inserted into a $(Size + 1) \times (Size + 1)$ auxiliary map, provided that it did not violate the triangle inequality. After generating the map, we randomly selected $Density \times (Size \times (Size - 1))/2$ entries from the lower triangular part of the $Size \times Size$ matrix of interobject distances, excluding the target. (Entries in the $(Size + 1)$th row and $(Size + 1)$th column represented distances between the target and objects in the file.)

Figure 4 presents the result, where distances between objects (including the target) were drawn from the range [0, 10000].[5] Files with sizes 100, 150, 200, 250, 300 were examined. It can be seen that *PERFO* drops (i.e., improves) as the density of a map increases; the improvement slows down after the density is greater than 0.3. No trend is evident with regard to file sizes, though *PERFO* seems slightly lower (i.e., better) for larger files. This probably happens because the larger the file, the more entries need to be generated. Since the distance range is fixed, it becomes more likely that several objects have the same distance from the currently examined object (which may cause them to be eliminated at the same time).

One interesting finding is that most irrelevant objects tend to be discarded in the initial stages of a search. For example, for $Density = 0.9$ and $Size = 150$, our

[5] In this, and subsequent figures, each point of the graph represents the average value over thirty maps.

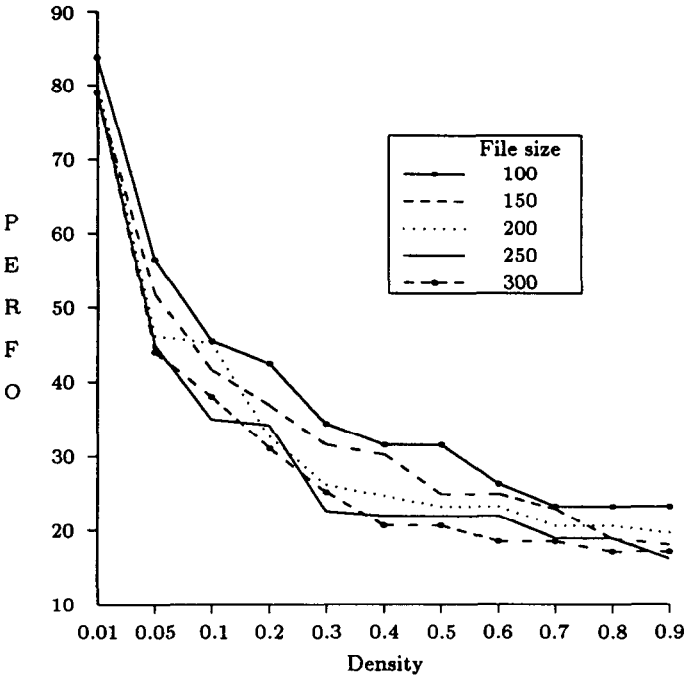


Fig. 4. Effect of densities for uniformly distributed distances; distances between objects (including the target) are drawn from [0, 10000].

algorithm completes a search in 27 stages, eliminating 123 objects on the average, among which nearly 80 objects are eliminated in the first 10 stages; only 43 objects are discarded in the remaining 17 stages. A possible explanation for this behavior is that most objects whose lower bounds are greater than the current minimum distance are eliminated in the earlier stages, leaving few to discard in the later stages.

We next examined the behavior of our search algorithm for varying distance ranges. It was expected that the parameters *MinDistance* and *MaxDistance* have strong influence on the performance of the algorithm. If $MaxDistance \leq 2 \times MinDistance$, the algorithm cannot eliminate any object because the greatest lower bound that could possibly be attained is $(MaxDistance - MinDistance)$: if this difference is less than or equal to the minimum distance a best match could have, our cut-off procedure becomes useless (cf., Figure 3, step 6).[6] On the other hand, if one enlarges the difference, the algorithm should improve, as more objects farther away from the target than the current minimum distance can be generated, which may cause more objects to be eliminated at each stage.

Figure 5 confirms this speculation. Here, *PERFO* is plotted as a function of $\ln(MaxDistance/MinDistance)$ for different densities. The *MinDistance* is fixed

[6] An extremum situation is that all objects (including the target) are equally distant from each other, in which case the proposed algorithm degenerates to the brute-force method (linear search over all objects in the file).

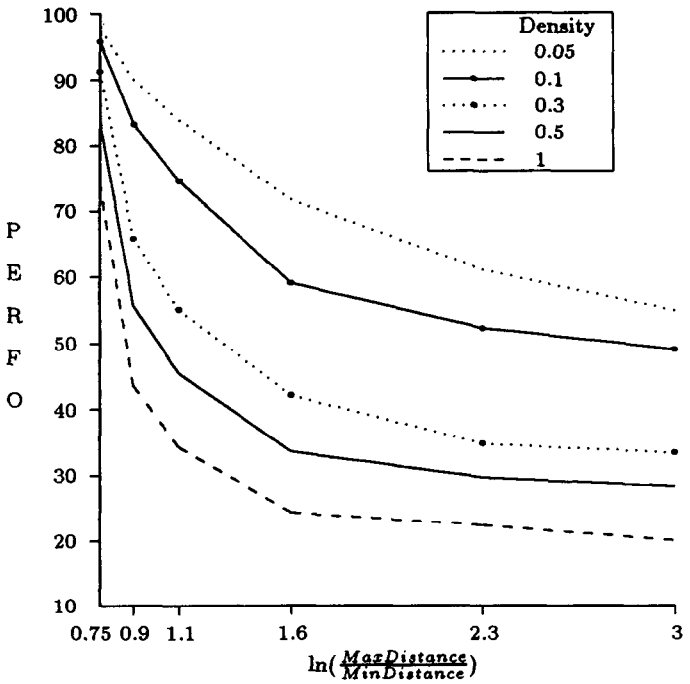


Fig. 5. Effect of distance ranges for uniformly distributed distances; file size = 150, minimum distance between objects (including the target) = 1000.

at 1000. We can see from the figure that when the value of *MinDistance* is large relative to *MaxDistance*, a small increase in the distance range improves *PERFO* considerably. The improvement slows down asymptotically.[7]

We also compared the number of distance computations our heuristic (for picking objects) required with the minimum number possible. That number was obtained by trying all possible permutations of distance calculations and finding the one that answered the query with the fewest possible of such calculations. The experiment was conducted using 30 files of size 20; the distances were drawn from the range [0, 10000]. It was found that our heuristic was within 20%, on the average, of the optimum. This result is encouraging, given the fact that we have no way of knowing a priori what the best distances to calculate are.

4.2 Nonuniformly Distributed Distances

To see the effect of nonuniform distribution for distances between objects, two extremum experiments were performed on maps with size 150 and densities 1, 0.5, and 0.01 (they represent complete, half complete, and very sparse maps, respectively).

[7] We repeated the same size and range experiments for Burkhard et al.'s algorithms and obtained similar results. This was unsurprising because all the algorithms essentially depend on the triangle inequality to eliminate distance calculations when searching a file.

In the first experiment, we considered files in which objects are far from one another, but one is very close to the target T. We generated a target–object distance value from the range [0, 100],[8] and then generated all other distances from the range [1000, 10000].[9] Our results show that the values of *PERFO*, in this case, are very low (1.3% for *Density* = 1, 3.4% for *Density* = 0.5, and 60.4% for *Density* = 0.01, respectively). In particular, for complete maps, at most two comparisons are needed to get the closest object. This is not surprising, since our heuristic for picking objects can always make the right choice after its first try.

In the second experiment, we considered files in which objects are close to one another, but far from T. We generated distances between file objects from the range [0, 1000], and then generated target–object distances from the range [10000, b]. Figure 6 shows how *PERFO* varies with b (the maximum distance from file objects to T).[10]

We see from the figure that *PERFO* improves dramatically as b increases. The reason is similar to that for the distance range experiments presented earlier (i.e., the larger the maximum distance to T, the more objects farther away from T than the current minimum distance there are, and hence the more objects that can be eliminated at each stage). Notice also that, under this circumstance, our algorithm suffers when objects in a file have approximately the same distance from T.

4.3 Results for Protein Data

To learn more about the performance of our algorithm in real applications, we have run it on a set of proteins. The data set was chosen, since protein comparison is expensive, as marked in Section 1. One hundred fifty-one proteins were randomly selected from the sequence database of Thinking Machines. Each protein has between 4 and 20 amino acids. (An amino acid is represented by a numerical or alphabetical character.) The interprotein distances were computed based on the *dayhoff* score metric [18].[11] (Unlike the data generated in Section 4.1, we found that the interprotein distances were distributed very unevenly. In the sample database, there are lots of small clusters in which proteins are close to one another. Clusters and all other nonclustered proteins are (roughly) equally distant from each other.)

[8] Here, "generated distances from the range [a, b]" actually means "used a random-number generator to produce distances, which were distributed uniformly over the range [a, b]." The distance maps were synthesized as described in Section 4.1.

[9] This type of distance distribution is analogous to what Perry and Willett described in [27, p. 61], where they considered the distributions of the similarity values for a set of documents and queries. They observed an extremely skewed distribution for the test collections: the great majority of the documents have a very small, or zero, similarity with the query. Few documents (less than 3% of the documents on the average) are close to the query (having similarity values greater than 0.2, measured by the Dice coefficient).

[10] Due to the triangle inequality, b in this case cannot be greater than 11000.

[11] The dayhoff score metric differs from, albeit is isomorphic to, a distance metric in the sense that the higher the score between two proteins, the closer they are. We used the following formula to compute the distance between two proteins based on their scores: $d(p_1, p_2) = c - s(p_1, p_2)$, where c is an empirical constant assuring that the difference satisfies the conditions of distance metrics, and $s(p_1, p_2)$ is the score between proteins p_1 and p_2.

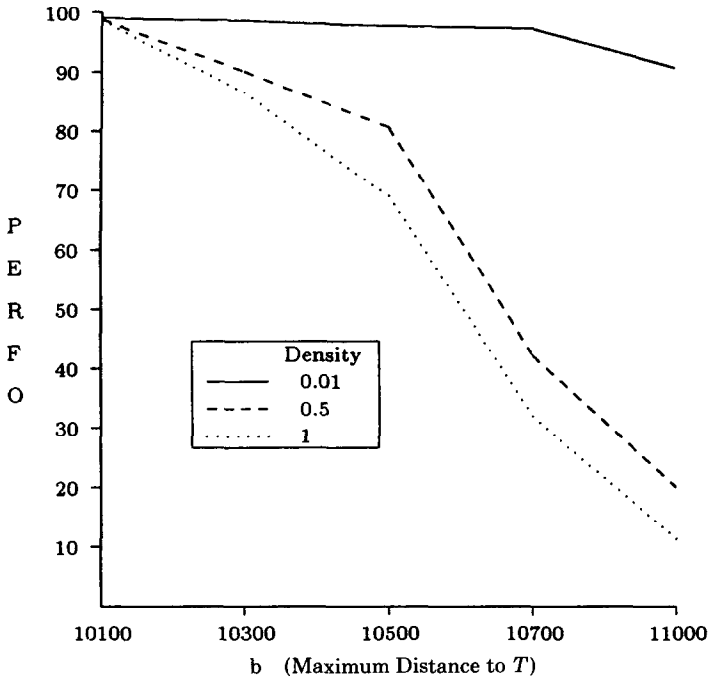


Fig. 6. Effect of varying maximum distance from file objects to T for a nonuniformly distributed distance case; file size = 150, minimum distance from file objects to T = 10000, distances between file objects were drawn from [0, 1000].

Our algorithm was run on these proteins thirty times, each time a randomly selected (distinct) protein was used as the target. Table II shows the means and standard deviations of the number of proteins compared for various densities.

Comparing Table II with Figure 4, the values of *PERFO* obtained from the proteins are higher (i.e., worse) than those from the generated data. Moreover, it was observed that proteins are eliminated rather unstably—in some runs no protein is eliminated; in others a large number of proteins are suddenly eliminated within some stage. The reasons are obvious in retrospect. If the target is a member of a cluster, our heuristic (for picking objects) can quickly locate best matches, yielding a very low *PERFO*. On the other hand, if the target does not belong to any cluster, our cut-off procedure becomes ineffective, which results in many proteins being computed.

4.4 Comparison to Previous Methods

We performed a number of experiments to compare the relative performance of our search algorithm to those proposed by Burkhard and Keller [5] and Shapiro [33], with the condition that the same distance information is given. The experiments were carried out using both uniformly distributed data and proteins. The data were made up as described in Sections 4.1 and 4.3. (In the following,

Table II. Statistics for Protein Data

	Density							
	1	0.9	0.7	0.5	0.3	0.1	0.05	0.01
Mean	95	98	106	111	125	138	143	147
Deviation	66	62	53	46	32	14	7	3
PERFO	63.3%	65.3%	70.7%	74.0%	83.3%	92.0%	95.3%	98.0%

Burkhard and Shapiro's algorithms are collectively referred to as BKS and ours is called SW.)

Figure 7a illustrates the behavior of these algorithms for various numbers of reference points. The reference points were chosen arbitrarily and, as suggested in [33], were kept outside clusters when running proteins. To better exploit the precomputed distances, instead of picking its first object randomly, SW was modified to first pick the reference points, and then pick objects with the least lower bound in subsequent stages (cf., Figure 3, step 3).

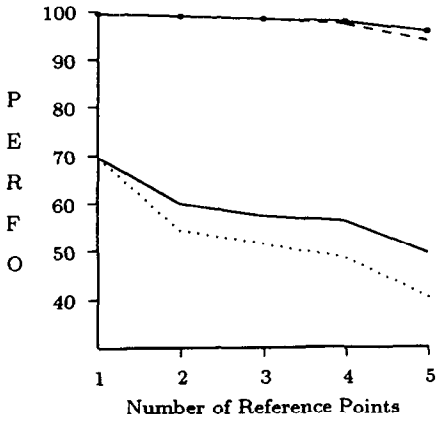
Recall that BKS picks objects with the smallest lower bound first, estimated on the basis of the first reference point, and eliminates objects O_k if there exists a reference point O^j such that $|d(O_k, O^j) - d(T, O^j)| > \xi$, whereas SW picks objects O_i with the smallest ADM$[n + 1, i]$ value first and eliminates O_k if ADM$[n + 1, k] > \xi$. Figure 7b shows the relative performance of the two heuristics (for picking objects) employed by each algorithm. To isolate the effect of cut-off procedures, the same BKS's cut-off criterion was used for both algorithms. Figure 7c shows the relative performance of the cut-off procedures employed by each algorithm, where the same BKS's heuristic was used.

These figures show that SW is better than BKS, and that the more reference points that are used (i.e., the more distances that are precomputed), the greater the improvement. Figure 7b shows that the SW's superiority is primarily due to a better heuristic for choosing objects at each stage. The different cut-off criteria for the two algorithms has only a minor influence on performance (Figure 7c). Notice that the *PERFO* is very high for both algorithms on the protein data, being close to that of the brute-force method. This happens because the density of precomputed distances we considered is low—lower than 0.1, even when five reference points are used.

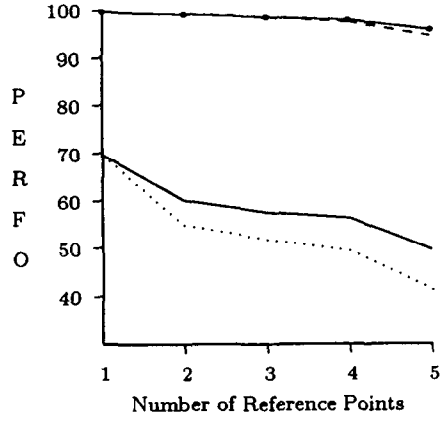
In practice, when distance calculation is cheap, using BKS may be advantageous, as it incurs little overhead. However, when distance calculation is extremely expensive, SW should have better overall performance.

4.5 Effect of Data Structures

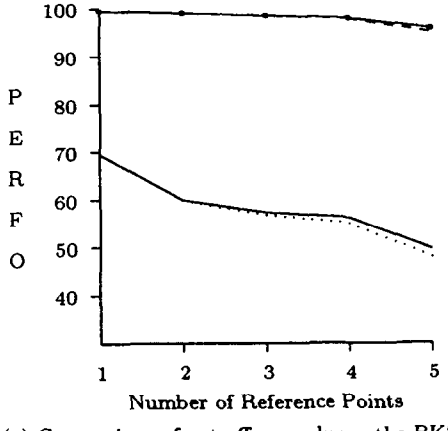
An interesting question arises naturally, based on Burkhard's work: Given a fixed number of distances, say, M, that one is allowed to calculate beforehand, what is the best set of distances (or edges) to choose? That is, what kinds of data structures for a file can yield best performance? We have tested our algorithms on some possible candidate structures with uniformly distributed data (similar



(a) Comparison of algorithms.



(b) Comparison of heuristics (for picking objects); the BKS's cut-off procedure was used.



(c) Comparison of cut-off procedures; the BKS's heuristic (for picking objects) was used.

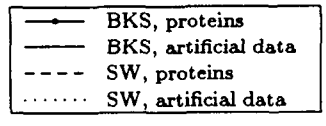


Fig. 7. Comparison of previous algorithms with ours, for both uniformly distributed data and proteins; file size = 150. For the generated data, distances between objects (including the target) were drawn from [0, 10000].

results were obtained for proteins, and are not shown here):

—*Stars*: As described in [5 and 33].

—*Bipartite graphs*: Objects in the file are evenly distributed into two disjoint groups, with those in group i being denoted as O_j^i, $i = 1, 2$, $j = 1, \ldots, Size/2$. Edges are allocated evenly and are constructed according to the following program:

```
for i := 0, 1, ... do begin
for j := 1 to Size/2 do
  compute d(O_j^1, O_{j+i}^2);
for j := 1 to Size/2 do
  compute d(O_j^2, O_{j+i}^1);
end;
```

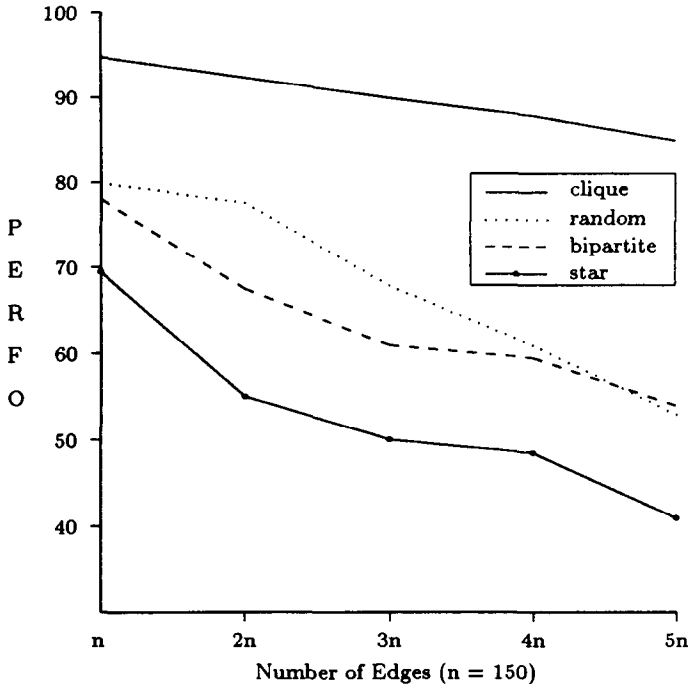


Fig. 8. Effect of data structures for uniformly distributed distances; file size = 150, distances between objects (including the target) were drawn from [0, 10000].

—*Cliques*: Arbitrarily choose k objects and compute distances between any two of them, with the constraint that $\binom{k}{2} < M$. Arbitrarily choose the remaining edges.

—*Random graphs*: Randomly choose M edges.

We first processed these structures by algorithm APPROXIMATE, and then ran our search algorithm based on the resulting maps. Our heuristic for picking objects was modified for cliques where objects in a clique are always examined prior to those outside the clique. (This is because we wanted to fully exploit the precomputed information.) Figure 8 illustrates the relative performance of these structures for various numbers of edges.

It is evident that stars outperform all the other structures, and are the data structure of choice. Cliques behave poorly. This is mainly due to the fact that there are no edges between objects outside a clique, and hence the distance bounds between them and the target are very low; consequently, few objects can be eliminated at each stage. It is also interesting to note that random graphs tend to be better than bipartite graphs as the number of edges increases.

5. CONCLUSIONS AND DISCUSSIONS

Burkhard and Keller [5], and later Shapiro [33], employed the triangle inequality to reduce the computational effort for best-match file searching. Their algorithms depended on having certain distances precomputed. In this paper, we proposed

an algorithm that can take advantage of any precomputed information. Further, our heuristic for choosing objects for comparison outperforms Burkhard et al.'s and shows a better improvement the more precomputed data is available. Simulation shows that it requires only 20% more distance calculations than the fewest possible for uniformly distributed distances.

To gain an insight into the behavior of these algorithms, we conducted extensive experiments and tested them in various settings. The results reveal that the performance of all the algorithms is not only dependent on the amount of precomputed data, but is strongly influenced by how distances distribute over a range and how large the range is. They suffer, for example, when distances between objects (including the target) are approximately the same.

Our results show that the multiple star topology of computed distances proposed by Burkhard et al. leads to good performance. We conjecture it is the topology to use if one is allowed to precompute a fixed number of distances. The reason is that every object is connected to every other by a set of length-two paths. This topological conjecture is left to theorists interested in average case complexity.

Our algorithm for approximating a distance map requires $O(n^3)$ time, where n is the size of the file. In practice, it may be infeasible to perform such approximation in a single run when files are very large. For this case, we suggest dividing the whole file into subfiles, applying our scheme to each to find the desired objects, and then comparing them to get the final result. We also note that setting up the complete distance matrix requires $O(tn^2)$, where t is the cost of a distance calculation. There is thus a trade-off between approximating the map as against setting up the complete distance matrix. Under the assumption that distance calculation is the dominant cost, maintaining an ADM is no doubt the method of choice. On the other hand, as t becomes smaller or n becomes larger, the latter option may become more attractive. (In the latter case the ADM, as well as algorithm APPROXIMATE, become redundant.)

The work reported here is based on a single processor architecture. Recently, there have been attempts to extend techniques for informational retrieval to multiprocessor architectures [16, 26, 28, 37]. Stewart and Willett [36], for example, describe three techniques that allow parallel searching of $k - d$ trees. Their results support the use of multiprocessor systems for searching applications in information retrieval. In [40], we investigated the possibility of parallelizing our search algorithm. We assumed that a set of processors would work by comparing objects with the target and eliminating distance calculations concurrently. We found that to use these processors effectively, one should delay multiprocessing to later stages of a search. This result agrees with our experimental results that most objects are eliminated in the first several stages of a search. If many processors execute at these stages, most compute distances to objects that the serial algorithm would eliminate. In the later stages the serial algorithm eliminates fewer objects, so parallelism is more useful.[12]

[12] The question of why most eliminations occur in the early stages of the algorithm may also be interesting to average case theorists.

ACKNOWLEDGMENTS

We are grateful to the anonymous referees and to Kaizhong Zhang for their useful comments on the preliminary versions of this paper. We wish to thank Jill Mesirov of Thinking Machines, who provided the proteins and their pairwise distances used for performing some of the experiments. Thanks also to Richard Cole, Rakesh Agrawal, Mitra Basu and Deepak Sherlekar for their help in doing this work.

REFERENCES

1. AGRAWAL, R., AND JAGADISH, H. V. Efficient search in very large databases. In *Proceedings of the 14th International Conference on Very Large Data Bases* (1988), 407–418.
2. AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. D. *Data Structures and Algorithms.* Addison-Wesley, Reading, Mass., 1983.
3. ANDERSON, D. P. Techniques for reducing pen plotting time. *ACM Trans. Gr. 2*, 3 (July 1983), 197–212.
4. BENTLEY, J. L., WEIDE, B. W., AND YAO, A. C. Optimal expected-time algorithms for closest point problems. *ACM Trans. Math. Softw. 6*, 4 (Dec. 1980), 563–580.
5. BURKHARD, W. A., AND KELLER, R. M. Some approaches to best-match file searching. *Commun. ACM 16*, 4 (Apr. 1973), 230–236.
6. CLAUS, V., EHRIG, M., AND ROZENBERG, G. *Graph-Grammars and Their Application to Computer Science and Biology.* Springer, New York, 1979.
7. DU, H. C., AND LEE, R. C. T. Symbolic Gray code as a multikey hashing function. *IEEE Trans. Pattern Anal. Mach. Intell. 2*, 1 (Jan. 1980), 83–90.
8. DUDA, R. O., AND HART, P. E. *Pattern Classification and Scene Analysis.* Wiley, New York, 1973.
9. EASTMAN, C. M., AND WEISS, S. F. A tree algorithm for nearest neighbor searching in document retrieval systems. In *Proceedings of the ACM SIGIR International Conference on Information Storage and Retrieval* (1978). ACM, New York, 1978, 131–149.
10. EASTMAN, C. M., AND WEISS, S. F. Tree structures for high dimensionality nearest neighbor searching. *Inf. Syst. 7*, 2 (1982), 115–122.
11. EASTMAN, C. M., AND ZEMANKOVA, M. Partially specified nearest neighbor searches using k − d trees. *Inf. Process. Lett. 15*, 2 (1982), 53–56.
12. FEUSTEL, C. D., AND SHAPIRO, L. G. The nearest neighbor problem in an abstract metric space. *Pattern Recognition Lett. 1*, 2 (1982), 125–128.
13. FLOYD, R. W. Algorithm 97: Shortest path. *Commun. ACM 5*, 6 (June 1962), 345.
14. FRIEDMAN, J. H., BENTLEY, J. L., AND FINKEL, R. A. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw. 3*, 3 (Sept. 1977), 209–226.
15. FUKUNAGA, K., AND NARENDRA, P. M. A branch and bound algorithm for computing k-nearest neighbors. *IEEE Trans. Comput. 24*, 7 (July 1975), 750–753.
16. HOLLAAR, L. A. The Utah text retrieval project. *Inf. Technol. Res. Dev. 2* (1983), 155–167.
17. ITO, T., AND KIZAWA, M. Hierarchical file organization and its application to similar-string matching. *ACM Trans. Database Syst. 8*, 3 (Sept. 1983), 410–433.
18. LANDER, E., MESIROV, J. P., AND WASHINGTON, T. Protein sequence comparison on a data parallel computer. In *Proceedings of the IEEE 1988 International Conference on Parallel Processing* (Aug. 1988). IEEE, New York, 1988, 257–263.
19. LIPMAN, D. J., AND PEARSON, W. R. Rapid and sensitive protein similarity searches. *Science 227* (1985), 1435–1441.
20. LUCARELLA, D. A document retrieval system based on nearest neighbor searching. *J. Inf. Sci. 14* (1988), 25–33.
21. MOHAN, K. C., AND WILLETT, P. Nearest neighbor searching in serial files using text signatures. *J. Inf. Sci. 11* (1985), 31–39.
22. MURTAGH, F. A very fast exact nearest neighbor algorithm for use in information retrieval. *Inf. Technol. Res. Dev. 1* (1982), 275–283.

23. MURTAGH, F. Expected-time complexity results for hierarchic clustering algorithms which use cluster centers. *Inf. Process. Lett. 16*, 5 (June 1983), 237–241.
24. MURTAGH, F. A survey of recent advances in hierarchical clustering algorithms. *IEEE Computer 26*, 4 (1983), 354–359.
25. MURTAGH, F. Multidimensional clustering algorithms. In *Lectures in Computational Statistics*, J. M. Chambers, J. Gordesch, A. Klas, L. Lebart, and P. P. Sint, Eds., Physica-Verlag, Vienna, 1985.
26. PAIGE, R. C., AND KRUSKAL, C. P. Parallel algorithms for shortest path problems. In *Proceedings of the IEEE 1985 International Conference on Parallel Processing* (1985). IEEE, New York, 1985, 14–19.
27. PERRY, S. A., AND WILLETT, P. A review of the use of inverted files for best match searching in information retrieval systems. *J. Inf. Sci. 6* (1983), 59–66.
28. POGUE, C. A., AND WILLETT, P. An evaluation of document retrieval from serial files using the ICL Distributed Array Processor. *Online Rev. 8* (1984), 569–584.
29. ROHLF, F. J. A probabilistic minimum spanning tree algorithm. *Inf. Process. Lett. 7* (1978), 44–48.
30. SALTON, G., AND MCGILL, M. J. *Introduction to Modern Information Retrieval.* McGraw-Hill, New York, 1983.
31. SHAMOS, M. I., AND HOEY, D. Closest-point problems. In *Proceedings of the 16th IEEE Symposium on Foundations of Computer Science* (Oct. 1975). IEEE, New York, 1975, 151–162.
32. SHAPIRO, B. A., AND ZHANG, K. Comparing multiple RNA secondary structures using tree comparisons. Manuscript, Division of Cancer Biology and Diagnosis, NIH, Frederick, Md., 1989.
33. SHAPIRO, M. The choice of reference points in best-match file searching. *Commun. ACM 20*, 5 (May 1977), 339–343.
34. SHASHA, D., AND WANG, T.-L. Optimal best-match retrieval. Tech. Rep. TR 480, Courant Institute of Mathematical Sciences, New York Univ., New York, Dec. 1989.
35. SMEATON, A. F., AND VAN RIJSBERGEN, C. J. The nearest neighbor problem in information retrieval: An algorithm using upperbounds. *ACM SIGIR Forum 16* (1981), 83–87.
36. STEWART, M., AND WILLETT, P. Nearest neighbor searching in binary search trees: Simulation of a multiprocessor system. *J. Doc. 43*, 2 (June 1987), 93–111.
37. TESKEY, F. N. Novel computer architectures for data storage and retrieval. Rep. 5845, British Library Research and Development Dept., London, 1986.
38. VAN RIJSBERGEN, C. J. The best-match problem in document retrieval. *Commun. ACM 17*, 11 (Nov. 1974), 648–649.
39. VAN RIJSBERGEN, C. J. *Information Retrieval.* 2nd ed. Butterworths, London, 1979.
40. WANG, T.-L., AND SHASHA, D. Query processing for distance metrics. In *Proceedings of the 16th International Conference on Very Large Data Bases* (Brisbane, Australia, Aug. 1990).
41. WARSHALL, S. A theorem on boolean matrices. *J. ACM 9*, 1 (Jan. 1962), 11–12.
42. YU, C. T., LUK, W. S., AND SIU, M. K. On the estimation of the number of desired records with respect to a given query. *ACM Trans. Database Syst. 3*, 1 (Mar. 1978), 41–56.

Received February 1990; revised July 1990; accepted August 1990