

Exact and Approximate Reverse Nearest Neighbor Search for Multimedia Data

Jessica Lin David Etter
jessica@ise.gmu.edu detter@gmu.edu

David DeBarr
Dave.DeBarr@microsoft.com

George Mason University
 Fairfax, VA 22030

Microsoft Corporation
 Redmond, WA 98052

Abstract

Reverse nearest neighbor queries are useful in identifying objects that are of significant influence or importance. Existing methods either rely on pre-computation of nearest neighbor distances, do not scale well with high dimensionality, or do not produce exact solutions. In this work we motivate and investigate the problem of reverse nearest neighbor search on high dimensional, multimedia data. We propose exact and approximate algorithms that do not require pre-computation of nearest neighbor distances, and can potentially prune off most of the search space. We demonstrate the utility of reverse nearest neighbor search by showing how it can help improve the classification accuracy.

1 Introduction

The nearest neighbor (NN) search [1, 2, 3, 7, 9, 14] has long been accepted as one of the classic data mining methods, and its role in classification and similarity search is well documented. Given a query object, nearest neighbor search returns the object in the database that is the most similar to the query object. Similarly, the k nearest neighbor search [15], or the k -NN search, returns the k most similar objects to the query. NN and k -NN search problems have applications in many disciplines: information retrieval (find the most similar website to the query website), GIS (find the closest hospital to a certain location), etc.

Contrary to nearest neighbor search, less considered is the related but much more computationally complicated problem of reverse nearest neighbor (RNN) search [8, 16, 17, 18, 20]. Given a query object, reverse nearest neighbor search finds all objects in the database whose nearest neighbors are the query object. Note that since the relation of NN is not symmetric, the NN of a query object might differ from its RNN(s). Figure 1 illustrates this idea. Object B being the nearest neighbor of object A does not automatically make it the reverse nearest neighbor of A , since A is not the nearest neighbor of B .

The problem of reverse nearest neighbor search has many practical applications that can be applied to areas such as business impact analysis and customer profile analysis. An example of business impact analysis using

RNN search can be found in the selection of a location for a new supermarket. In the decision process of where to locate a new supermarket, we may wish to evaluate the number of potential customers who would find this store to be the closest supermarket to their homes.

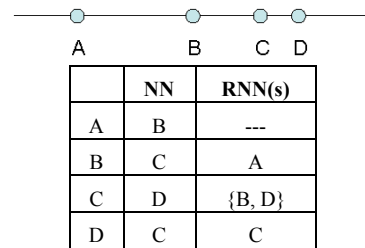


Figure 1. B is the nearest neighbor of A ; therefore, the RNN of B contains A . However, the RNN of A does not contain B since A is not the nearest neighbor of B .

Another example can be found in the area of customer profiling. Emerging technologies on the internet have allowed businesses to push information such as news articles or advertisements to their customer's browser. An effective marketing strategy will attempt to filter this data, so that only those of items of interest to the customer will be pushed. A customer who is flooded with information that is not of interest to them will soon stop viewing the data or stop using the service. In this example, an RNN search can be used to identify customer profiles that find the information closest to their liking.

It is interesting to note that the examples above present a slight variation of the RNN problem known as the *Bichromatic RNN* [17]. In the Bichromatic problem, we consider two classes in our set of objects S . The set has been subdivided into supermarkets and potential customers for the first example, and information/products and internet viewers for the second example. We are interested in distances between objects from different classes.

On the other hand, the *Monochromatic RNN* search [17] is one in which all objects in the database are treated the same and we are interested in similarities between them. For example, in the medical domain, doctors might wish to identify all patients who exhibit certain medical

conditions such as heartbeat patterns similar to a specific patient or a specific prototypical heartbeat pattern.

In all the examples described above, it is important to note the distinction between (k -)NN queries, range queries, and RNN queries. If we wish to find the objects similar to our query point, often our search needs to expand beyond the simple notion of nearest neighbor or k -nearest neighbors. For example, we might be interested in objects that may not be in the k -NN search neighborhood, but find our query to be closest to them. Furthermore, in all cases above, it's difficult to specify a range for similarity (as in the case for range queries), or a desired number of matching objects (i.e. k for k -NN) to be returned. The RNN queries will provide exactly the results to the problems described above.

In addition, the number of RNNs for an object can potentially capture the notion of its *influence* or *importance*. For the supermarket example, the location with the highest RNN count (i.e. the highest number of potential customers) can be said to be of significant influence. This novel notion of the influence of a data object in the database was introduced by Korn and Muthukrishnan [8]. The influence set of a data object Q is the set of objects in the database that are likely to be affected by Q . To solve the problem, the authors introduced the notion of RNN. The RNNs of a query object Q is the set of objects whose nearest neighbors are Q . If we consider the nearest neighbor of any object to be of certain significance with respect to the object, then intuitively, Q has significant influence on its RNNs, since the objects in this set all consider Q to be closest to them.

1.1 RNN on multimedia data

With the rapid advancement of technology today, the past decade has seen an increasing interest in generating and mining multimedia datasets. This type of data includes time series, images, video sequences, audio, etc. Since they are typically very high dimensional, many mining algorithms or dimensionality reduction techniques are proposed to cope with the high dimensionality. To date, existing algorithms for exact RNN search work for only low-dimensional data. Some algorithms are proposed for arbitrary dimensionality, but they rely on indices such as R-tree or its variants, which do not scale well for high dimensionality [1]. Our goal is thus to design algorithms that can scale to high dimensionality.

In this work we focus on multimedia sequence data to demonstrate the effectiveness of our algorithms. Apart from its ubiquity and simplicity, multimedia data such as images can potentially be represented as sequences of values as well. For example, one can construct a color histogram for each of the RGB components for color images, and concatenate the histograms together to form one series, which can then be treated the same as conventional time series data [12]. Figure 2 shows such an

example. In Figure 3, a shape is converted into a sequence of values [10]. Recent studies have shown that such representations produce promising results for various data mining tasks [10, 12, 19]. In fact, some of the time series datasets we use for experimental evaluation are shape-converted sequence data. In addition, even without the conversion to time series, the algorithms proposed here can potentially be adapted for many signal-typed, continuous data, as they share some common characteristics, such as high autocorrelation, with time series. For the rest of the paper, we will use the terms sequences and time series interchangeably.

Thus, we motivate, and show the utility of RNN search on time series, and propose several heuristics for finding exact solutions. We also propose heuristics that find approximate solutions, which are useful when efficiency rather than exactness is of critical concern. In addition, we demonstrate how R(k)NN search can be applied for anomaly detection and thus improve classification accuracy.

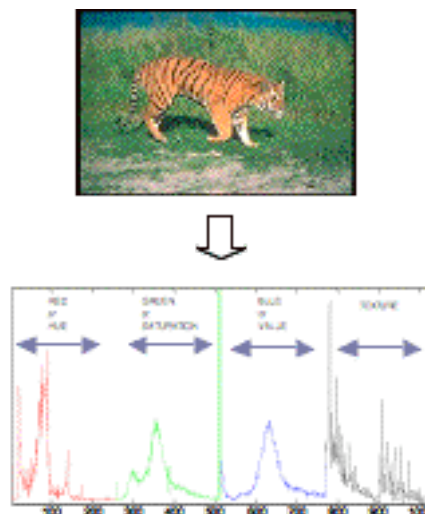


Figure 2. The RGB histograms, as well as the texture information, from the image are extracted and concatenated together to form a long series.

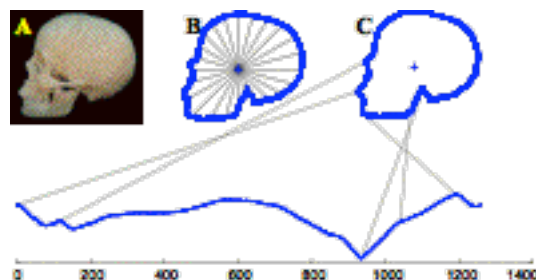


Figure 3. A shape is converted to time series. The y-axis is the distance from the center of mass to the outline of the shape

The paper is organized as follows. Section 2 provides background information on time series, as well as related work on reverse nearest neighbor search. In Section 3 we formally define the RNN search problem for time series and describe the brute-force algorithm to find RNNs given a query. In Section 4, we describe four different heuristics to help speed up the search process. In Section 5, we show results of experimental evaluation on our heuristics in terms of both accuracy and efficiency. Section 6 concludes and offers suggestions for future work.

2 Related Work and Background

The earlier work on RNN focuses mostly on 2-D objects. The pioneering work on RNN search [8] proposes algorithms that are based on pre-computed nearest neighbor distances for all objects. Such pre-computation is inefficient and is expensive for dynamic databases. Stanoi et al [17] proposed an algorithm that does not require pre-computation of NN distances; however, their approach does not work for dimensionalities higher than two.

Tao et al [18] developed a solution for arbitrary dimensionality. The technique, TPL, cuts the data space into two half-planes by drawing a perpendicular bisector between the query object Q and an arbitrary data object P . The intuition is that all the objects that reside on the opposite of Q cannot be the candidates for Q 's RNN since they are closer to P than they are to Q . These objects can thus be eliminated from the search without computing the actual nearest neighbor distances. This simple and elegant approach works well for low dimensionality. However, in high dimensionality, truncating the search space by drawing the hyper-curve bisectors can be computationally expensive [16]. The same authors further proposed another algorithm that works on higher dimensionality. However, since it utilizes R-trees, the dimensionality suitable for the algorithms is limited by that of R-trees. Furthermore, the highest dimensionality shown in their experiments is 6 [18]. An approximate version of the algorithm is available; however, it works only with Euclidean distance [18].

Singh et al [16] proposed an approximate algorithm for high dimensional data, based on the relationship between k -NN and RNN; however, it cannot guarantee exact results. In addition, although the claim is to find RNNs on high-dimensional data, the algorithm still relies on pre-processing the data with Singular Value Decomposition (SVD) to reduce the dimensionality. This approach is clearly infeasible for large or dynamic data, as SVD is known to be computationally expensive.

2.1 Notation

For concreteness, we begin with a definition of time series:

Definition 1. *Time Series:* A time series $T = t_1, \dots, t_m$ is an ordered set of m real-valued variables.

Some distance measure $Dist(C, M)$ needs to be defined in order to determine the similarity between time series objects.

Definition 2. *Distance:* $Dist$ is a function that has C and M as inputs and returns a nonnegative value R , which is said to be the distance from M to C . We require that the function $Dist$ be symmetric, that is, $Dist(C, M) = Dist(M, C)$.

Although there are dozens of distance measures proposed for time series in the literature, Euclidean distance has been shown to outperform many of the more complicated distance measures [5]. Therefore, we will use Euclidean distance as our choice of distance measure in this paper.

Definition 3. *Euclidean Distance:* Given two sequences Q and C of length n , the Euclidean distance between them is defined as:

$$Dist(Q, C) = \sqrt{\sum_{i=1}^n (q_i - c_i)^2}$$

Each sequence object is normalized to have mean zero and a standard deviation of one before calling the distance function, because it is well understood that in virtually all settings, it is meaningless to compare time series with different offsets and amplitudes [5].

3 Finding RNN in Time Series

As mentioned, the existing work on RNN fails to address high dimensional data such as sequences, since the algorithms proposed typically scale poorly with increasing dimensionality. Here we consider the RNN search problem for time series data, and propose an efficient algorithm on the discretized representation of data. We further propose an approximate version of the algorithm that improves the efficiency even more.

We begin by formally defining the RNN search problem for time series.

There are two types of RNN queries: monochromatic and bichromatic.

Definition 2. *monochromatic-RNN:* Given a time series database TT and a query time series Q , $monoRNN(TT, Q)$ is the set of time series objects B in TT that have Q as their nearest neighbors.

Definition 3. *bichromatic-RNN:* Given two time series databases TT and SS , and a query time series $Q \in SS$, $biRNN(TT, SS, Q)$ is the set of time series B in TT that have Q as their nearest neighbor from SS .

For simplicity and clarity, for the rest of the paper we will refer to RNN as in the monochromatic case. Extension to the bichromatic case is straight-forward, thus omitted from our description.

In addition, as with the nearest-neighbor search problem, RNN search can be extended to Rk NN search. Applying our heuristics to Rk NN is straight-forward. We

will conclude our discussion of $RkNN$ with the definition below, and will revisit $RkNN$ in the experimental section.

Definition 4. $RkNN$: Given a time series database TT and a query time series Q , $RkNN(TT, Q, k)$ is the set of time series in TT who have Q as one of their k -nearest-neighbors.

Let's first consider the simplest scenario for the RNN search problem. Similar to the examples described in the previous section, a query object Q (e.g. the heartbeats of a patient) is given, and the goal is to retrieve all objects that consider the query object their nearest neighbor. Intuitively, the definition of RNN implies that in order to find the RNN of Q , one may need to find the nearest neighbor for every object in the dataset.

The brute-force algorithm for finding RNN, as well as the earlier algorithms proposed that require pre-computation, work by finding the nearest neighbor for each object and then determining which objects have Q as their nearest neighbors. The computation of all nearest neighbors, whether they be stored on hard disk (i.e. precomputed), or be computed on the fly, require double nested loops, where the outer loop considers each candidate object C in the dataset, and the inner loop is a linear scan to identify the candidate's nearest neighbor. The brute-force algorithm is easy to implement and produces exact results. However, its quadratic time complexity makes this approach impractical for large and/or dynamic datasets.

Once the nearest neighbors for all objects are identified, some of the existing approaches then record the nearest neighbor distances, and use index structures to determine if the query is closer to an object than the object's previously-computed nearest neighbor is. If the database is large, frequently updated, or streaming (which are all typical cases for time series), then a more efficient algorithm that does not need computation of all pair-wise distances is desirable.

Fortunately, the following observations offer hope for improving the algorithm's running time. Recall that the goal is to identify the objects whose distances to Q are smaller than their distances to all other objects in the dataset. This implies that we might not need to know the actual nearest neighbor or the nearest neighbor distance for every candidate. The only piece of information that is crucial is whether or not a given object in the dataset *could* be the candidate for $RNN(TT, Q)$.

Consider the following scenario. Suppose we start with the candidate object C_i from TT , and $Dist(C_i, Q)$ is 10. We would like to find the nearest neighbor for C_i so we can determine if C_i is an RNN for Q . In the process of identifying the nearest neighbor for C_i , suppose the first object we examine is C_j , and suppose we find that $Dist(C_i, C_j)$ is 2. At this point, we know that C_i could not be the candidate for $RNN(TT, Q)$, since its nearest neighbor is obviously not Q . We can therefore safely abandon the rest

of the search for C_i 's nearest neighbor. In other words, when we consider a candidate C_i in TT , we don't actually need to find its true nearest neighbor. As soon as we find any object that has a smaller distance to C_i than $Dist(C_i, Q)$, we can abandon the search process for C_i , knowing that it could not be in $RNN(TT, Q)$. The steps are outlined in Table 1.

Table 1. Outline for the improved RNN algorithm

1	Insert all objects in TT to the Candidate List CL . Initialize the current candidate, C_i , to be the first item in CL .
2	Compute $Dist(C_i, Q)$.
3	Start the nearest neighbor search for C_i by computing $Dist(C_i, C_j)$ for all $j \neq i$, until we encounter $Dist(C_i, C_j) < Dist(C_i, Q)$ – in which case, stop the search for C_i . If the nearest neighbor search is run to completion for C_i , insert C_i to the set $RNN(TT, Q)$. Remove C_i from CL .
4	Go back to Step 1 and repeat the process until CL is empty.

The idea of filtering out candidates that could not be the solution is not new. In fact, it is the basis for many of the existing approaches [16, 17]. However, since their techniques do not scale well with high dimensionality, here we offer a different approach to determine which candidates should be pruned off.

Clearly, for each candidate considered, the most time-consuming step is Step 3. In addition, the utility of the optimization depends on the number of distances we need to compute before we can call off the nearest neighbor search for C_i : the earlier we examine an object that has a smaller distance to C_i than $Dist(C_i, Q)$, the earlier we can abandon the search process for the current candidate. Simply stated, our goal is thus the following: if there is a reason that a candidate could not be in the answer set, we would like to detect that reason as soon as possible.

Note that the order in which the candidates are examined (i.e. the objects in CL) does not matter, as each search is independent. It is the ordering of objects when searching for C_i 's nearest neighbor, as in Step 3, that we should try to optimize. However, for the candidates in CL , we can take advantage of the already-computed distances so far, and continue the search process with the objects previously considered.

Continuing from the example described above, after we abandon the nearest neighbor search for C_i , we can go on and compute $Dist(C_j, Q)$ since we already know $Dist(C_i, C_j)$. If $Dist(C_i, C_j) < Dist(C_j, Q)$, then we can abandon the search for C_j as well, otherwise the search continues.

Our task now is narrowed to predicting the likely near-neighbors for a given candidate, and designing a heuristic that determines the ordering of objects examined for Step 3 such that we can abandon the searches for non-RNNs as soon as possible. Note that the optimal ordering for each candidate C_i completely depends on its relations

with other objects in the dataset; therefore, such heuristic will need to be applied to each candidate to determine the optimal ordering.

3.1 Best-case scenario

In the best case scenario, the number of distance computations we need to make is approximately

$$\begin{aligned} \# \text{ dist_calls} &= |RNN(TT, Q)| \times |TT| + 2 \times |TT| \\ &= (2 + |RNN(TT, Q)|) \times |TT| \end{aligned}$$

where $|RNN(TT, Q)|$ is the number of RNNs for Q , and $|TT|$ is the number of objects in the dataset. Since all members in $RNN(TT, Q)$ have Q as their nearest neighbor, it's inevitable that for these objects, their distances to *all* other objects in the dataset need to be computed in order to conclude that their distances to Q is indeed the minimum. Therefore, the $|RNN(TT, Q)| \times |TT|$ distance calls are necessary. For the rest of the candidates, the best case occurs when the first object we examine in Step 3 is an object closer to the candidate than the query. In this case, we can ensure that at most two distance computations are needed before abandoning the search (i.e. the first one computes $Dist(C_i, Q)$; the second one, if not already computed previously, computes a distance smaller than $Dist(C_i, Q)$, thus allowing early abandonment of search). Note that this is only an approximation, for simplicity, as the computations for $RNN(TT, Q)$ are double-counted.

We can also record the distances computed so far to avoid redundant computation, but such optimization has limited improvement if the number of RNNs for Q is small relative to the number of objects in the dataset – as we can assume to be the case for most real-life datasets.

The time complexity for the best case scenario is thus $\Omega(rN)$, where r is the number of RNNs, and N is the size of the dataset. Since we expect that $r \ll N$, the best-case scenario for the ordered search is by far more efficient than the $O(N^2)$ complexity of the brute-force approach.

In the next section we describe several heuristics that approximate the optimal ordering without having any information about the actual distances.

4 Approximating the Optimal Ordering

The time series research community has significant experience in dealing with high dimensional data and has introduced numerous dimensionality reduction techniques. In this work, we will utilize Symbolic Aggregate Approximation (SAX) [11] to reduce the dimensionality of our data. This method has been adopted by numerous researchers and has proven to be a very effective technique, and the only symbolic one that provides a lower bounding distance measure. Our review of this technique is brief and we refer the interested reader to [11] for expanded analysis.

4.1 Symbolic Aggregate approximation (SAX)

Given a time series of length n , SAX produces a lower dimensional representation of a time series by transforming the original data into symbolic words. Two parameters are used to specify the size of the alphabet to use (i.e. α) and the size of the words to produce (i.e. w). The algorithm begins by using a normalized version of the data and creating a Piecewise Aggregate Approximation (PAA). PAA reduces the dimensionality of a time series by transforming the original representation into a user defined number (i.e. w , typically $w \ll n$) of equal segments. The segment values are determined by calculating the mean of the data points in that segment. The PAA values are then transformed into symbols by using a breakpoint table based on a Gaussian distribution. In [4] the authors found that normalized time series subsequences had a highly Gaussian distribution. A symbolic transformation table could be created by defining breakpoints that would result in regions of equal-probability on the Gaussian distribution. These breakpoints may be determined by looking them up in a statistical table. For example, Table 2 gives the breakpoints for values of α from 3 to 5.

Table 2. A lookup table that contains the breakpoints that divides a Gaussian distribution into an arbitrary number (from 3 to 5) of equiprobable regions.

β_i	α	3	4	5
β_1		-0.43	-0.67	-0.84
β_2		0.43	0	-0.25
β_3			0.67	0.25
β_4				0.84

Figure 4 summarizes how a time series is converted to PAA and then symbols.

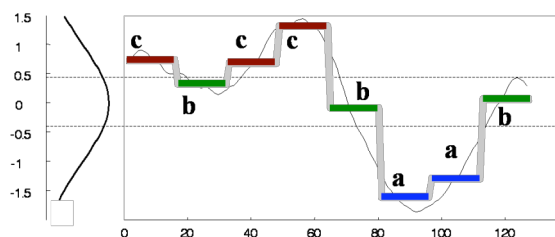


Figure 4. Example of SAX for a time series. The time series above is transformed to the string *cbccbaab*, and the dimensionality is reduced from 128 to 8.

4.2 Ordered search for RNN

We will first describe the basic heuristic that helps identify candidates that are closer to other objects than they are to the query. Then we will describe how we can further prune off the search space by considering only a subset of candidates that can potentially be in the solution set.

H1: same string heuristic

Recall that our goal is to identify objects whose nearest neighbors is the query object. This means these objects are closer to the query than they are to any other objects in the database. Therefore, if we see that the candidate being examined has a smaller distance to another object than its distance to the query, then we can conclude that this candidate could not be the RNN for the query. Our goal is thus to encounter *any* object close to the current candidate as early as possible. To predict which objects are close to the given candidate, we can look at their SAX representation. As shown in Figure 4, the shape of the given time series is approximately preserved by its symbolic representation. Therefore, we can expect that the time series encoded into the same SAX representation are likely to be similar to each other. Considering those with the same symbolic representation first seems a reasonable starting point. Among this subset, whose size is expected to be much smaller than the original dataset, we can potentially find one object to which the candidate C_i is closer than it is to the query, thus helping us eliminate C_i from the search.

What we need to consider now, then, is to efficiently build the subset in which all the time series have the same string representation as the candidate, for each candidate examined. One obvious approach to identify such time series is to sequentially scan through the entire string representations of time series. However, having a linear scan for each object in the dataset clearly is not an efficient solution. Fortunately, the symbolic nature of SAX allows hashing, for which constant-time look-up can be expected.

We begin by creating two data structures to support our heuristics. As a pre-processing step, each time series in TT is converted to a SAX word, which is stored in an array along with a pointer to the object. Note that the conversion/dimensionality reduction for each time series is independent of other objects in the dataset. Therefore, insertions of new data points would not require re-processing of the entire dataset.

Once we have this list of SAX words, we construct a hash table. Each bucket in the hash table represents a word and contains a linked-list index of all word occurrences that map to the corresponding string. The address of a SAX word is computed by using the following hash function:

$$h(C, w, \alpha) = \sum_{i=1}^w (\text{ord}(\hat{c}_i) - 1) \times \alpha^{w-i}$$

where $\text{ord}(\hat{c}_i)$ is the ordinal value of \hat{c}_i , i.e. $\text{ord}(a) = 1$, $\text{ord}(b) = 2$, and so forth; w is the word size; and α is the alphabet size. For example, for the string caa , where $w = 3$, $\alpha = 3$,

$$h(caa, 3, 3) = 2 \times 3^2 + 0 \times 3^1 + 0 \times 3^0 = 18$$

This hash function assigns each SAX word a unique address ranging from 0 to $\alpha^w - 1$, hence guarantees minimal perfect hashing.

The size of the hash table is independent of the size of the dataset – it depends only on the two parameters of SAX: w and α . According to [11], it's typically sufficient for α to take on small values such as 3 or 4; and while w is usually data-dependent, oftentimes we can expect it to remain reasonably small as well for the following reasons. First, it is generally not very meaningful to compare time series that are too long – for long time series, we are typically only interested in their local structures extracted via sliding windows. In addition, [11] has shown that SAX approximation produces competitive results comparing to other dimensionality reduction techniques such as DFT and DWT, or even the raw data.

That being said, despite the optimality of the suitable choice of w , the memory consumption for creating an empty hash table is considerably small and negligible, compared to the typically massive amount of time series data. For example, for $w = 10$ and $\alpha = 3$, the table size is $3^{10} = 59,049$. The memory consumption for creating an empty hash table of this size is less than 1 MB. The size of the resulting hash table, with subsequence indices hashed to the buckets of the corresponding SAX strings, is linear to the size of the time series datasets.

Once the hash table is constructed, we can approximate the optimal ordering for the current candidate's nearest neighbor search as follows. For each candidate C_i being considered, we check in the SAX words array for its SAX word, and since our purpose is to identify the objects that are likely to be similar to C_i , we then check the hash table for the corresponding bucket, and retrieve the list of objects that are encoded to the same SAX string. This list of objects will be examined prior to the rest of the objects which are then ordered sequentially. Hopefully among these *preferred* objects, we will find one that is closer to C_i than the query is to C_i . This process is repeated for each candidate examined. This simple heuristic is shown to work well and reduce the search space by a large amount.

H2: Zero MINDIST heuristic

One of the advantages of SAX that makes it more desirable than all the other symbolic representation proposed in literature is that it provides lower-bound distance measures. As shown in [11], given a lookup table for the breakpoints as in Table 3, the lower-bounding distance for the Euclidean distance, MINDIST, between two strings S_1 and S_2 can be determined as follows:

$$MINDIST(S_1, S_2) = \sqrt{\frac{n}{w}} \sqrt{\sum_{i=1}^w (dist(S_{1i}, S_{2i}))^2}$$

where

$$dist(r, c) = \begin{cases} 0, & \text{if } |r - c| \leq 1 \\ \beta_{\max(r,c)-1} - \beta_{\min(r,c)}, & \text{otherwise} \end{cases}$$

and $\beta_{\max(r,c)-1} - \beta_{\min(r,c)}$ is the difference in breakpoints for the regions associated with the alphabetic symbols.

Table 3. Lookup table for alphabet distances

	a	b	c
a	0	0	0.86
b	0	0	0
c	0.86	0	0

Simply speaking, the minimum distance between two alphabets is zero if they are the same, or neighboring alphabets (e.g. $dist(a, a) = 0$, $dist(a, b) = 0$). Otherwise, their distance is the height of the region between them. For an example alphabet of size 3, Table 3 shows that $dist(a, c) = 0.86$ (cf. Figure 4).

The heuristic described in the previous section looks for time series objects that are encoded to the same string. It might be the case that a given candidate has a string representation not shared by many others. In this case, we might exhaust this preferred, same-string list pretty fast. When this happens, instead of sequentially scanning the rest of the remaining objects, we consider a simple alternative.

As Figure 4 suggests, segments that are encoded as neighboring alphabets (e.g. a and b) might be close to each other. This is also the reason that neighboring alphabets have a minimum distance of zero. Therefore, we propose that, after exhausting the same-string list, we iteratively examine the buckets for which the ordinal values of all symbols differ by no more than one from the corresponding symbols in the candidate string. For example, if the candidate string is “ bab ,” after exhausting the bucket containing objects encoded to “ bab ,” we might retrieve the bucket for “ aaa ,” then “ aab ” and so forth. If we exhaust all the buckets for strings of MINDIST zero from the candidate string, then we examine the rest of the objects in sequential order (we can have a vector of flags, telling us which object has been visited). This strategy is not much more costly than the base heuristic, except for the bucket-lookup, and is shown to be more efficient than the base heuristic.

Note that we can effectively prune off the search space (not just for individual candidate’s nearest neighbor search, but for the entire RNN search) by taking advantage of the lower-bounding property of SAX, and/or using the triangle inequality property. For example, we can eliminate unnecessary distance computations if the lower-bounding

distance between C_i and C_j is larger than $Dist(C_i, Q)$. We can envision such optimization to be extremely useful when $Dist(C_i, Q)$ is small (i.e. when C_i is one of Q ’s RNNs). Such case is also the only case when the whole dataset needs to be scanned in order to conclude that Q is C_i ’s nearest neighbor. We will defer the effectiveness of lower-bounding distances for future research. In the remaining of this section, we describe two more heuristics that return approximate answers.

H3: MINDIST partial candidates heuristic

So far, the two heuristics proposed determine the ordering of the preferred list before examining the remaining objects. Nevertheless, both heuristics still require that we check every single object before the search of RNN ends. For this heuristic, instead of going through every object in the entire dataset (cf. Step 1 in Table 1), we only consider candidates whose MINDIST to Q is zero. Then for each candidate, we retrieve objects the same way as in H2.

The intuition behind this heuristic is that objects that can potentially be RNNs of the query are probably similar to the query. Therefore, they are likely to be encoded to similar strings as the query (i.e. same string or strings with MINDIST of zero). While we cannot guarantee to always find exact solutions with this heuristic unless we incorporate the knowledge from lower- or even upper-bounding distances, experiments show that this heuristic produces close-to-exact output.

In a broad sense, our approximate search is similar to the approach in [16]. In their work, the search for approximate RNNs is done by the following steps:

1. A small subset of objects that could potentially be the RNNs of the query is identified. The authors argue that due to the close relationship between k -NN and RNN, an object’s RNNs are likely to be also its k -NNs, given an appropriate k .
2. For this subset of objects, find their global nearest neighbors so that it can be determined if they are closer to the query than they are to their respective nearest neighbors. The author proposed two filtering approaches to speed up the search.

There is no guarantee that the query’s RNNs are always among its k -NNs, since it’s dependent on the value of k . Therefore, the results produced are only an approximate. The authors showed that with a relatively small k (e.g. 30), they achieve 85%-95% recall on 4 different datasets.

This part of our work is similar to theirs in the sense that we also try to produce a small subset of objects that likely contain the true RNNs. However, we do so by predicting such likelihood from the objects’ symbolic representations. Our approach is not restricted by the choice of k . As will be shown later, it is difficult to determine the value of k that works well for all datasets. If k is too large, then we end up with a larger subset than

necessary. On the other hand, if k is too small, than we might miss the true RNNs. Instead of specifying the number of objects to be our potential candidates, the size of such subset depends on the data. Our approach only retrieves the “similar” ones according to the symbolic representations.

H4: MINDIST subset only heuristic

For the MINDIST-Partial Heuristic, although we have reduced the search space by only considering the candidate objects that fulfill the MINDIST requirement (let’s call this subset of objects A), the search space for individual candidates is still the same (i.e. we need to potentially consider the entire dataset for each candidate). With the same argument provided above, for this heuristic, we consider only objects whose MINDIST to the current candidate object is zero (let’s call this subset of objects B). Note that A and B are not necessarily the same.

Let’s consider an example. Suppose the SAX word for the query is “ aba .” Furthermore, suppose that among the subset A that fulfills the MINDIST requirement for the query, we consider a candidate “ bba .” Note $MINDIST(“aba”, “bba”) = 0$. We then need to construct subset B that fulfills the MINDIST requirement with respect to “ bba .” In this subset B , we might have an object whose SAX word is “ cba .” Again, verify that $MINDIST(“bba”, “cba”) = 0$. However, “ cba ” is not in A , since $MINDIST(“cba”, “aba”) > 0$. This is the reason that for each candidate, we need to construct its own subset of potential candidates.

Again, this heuristic produces approximate solutions; however, the experiments show that the accuracy is very high. As a matter of fact, the accuracy for this heuristic is the same as the previous heuristic, i.e. whatever objects retrieved by H3 will be guaranteed to be retrieved by H4. Since we only consider a subset of the dataset, the declaration of a candidate as an RNN after exhausting the subset and not finding an object closer to the query might be a false positive (i.e. a closer object might be among the remaining subset that we do not consider). However, this heuristic improves the search time tremendously, and is particularly useful when speed rather than exactness is of critical concern.

Since the MINDIST subset might still be larger than we desire, we can put more restrictions on the strings to be considered. For example, instead of allowing the alphabets at each location to differ at most by 1, we can allow only one or two such “don’t-care” location for the entire string. For H3 and H4, the string “ aba ” would consider “ bab ” similar since $MINDIST(“aba”, “bab”) = 0$, even though at all locations, the alphabets are different. If we allow only one or two differing alphabets for the entire string, then the potential subset will be considerably smaller. We defer the analysis of trade-off between efficiency and accuracy for future research.

We summarize all heuristics in Table 4 below. All but the brute-force approach benefits from early termination for each candidate’s NN search. Heuristics #1-#4 were described in this section, whereas Heuristic #0B was described in Table 1. The brute-force approach computes all $O(N^2)$ distances.

Table 4. Summary of the heuristics described so far.

		Outer loop (candidates)	Inner loop (NN search for candidate)
0A	Brute Force (no early termination)	Sequential – all	Sequential – all
0B	Basic w/o SAX	Sequential – all	Sequential – all
1	Basic w/ SAX (same string)	Sequential – all	Same string subset first; the rest sequential
2	Zero MINDIST	Sequential – all	Zero MINDIST subset first; the rest sequential
3	Zero MINDIST w/ Partial Candidates	Zero MINDIST subset only	Zero MINDIST subset first; the rest sequential
4	Zero MINDIST Subset Only	Zero MINDIST subset only	Zero MINDIST subset only wrt current candidate

4.3 Parameter selection

SAX requires two parameters: alphabet size α , and the SAX word size w . While large values of α and/or w result in more selective encoding, they also result in small preferred lists (since the criteria for objects to be mapped to the same or similar strings are more strict). On the other hand, small values of α and/or w will likely result in large preferred lists, with few of the objects being the true similar ones. Neither of these situations is desirable for our heuristics.

Extensive experiments carried out by [11] suggest that typically, a value of either 3 or 4 for α works well for virtually any task on any dataset. For this reason, we use $\alpha = 4$ for all our experiments.

Having fixed α , we now have to determine the value for w . While the best choice of w is data-dependent, generally speaking, time series with smooth patterns can be described with a small w , and those with rapidly changing patterns prefer large w to capture the critical changes.

5 Empirical evaluation

All heuristics we propose in this paper can be extended to Rk NN search very easily. In the first part of our experiments, we evaluate the accuracy of the *approximate* algorithms of Rk NN search, and its efficiency/scalability in the second part. Note that H1 and

H2 produce exact solutions, so they are excluded from the accuracy experiments. In the third part of our experiments, $RkNN$ is used to identify and remove outliers from training sets for pattern recognition by kNN .

5.1 Accuracy

We use the datasets from UCR Time Series Archive [4]. The datasets can be found from the UCR Time Series Classification/Clustering Page [6]. Other than regular time series data, the datasets also contain shape (e.g. Face, Leaf) and video (e.g. yoga) data that are represented as sequence data. Since we do not need the class information, we combine the training and test sets to form larger datasets. For each dataset, we randomly draw 50 objects as queries, and run $RkNN$ on these objects in separate runs. Arbitrarily, we choose $k = 3$, $\alpha = 4$, and $w = 4$. We then compute the average error and false alarm rates. The error rate is one minus the percentage of $RkNN$ s successfully found; and the false alarm rate is the percentage of items incorrectly identified as the $RkNN$ s. As noted in [6], these datasets are too small to make any meaningful claims on efficiency. Experiments on scalability will be reported in the next section.

We compare our results with the kNN based strawman approach as described in Section 4.2, under heuristic H3 [16]. We need to determine one parameter; namely, the value for k . In order to avoid confusion with the k in $RkNN$, we will use kl to denote the k in k -NN based approach. In the paper [16], the authors stated that 90% recall can be achieved by using $kl = 10$. We note that this is only true for two out of four experiments conducted in the original paper. For one of the datasets, the recall is around 72% for $kl = 10$. Therefore, we choose $kl = 30$, in hope that better accuracy can be obtained.

The accuracy reported here for the strawman approach, shown in Table 5, is expected to be higher than the accuracy in the original paper, since instead of performing SVD on the data, we use the raw data directly. The reason that we skip the SVD step is that SVD becomes impractical for large and/or dynamic datasets. We feel that this is a fair comparison, since in this section, we are only comparing accuracy, and by deliberately removing the step that causes deteriorating results for their approach, the results we report here are actually optimistic compared to the original ones. Nevertheless, even in this case, our heuristics result in considerably higher accuracy. Note that both H3 and the k -NN based approach should have no false positives, as they consider the entire dataset for each candidate (i.e. find global NN), whereas H4 considers only the MINDIST subset.

As mentioned earlier, it's difficult to determine the effective value for kl . One naïve approach of choosing kl as a function of the size of the dataset does not work, as we can see from the results that the error rates for the k -NN based approach are not correlated to the data size.

Fortunately, our approaches allow automatic determination of the subset size, according to the structure of data.

From the results, we can see that our accuracy is nearly 100%, except for two datasets. For H4, there are some false positives, but the average false alarm rates are very small.

Table 5 Error and false positive rates for the approximate heuristics (H3 and H4).

Datasets (# records x length)	H3		H4		k-NN	
	Error rate	False Pos. rate	Error rate	False Pos. rate	Error rate	False Pos. rate
50words (905x270)	0.0	0.0	0.008	0.01	0.015	0.0
Adiac (781x176)	0.0	0.0	0.0	0.0	0.124	0.0
Beef (60x470)	0.0	0.0	0.0	0.0	0.0	0.0
CBF (930x128)	0.0	0.0	0.0	0.0	0.004	0.0
Coffee (56x286)	0.0	0.0	0.0	0.0	0.0	0.0
ECG200 (200x96)	0.0	0.0	0.0	0.0	0.024	0.0
FaceAll (2250x131)	0.0	0.0	0.0	0.0	0.034	0.0
FaceFour (112x350)	0.0	0.0	0.0	0.0	0.012	0.0
Gun_Point (200x150)	0.0	0.0	0.0	0.0	0.005	0.0
Lighting2 (121x637)	0.0	0.0	0.0	0.0	0.05	0.0
Lighting7 (143x319)	0.0	0.0	0.0	0.0	0.025	0.0
OSULeaf (442x427)	0.0	0.0	0.0	0.0	0.038	0.0
OliveOil (60x570)	0.0	0.0	0.0	0.0	0.032	0.0
SwedishLeaf (1125x128)	0.0	0.0	0.0	0.0	0.2	0.0
Trace (200x275)	0.0	0.0	0.0	0.0	0.0	0.0
TwoPatterns (5000x128)	0.0	0.0	0.0	0.0	0.031	0.0
Fish (350x463)	0.0	0.0	0.0	0.0	0.065	0.0
Synthetic Control (600x60)	0.003	0.0	0.003	0.006	0.177	0.0
Wafer (7164x152)	0.0	0.0	0.0	0.0	0.036	0.0
Yoga (3300x426)	0.0	0.0	0.0	0.0	0.017	0.0

5.2 Efficiency and scalability

In this part of the experiments, we evaluate the efficiency of our heuristics on large datasets. We create random walk

datasets of 100,000 and 300,000 in size. We then generate 10 different queries and use them for RNN search. To determine the efficiency of the heuristics, we compare the total number of distance calls to the Euclidean distance function. Figure 5 shows the results, measured in the fraction of distance calls needed compared to the basic heuristic (H0B, early termination only; no heuristic ordering). Note that as the data size increases, the speedup is more obvious. Also, since we should expect that real-world datasets have more structure than random walk data (thus easier to come across similar object which provokes early termination), the results shown here are likely to be an underestimation of true potential of speedup.

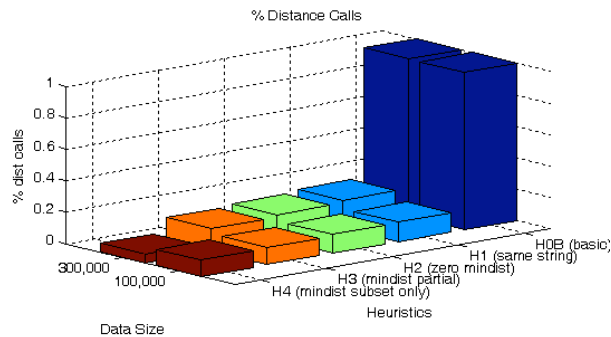


Figure 5. The speed up is shown as the percentage of distance calls needed for each heuristic, compared to the basic heuristic (H0B: early termination only; no heuristic ordering).

5.3 Outlier Detection

In addition to determining the influence of a given query point, the concept of RNN has many useful applications. For example, RNN can potentially be applied on clustering outlier detection. While most anomaly detection algorithms [13, 18] work efficiently in detecting a global anomaly, they cannot detect local outlier for a given cluster (e.g. an outlier that is not necessarily farthest away from the rest of the data).

As illustrated in Figure 6, there are two clusters with two outliers. Observation 8 is considered to be a global outlier, while observation 7 is considered to be a local outlier. Most anomaly detection algorithms can identify observation 8 as the anomaly, but will fail to identify observation 7.

In addition, in pattern recognition (classification) tasks, it can be useful to avoid making inferences based on unusual training set members. For example, in Figure 6, observation 7 changes the position of the maximum margin hyperplane that separates the squares on the left from the circles on the right.

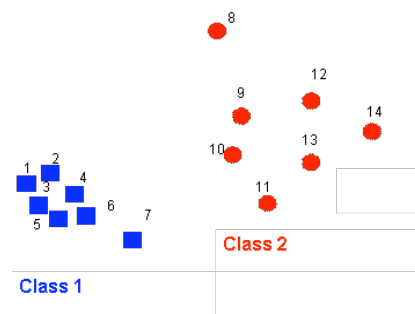


Figure 6. Two clusters of data with one global outlier (observation 8) and one local outlier (observation 7). Both outliers can be detected using an $RkNN$ -based algorithm.

Nanopoulos et al [13] proposed an algorithm based on $RkNN$ to identify anomalies in data. The idea is that, given an appropriate value of k , objects that do not have any $RkNN$ are considered anomalies. In this section we investigate the effectiveness of the $RkNN$ -based outlier detection algorithm

We used the data sets from the UCR Classification/Clustering page [6] to see if using $RkNN$ for outlier detection and removal improves query classification performance. In our training step, we try different values of k starting from 2, and remove all the objects with zero $RkNN$ count. We stop the process when no object has zero $RkNN$ count (typically when $k = 5$ or 6) or when we reach the user-defined maximum value of k . Note that as k increases, the number of such objects returned decreases.

After identifying and removing the outliers (if any), we validate the results and show the actual classification accuracy on the test sets in Table 6. The first column of numbers is the original accuracy, computed before any outlier removal. The second column of numbers is the accuracy for the $RkNN$ -based outlier detection algorithm.

The bolded text indicates that the removal of outliers by the corresponding method improves the accuracy, whereas the shaded cell indicates that the accuracy deteriorates after the removal. To get an idea of how effective our algorithms perform, we repeat the classification task using various parameters examined during the training process, and record the best possible accuracy (shown as the second number, if the actual accuracy is not the best).

Table 6. Error rate before and after outlier removal, if any.

Dataset	Original	RkNN/best possible
50words	0.2527	0.2527 / 0.2505
Adiac	0.4194	0.4194
Beef	0.33	0.33
CBF	0.0233	0.06/0.0233
Coffee	0	0
ECG200	0.1900	0.1800
FaceAll	0.2154	0.2154 / 0.1811
FaceFour	0.1591	0.1591
GunPoint	0.04	0.04 / 0.033
Lighting2	0.1311	0.1311
Lighting7	0.2329	0.2740 / 0.2329
OSULeaf	0.4463	0.4463
OliveOil	0.1333	0.1333
Swedish	0.2192	0.2192 / 0.2176
Trace	0.01	0.01
2Patterns	0.0008	0.0008 / 0.0005
Fish	0.1657	0.1543
Synthetic	0.0367	0.0367
Wafer	0.0092	0.0117 / 0.0092

In some cases, removal of “outliers” worsens the accuracy. We believe that one way to improve the flexibility of the RkNN-based approach is to add a filtering step and treat the “outliers” returned from the algorithm as candidates only, rather than removing them right away. Once we get the set of candidates, we can then remove them one at a time until the accuracy deteriorates. We will defer the investigation of such refinement for future research.

We conclude this section by noting that while most anomaly detection algorithms can benefit from knowing the class labels (i.e. anomaly detection can be applied individually on each class), the RkNN-based anomaly detection approach does not need such information and therefore has potential for a broader range of applications such as pre-clustering outlier removal.

6 Conclusions and future work

In this work we propose several heuristics that speed up the reverse nearest neighbor search by pruning off a large part of the search space. While the focus is mainly on sequence data, we note that many multimedia data types such as images can be represented as sequences as well. In addition, the same heuristics can potentially be adapted to other signal-typed, continuous data even without representing them as time series.

For future work, we plan to investigate the following:

- Efficient algorithms when multiple queries are given.
- As shown in the experimental results, the speedup becomes more significant when the data size increases. We plan to run more scalability experiments, using large, disk-resident datasets.
- Other heuristics as mentioned in Section 4.

7 References

- [1] S. Bertchtold, B. Ertl, D. A. Keim, H. P. Kriegel, and T. Seidl. Fast Nearest Neighbor Search in High-Dimensional Space. In proceedings of the 14th International Conference on Data Engineering. Orlando, FL. Feb 23-27, 1998. pp. 209-218
- [2] K. L. Cheung and A. W. Fu. Enhanced Nearest Neighbour Search on the R-Tree. SIGMOD Record. vol. 27. pp. 16-21. 1998.
- [3] H. Ferhatosmanoglu, I. Stanoi, D. Agrawal, and A. E. Abbadi. Constrained Nearest Neighbor Queries. In proceedings of the 7th International Symposium on Spatial and Temporal Databases. Redondo Beach, CA. July 12-15, 2001. pp. 257-278
- [4] The UCR Time Series Data Mining Archive. <http://www.cs.ucr.edu/~eamonn/TSDMA/index.html>
- [5] E. Keogh and S. Kasetty. On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. In proceedings of the 8th ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining. Edmonton, Alberta, Canada. July 23-26, 2002. pp. 102-111
- [6] The UCR Time Series Classification/Clustering Homepage: http://www.cs.ucr.edu/~eamonn/time_series_data.
- [7] G. Kollios, G. Gunopulos, and V. J. Tsotras. Nearest Neighbor Queries in a Mobile Environment. In proceedings of the International Workshop on Spatio-Temporal Database Management. Edinburgh, Scotland. Sept 10-11, 1999. pp. 119-134
- [8] F. Korn and S. Muthukrishnan. Influence Sets Based on Reverse Nearest Neighbor Queries. In proceedings of the 19th ACM SIGMOD International Conference on Management of Data. Dallas, TX. May 14-19, 2000. pp. 201-212
- [9] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. Fast Nearest-Neighbor Search in Medical Image Databases. In proceedings of the 22nd International Conference on Very Large Data Base. Bombay, India. Sept 3-6, 1996. pp. 215-226
- [10] J. Lin and E. Keogh. Group SAX: Extending the Notion of Contrast Sets to Time Series and Multimedia Data. In proceedings of the 10th European Conference on Principles and Practice of Knowledge

Discovery in Databases. Berlin, Germany. Sept 18-22, 2006. pp. 284-296

- [11] J. Lin, E. Keogh, W. Li, and S. Lonardi. Experiencing SAX: A Novel Symbolic Representation of Time Series. *Data Mining and Knowledge Discovery*. 2007.
- [12] J. Lin, M. Vlachos, E. Keogh, and G. Gunopulos. Multi-Resolution K-Means Clustering of Time Series and Application to Images, *Workshop on Multimedia Data Mining, the 4th SIGKDD International Conference on Knowledge Discovery and Data Mining*. Washington D.C., 2003.
- [13] A. Nanopoulos, Y. Theodoridis, and Y. Manolopoulos. C2p: Clustering Based on Closest Pairs. In proceedings of the 27th International Conference on Very Large Data Bases. Rome Italy. Sept 11-14, 2001. pp. 331-340
- [14] S. Roussopoulos, S. Kelley, and F. Vincent. Nearest Neighbor Queries. In proceedings of the 1995 ACM SIGMOD International Conference on Management of Data. San Jose, CA. May 22-25, 1995. pp. 71-79
- [15] T. Seidl and H. P. Kriegel. Optimal Multi-Step k-Nearest Neighbor Search. In proceedings of the 1998 SIGMOD International Conference on Management of Data. Seattle, WA. June 2-4, 1998. pp. 154-165
- [16] A. Singh, H. Ferhatosmanoglu, and A. Tosun. High-Dimensional Reverse Nearest Neighbor Queries. In proceedings of the 2003 ACM CIKM International Conference on Information and Knowledge Management. New Orleans, LA. Nov 2-8, 2003.
- [17] I. Stanoi, D. Agrawal, and A. E. Abbadi. Reverse Nearest Neighbor Queries for Dynamic Databases. In proceedings of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery. Dallas, TX. May 14, 2000. pp. 44-53
- [18] Y. Tao, D. Papadias, and X. Lian. Reverse knn Search in Arbitrary Dimensionality. In proceedings of the 30th International Conference on Very Large Data Bases. Toronto, Canada. Aug 31-Sept 3, 2004. pp. 744-755
- [19] L. Wei, E. Keogh, and X. Xi. SAXually Explicit Images: Finding Unusual Shapes. In proceedings of the 2006 IEEE International Conference on Data Mining. Hong Kong. Dec 18-22, 2006.
- [20] C. Yang and K.-I. Lin. An Index Structure for Efficient Reverse Nearest Neighbor Queries. In proceedings of the 17th International Conference on Data Engineering. Heidelberg, Germany. April 2-6, 2001. pp. 485-492