Efficiently Evaluating Order Preserving Similarity Queries Over Historical Market-Basket Data

Submitted to ICDE'06 - Paper ID: 546

Reza Sherkat
Department of Computing Science
University of Alberta
reza@cs.ualberta.ca

Davood Rafiei
Department of Computing Science
University of Alberta
drafiei@cs.ualberta.ca

Abstract

We introduce a new domain-independent framework for formulating and efficiently evaluating similarity queries over historical data, where given a history as a sequence of timestamped observations and the pair-wise similarity of observations, we want to find similar histories. For instance, given a database of customer transactions and a time period, we can find customers with similar purchasing behaviors over this period. Our work is different from the work on retrieving similar time series; it addresses the general problem in which a history cannot be modeled as a time series, hence the relevant conventional approaches are not applicable. We derive a similarity measure for histories, based on an aggregation of the similarities between the observations of the two histories, and propose efficient algorithms for finding an optimal alignment between two histories. Given the non-metric nature of our measure, we develop some upper bounds and an algorithm that makes use of those bounds to prune histories that are guaranteed not to be in the answer set. Our experimental results on real and synthetic data confirm the effectiveness and efficiency of our approach. For instance, when the minimum length of a match is provided, our algorithm achieves up to an order of magnitude speed-up over alternative methods.

1 Introduction

Traditional databases store a single, often recent, snapshot of a modeled real world. With recent developments in the areas of data warehousing and data mining, there has been an increasing interest in querying multiple snapshots of data, often stored in temporal databases [22, 24], semi-structured document collections [8, 7, 9], and OLAP applications [15]. The focus has been mainly on detecting and representing changes in order to provide a better support for selection and projection queries over multiple versions of data. Support for similarity queries on histories enables various forms of analysis on time evolving data. There has been an increasing interest in efficiently retrieving similar histories where a history is described as a time series [1, 18]. However, there are many real-life scenarios in which the history cannot be modeled as a time series. The Internet Archive [12], for instance, stores a snapshot of the Web approximately every six months. An interesting query over this collection is: *find Web pages with change histories similar to the change history of a given page*. Naturally it is quite possible to find Web pages that are similar in content at one or more points in time but have different change histories. It is also possible to find rather dissimilar Web pages with similar change histories; this can happen for two relatively static pages, perhaps maintained by the same authority, where small fractions of both pages change similarly, but the two pages are otherwise different.

1.1 Motivating Examples

Example 1. In a hospital, routine observations are made about patients. These observations can be made by doctors or nurses and may include general symptoms such as "high fever," "rash," "high blood cholesterol," "bleeding," the medications used, responses to the medications, and the medical advice given. If each sign, symptom, or medication is assigned a symbol, then an observation simply becomes a set of symbols, and the medical history of a patient can be described as a sequence of sets. The example in Table 1.1 shows this scenario for three histories over a period of 4 days. An interesting query is "*find medical histories similar to h2.*" Suppose the query returns the medical histories *h1* and *h3*. We expect to find some common patterns between similar histories so the next interesting query can be "*in what respect are histories h2 and h1 similar?*" For day 1, the symbol *b* is observed in both *h1* and *h2*. There is no

Table 1. An example showing three histories over a period of three days

	h1	*h2*	*h3*
Day 1	{a, b}	{b, c, d}	{b, c, d}
Day 2	{c, d}	{f, g}	{a}
Day 3	{f, g}	{g, h, i}	{i}
Day 4	{h, i, j}	{h, k}	{g, h, i}

common symbol for day 2, but the two histories also share a symbol for days 3 and 4. We can find a larger overlap between the two histories if we compare days 2, 3 and 4 of *h1* respectively with days 1, 2, and 3 of *h2* where the common pattern will be $\langle\{c, d\}, \{f, g\}, \{h, i\}\rangle$. Similarly, the common pattern with the largest overlap between *h2* and *h3* is $\langle\{b, c, d\}, \{g, h, i\}\rangle$. We might be also interested in a common pattern that covers at least three days of *h2* and *h3* (i.e. $\langle\{b, c, d\}, \{i\}, \{h\}\rangle$).

Example 2. In retail, customer transactions are often recorded in a data warehouse for further querying and analysis. The purchase history of a customer, in particular, may show changes of the needs and the preferences over time. To provide personal recommendations to customers, we may want to find customers that have a purchase history similar to a given customer. Another interesting query is an all-pair query, which can be used to cluster customers based on their purchase history. Each cluster might be further analyzed in order to find patterns specific to each group.

1.2 Problem Statement and Our Contributions

The problem to be addressed in this paper is to formalize the notion of a similarity search over historical market basket data, where each history is encoded as a sequence of timestamped observations, and to develop efficient algorithms for evaluating our queries. Our contributions are as follows:

- We introduce a measure of similarity which generalizes the idea of an edit distance to histories and which is useful in many practical settings (including those reported in our introduction and experiments).
- We develop a notion of an optimal alignment between two histories and an efficient dynamic programming algorithm that finds the score of an optimal alignment of any given length.
- Given the length and the score of an optimal alignment for two histories, our enumeration algorithm efficiently finds a set of common signatures that can give rise to such an optimal alignment. These signatures show the common patterns that are observed in the same order in two histories, hence generalizing the concept of the longest common subsequence to histories.
- To efficiently evaluate our similarity queries over large collections of histories, we develop a few upper bounds that help in our pruning of non-qualifying histories. In particular, one of our upper bounds targets the cases in which only a small fraction of items from the set of all possible items appear in each observation; such cases are common in many datasets that we have been experimenting with.
- We have conducted extensive experiments to evaluate both the effectiveness of our similarity measure and the efficiency of our algorithms using both real and synthetic data.

The rest of the paper is organized as follows. In Section 2 we present some preliminary definitions and notations. In Section 3, we present our similarity model for histories. Section 4 presents our approach to process similarity queries over large collections of histories. Performance evaluation and experimental results are reported in Section 5. Related work is surveyed in Section 6, followed by conclusions and directions for future work in Section 7.

2 Background

In this section, we introduce our notation and provide the background for the rest of the paper.

Definition 1 (Observation) Let $I = \{t_1, \ldots, t_n\}$ be a set of items. An observation is a set of pairs (t_i, w_i) such that $t_i \in I$ is an item and w_i, a real number, is the weight of the item in the observation. Given an ordering of the items in I, an observation can be represented as a vector:

$$[w_1, w_2, \ldots, w_n]^T.$$

We assume that the weight of an item which is not in the observation is zero. An observation x may be associated with a timestamp in which case we use $ts(x)$ to refer to this timestamp. The similarity between two observations x and y can be quantified using a similarity function, denoted by $\sigma(x, y)$, which can be, for instance, the cosine measure [19], or the Jaccard coefficient, or its extensions [23].

Definition 2 (History) A history is a chronologically-ordered sequence of observations denoted as:

$$X = \langle x_1, x_2, \ldots, x_m\rangle$$

such that $ts(x_i) \leq ts(x_j)$ iff $i < j$. The length of the history, denoted by $|X|$, is the number of observations in the history.

Alternatively, if we are interested in *changes* rather than the content of observations, we may consider the history of changes, instead of contents. For a Web page, an observation can be either the content or the updates to a previous version and a timestamp can be the time the page is crawled or the changes are observed. For a customer, an observation can be the set of items purchased within a transaction and the weight of an item may indicate the importance of the item (e.g. quantity, price, profit) to the transaction.

Since a history is modeled as a sequence of observations, alignment techniques can be used to measure the similarity between two histories. An alignment is a way to line up subsequences of two histories where each observation of a history is matched with either an observation in the other history or a gap.

Definition 3 (Alignment) An alignment of two histories is a sequence of the following edit operations:

- $(\alpha \rightarrow \epsilon)$ denotes the deletion of observation α,
- $(\epsilon \rightarrow \beta)$ denotes the insertion of observation β, and
- $(\alpha \rightarrow \beta)$ denotes the matching of α with β

where α is an observation in one history, β is an observation in the other history and ϵ denotes a null observation. An alignment may be assigned a score and this score may be used to compare two alignments.

Definition 4 (Alignment Score) Let $\sigma(\alpha \rightarrow \beta)$ denote the score of matching two observations, with the constraint that at most one observation can be null. The score of an alignment is defined as an aggregate score of the matches in the alignment.

If the aggregation function is fixed to *sum*, which is commonly used to compare strings [11], the alignment score of an alignment $\langle \alpha_1 \rightarrow \beta_1, \ldots, \alpha_r \rightarrow \beta_r \rangle$ is defined as:

$$\sum_{i=1}^{r} \sigma(\alpha_i \rightarrow \beta_i) \tag{1}$$

Next, we discuss the limitations of existing solutions for aligning two histories and our proposed solutions to avoid these limitations.

3 Optimal Alignments of Histories

In general, it is possible to find multiple alignments between two histories, but we are often interested in an alignment with some desired properties. For instance, we may want to find an alignment with the highest score or the longest possible alignment. Such properties of an alignment may be specified in a query in the form of some conditions on the length and/or the score of an alignment.

3.1 Limitations of the Existing Solutions

A related problem is string alignment which has been extensively studied in bioinformatics [20], approximate string matching, speech processing, etc. The Smith-Waterman (SW) algorithm [21] is commonly used to find an alignment with the highest score. This algorithm, when applied to two histories of lengths m and n, can find an optimal alignment in $O(mn)$ time using a dynamic programming approach, assuming that two observations can be compared in constant time. However, there are two problems when the SW algorithm is applied to histories. First, it may not be realistic to assume that two observations can be compared in constant time, in particular when the observations are long or the similarity function σ is not trivial. The number of possible observations is also typically huge, and it is not an option to pre-compute the pair-wise similarity between all observations. Second, we may not be interested in an alignment with the highest score. Instead, we might be interested in an alignment of a specific length with the highest score or the longest alignment(s) with a score greater than a threshold. In both cases, the desired alignment is not necessarily an extension of an alignment found by the SW algorithm. Consider, for instance, the histories *h2* and *h3* in Table 1.1. Given σ as the fraction of items common to two observations, an alignment that maximizes the score in Equation 1 and can be found by the SW algorithm is $\langle \{b,c,d\} \rightarrow \{b,c,d\}, \{g,h,i\} \rightarrow \{g,h,i\} \rangle$. However, if we are interested in an optimal alignment of length three, i.e. $\langle \{b,c,d\} \rightarrow \{b,c,d\}, \{g,h,i\} \rightarrow \{i\}, \{h,k\} \rightarrow \{g,h,i\} \rangle$, the result of the SW algorithm cannot be extended to find this alignment.

Sequence alignment has been an active research in bioinformatics and there are a number of improvements over the SW algorithm. FASTA [17] and BLAST [3], in particular, rely on the so-called *hit-and-extend* heuristic to speed up the search for an optimal alignment. The idea here is to construct a deterministic finite automaton (DFA) for one sequence and run this automaton on the other sequence to find every substring of a fixed length which is similar to a substring in the former sequence. Unfortunately, the size of the DFA grows exponentially with the size of the alphabet. For historical market basket data, for instance, the alphabet is the power set of the set of items. Even when the number of items is as small as $1,000$, the alphabet size is $\simeq$ 1.07e+301, compared with 4 and 20 for DNA and protein sequences respectively. Therefore, this approach cannot be used to speed up the dynamic programming scheme in our problem. For the same reason, approaches that use a suffix tree to speed up the search (e.g. [6]) are not also applicable. Next, we discuss our algorithm for finding the score of an optimal alignment of a given length l, here referred to as l-alignment.

3.2 Finding the Score of an Optimal l-Alignment

Given two histories, we want to find the score of an optimal alignment of a given length l. We can relate the problem of finding the score of an optimal l-alignment to the problem of finding the score of a shorter optimal alignment if the alignment scoring function satisfies the principle of optimality [4]. Let h_1 and h_2 be two histories and denote an optimal alignment of the two histories with A^*. An alignment scoring function $f(\cdot)$ satisfies the principle of optimality if for any pair of non-overlapping prefixes and suffixes of h_1 and h_2:

$$f(A^*) \geq f(A_p^* \oplus A_s^*) \tag{2}$$

where A_p^* is an optimal alignment between the two prefixes, A_s^* is an optimal alignment between the two suffixes and $\oplus$ denotes the concatenation operator. A large class of functions, including Eq. 1, satisfy the principle of optimality; a detailed discussion of these functions can be found in [4]. For the sake of presentation clarity, from now on we will use Eq. 1 as our scoring function, but the algorithm discussed here should be applicable to any function that satisfies the principle of optimality. Next, we propose a divide-and-conquer approach to find the score of an optimal l-alignment.

Lemma 1 *Let $G_{i,j}^l$ be the score of an optimal l-alignment of two suffixes $\langle x_i, \ldots, x_m \rangle$ and $\langle y_j, \ldots, y_n \rangle$:*

$$G_{i,j}^l = max \begin{cases} \sigma(x_i \rightarrow \epsilon) + G_{i+1,j}^l \\ \sigma(\epsilon \rightarrow y_j) + G_{i,j+1}^l \\ \sigma(x_i \rightarrow y_j) + G_{i+1,j+1}^{l-1} \end{cases} \tag{3}$$

where $1 \leq l \leq min(m,n)$, $i \leq m-l+1$ and $j \leq n-l+1$. $G_{i,j}^l$ is zero for $i > m-l+1$ or $j > n-l+1$.

Proof: One of the following constructions gives the score of an optimal l-alignment of $\langle x_i, \ldots, x_m \rangle$ and $\langle y_j, \ldots, y_n \rangle$:

- Leave x_i unmatched and find the score of an optimal l-alignment of $\langle x_{i+1}, \ldots, x_m \rangle$ and $\langle y_j, \ldots, y_n \rangle$. The score of this alignment is $G_{i+1,j}^l$ plus the penalty of leaving x_i unmatched, i.e. $\sigma(x_i \rightarrow \epsilon)$. A similar argument applies when y_j is left unmatched and we omit here for brevity.

- Match x_i with y_j and find the score of an optimal $(l-1)$-alignment of $\langle x_{i+1}, \ldots, x_m \rangle$ and $\langle y_{j+1}, \ldots, y_n \rangle$. The score of this alignment is $G_{i+1,j+1}^{l-1}$ plus the score of matching x_i with y_j. ∎

Note that $G_{1,1}^l$ gives the score of an optimal alignment of length l, which can be found using a dynamic programming algorithm in $O(mnl)$ time and $O(mn)$ space. This approach, however, will become expensive for long histories. Therefore, we identify some special cases in which these time and space complexities can be reduced.

There are often scenarios in which the two observations cannot be matched if they are recorded far apart. For instance, when aligning the histories of two customers, it may not be reasonable to match purchase transactions that are recorded more than a month apart. Therefore, to preserve a temporal locality among matched observations, we may enforce a constraint similar to the Sakoe-Chiba band used to restrict the warping window in dynamic time warping [5]:

Definition 5 (r-neighborhood constraint) An alignment satisfies the r-neighborhood constraint if for all matches $(x_i \rightarrow y_j)$ in the alignment, y_j is in r-neighborhood of x_i, i.e. $|ts(x_i) - ts(y_j)| \leq r$.

When $r = 0$, only observations that are recorded at the same time could be considered for a match. Increasing r adds some flexibility in matching observations that are recorded within a time frame.

Our first improvement takes advantage of an r-neighborhood constraint, when it is present. Let X and Y be two histories of lengths m and n respectively and let $m \geq n$ (the role of X and Y can be interchanged otherwise). For $r > 0$, let m_{2r} denote the maximum number of observations in X that are recorded in a time frame of length $2r$. Since each observation of Y can be matched to one of at most m_{2r} observations in X, the time complexity and the space requirement of computing $G_{1,1}^l$ reduce to $O(m_{2r}nl)$ and $O(m_{2r}n)$, respectively. The improvement is significant when X is long and $m_{2r} \ll m$.

Our next improvement is useful if a minimum threshold is specified for the score of matches of an alignment. The idea is to remove observations that cannot participate in an alignment before running a dynamic programming algorithm. More specifically, given two histories X and Y, X is transformed into possibly a shorter history by removing all observations x_i in X such that $\sigma(x_i \rightarrow y_j)$ is less than the given threshold for all observations y_j in Y; a similar transformation can be applied to Y. The transformations can be applied to X and Y in $O(mn)$ time.

3.3 Finding Common Patterns of Two Histories

Further to fining a degree of similarity between two histories, it is interesting to find out in what respect two histories are similar. More specifically, we want to identify the common patterns that arise in two histories and may give rise to a similarity. Finding such patterns for histories is related to the problem of finding the longest common subsequence (LCS) for strings. However, the idea of only matching identical observations in LCS is too restrictive;

we can hardly find any identical observation in two histories. Therefore, we generalize LCS by relaxing the condition that the matched observations must be identical. We do this in two phases: first, we find an alignment of a desired length and score; this is discussed in the next subsection. Then, we can identify the common items in the matched observations and construct a common pattern, referred to here as a *common signature*.

In our setting, an observation is a set. Therefore, given an alignment, a common signature can be a sequence of sets, each being the intersection of two observations matched in the alignment. The common signature for histories is a generalization of the LCS for strings. However, unlike the LCS, we are interested in finding optimal alignments that contain a desired number of matches since the alignment score depends on the number of matches in the alignment.

3.4 Enumerating Optimal Alignments

Given two histories X and Y, we want to enumerate all optimal l-alignments whose score is greater than a desired threshold s.

Lemma 2 *Let X and Y be two histories with m and n observations each. For any $1 \leq r \leq l$, if $\langle x_{i_1} \rightarrow y_{j_1}, \ldots, x_{i_r} \rightarrow y_{j_r}, \ldots, x_{i_l} \rightarrow y_{j_l} \rangle$ is an optimal l-alignment of X and Y, then*

- *$\langle x_{i_1} \rightarrow y_{j_1}, \ldots, x_{i_{r-1}} \rightarrow y_{j_{r-1}} \rangle$ is an optimal $(r-1)$-alignment of two prefixes of X and Y, i.e. $\langle x_1, \ldots, x_{i_r-1} \rangle$ and $\langle y_1, \ldots, y_{j_r-1} \rangle$.*
- *$\langle x_{i_{r+1}} \rightarrow y_{j_{r+1}}, \ldots, x_{i_l} \rightarrow y_{j_l} \rangle$ is an optimal $(l-r)$-alignment of two suffixes of X and Y, i.e. $\langle x_{i_r+1}, \ldots, x_m \rangle$ and $\langle y_{j_r+1}, \ldots, y_n \rangle$.*

This lemma is a direct result of the principal of optimality. To find an optimal l-alignment of two histories, we can first locate a match $(x_{i_r} \rightarrow y_{j_r})$ as a *pivot*. An optimal alignment then can be formed by concatenating the optimal $(r-1)$-alignment of the prefixes, the pivot, and the optimal $(l-r)$-alignment of the suffixes. Since Lemma 2 holds for any $1 \leq r \leq l$, we assume $r = 1$, i.e. the pivot is the first match of an alignment to be constructed. To construct an l-alignment of score equal or greater than s, a match $x_i \rightarrow y_j$ can be a pivot if $\sigma(x_i \rightarrow y_j) + G^l_{i+1,j+1} \geq s$. Algorithm 1 conducts a branch-and-bound search to enumerate those alignments. In each step, the algorithm identifies possible pivots, provided that $G^l_{i,j}$ is computed ahead using Eq. 3, and effectively prunes all alignments of suffixes $\langle x_{p+1}, \ldots, x_m \rangle$ and $\langle y_{q+1}, \ldots, y_n \rangle$ that cannot contribute to form a desired alignment. Algorithm 1 can be parametrized to find the following alignments:

1. All alignments of length l_1: call $Enum(a, b, l_1, 0, \langle\rangle)$.

Algorithm 1: Enumerate

Input : $X = \langle x_1, \ldots, x_m \rangle$, $Y = \langle y_1, \ldots, y_n \rangle$, l, s, A
Output: Enumerates desired alignments.

l: length of desired alignment
s: minimum score of desired alignment
A: empty ($\langle\,\rangle$) or $\langle x_{i_1} \rightarrow y_{j_1}, \ldots, x_{i_r} \rightarrow y_{j_r} \rangle$

Procedure Enum(X, Y, l, s, A)
if $l = 0$ **then**
  if $s \leq 0$ **then** print A ;
  return ;
if $A = \langle\,\rangle$ **then**
  $R_1 \leftarrow \{1, \ldots, m - l + 1\}$;
  $R_2 \leftarrow \{1, \ldots, n - l + 1\}$;
else
  $R_1 \leftarrow \{i_r + 1, \ldots, m - l + 1\}$;
  $R_2 \leftarrow \{j_r + 1, \ldots, n - l + 1\}$;
foreach *(p,q)* $\in R_1 \times R_2$ **do**
  if $(x_p \rightarrow y_q)$ *is a pivot* **then**
    A' = concat(A, $x_p \rightarrow y_q$) ;
    Enum(X, Y, $l-1$, $s - \sigma(x_p \rightarrow y_q)$, A');

2. k alignments of length at least l_1 with the highest scores: call Algorithm 1 with the top k scores of $G^l_{i,j}$ such that $l \geq l_1$, $i \leq m - l + 1$ and $j \leq n - l + 1$.

3. k longest alignments with a score more than a threshold: call Algorithm 1 for k pairs of length and score, where the length varies from $min(m, n)$ to 1 and the score is the desired threshold.

Modifying Algorithm 1 to accommodate a gap constraint is straightforward and we omit it here for brevity. In the next section, we show how to process similarity queries efficiently over a large database of histories.

4 Queries over Large Database of Histories

Consider the problem of efficiently evaluating similarity queries over a large collection of histories. A query is stated as a history with two parameters r and l. A history in the database is a candidate if it can form an alignment with the query history such that the alignment satisfies the r-neighborhood constraint and contains at least l matches. Ideally, we want to construct an index on histories, but this require having a metric distance function between histories. In an attempt to form a distance function from our alignment score in Eq. 1, we first define the similarity of two histories as a normalized score of their optimal alignments. Let A denote an optimal l-alignment of two histories X and

Y,

$$sim_l(X,Y) = \frac{f(A)}{min(|X|,|Y|)}.$$

Clearly $0 \leq sim_l(X,Y) \leq 1$, and a distance function between X and Y can be defined as

$$diss_l(X,Y) = 1 - sim_l(X,Y).$$

Proposition 1 *$diss_l(X,Y)$ does not satisfy the triangle inequality.*

Proof: As a counter example, let $h_1 = \langle\{a\},\{b\},\{x\}\rangle$, $h_2 = \langle\{b\},\{x\}\rangle$, and $h_3 = \langle\{b\},\{c\},\{x\}\rangle$. If we measure the similarity of observations using the Jaccard coefficient, and for $l = 2$:

$$diss_l(h_1,h_3) > diss_l(h_1,h_2) + diss_l(h_2,h_3) \quad \blacksquare$$

As a consequence of this proposition, any spatial access method for indexing histories may lose some qualifying candidates from the result set of queries. A straightforward alternative is to do a sequential scanning; but this is not efficient due to the large number of I/Os and the complexity of comparing two histories. A B+tree index on the length can filter those histories of length less than l which cannot be candidates. This will save a fraction of I/Os and computations; however, the similarity is still computed for non-qualifying histories. To prune some of those similarity computations, we propose two upper bounds for $sim_l(X,Y)$. The first upper bound is quick to compute but requires reading the histories. The second upper bound can be evaluated efficiently for a subset of the database using an index structure. In what follows, we assume that X is the query history and Y is a data history in the database, with m and n observations each.

4.1 A General Upper Bound

For each observation y_i of Y, let x_{i^*} be an observation with the highest similarity to y_i among all observations of X that are recorded in the r-neighborhood of y_i. In case no such observation exists, we assume that x_{i^*} is a null observation and $\sigma(x_{i^*} \rightarrow y_i) = 0$. Let S_l be a set that contains l observations of Y, such that for every pair of observations y_i and y_j, if $y_i \in S_l$ and $y_j \notin S_l$:

$$\sigma(x_{i^*} \rightarrow y_i) \geq \sigma(x_{j^*} \rightarrow y_j)$$

Lemma 3 *For two histories X and Y, $usim_l(X,Y)$ defined as*

$$usim_l(X,Y) = \frac{\sum_{y_i \in S_l} \sigma(x_{i^*} \rightarrow y_i)}{min(m,n)} \qquad (4)$$

provides an upper bound for $sim_l(X,Y)$.

Informally, $usim_l(X,Y)$ can be seen as the score of an optimal *relaxed* l-alignment. Each observation in X could be potentially matched with more than one observation of Y. The order of observations in individual histories may not be preserved completely in the alignment, i.e. there could be two observations x_{i_1} and x_{i_2} in X that are respectively matched with y_{j_2} and y_{j_1} of Y.

Compared to $sim_l(X,Y)$, $usim_l(X,Y)$ can be computed in less time. Let m_{2r} be the maximum number of observations recorded for X in a time interval of length $2r$. The upper bound can be computed in $O(n(m_{2r} + log\ l) + l)$ time, compared to $O(m_{2r}nl)$ which is required to compute the actual similarity. The upper bound can be used to prune non-qualifying histories before a similarity computation.

In the case of a k nearest neighbor (k-NN) query X, a data history Y can be filtered out safely when the upper bound $usim_l(X,Y)$ is less than the score of the k-th best candidate found so far. Otherwise, the similarity of the history and the query is computed and the list of k best candidates is updated if the similarity is more than the score of the k-th best candidate. After processing all histories, the result of the query is the list of k best candidates.

In the case of a range query X, a data history Y can be filtered out safely if its upper bound $usim_l(X,Y)$ is less than the threshold of the range query. Otherwise, the similarity of the history and the query is computed and the data history is included in the result of the query if the similarity is greater than the threshold.

For both queries, every history must be read before we can decide if a history can be pruned. In fact, the upper bound is computed even for histories that have no observation similar to any observation of the query. An interesting question is if it is possible to filter some histories prior to computing the upper bound. We believe that an exact answer to this question depends on functions σ and f. In the next subsection, we provide an affirmative answer to this question when σ is either the cosine measure or the extended Jaccard coefficient.

4.2 An Index-based Upper Bound for Sparse Observations

Our proposed upper bound in this section is aimed at sparse observations which are common, for instance, in market basket data where a transaction typically consists of a few items (out of the set of all possible items). Suppose the items are ordered, and an observation is described as a vector where the ith element of the vector gives the weight of item i; let's use the notation $\vec{x}$ to denote an observation. We propose an upper bound for $sim_l(\cdot)$ that takes advantage of the sparsity of the observations to reduce the number of histories that need to be scanned or compared to the query. Unlike $usim_l(\cdot)$, this new upper bound can be ef-

ficiently computed using an inverted index on observations. In many real-life applications where only a small subset of the histories in the database are similar to a query, our approach turns out to be more efficient than a sequential scan (as shown in our experiments).

Lemma 4 *Let I_X denote the set of every item with a non-zero weight in at least one observation of X. $\sigma^*(\vec{x}_{i^*} \rightarrow \vec{y}_i)$, defined as*

$$\sum_{t \in I_X} \vec{y}_i[t] \cdot max_j \left\{ \vec{x}_j[t] \;\middle|\; |ts(\vec{y}_i) - ts(\vec{x}_j)| \leq r \right\}$$

overestimates $\sigma(\vec{x}_{i^*} \rightarrow \vec{y}_i)$, where $\vec{y}_i[t]$ and $\vec{x}_j[t]$ denote the weights of an item t in $\vec{y}_i$ and $\vec{x}_j$, respectively.

Proof: Appears in the appendix.

Lemma 5 *For two histories X and Y, if $|Y| \geq l$,*

$$Usim(X,Y) = \frac{\sum_{i=1}^{|Y|} \sigma^*(\vec{x}_{i^*} \rightarrow \vec{y}_i)}{min(m,n)} \qquad (5)$$

provides an upper bound for $sim_l(X,Y)$.

Proof: While $usim_l(X,Y)$ considers the score of l best matches for the optimal relaxed l-alignment, $Usim(X,Y)$ considers $|Y| \geq l$ best matches. Furthermore, for each match, an upper bound of the score is considered. Therefore, $Usim(X,Y) \geq usim_l(X,Y) \geq sim_l(X,Y)$. ∎

Intuitively, this upper bound is the score of an optimal *relaxed* alignment that matches each observation $\vec{y}_i$ with the best observation that can be constructed from all observations of X in an r-neighborhood of $\vec{y}_i$. Indeed, $Usim(X,Y)$ can be computed efficiently using an inverted index that maps each item t in the domain to a list of (h_{id}, ts, w) triplets. Each such triplet indicates that for a history h_{id}, item t has a non-zero weight w in an observation recorded at timestamp ts. For each query X, the set of items I_X is extracted and $Usim(X,Y)$ is initially set to zero for all histories Y in the database. For each item t in I_X, the list associated with t is scanned from the inverted index. For each triplet $(Y, ts(\vec{y}_i), \vec{y}_i[t])$ in this list, the maximum weight of t in any observation of the query in the r-neighborhood of $\vec{y}_i$ is identified and $Usim(X,Y)$ is updated accordingly.

To use this upper bound for a range query, it is necessary to retrieve a history Y only if $Usim(X,Y)$ is greater than the threshold of the range query where X is a query history. Similarly, for a k-NN query, it is necessary to retrieve a history Y only if $Usim(X,Y)$ is greater than the similarity of the query and the k-th best candidate found so far. In both cases, $Usim(X,Y)$ can be computed using an inverted index on observations only. Both queries can also use $usim_l$ to further prune some histories not already filtered by $Usim$, since $Usim$ overestimates $usim_l$ and there can be still false positives.

5 Experimental Evaluation

In this section, we present the result of an experimental study of our approach on both real and synthetic data sets. We also examine some of the solutions developed for time series data and show why they are not applicable to the general problem discussed in this paper. We run experiments to show both the effectiveness of our scheme and the efficiency of our approach to process queries. The results confirm that our similarity measure is effective to retrieve histories with similar patterns and that our algorithms are efficient and scalable with the number of histories and the number of items. The experiments are performed on a machine with a single AMD/XP2600 CPU running Red Hat Linux, and all algorithms are implemented in C.

5.1 Datasets

We used three data sets in our experiments: a real dataset (the DBLP collection) and two synthetic datasets. A synthetic dataset was generated using a simplified model for changes between consecutive observations of a history and another dataset was generated using a modified version of the data generator for sequential market basket data [2].

In the DBLP collection [10], each journal or conference is treated as a sequence of observations. Each observation contains the set of terms in the table of contents of a journal issue or a conference proceeding (except author names). Terms are assigned weights using the *tf.idf* scheme. The timestamp of each observation is the year that the journal is published or the conference is held. The history for the VLDB conference, for instance, has 29 observations(as of March 20, 2004). From this dataset, we could extract $2,784$ histories which we will use to provide anecdotal examples of the naturalness of our similarity measure.

Synth1: is a synthetic dataset that contains histories of documents. We model each document as a set of terms, which in turn, can be represented using a bit string of length n with a *one* in position i indicating that term i is present and a *zero* indicating the absence of the corresponding term in the document. We further assume that the number of changes between two consecutive versions of a document (i.e. the insertion of new terms or the removal of some existing terms) follows a Poisson distribution [16], in that the numbers of changes in non-overlapping intervals are independent for all intervals. To make the next version of a document predictable from the current version, we assume that changes follow the gray code order, although other orders could also be considered. In other words, if the number of changes between version v_i and v_{i+1} is k, the bit string representation for v_{i+1} corresponds to the k-th bit string that follows v_i in gray code order. The dataset contains $20,000$ histories, $n = 8$, and the first observation of each history is

selected uniformly at random from the first 16 gray codes. For each history, the parameter for the Poisson distribution that generates the number of changes is selected uniformly from $[1, 10]$. The number of observations in each history is uniformly distributed in the range $[32, 64]$. We use this dataset to evaluate the effectiveness of our similarity measure in retrieving histories that are generated using nearly the same parameter.

Synth2: This dataset simulates a collection of customer purchase histories. We used the synthetic data generator introduced in [2], but also assigned a hypothetical timestamp to each transaction and a weight to each item in a transaction. This dataset contains $8,000$ histories. The number of distinct items is $1,000$. The average number of observations in each history and the average number of items in each observation is 10, which is the default setting used in data mining experiments. For each history, the timestamp for the first transaction, as well as the difference in timestamps for two consecutive transactions, is a uniformly distributed discrete random number in $[1, 5]$. The weight for each item in a transaction is a uniformly distributed random number in $[0, 1]$. Each observation is normalized so have a unit norm, so that we could use the upper bound we proposed in Section 4.2. We use this dataset to measure the performance and scalability of our algorithms.

5.2 Limitation of Techniques Developed for Time series

The large number of methods developed for indexing time series databases motivated us to model our problem as an equivalent similarity search problem in the time series domain. We treat each history as a trajectory in d-dimensional space, where d is the number of distinct items. The number of items we encountered in our experiments ($98,451$ for DBLP collection) was far beyond the number of dimensions that could be handled efficiently by methods developed to index multidimensional time series (e.g. [26, 25, 13]). Even considering dimensionality reduction cannot mitigate the problem completely. The Johnson-Lindenstrauss lemma [14] states that for any set of n points, there is a mapping from d-dimensional space to k-dimensional space such that the distances between points are not distorted by more than a factor of $(1 \pm \epsilon)$ with probability $O(n^{-2})$ provided that $k \geq 4(\epsilon^2/2 - \epsilon^3/3)^{-1} ln(n)$. We used this lemma since it provides a bound for the amount of distortion in terms of distance. From the DBLP dataset, we removed frequent terms and stopwords. We applied the Johnson-Lindenstrauss lemma and reduced the number of dimensions from $42,173$ to 515. The amount of distortion introduced in the dissimilarity of observations was close to 50%. Moreover, the dimensionality of the data was still too high for the algorithms that index multi-dimensional time series.

Table 2. Mean and Stand. Dev. of $MD(\lambda_q, n)$ for 1-NN and 10-NN queries

	$MD(\lambda_q, 1)$		$MD(\lambda_q, 10)$	
	Mean	Stand. Dev.	Mean	Stand. Dev.
sim_l	0.30	0.49	0.30	0.29
UnionAll	1.81	1.69	1.90	1.14
LAST	3.05	2.36	3.12	1.03

5.3 Effectiveness of sim_l

In this experiment, we wanted to investigate how sim_l compared to other alternatives in retrieving similar histories. We generated $2,000$ queries using the same mechanism used to generate Synth1. Using a k-NN query with $k = 10$, we retrieved 3 ranked lists, based on sim_l and two alternative similarity measures: LAST , which measured the similarity between the last observations of two histories, and UnionAll , which measured the similarity between two observations each formed by performing a union of all observations in the corresponding history. We used the Jaccard coefficient to measure the similarity between observations and l, the desired length of an alignment, is selected randomly from $[32, 64]$. Let λ_q be the parameter used to generate the query and λ_{r_i} be the parameter used to generate the history that is ranked i-th in the answer set. Since λ_q and λ_{r_i} are responsible for the change pattern of the corresponding histories, the difference between λ_q and λ_{r_i} must be small for two histories that have a similar change pattern. Therefore, we evaluated the mean deviation of λ_{r_i} from λ_q, defined as

$$MD(\lambda_q, n) = \frac{\sum_{i=1}^{n} |\lambda_{r_i} - \lambda_q|}{n}$$

to assess the effectiveness of the similarity measures. Table 2 shows the average and standard deviation of $MD(\lambda_q, n)$ for the best and top 10 results. According to the results, λ_{r_i} is expected to be closer to λ_q for sim_l. Since LAST only considers the last observation of each history, there is a high chance that two histories with different initial observations and generating parameters have exactly the same last observations. The major drawback of UnionAll is that the union of observations may include all the terms, which makes a history pretty much similar to any other history independent of the generating parameter. Moreover, LAST is not sensitive to the order of observations, i.e. all the permutations of a given history will be treated as if they are identical.

We conducted some experiments to examine the effectiveness of our similarity measure on real data (the DBLP

Table 3. Result of 10-NN query for the VLDB, the KDD, and the AAAI conferences

VLDB	KDD	AAAI
VLDB	KDD	AAAI
ICDE	PKDD	IJCAI
SIGMOD	DaWak	ECAI
DEXA	ICML	AAAI/IAAI
IDEAS	CIKM	FLAIRS
IEEE TKDE	FLAIRS	PRICAI
DASFAA	IEEE TKDE	Artificial Intelligence
CIKM	ICTAI	ICTAI
EDBT	ICDE	IEA/AIE
DEXA Workshop	ISMIS	GECCO

collection). We posed publications, either conferences or journals, as queries and retrieved a ranked list of similar publications as reported by our similarity measure. Similar publications share one or several topics of interest that change in time. Also, new approaches and ideas are mostly introduced and developed in similar publications. Therefore it is likely that similar change trends are observed in publications that belong to the same or related communities. Table 3 lists the result of 10-NN query for the VLDB, the KDD, and the AAAI conferences. The publications are focused on topics related to *databases* for the VLDB, topics related to *data mining* for the KDD, and topics related to *artificial intelligence* for the AAAI.

5.4 Pruning Power and Efficiency

In this section, we evaluate the performance of processing similarity queries over Synth2 dataset. Each query is a history which is selected randomly from the dataset. The parameters for each query (either k-NN or range query) are r and l, which specify an r-neighborhood constraint and the minimum number of desired matches in an alignment between a query and a data history, respectively. We compared a naive scan with three pruning schemes. Lp uses a B^+tree and evaluates similarity for histories that have more than l observations. LUBp uses $usim_l$ defined in Eq 4, in addition to the number of observations, to prune unnecessary computations. LINDp uses $Usim$, as defined in Eq 5, and the length of histories to read and evaluate the similarity only for a fraction of the histories of the dataset. Since our proposed approaches guaranty the returning of all qualifying histories (our upper bounds overestimate sim_l), we measure only pruning power (the fraction of dataset for which actual similarity is evaluated) and query response time. In each case, we report the average of preforming each experiment for 200 queries. For brevity, we report only the results when the cosine measure was used to quantify the similarity

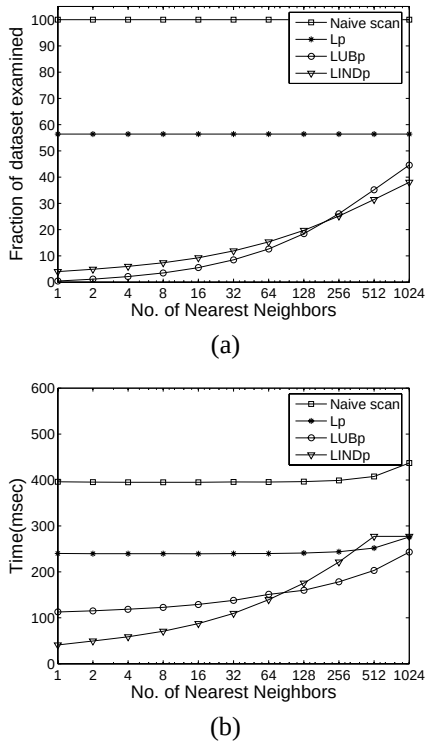
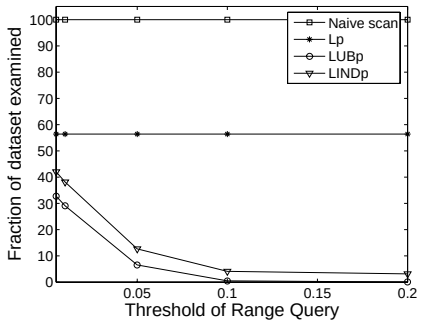


Figure 1. Pruning power (a) query processing time (b) for k-NN query.

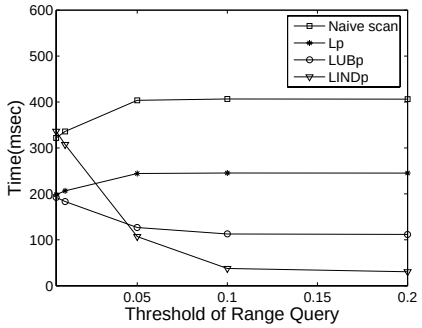
of observations, but we obtained very similar results when we also used the the Jaccard measure instead.

In the first experiment, we selected r and l randomly from $[1, 4]$ and $[1, 20]$, respectively. Figure 1(a) compares the pruning power for k-NN queries when k varies from 1 (i.e. nearest neighbor) to $1,024$ (which returns approximately 12% of the dataset). Using the length of the history results in pruning about 45% of the histories safely. Using the proposed upper bounds in addition to length results in a remarkable pruning. The pruning decreases as k increases since the similarity needs to be evaluated for a larger fraction of the database. The pruning reduces the response time for k-NN queries (Figure 1(b)). Note that although we observe a better pruning for LUBp when $k < 64$, LINDp has a better response time showing up to an order of magnitude speed-up over the naive scan for nearest-neighbor queries. This speedup occurs because LINDp avoids reading some of the non-qualifying histories. However, the speedup comes with the extra cost of performing random disk access, which dominates the cost of performing a sequential scan by LUBp when $k > 82$.

Figure 2(a) compares the pruning power for a range query when the threshold of the query is increased from 0.005 to 0.2. Note that LUBp shows a better pruning power

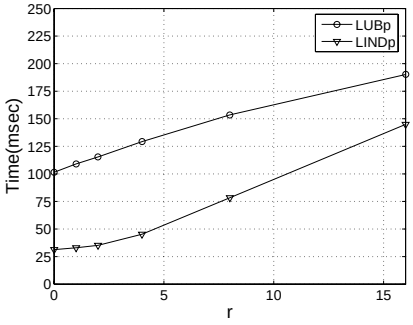


(a)

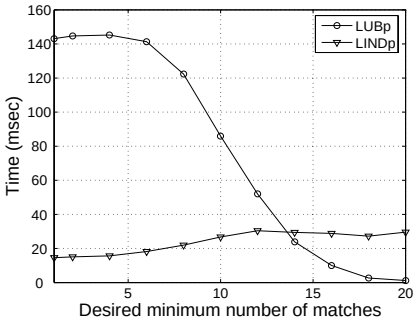


(b)

Figure 2. Pruning power (a) query processing time (b) for range query.



(a)



(b)

Figure 3. Time per query varying (a) the size of the r-neighborhood constraint (b) the desired minimum number of matches in an alignment.

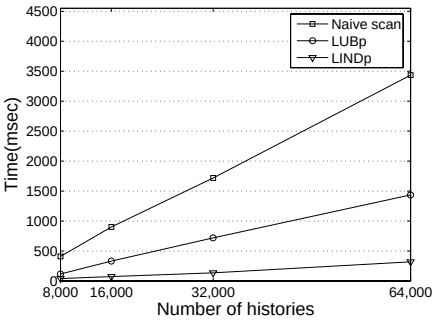
compared to LINDp since LINDp uses an over-estimation of the upper bound used by LUBp. However, LINDp reads a smaller fraction of the database, and shows a better response time as the number of histories to be scanned decreases (Figure 2(b)), making it more efficient than LUBp when the threshold is greater than 0.05.

We next investigate how the parameters r and l could affect the performance of our proposed methods. We report only the results for nearest-neighbor queries; however, we obtained very similar results for k-NN and ranges queries. First we varied r from 0 to 16 and for each r, we selected l randomly from $[1, 20]$ and picked (randomly) a history that had more than l observations as a query. According to Figure 3(a), increasing r increases response time for both LUBp and LINDp. However, LINDp slows down more quickly. Note that both methods overestimate sim_l using the score of a relaxed alignment. The chance of matching an observation of a data history with more than one observation of a query increases with r, making the upper bound less tight and consequently increasing the response time for both approaches. However, LINDp is more influenced since $Usim(X, Y) \geq usim_l(X, Y)$.

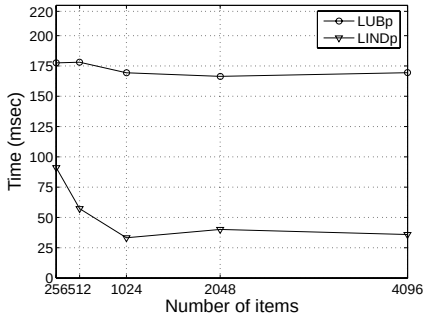
Next we changed l from 1 to 20 and selected r randomly from $[0, 4]$. For each l, we randomly selected a history (from the database) with at least l observations as a query. Figure 3(b) compares the running time of a nearest neighbor query using LUBp and LINDp for pruning. The upper bounds employed in both methods are close to actual similarity when l is small. However, the number of histories with more than l observations decreases as l increases, which helps LUBp to reduce the number of sequential disk accesses and redundant computations. For LINDp, the number of random disk accesses (due to using an index) does not change, but the time required for computing similarity increases, which justifies the observed trend.

5.5 Scalability Test

To compare the scalability of LUBp and LINDp compared to a naive scan, we increased the number of histories in the database from $8,000$ to $64,000$ and measured the average response time for a nearest neighbor query. Both r and l were selected randomly from ranges $[1, 4]$ and $[1, 20]$ respectively. As shown in Figure 4(a), both LUBp and LINDp scale up linearly, and the performance gap between these methods and the naive scan increases with the number of histories in the database. In another experiment, we



(a)



(b)

Figure 4. Time per query varying (a) the number of histories (b) the number of distinct items in database.

kept the number of histories fixed to $8,000$ but varied the number of items from 256 to $4,096$. Figure 4(b) depicts the average response time of a nearest neighbor query over this collection. Although LUBp is not significantly affected by this increase, we observe a response time for LINDp that decreases when the number of items increases from 256 to $1,024$ and remains unaffected after that. This is mainly because the dataset becomes sparse as the number of items increases. For instance, the probability that two observations, each with 10 random items, have a non-zero similarity is $1 - \prod_{i=1}^{10} \frac{247-i}{256} \approx 0.4422$ for 256 items and 0.1329 for $1,024$ items; LINDp takes advantage of this sparsity to reduce the query response time.

6 Related Work

Related research includes the work on detecting, representing, and querying changes. Chawathe et al. [7] propose a framework to represent changes by annotating the changed data using tags. A tag contains the type of change, a timestamp, and a reference to the modified values. The queries supported in this framework have the familiar *select-from-where* syntax over the annotated-graph that represents historical semistructured data. Chien et al. [9] represent the history of an evolving XML document using another XML document. Temporal and content-based queries are supported on the versions or changes of XML documents. Our work differs from the aforementioned work in that we focus on similarity queries on historical data, where the query itself is a history.

Similarity-based sequence matching has been an active research area in bioinformatics. There are several algorithms for solving sequence alignment problems. BLAST [3] and FASTA [17] are commonly used algorithms that employ heuristics to speed up similarity search on biological sequences. These algorithms are applicable to domains where the size of alphabet is small. Also, several algorithms, e.g. [6] have been proposed that use suffix trees to speedup the search. Constructing a suffix tree is not feasible for sequences of large alphabets.

For sequences of events, Wang et al. [27] study the problem of retrieving sequences that match a query sequence in terms of both the events and the timestamp of the events. Unlike our work, only identical events could be matched.

The idea of the longest common subsequence (LCS) has been used to measure the similarity between time series. Agrawal et al.[1] model time series as a sequence of equal length segments, and use the normalized length of the LCS of the resulting sequences of segments to measure the similarity. Vlachos et al.[25] use the normalized length of the LCS of time series to measure the similarity of multidimensional trajectories. In another work [26] they use the LCS of the minimum bounding envelope of trajectories to filter out a fraction of unnecessary LCS computations. Our work differs in that we measure the similarity when a history cannot be modeled as a time series.

7 Conclusions and Future Work

We have introduced a new domain-independent framework to both formulate and efficiently evaluate similarity queries over historical data. Our work generalizes a few concepts including the edit distance and the longest common subsequence to histories. This generalization is helpful; for instance, it enables us to find a common signature between histories based on their optimal alignments. We have developed some upper bounds for our similarity queries and one of our upper bounds has this interesting property that it makes use of an index even though it is not metric. Finding similar histories over order preserving data has many potential applications, of which we have considered historical market basket data and multi-version documents in our experiments. Our experiments on real and synthetic data confirm the effectiveness of our proposed scheme and the efficiency of our algorithms.

Many approaches to indexing time series use the GEM-

INI framework proposed by Faloutsos et al. In this framework, a high level representation of data (e.g. Minimum Bounding Regions for time series) and a metric measure, that underestimates the distances, are defined to prune the search. We are investigating the feasibility of applying one such framework to histories to possibly improve the performance of our similarity queries. Developing a metric measure to compare histories can also be useful for a possible clustering of the data.

References

[1] R. Agrawal, K.-I. Lin, H. S. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *Proceedings of the VLDB Conference*, pages 490–501, 1995.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the ICDE Conference*, pages 3–14, 1995.

[3] S. Altschul, W. Gish, W. Miller, E.Myers, and D. Lippman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[4] J. Bather. *Decision Theory: An Introduction to Dynamic Programming and Sequential Decisions*. John Wiley & Sons, 2000.

[5] D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *KDD Workshop*, pages 359–370, 1994.

[6] P. Bieganski, J. Riedi, J. V. Carlis, and E. F. Retzel. Generalized suffix trees for biological sequence data: Applications and implementation. In *Proceedings of the HICSS Conference*, pages 35–44, 1994.

[7] S. S. Chawathe, S. Abiteboul, and J. Widom. Representing and querying changes in semistructured data. In *Proceedings of the ICDE*, pages 4–13, 1998.

[8] S. S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom. Change detection in hierarchically structured information. In *Proceedings of the ACM SIGMOD Conference*, pages 493–504, 1996.

[9] S. Y. Chien, V. J. Tsotras, and C. Zaniolo. Efficient management of multiversion documents by object referencing. In *Proceedings of the VLDB Conference*, pages 291–300, 2001.

[10] DBLP. http://www.informatik.uni-trier.de/ ley/db/.

[11] D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.

[12] Internet Archive. http://www.archive.org.

[13] J. JáJá, J. Kim, and Q. Wang. Temporal range exploration of large scale multidimensional time series data. In *Proceedings of the SSDBM Conference*, pages 95–, 2004.

[14] W. B. Johnson and J. Lindenstrauss. Extensions of lipshitz mappings into a hilbert space. In *In Conference in modern analysis and probability*, pages 189–206, 1982.

[15] A. Mendelzon and A. Vaisman. Temporal queries in olap. In *Proceedings of the VLDB Conference*, pages 242–253, 2000.

[16] A. Papoulis. *Probability, random variables, and stochastic processes*. McGraw-Hill, third edition, 1991.

[17] W. Pearson. Rapid and sensitive sequence comparison with fastp and fasta. In *Methods in Enzymology*, volume 183, pages 63–98. Academic Press, 1990.

[18] D. Rafiei and A. Mendelzon. Similarity-based queries for time series data. In *Proceedings of the SIGMOD Conference*, pages 13–25, 1997.

[19] C. J. V. Rijsbergen. *Information Retrieval*. Butterworths, 1979.

[20] D. Sankoff and J. Kruskal. *Time warps, string edits, and macromolecules, the theory and practice of sequence comparison*. CSLI Publications, 1999.

[21] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[22] R. Snodgrass. *The TSQL2 temporal query language*. Kluwer Academic Publishers, 1995.

[23] A. Strehl and J. Ghosh. Value-based customer grouping from large retail data-sets. In *Proceedings of the SPIE Conference on Data Mining and Knowledge Discovery*, volume 4057, 2000.

[24] D. Toman. Point-based vs. interval-based temporal query languages. In *Proceedings of the ACM PODS Conference*, pages 58–67, 1996.

[25] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh. Indexing multi-dimensional time-series with support for multiple distance measures. In *Proceedings of the ACM SIGKDD Conference*, pages 216–225, 2003.

[26] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *Proceedings of the ICDE*, pages 673–684, 2002.

[27] H. Wang, C. S.Perng, W. Fan, S. Park, and P. S. Yu. Indexing weighted-sequences in large databases. In *Proceedings of the ICDE*, pages 25–36, 2003.

Appendix: Proof of Lemma 4

For two vectors $\vec{x}$ and $\vec{y}$, the cosines measure and the extended Jaccard coefficient are defined as:

$$\begin{aligned} s_{cosine}(\vec{x},\vec{y}) &= \frac{\vec{x}^T\vec{y}}{\|\vec{x}\|_2\|\vec{y}\|_2} \\ s_{Jaccard}(\vec{x},\vec{y}) &= \frac{\vec{x}^T\vec{y}}{\vec{x}^T\vec{x}+\vec{y}^T\vec{y}-\vec{x}^T\vec{y}} \end{aligned}$$

Where $\|\vec{x}\|_2$ denote the L_2-norm of $\vec{x}$. For vectors $\vec{y_i}$ and $\vec{x_{i^*}}$ of unit norm:

$$\begin{aligned} s_{Jaccard}(\vec{x_{i^*}},\vec{y_i}) &\leq s_{cosine}(\vec{x_{i^*}},\vec{y_i}) \\ &= \sum_{t\in I} \vec{x_{i^*}}[t]\vec{y_i}[t] \end{aligned} \quad (6)$$

For any observation $\vec{x}_{i^*}$:

$$\vec{x}_{i^*}[t] \leq max_j \left\{ \vec{x}_j[t] \;\middle|\; |ts(\vec{y_i}) - ts(\vec{x}_j)| \leq r \right\} \quad (7)$$

Replacing $\vec{x_{i^*}}[t]$ in Eq. 6 with the right hand side of Eq. 7 will establish the proof. ∎