# Optimal Distance Bounds for Fast Search on Compressed Time-Series Query Logs

MICHAIL VLACHOS
IBM Zürich Research Lab
SULEYMAN S. KOZAT
Koç University
and
PHILIP S. YU
University of Illinois at Chicago



Consider a database of time-series, where each datapoint in the series records the total number of users who asked for a specific query at an internet search engine. Storage and analysis of such logs can be very beneficial for a search company from multiple perspectives. First, from a data organization perspective, because query Weblogs capture important trends and statistics, they can help enhance and optimize the search experience (keyword recommendation, discovery of news events). Second, Weblog data can provide an important polling mechanism for the microeconomic aspects of a search engine, since they can facilitate and promote the advertising facet of the search engine (understand what users request and when they request it).

Due to the sheer amount of time-series Weblogs, manipulation of the logs in a compressed form is an impeding necessity for fast data processing and compact storage requirements. Here, we explicate how to compute the lower and upper distance bounds on the time-series logs when working directly on their compressed form. Optimal distance estimation means tighter bounds, leading to better candidate selection/elimination and ultimately faster search performance. Our derivation of the optimal distance bounds is based on the careful analysis of the problem using optimization principles. The experimental evaluation suggests a clear performance advantage of the proposed method, compared to previous compression/search techniques. The presented method results in a 10–30% improvement on distance estimations, which in turn leads to 25–80% improvement on the search performance.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database Applications—*Data mining*; E.4 [**Data**]: Coding and Information Theory—*Data compaction and compression*

General Terms: Algorithms

Authors' addresses: M. Vlachos, IBM Zürich Research Lab, Säumerstrasse 4, Rüschlikon, Switzerland; email: michalis0@gmail.com; S. S. Kozat, Department of Electrical Engineering, Koç University, Istanbul, Turkey; P. S. Yu, Department of Computer Science, University of Illinois at Chicago, Chicago, IL.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2010 ACM 1559-1131/2010/04-ART6 $10.00
DOI 10.1145/1734200.1734203 http://doi.acm.org/10.1145/1734200.1734203

**ACM Reference Format:**
Vlachos, M., Kozat, S. S., and Yu, P. S. 2010. Optimal distance bounds for fast search on compressed time-series query logs. ACM Trans. Web 4, 2, Article 6 (April 2010), 28 pages.
DOI = 10.1145/1734200.1734203 http://doi.acm.org/10.1145/1734200.1734203

## 1. INTRODUCTION

Internet search engines receive daily vast numbers of user queries. These search requests are typically recorded in some aggregate form, which the search engines later analyze in an effort to constantly improve the user experience. This is achieved through effective modeling and understanding of the user search preferences and their evolution over time. The work presented here deals with the compression of time-series weblogs and their efficient search. We consider temporal sequences that capture the daily demand of search queries. Figure 1 depicts two such sequences, describing the demand at a search engine for the queries *xbox* and *playstation*, during the period of one year. This representation of a query can highlight important data parameters; first, one can notice that the two queries exhibit a similar demand pattern. This implies that the two queries are semantically related, which is true since both keywords describe gaming consoles. Secondly, the temporal representation of a query reflects important trends. For the specific example, one can safely deduce that there is a greater demand for game consoles during Christmas. Generally, as previous studies note: "user behavior is deeply related to search keyword[s]" [Otsuka et al. 2004]. It is great interest to distill a user's search behavior because it can prove beneficial in a variety of applications:

(1) *Search engine optimization:* Understanding the semantic similarity between keywords can assist in constructing more accurate keyword taxonomies and achieving better clustering of keywords [Otsuka and Kitsuregawa 2006; Ziegler et al. 2006]. This can serve in providing better search results [Kage and Sumiya 2006] and ultimately help understand the true relationship between Web pages. A number of features can assist in this process, such as repetition in the search behavior [Sanderson and Dumais 2007], something that is easily conveyed by the temporal representation of the query demand.
(2) *Keyword recommendation:* Related queries are manifested by similar demand patterns. A search engine can exploit this characteristic for offering a "maybe you would also be interested in this" functionality. As an illustrative example, Figure 2(a) shows some of the queries with similar demand patterns as the keyword 'cinemas'. All the results are highly interpretable and include queries such as *movie guide*, *hollywood.com*, and *roger ebert*.
(3) *Better spelling correction:* No dictionary or ontology can cover the wide range of keywords that appear on the web. However, relationships between

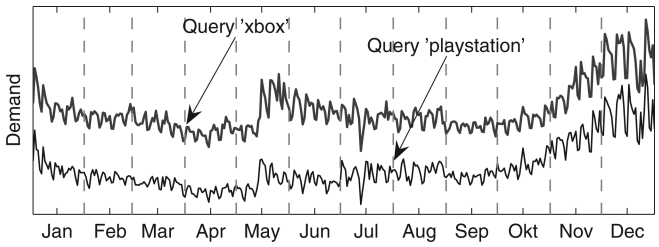


Fig. 1. Daily demand over a year for queries *xbox* and *playstation*.

keywords can be deduced by the systematic study of query logs and search engines [Chen et al. 2007; Bollegala et al. 2007]. Figure 2(b) illustrates an instance of such an example, for the query *florida* and the misspelled keyword *flordia*.

(4) *Identification of news events:* Query logs can help understand and predict behavioral patterns [Adar et al. 2007]. Important events usually manifest themselves as bursts in the query demand [Kleinberg 2002; Vlachos et al. 2004; Wang et al. 2007]. News travel fast, and web queries travel even faster. By identifying increasing demand in a query, search engines can accurately pinpoint developing news events. Such an example is indicated on Figure 3, which shows that during May 2002 there was an increase in the demand for the keyword *Spiderman*. This occurred because the movie *Spiderman 1* was released in the theaters during May of that year.

(5) *Advertising impact:* The financial aspect of search engines is materialized by the carefully selected matching of keywords to advertisements. Semantic clustering of queries can, first, assist the search engine in recommending related keywords to the advertisers. Secondly, seasonal query demand can help define in a more relevant way the price of an advertisement, by elevating the price during times of greater demand for the keyword. This paradigm is similar to the pricing of the TV or radio advertisements, where prime-time commercials are valued higher than the remaining time slots.

A common denominator in all of the above applications is a set of operations that allow for the effective storage and retrieval of the weblog data. Given the excessive amount of collected data, there is a pragmatic need for effective data compression. Popular search engines like Google, MSN, and Yahoo! have data retention periods that lie in the range between 13 and 18 months.[1] Very recently (Dec. 2008), Yahoo! announced that it will reduce the retention time of users search record to 3 months.[2] This is apparently another

[1]`http://googleblog.blogspot.com/2007/06/how-long-should-google-remember.html`

[2]`http://googlewatch.eweek.com/content/data_retention_policy/yahoos_data_retention_move_puts_pressure_on_google_microsoft.html`

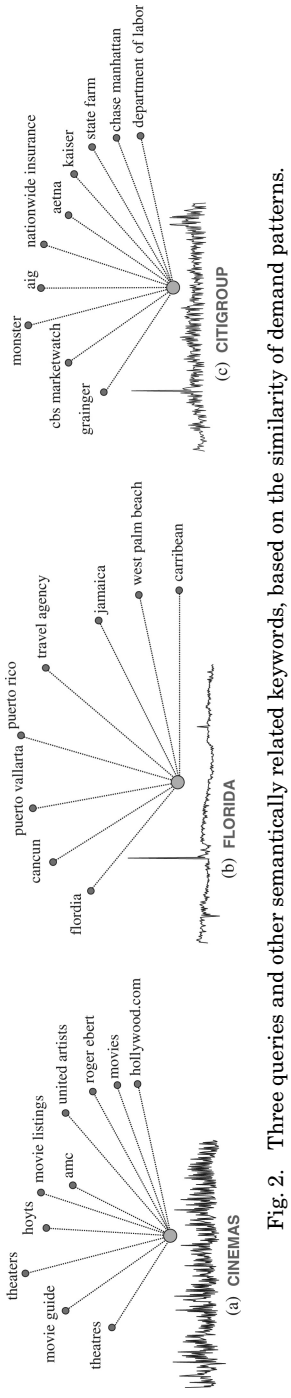


Fig. 2. Three queries and other semantically related keywords, based on the similarity of demand patterns.

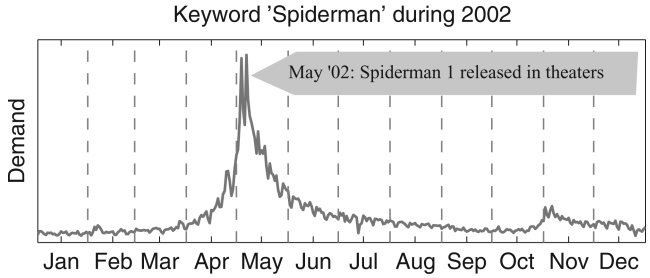


Fig. 3. Daily demand for the query *Spiderman* during 2002.

indication of the exploding data sizes and the forceful need for data compression. However, data compression on its own has little to offer if it cannot be combined with analytics and search mechanisms that work directly on compressed domain.

We propose to leverage the smooth and periodic nature of the Weblog data, to offer a highly effective data compression scheme of the temporal patterns. The Fourier coefficients with the highest energy are utilized which can effectively capture most of underlying signal's energy. We demonstrate with various examples that such a compressed data representation can accurately describe most of the data variability and also capture important patterns in the log files. While this provides an excellent compression technique, comparison between the compressed sequences is difficult since they are described by a (possibly) diverse set of coefficients, according to their dominant frequencies. In this work we present techniques that overcome this obstacle. A major contribution of this work is a technique for calculating the *optimal* distance bounds that can be derived using the aforementioned compressed representations. The algorithm is based on solid optimization principles and offers a significant boost in the search performance compared to the current state of the art. The technique that we propose here is also of independent interest for general time-series data; it is applicable on any numeric sequence data and on any orthonormal data transformation.

The article is structured as follows; we begin with an overview of relevant work from the web search, information retrieval and data mining communities in Section 2. We continue with a discussion about techniques for compressing Weblog sequences using orthonormal transforms and how to improve the search performance through use of upper and lower bounding techniques (Section 3). Subsequently, Section 4 delineates our algorithm for optimally bounding the distance between compressed Weblog sequences, first providing the intuition behind it, and then a formal proof. We illustrate how our scheme could be combined with indexing techniques and also provide performance optimizations through use of "early termination." Detailed experiments examining convergence properties, scalability, tightness of bounds and pruning efficiency are the focus of Section 5. Finally, Section 6 concludes this paper and elucidates directions for future work.

## 2. RELATED WORK

Previous work considered various applications on temporal query sequences. Sun et al. [2007] examine the discovery of causal relationships across query logs by deploying an event causality test. Chien and Immorlica [2005] and Lie et al. [2006] study similarity search and clustering in query data based on metrics such as correlation and periodicity. While these utilize linear metrics to quantify the similarity, Adar et al. [2007] examines the use of nonlinear metrics such as Time-Warping. In Liu et al. [2006] the similarity between query logs is quantified by comparing the similarity of ARIMA coefficients. Richardson [2008] examines how long-term query patterns can help build concept hierarchies and also assist in deciphering the sociological behavior of Web users. Rode and Hiemstra [2006] examine the applicability of temporal query profiles as relevance feedback mechanism. Finally, Zhao et al. [2006] examine a similar application of search on temporal logs but using click-through data. An extensive study of temporal query logs is presented in Beitzel et al. [2007], and techniques are delineated for analyzing the change of trends over time. However, none of this work examines how to tailor search based on compressed representations of the Weblogs. Our work, in that sense, is complementary to all the above approaches, by allowing them to scale up to even larger dataset sizes.

In the data-mining community, search on time-series under the Euclidean metric has been broadly studied [Agrawal et al. 1993; Rafiei and Mendelzon 1998; Wang and Wang 2000] but, typically, compression using the first Fourier or wavelets is considered. Vlachos et al. [2004] study the use of diverse sets of coefficients, but this is the first work that offers the *tightest possible* lower/upper bounds. In the experimental section we offer a thorough performance comparison of our approach across the most predominant methodologies in the time-series literature.

Indexing of time-series data has also been a well-studied topic in the data-mining literature [Assent et al. 2008; Kahveci and Singh 2001; Yi and Faloutsos 2000; Cai and Ng 2004]. The majority of such approaches consider that each compressed sequence is represented by the same set of coefficients thus are not applicable for our setting. In the course of the article we show how metric tree structures can be modified to operate on bounded pairwise distances and hence accommodate our technique.

## 3. SEARCHING TEMPORAL LOG DATA

Before we explain how Nearest-Neighbor searches are performed over query logs, we begin with a short overview of how these temporal demand patterns are constructed on the first place.

### 3.1 Query-Log Construction

Search engines operate multiple proxy sites that service local search requests. When a user enters a query term, a hash counter corresponding to the specific query entry is increased, and for each day the total number of users having requested the specific term(s) is recorded. The values from each site are

aggregated and the total count of daily requests per query is recorded. In order to avoid the excessive storage of query logs, only those query terms that are above a certain threshold are stored. This process is repeated daily, and the global temporal trend of the total set of posed queries is stored in a sequence form. Such a representation is also privacy aware only the aggregate trend is recorded, but no user-specific query behavior is ever stored.

### 3.2 Generic Search Algorithm

We consider a database $\mathcal{DB}$ that stores the temporal query sequences $x^{(i)}$, $i = 1 \ldots M$. The general problem that we examine can be abstracted as follows: A user is interested in finding the $k$ most similar sequences to a given query sequence $q$, under a certain distance metric $d$. This operation is also known as $k$-Nearest-Neighbor (NN) search. It is a core function in database search and also a fundamental operation in many data-mining and machine-learning algorithms, including classification (NN-classifier), clustering, and so on. Therefore, the provision of such functionality is important for any system that attempts to analyze the data or make useful deductions. The distance function $d$ that we consider in this work is the Euclidean distance. More flexible measures, such as time-invariant distances [Vlachos et al. 2005] (essentially a Euclidean distance on the periodogram) could also be used with little to no modifications of our main algorithm. However, for ease of exposition here we focus on the Euclidean distance,[3] which is also the distance measure of preference in most of the related work [Chien and Immorlica 2005; Lie et al. 2006].

In Figure 2 we plot some of the nearest neighbors of 3 queries; *cinemas*, *florida* and *citigroup*. We observe that the results have a high semantic affinity with the posed query. For example, the query citigroup (Figure 2(c)) returns other financial or insurance companies. Additional examples on the quality of the retrieved matches can be found in the experimental section.

Search operations can be quite costly, especially for cases where the cardinality of the sequences is quite extensive and the sequence length is also substantial (both statements are true for our scenario). This is observed, because sequences need to be retrieved from disk in order to be compared to the query $q$. An effective way to mitigate this cost is to retain a smaller, compressed representation of the sequences, which will be used as an initial prefiltering step. The set of compressed sequences could be small enough so that it can be kept in-memory, hence lending an even greater performance speedup. Essentially, one is employing a *multilevel* filtering mechanism. When examining the compressed sequences, we obviously cannot derive the exact distance between the query $q$ and any sequence $x^{(i)}$ in the database. Underestimates and upper estimates of the distance will be calculated, which in the literature are also known as *lower* and *upper bounds* on the distance function. Using these bounds, a superset of the $k$-NN answers will be returned, which will be verified against the uncompressed disk-resident sequences. These will be fetched and compared with the query, so that the exact distances can be

[3]Note that correlation is also an instance of Euclidean distance on properly normalized sequences.

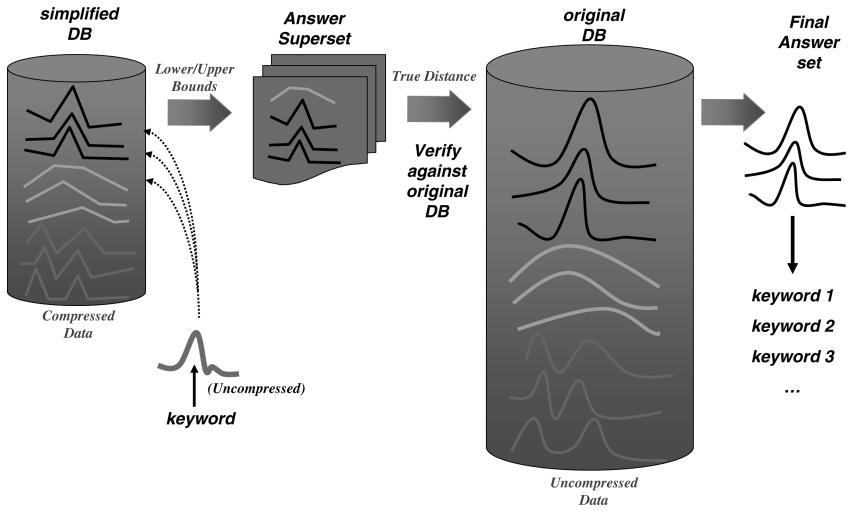


Fig. 4. General framework for speeding up Nearest-Neighbor search operations.

computed. This methodology is very widely used in the data mining field and it is the methodology also used in this work. These steps are summarized in Figure 4.

### 3.3 Use of Upper and Lower Bounds

Lower/upper bounds on the distance function serve three purposes: (1) eliminate from examination candidate sequences that are provably worse than the current best match during the search procedure; (2) dictate a search order of the disk-resident sequences, so that more promising candidates are examined first, hence providing at an early stage of the search a good candidate match. This will help eliminate subsequent distance sequences from examination; (3) provide guarantees that the initial data filtering using the compressed data, will return the same outcome as when scanning sequentially the original uncompressed data.

Consider that we are seeking the 1-NN match of the query $q$. By swiftly scanning the compressed representations lower and upper bounds of $q$ against all sequences can be derived. We extract the minimum upper bounds $UB_{min}$ and all sequences that have lower bound greater than $UB_{min}$ can be safely discarded, since obviously a better match can be found (in the form of the sequence with upper bound equal to $UB_{min}$). Next, the uncompressed sequences are retrieved from disk in the order suggested by the lower bounds (LB's), since sequences with smaller LB's are more likely to be closer to the query $q$. The true distance of each sequence to the query is evaluated and the best-so-far match is possibly updated. Once the LB of the currently retrieved sequence is greater than the (true) distance of the best-so-far match, then the search can be terminated, since all the remaining sequences are guaranteed to have greater distance than the best-so-far candidate sequence.

In the general case, where one is searching for the k-Nearest-Neighbors ($k > 1$), the only change introduced in the preceding process is the introduction of a priority queue that holds the $k$ best results, and the algorithm prunes the search space based on the distance of the $k$-th best-so-far match.

The given search procedure can be further improved through various techniques, such as, for example, creation of an indexing scheme. However, the steps that we described are rudimentary in most all search and indexing techniques [Keogh 2002; Weber et al. 1998]. Additionally, the aforementioned search procedure constitutes a bias-free approach to evaluate the search performance of a technique, since it does not depend on any implementation details. We employ the aforementioned search procedure in the experimental section, in order to provide an unbiased performance estimator between various lower/upper bounding techniques, since it does not depend on any implementation details, but merely relies on the tightness of the derived bounds.

Obviously, techniques that provide tighter bounds will be able to offer better pruning power and enhanced search performance. Later on, we will provide an algorithm that computes the *tightest possible* lower and upper bounds, when utilizing the high-energy coefficients of weblog sequences. In the upcoming section we describe how this compression is achieved.

### 3.4 Compressing Weblogs

Query demand patterns exhibit a smooth and highly periodic nature [Vlachos et al. 2004], therefore it is natural to compress such temporal data utilizing the Fourier transform. Wavelets or Principal Components Analysis (PCA) could have also invariably been used without any change in the algorithms that will be described later on. In fact, everything that will be mentioned henceforth is applicable on any *orthonormal transform*.

We begin with some notation first and a brief overview of the Fourier transform. We denote each query as a time-series $x = \{x_0, x_1, \ldots, x_{N-1}\}$ and the Fourier transformation of $x$ by the capital letter $X$.

*Discrete Fourier Transform*. The normalized Discrete Fourier Transform (DFT) of a sequence $x$ is a vector of complex numbers $X(f)$:

$$X(f_{n/N}) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N}, \quad n = 0, 1, \ldots, N-1.$$

Each of the complex numbers encodes the amplitude and phase of a sinusoid with frequency $f$ and the sum of all sinusoids reconstructs the original sequence.

*Periodogram*. The energy of all Fourier coefficients is denoted by the periodogram $P$, a vector comprised of the squared magnitude of the coefficients:

$$P(f_{n/N}) = \left|\left|X(f_{n/N})\right|\right|^2, \quad n = 0, 1, \ldots, \left\lceil \frac{N-1}{2} \right\rceil.$$

The most dominant frequencies appear as peaks in the periodogram and correspond to the coefficients with the highest magnitude. From here on, when we

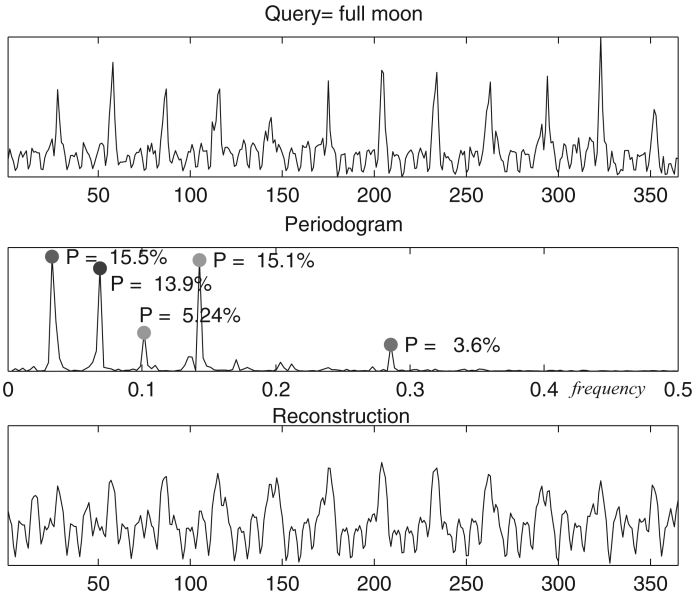


Fig. 5. We observe that many of the query weblogs can be characterized by few dominant frequencies. For the query *Full Moon* 5 major frequencies account for more than 50% of the signal's energy.

refer to the best or largest coefficients, we mean the ones that have the highest energy and correspond to the tallest peaks of the periodogram. One could construct an approximation of the original signal using any set of frequencies, but typically the peaks on the periodogram would correspond to the important data frequencies. These are the coefficients that we use to compress each query sequence.

*Example*. Consider the demand pattern of the query *Full Moon* in Figure 5. Its periodogram (which summarizes the energy of the signal frequencies) consists of very few frequencies that hold the majority of the signal energy. For example, 2 of the peaks on the periodogram correspond to a monthly and a weekly periodicity. That is, people ask for the specific query approximately every month and every week. Since the data is composed of few frequencies, the data can be compressed or summarized accurately using minimal information. At the bottom of Figure 5 we also show the high quality reconstruction that is achieved when using the 5 coefficients with the highest energy. We will utilize the Fourier coefficients with the highest energy signature for compressing the underlying temporal demand pattern of each query.

Therefore, each compressed query sequence $X$ will be described by a set of $c$ coefficients that hold the largest energy. The vector describing the positions of those coefficients in $X$ is denoted as $p^+$, while the positions of the remaining ones as $p^-$ (that is $p^+, p^- \subset [1, \ldots, N]$). For any sequence $X$, we will also store in the database the vector $X(p^+)$ or equivalently $X^+$. If $Q$ is a query in the transformed domain, then $Q(p^+)$ (or $Q^+$) describes a sequence holding the

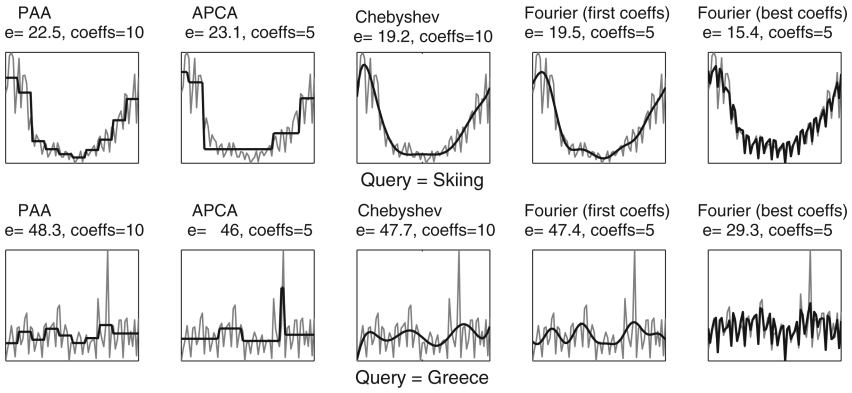


Fig. 6. Comparison of various compression techniques on query weblogs. The approximation error $e$ is very low when using the Fourier coefficients with the highest energy.

equivalent coefficients as the vector $X(p^+)$. Similarly, $Q(p^-) \equiv Q^-$ is the vector holding the analogous elements of $X(p^-) \equiv X^-$.

*Example*. Suppose $X = \{(1+2i), (2+2i), (1+i), (5+i)\}$ and $Q = \{(2+2i), (1+i), (3+i), (1+2i)\}$. The magnitude vector of $X$ is: $||X|| = \{2.23, 2.82, 1.41, 5.09\}$. Then, $p^+ = \{2, 4\}$, $X(p^+) = \{(2+2i), (5+i)\}$ and $Q(p^+) = \{(1+i), (1+2i)\}$.

Previous work considers compression of time-series data using the same set of orthogonal coefficients, because this allows for easier comparison of the respective coefficients and direct adaptation of traditional indexing structures, such as R-trees. Figure 6 depicts two query examples and their approximation for various time-series compression techniques, such as Piecewise Aggregate Approximation (PAA) [Yi and Faloutsos 2000], Adaptive Piecewise Constant Approximation (APCA) [Keogh et al. 2001], Chebychev Polynomials [Cai and Ng 2004] and the use of first Fourier coefficients [Agrawal et al. 1993; Rafiei and Mendelzon 1998]. We observe that the sequence reconstruction error $e$ using the best Fourier coefficients (last column) is very low, indicating the merits of the proposed compression technique.

In addition, to the best coefficients of a sequence we will also record the energy of the discarded coefficients: $e_X = ||X^-||^2$, which corresponds to the sum of squares of the omitted coefficients. This quantity represents the error in the compressed representation, or, equivalently, the amount of energy in the discarded coefficients.

### 3.5 Searching Compressed Weblogs

We now have all the elements for describing our problem setting. Given an uncompressed query $q$, we need to find the closest sequences $x$ in the database, based on the Euclidean distance ($L_2$-Norm). Parseval's theorem dictates that the Euclidean distance is the same whether computed in the time or in the frequency domain. The preservation of energy holds for any orthonormal transform (wavelets, PCA, etc), so anything mentioned from now on, is applicable on

a variety of data transforms. The distance can be decomposed as follows:

$$\begin{aligned} D(x,q)^2 &= D(X,Q)^2 \quad \text{(Parseval)} \\ &= D(X^+,Q^+)^2 + D(X^-,Q^-)^2 \\ &= \|X^+ - Q^+\|^2 + \|X^- - Q^-\|^2. \end{aligned} \tag{1}$$

The computation of the first part of the distance is trivial since we have all the required data. However, the second part $\|X - Q\|^2$ cannot be calculated since $X^-$ (the discarded coefficients) is unknown. Nonetheless, because we have compressed each sequence $X$ using the best coefficients, by construction we know that the magnitude of each of the coefficients in $X^-$ is less than the smallest magnitude in $X^+$. We use $minPower = ||X^+_{min}||$ to denote the magnitude of the smallest coefficient in $X^+$.

We can estimate the range of values within which $\|X^- - Q^-\|^2$ lies, by expressing it as an an optimization problem; in specific, as two optimization subproblems. As a maximization problem when considering the upper-bound distance, and as a minimization problem when attempting to establish the lower-bound distance

$$\|X^+ - Q^+\|^2 + \min_{X^-} \|X^- - Q^-\|^2 \le \|X - Q\|^2 \text{ and}$$
$$\|X - Q\|^2 \le \|X^+ - Q^+\|^2 + \max_{X^-} \|X^- - Q^-\|^2.$$

For example, if we wish to discover a tight upper bound on the distance, we need to provide a solution for the following optimization problem:

$$\max_{X^-} \|X^- - Q^-\|^2 \text{ such that} \tag{2}$$
$$\|X^-\|^2 = e_X \tag{3}$$
$$\|X_i^-\| \le minPower, \tag{4}$$

and

$$\min_{X^-} \|X^- - Q^-\|^2 \text{ such that} \tag{5}$$
$$\|X^-\|^2 = e_X \tag{6}$$
$$\|X_i^-\| \le minPower, \tag{7}$$

where $X_i^-$ is the $i$th component of the $X^-$.

The algorithm that we provide is the *optimal*, that is, the bounds that we compute are the *tightest possible to the original distance under the Euclidean Metric*. We provide this result when a sequence is compressed using the $k$ coefficients with the highest energy. To our best of knowledge, this is the first work that offers such bounds. First we provide an intuition regarding our solution to the problem. Initially, on 2-dimensions and then on n-dimensions. In the appendix we include a formal proof regarding the optimality of our solution.

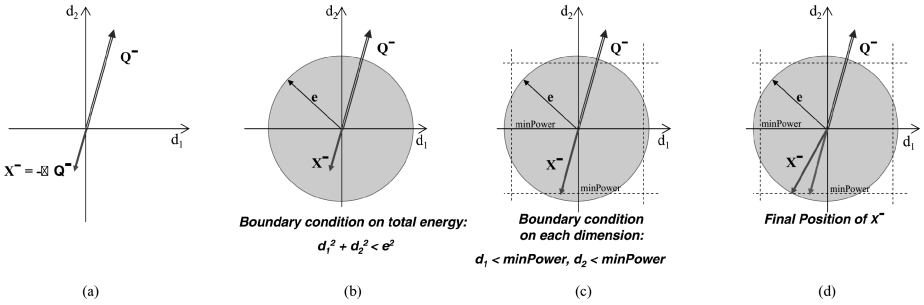


Fig. 7. Illustration of the intuition behind our algorithm on 2 dimensions.

## 4. OPTIMAL DISTANCE BOUNDS

### 4.1 Algorithm Intuition on 2D

We demonstrate the optimal solution with a simple example. For this example we assume that $X$ and $Q$ are 2-dimensional real vectors. We first find the optimal upper bound and later the optimal lower bound. For the optimal upper bound calculation, $\|Q^+ - X^+\|$ is known and we want to find

$$\max_{X^-} \|X^- - Q^-\|^2 \tag{8}$$

such that $e_X = \sqrt{(X_1^-)^2 + (X_2^-)^2}$ and $\|X_i^-\| \leq minPower$, $i = 1, 2$, where $X^- = [X_1^- \ X_2^-]^T$ and $Q^- = [Q_1^- \ Q_2^-]^T$.

Intuitively, given the query $Q^-$, the vector that will maximize $\|Q^- - X^-\|^2$ should be on the opposite direction of $Q^-$, i.e., $X^- = -\alpha Q^-$ for some $\alpha > 0$, as seen in Figure 7(a). Let's also plot on the same figure the two constraints that we have:

(1) Notice that the constraint on the total available energy $e_X$ essentially is translated into a circle on the 2D plane (Figure 7(b)). Therefore the unknown vector $X^-$ should always lie within this circle, otherwise the energy constraints will be violated.
(2) The constraint on each coefficient of $X^-$ indicates that they should not exceed $minPower$, therefore cannot go further than the dotted vertical and horizontal lines at position $minPower$ on the two dimensions/axes, $d_1$ and $d_2$ (Figure 7(c)).

The algorithm proceeds as follows; we begin to scale $X^-$ in the opposite direction of the known $Q^-$ by increasing $\alpha$, so as maximize the distance $\|Q^- - X^-\|^2$. Now, one of two things may happen. Either we hit on the $minPower$ boundary on one of the axes or we cross the circle circumference indicating the total energy (whichever is violated first). As indicated in Figure 7(c), suppose that we encounter first the boundary condition on one of the axes, for example, on axis $d_2$. Then we keep the corresponding dimension fixed at $\|X_2^-\| = minPower$, that is, $X_2^- = -minPower$, and only scale the vector $X^-$ on the remaining dimensions until we use all the remaining energy or until we encounter another boundary

condition. So, now we start moving $X^-$ along the $d_1$ dimension. We can only scale it up to a certain point, because we are faced with the total energy boundary (the circle). At that point, the search stops because all conditions of the maximization problem have been satisfied.

In a similar fashion, if we want to find the *lower bound*, we have to solve

$$\min_{X^-} \|X^- - Q^-\|^2$$

such that $e_X = \|X^-\|$ and $\|X_i^-\| \leq minPower$, $i = 1, 2$. However, intuitively, given the query $Q^-$, the vector which will minimize $\|Q^- - X^-\|^2$ should be on the same direction of $Q^-$, i.e., $X^- = \alpha Q^-$ for some $\alpha > 0$. Since, the boundary conditions are symmetric, proceeding similarly to the maximization problem, we observe that the vector $-X^{-,*}$ yields the minimizer solution where $X^{-,*}$ is the solution to the maximization problem.

Therefore, we do not have to solve the optimization problem twice, but only once, since the two optimization problems are identical.

### 4.2 Algorithm on n-Dimensions

We now show how the algorithm operates in n dimensions, to allow better exposition of our ideas. We depict the maximization problem.

Figure 8(a) shows the known vector $Q^-$ and the (unknown yet) vector $X^-$ which we attempt to estimate. On the right side of the figure we also depict a bucket indicating the available energy that we can allocate on $X^-$. In the previous example, we mentioned that vector $X^-$ needs to be on the opposite direction of vector $Q^-$, which translates to creating a rescaled vector of $Q^-$ along that direction. $X^-$ is rescaled until all energy is used up (Figure 8(b)). If certain coefficients exceed the *minPower* constraint, they are truncated/fixed to *minPower* (Figure 8(c)). The energy that is alloted for the coefficients that are now fixed is subtracted from the total available energy (Figure 8(c)). For the remaining coefficients we repeat the same process, as shown in Figures 8(d),(e), and (f), until all the available energy is used, or all the unknown coefficients are approximated (fixed).

The configuration described above is a *water-filling* solution [Cover and Thomas 1991; Bertsekas 2000] and it is shown to be optimal in the Appendix. In Figure 9 we provide a pseudocode of the algorithm that we just described.

Therefore, the described water-filling algorithm will provide the *tightest* possible lower and upper distance bounds between the Euclidean distance on two compressed sequences, when the $k$ orthonormal coefficients with the highest energy are used in order to describe the compressed sequences (and information about the energy of the discarded coefficients is also retained).

### 4.3 Optimizations—Early Termination

Now we introduce a simple optimization that can further reduce the running time of the optimal algorithm. In the main body of the algorithm, there exists an iterative loop that assigns the remaining energy to the unknown coefficients according to the principles that we described. Even though in the experimental section we show that this loop converges very fast, when distance estimation

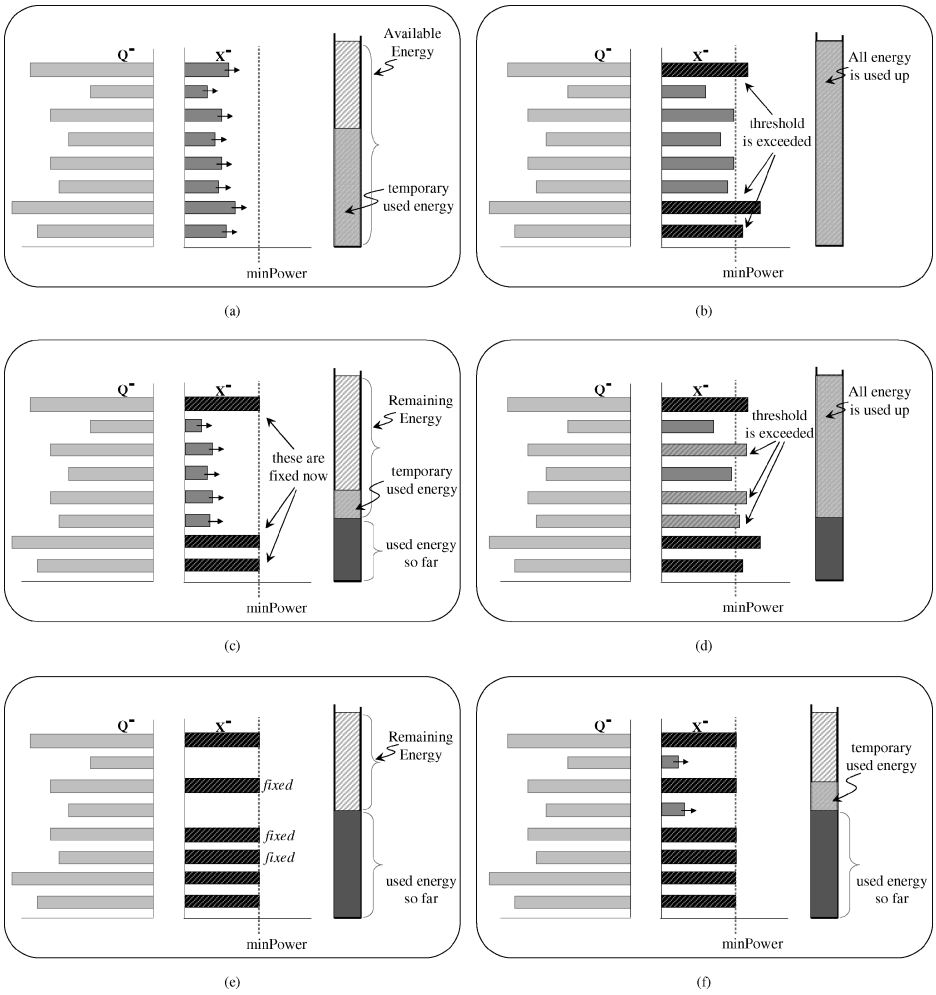


Fig. 8. Various steps of the optimization (water-filling) algorithm in n-dimensional space. The unknown vector $X^-$ is rescaled (opposite $Q^-$) until all energy is used (a),(b). Coefficients exceeding the given constraints are fixed (c), and the process is repeated for the remaining coefficients until all energy is used up (d),(e),(f).

occurs during a NN-search operation, the algorithm can break early out of the loop if the current lower-bound distance bound is already worse than the *bestSoFar* match on the ongoing search. We call this process *early termination* of the algorithm, and it can result in a significant speedup in the execution of the search process.

Notice, that before entering the loop (lines 23–32 of Figure 9) there is an initial distance estimate *distSq* computed based on the known coefficients. This is increased accordingly to estimate upper and lower bounds on the distance. Let's also dictate that the algorithm accepts as input the current *bestSoFar* match from the (ongoing) search process. We can update the current lower bound distance within the loop, and then stop improving the distance bounds,

```
// Q is uncompressed
// X is compressed + eX is the error of discarded coefficients
//
function [LB, UB] = optimal(Q, [X, eX])
{
   p+ = find(X.coeffs); // best coefficients of X
   minPower = min(abs(X(p+)));

   // distance of known coefficients
   distSq = sum( abs( X(p+) - Q(p+) ).^2);

   // distance from the remaining coefficients
   p- = setdiff([1:N], p+); // discarded coefficients

   eY = sum(abs([Q(p-)].^2); // eX is given

   directions = Q ./ norm(Q); // extract direction
   Xnew = directions .* norm(X); // rescale it

   violations = find(Xnew > minPower);

   // iterate until no energy is left
   while (~isempty(violations))
   {
      Xnew(violations) = minPower; // fix these dimensions

      remainingCoeff = find(Xnew < minPower);
      delta = sqrt(( eX - |Xnew| -|remainingCoeff|)*(minPower^2) ) /
                                          sum(directions(dest).^2));
      Xnew(remainingCoeff) = delta*directions(remainingCoeff);

      violations = find(Xnew > minPower);
   }

   LB = distSq + sum((Q - Xnew).^2); // updated lower bound
   UB = distSq + sum((Q + Xnew).^2); // updated upper bound

   LB = sqrt(LB);
   UB = sqrt(UB);
}
```

Fig. 9. An implementation of the solution to the optimization problem.

if the current lower bound is larger than the $bestSoFar$ match. This is possible, since every execution of the loop will only increase the lower bound, bringing it closer to the actual distance value between the two sequences $Q$ and $X$.

Therefore, using the early termination feature, the algorithm will still return the same results. However, it will refrain from estimating the tightest possible bound, if the sequence is *guaranteed* to be more distant than an already found match.

In the upcoming experimental section, we show that this simple optimization results in a significant reduction on the amount of required iterations executed by the algorithm.

### 4.4 Indexing in Metric Spaces

Even though indexing goes beyond the scope of this paper, we mention succinctly how the compression with different coefficients can utilize an indexing structure for additional performance benefits.

Due to the non-uniform use of coefficients for every sequence, traditional space partitioning indexing structures like R-trees [Guttman 1984], and its

variants, cannot be utilized since they require the same set of coefficients for all objects. However, metric trees which utilize distances between objects can be adapted for our purpose. For example, VP-trees [Yianilos 1992; Chee Fu et al. 2000; Bozkaya and Özsoyoglu 1997] are a popular instance of a metric tree structure that partition the space based on distances to selected *vantage points* of the dataset. A tree containing the objects is constructed by recursively separating the dataset into two distinct sets based on the median distance $\mu$ to the vantage point. The objects closest to the vantage point ($S_<$) are stored on the left subtree, and those that are further away from the median distance ($S_>$) are directed on the right subtree. The process is repeated recursively and a different vantage point is selected for each of the remaining subsets. Figure 10 illustrates this process on 2 dimensions for clarity (each point essentially represents one object).

After the tree is constructed and a query is posed, one only has to examine the proper subset based on the position of the query. Only if the query lies close to the median distance, both subsets need to be examined, otherwise one of them is discarded from examination.

For more details about such an approach, we direct the interested reader to Vlachos et al. [2005].

## 5. EXPERIMENTS

We evaluate various parameters of our algorithm; the convergence rate, the tightness of the estimated bounds, and the additional pruning power that is achieved when using the presented optimal algorithm. As our testbed we use search engine logs spanning a period of 3 years (3*365 points per sequence), which we trim down to 1024 points in order to simplify calculations and exposition of ideas. We had at our disposal approximately 60,000 query patterns. The sequences were studentized (mean value was subtracted and sequences normalized by the std), so as to remove any scale bias. In this way we are reverting the distance into a measurement of correlation and can discover more flexible patterns.

Before presenting any performance aspects of our approach, we depict some of the Nearest-Neighbor (NN) matches that resulted from the search procedure for various queries. In Figure 11 we illustrate some indicative results included within the 20-NN matches of various posed queries. The order of the reported results also corresponds to the relative order of the answers in the result set. However, we do not consider that an analyst should place importance on the actual order of the returned results, rather than focus on the set of the top results as a whole, in any attempt to analyze the connection to the posed query. The results reported are for the following queries: *abc*, *alexander the great*, *deutsche bank*, *formula 1 racing*, *toyota.com*, and *usps.com*. One can observe that the returned matches hold a semantic affinity to the posed query. For example, the results of the query *toyota.com* returns car-related queries, for example, other car manufacturers or even keywords relating to car insurance or automotive related Web sites. Similarly, the outcome of the query *deutsche bank* resulted in keywords related to financial companies and stocks. In general,

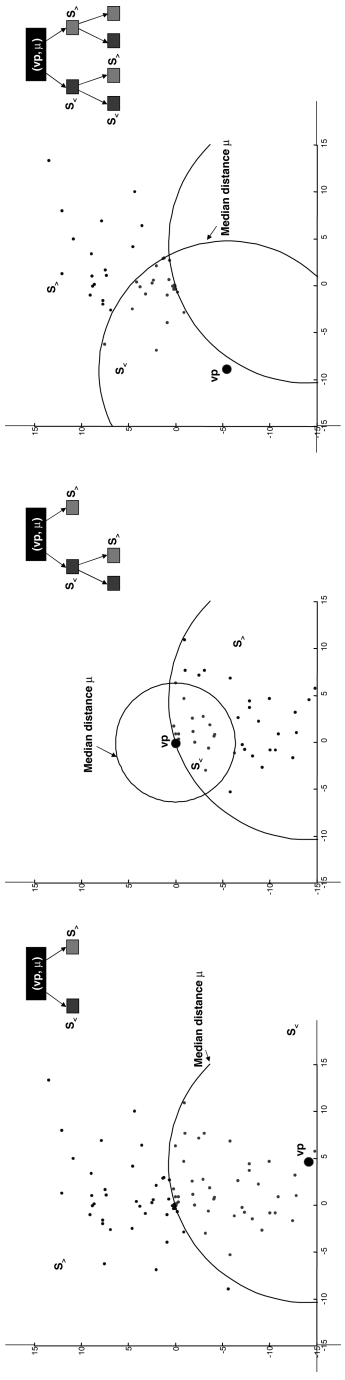


Fig. 10. Illustration of the creation of a VP-tree (metric tree) that can accommodate the compressed sequences.

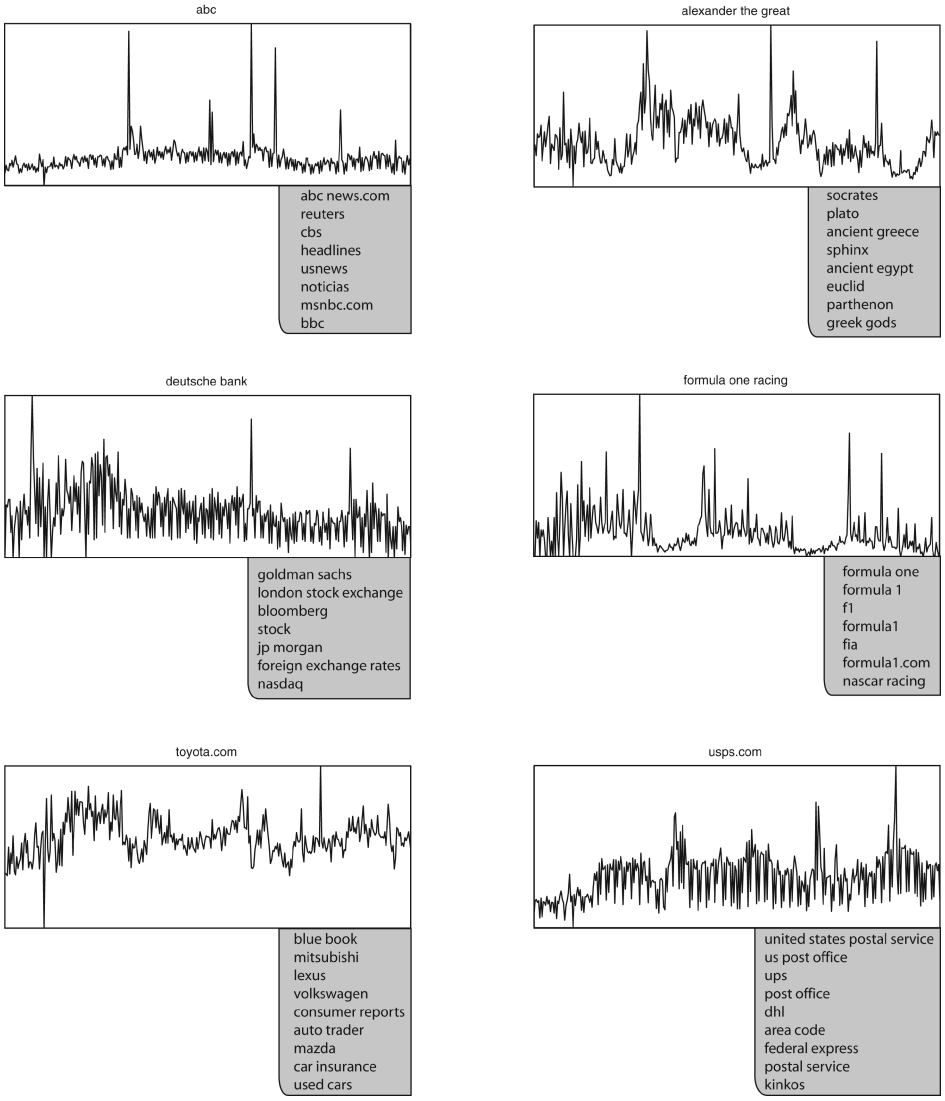


Fig. 11. Several NN-search results. Six queries and an indicative subset of their 20-nearest-neighbors based to the temporal similarity in the query demand.

search on the query logs returns highly interpretable and useful matches, something that was also attested by other relevant publications [Chien and Immorlica 2005; Adar et al. 2007; Vlachos et al. 2004; Vlachos et al. 2005; Hall 2009].

## 5.1 Runtime

Reporting the actual runtime of an algorithm, is not always an accurate indicator of performance, since wall-clock running time can be severely biased based

Table I. Runtime for Returning the 20NN

| Dataset cardinality | 8000 | 16000 | 32000 |
|---|---|---|---|
| Runtime (sec) | 4.0625 | 5.3958 | 6.7500 |

on implementation, programming language, hardware, etc. Additionally, since part of the experiments were conducted in Matlab, timing experiments cannot capture the raw performance achieved by the proposed techniques. However, for purposes of indicating the trend in runtime increase, in Table I we illustrate the time to return the 20-NN for increasing dataset cardinalities (8000 -16000 - 32000 sequences). The running cost is typically sublinear in the number of database sequences, because we are reutilizing the derived bounds from previously examined sequences.

One can observe that NN-search on 16,000 sequences is not twice as costly as the runtime on 8,000 sequences. Actually, searching 32,000 time-series adds only 1.6 times to the cost of iterating through 4,000 time-series, even though the dataset cardinality increased 4 times. From an implementation point of view, and because of the sublinear nature of the algorithm, the search can be easily architectured to perform in real-time. The search process can be trivially parallelized with minimal communication and merging costs through distributing portions of the dataset. NN queries can be issued at different sites and merged subsequently. In such a way, even when the dataset size gets increasingly large, real-time retrieval options are feasible.

In the upcoming sections we mostly focus on reporting the pruning power of various search algorithms, which is an unbiased indicator of performance [Keogh and Kasetty 2003; Keogh et al. 2001].

### 5.2 Convergence Rate

The proposed water-filling algorithm iteratively rescales subsets of the unknown coefficients, in the process of allocating the remaining signal energy. A number of iterations are required until convergence. Here, we empirically demonstrate that the algorithm reaches the solution in very few iterations (typically 2 to 3); therefore, performance of the algorithm is not adversely impacted. The experiment is conducted by computing 1000 distance calculations (lower and upper bounds) from a pool of randomly selected query logs. We repeat the experiment for various compressed representations, retaining from 8 to 64 coefficients per sequence, or in other words, for compression rates of $\frac{128}{1}$ to $\frac{16}{1}$. The histograms of the number of iterations are depicted in Figure 12. We observe that the algorithm converges very fast, typically in 1 to 4 iterations, with the majority of the cases being 2–3 iterations.

Notice, that each additional iteration incurs a CPU cost which is smaller than the previous one, because at each iteration we only evaluate coefficients that have not been previously estimated. Therefore, there is a diminishing CPU cost. Finally, we emphasize that most search operations are I/O bound and the small additional cost that our algorithm incurs is only CPU-based. As will been shown next, our algorithm ultimately achieves much better performance than previous approaches due to the much tighter distance bounds which lead to a

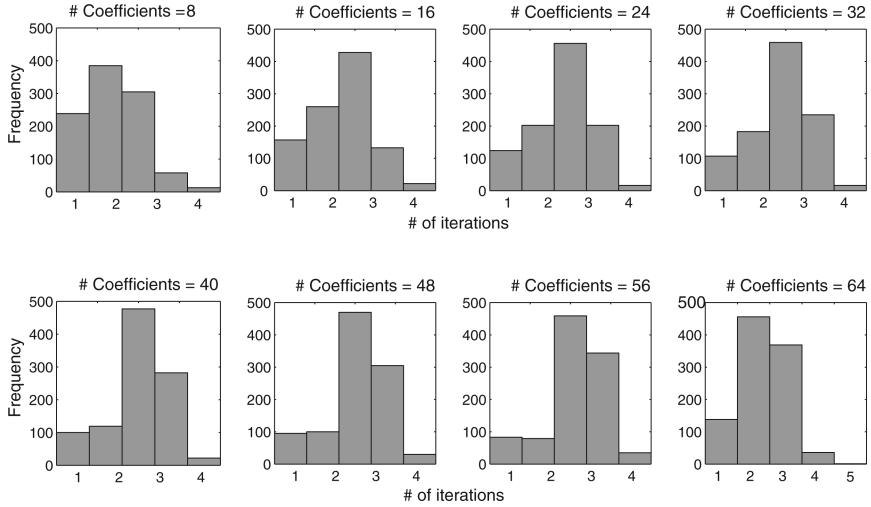


Fig. 12. Number of iterations for convergence on the optimization algorithm. The algorithm converges very fast, typically in 2-3 iterations.

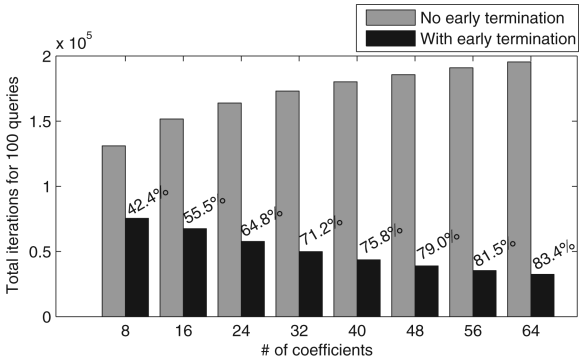


Fig. 13. Incorporation of early termination, reduces significantly the amount of iterations required by the algorithm. The shown percentages indicate the relative reduction rates in iterations.

significant reduction in the number of uncompressed sequences fetched from the disk.

*Early Termination:* The convergence rate of the optimal algorithm can be reduced even further by exploiting the early termination (ET) feature that was introduced in section 4.3. In order to observe the merits of this feature we need to perform an actual NN-search, so that we provide the current *bestSoFar* match to the algorithm. We search over 1000 sequences and we pose 100 queries. We record the total number of iterations over those 100 queries, with and without the early termination. The results of the experiment are shown in Figure 13. The regular optimal algorithm (without early termination) executes additional iterations for increasing number of coefficients, something that was already observed in the previous experiment. However, when early termination is

employed, the number of iteration in the algorithm actually *decreases*, for increasing number of coefficients. This is the case because as more coefficients are used there are more opportunities for estimating even tighter distance bounds, a fact that the optimal algorithm exploits. This automatically means that when more coefficients are utilized the bounds will progressively become less loose, giving better distance estimates and allowing for early termination. That is why we observe a reduction of iterations when utilizing more coefficients, under the ET feature.

On the same figure we also record the relative reduction of iterations under the ET, which is quite dramatic, and ranges from 40% to more than 80%.

### 5.3 Bound Tightness

Now, we compare the tightness of our bounds against widely used time-series search techniques, which have appeared in the data-mining literature. The strawmen approaches that we compare with are:

(1) *First Coefficients:* Techniques that compute bounds on the distance using the first coefficients, inherently make the assumption that the underlying signal contains primarily low frequency components [Agrawal et al. 1993; Rafiei and Mendelzon 1998]. Such approaches perform sufficiently on random walk signals, such as stock market data, but in general do not adapt well for generic signals. Additionally, such approaches only estimate lower bounds on the distance function, therefore in general cannot match the pruning performance that the combination of lower/upper bounds can achieve.

(2) *First Coefficients + error:* This approach augments the aforementioned methodology by recording also the reconstruction error (or remaining energy of the omitted coefficients), which improves upon the previous bounds. This work additionally utilizes upper bounds, which the previous approaches did not consider [Wang and Wang 2000].

(3) *Best Coefficients + error:* Similar to the previous approach, this technique exploits the coefficients with the highest energy plus the approximation error in order to bound the distance [Vlachos et al. 2004].

5.3.1 *Space Requirements.* Notice that it is not meaningful to directly compare the above approaches using the same number of coefficients, because each technique may require a different amount of storage space. We need to compare all approaches under the same memory requirements.

The storage of the first $c$ Fourier coefficients requires just $2c$ doubles (or $2c * 8$ bytes). However, when utilizing the $c$ best coefficients for each sequence, we also need to store their positions in the original DFT vector. That is, the compressed representation with the $c$ largest coefficients is stored as pairs of *[position-coefficient]*.

For our experiments, the sequences are composed of 1024 points, which means that we need to store 512 positions, if we consider the symmetric property of the Fourier coefficients. 9 bits would be sufficient to describe any of the coefficient positions, however, since on disk we can write only multiples

Table II. Requirements for Usage of Same Storage for Each Approach

| | |
|---|---|
| First Coeffs | c First Coeffs + Middle Coeff |
| First Coeffs + error | c First Coeffs + Error |
| Best Coeffs + error | $\lfloor c/1.125 \rfloor$ Best Coeffs + Error |
| Optimal | $\lfloor c/1.125 \rfloor$ Best Coeffs + Error |

of bytes, each position requires 2 bytes. Therefore, each approach that utilizes the best coefficients allocates $16 + 2$ bytes per coefficient. In other words, if an approach storing the first coefficients uses $c$ coefficients, then our method will use $\lfloor 16c/18 \rfloor = \lfloor c/1.125 \rfloor$ coefficients.

For some distance measures we also use one additional double to record the error (sum of squares of the remaining coefficients). For the measures that don't use the approximation error we need to allocate one additional number and we choose this to be the middle coefficient of the full DFT vector, which is a real number [Oppenheim et al. 1997] (since we have real data with lengths of power of two). If in some cases the middle coefficient happens to be one of the $c$ best ones, then these sequences just use 1 less double than all other approaches. Table II summarizes how the same amount of memory is allocated for each compressed sequence of every approach.

Therefore, when in the following figures we mention memory usage of [2*(32)+1] doubles, the number in parenthesis essentially denotes the coefficients used for the methods using the first coefficients (+1 for the middle coefficient or the error, respectively). For the same example, approaches using the best coefficients will use the 28 best coefficients, leading to the same memory requirements.

5.3.2 *Results.* We plot the lower and upper bounds derived by each approach and we normalize the results against the exact euclidean distance. Numbers closer to 1 indicate tighter bounds. We observe that in all cases the optimal algorithm returns the best distance estimates compared to the other approaches, even though it uses fewer coefficients than some of the competing methodologies. On the title of each graph of Figure 14 we also indicate how much the optimal algorithm improves on the First Coeffs + error approach. We observe that the optimal introduces an improvement of approximately 10% on the lower bounds and 12–15% on the upper bounds. As we will demonstrate in the following section, this reduction in the distance ambiguity can lead to very dramatic speedups in the overall search performance.

## 5.4 Pruning Power and Performance Improvement

For this experiment we assemble a large pool of query Weblogs consisting of 32000 temporal sequences. We pose 100 random queries that don't have exact matches in order to offer more realistic performance metrics. We search for the 1-Nearest-Neighbor of each query and we utilize both Lower and Upper bounds. For the First Coeffs approach we utilize only the lower-bounds, since no upper-bounds are offered.

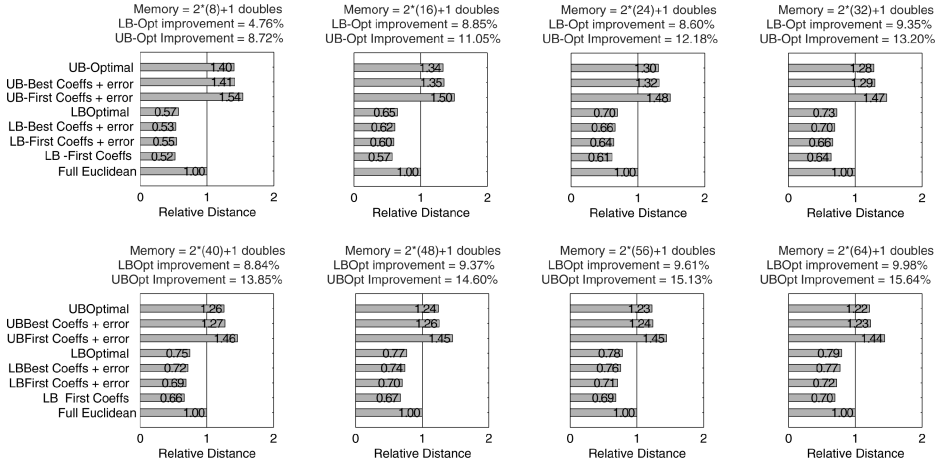


Fig. 14. Comparison of lower/upper bounds returned by various techniques, across different compression rates. The optimal algorithm exhibits the tightest possible bounds.

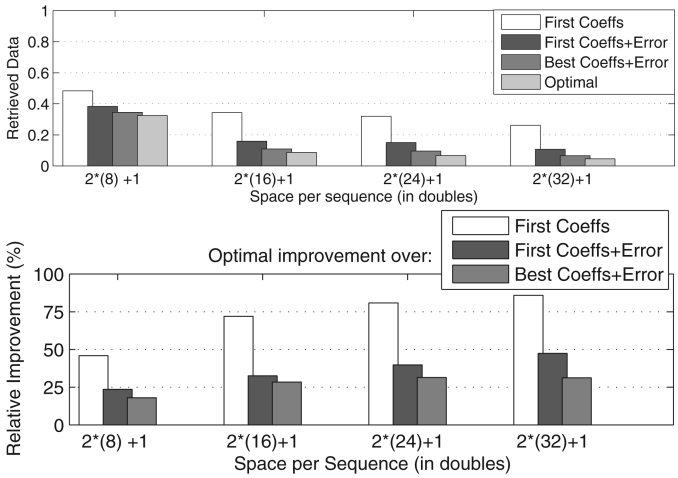


Fig. 15. Top: Ratio of uncompressed sequences retrieved from disk (smaller numbers are better). Bottom: Improvement of Optimal against all other approaches (higher numbers are better).

We evaluate the performance of each technique based on the search procedure presented in Section 3.3, which prunes the search space and directs the search according to the lower/upper bounds derived from the compressed sequences. Ultimately, we measure the cardinality of uncompressed sequences that each technique retrieved from disk. This essentially reflects the most important bottleneck of a search performance, because it is an I/O bound process. Figure 15 shows how many uncompressed sequences were retrieved for each of the search approaches, normalized by the total number of sequences. For clarity, the lower graph in the figure depicts the relative improvement in the search performance when the proposed optimal bounding algorithm is used.

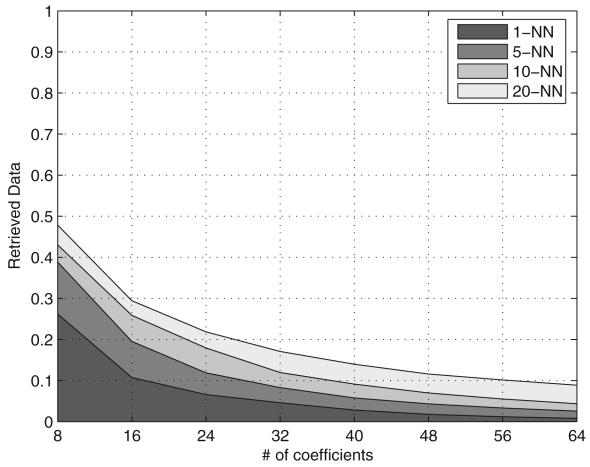


Fig. 16. Optimal Algorithm: Ratio of uncompressed sequences retrieved from disk for different number of coefficients and NN-search (1-NN, 5-NN, 10-NN, 20-NN).

When using [2*(32)+1] doubles per sequence we observe the largest improvement in performance; 80%, 50%, 30% improvement compared to the 3 other distance bounding methodologies. Therefore, we can achieve excellent performance compared to previous state of the art when utilizing the optimal distance bounds.

Lastly, we focus on the optimal algorithm and we depict its performance when searching for increasing number of nearest neighbors (parameter $k$). We measure the performance for 1-NN, 5-NN, 10-NN and 20-NN searches. We capture again the pruning power of the algorithm as the ratio of uncompressed sequences retrieved from disk. When searching for k-NN results the search algorithm is slightly modified in order to accommodate the use of a priority queue, where the $k$ current nearest matches are stored. Now the pruning is achieved using the $k$-th *bestSoFar* match. The results of this experiment are reported in Figure 16 and indicate a small and smooth sublinear increase in the number of retrieved sequences under increasingly larger $k$-NN searches. For example, when using 64 coefficients per sequence and searching for the 1-NN, 0.8% of the sequences are retrieved. Approximately 5 times more sequences are retrieved when searching for the 10-NN and 10-times as many sequences when a 20-NN query is posed.

In conclusion, with these experiments we have seen that the presented optimal distance estimation algorithm converges fast, provides the tightest possible distance bounds, and leads to significant benefits in the search performance.

## 6. CONCLUSIONS

This work examined techniques that can boost the search performance on temporal query Weblogs. We presented algorithms that compute the optimal lower and upper bounds on distance functions, when working directly on the compressed data. In addition to its theoretical underpinning, the algorithm is

easy to implement, it is lightweight in its execution, and results in a significant speedup of search operations.

Even though in this work we focused on a specific distance measure (Euclidean distance) and on a certain compression methodology (Fourier transform), our approach is directly applicable to other linear distance functions, and to any orthonormal data transformation (wavelets, principal components, etc). The presented techniques can also be combined with index structures (e.g. metric trees) for achieving an additional performance boost. Although this work was presented in the context of query demand patterns, search on other types of temporal Web data, such as sequences of tag usage [Rattenbury et al. 2007; Dubinko et al. 2006] and click-through data [Zhao et al. 2006] can also benefit from our contribution. Finally, our techniques are of independent interest for any time-series search application.

## REFERENCES


ADAR, E., WELD, D., BERSHAD, B., AND GRIBBLE, S. 2007. Why we search: Visualizing and predicting user behavior. In *Proceedings of the International World Wide Web Conference*.

AGRAWAL, R., FALOUTSOS, C., AND SWAMI, A. 1993. Efficient similarity search in sequence databases. In *Proceedings of the International Conference on Foundations of Data Organization and Algorithms (FODO)*. 69–84.

ASSENT, I., KRIEGER, R., AFSCHARI, F., AND SEIDL, T. 2008. The TS-tree: Efficient time series search and retrieval. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 252–263.

BEITZEL, S., JENSEN, E., CHOWDHURY, A., FRIEDER, O., AND GROSSMAN, D. 2007. Temporal analysis of a very large topically categorized Web query log. In *J. Amer. Soc. Inform. Sci. Tech. 58*, 2, 166–178.

BERTSEKAS, D. P. 2000. *Nonlinear Programming*. Athena Scientific.

BOLLEGALA, D., MATSUO, Y., AND ISHIZUKA, M. 2007. Measuring semantic similarity between words using Web search engines. In *Proceedings of the International World Wide Web Conference*. 757–766.

BOZKAYA, T. AND ÖZSOYOGLU, M. 1997. Distance-based indexing for high-dimensional metric spaces. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 357–368.

CAI, Y. AND NG, R. 2004. Indexing spatio-temporal trajectories with chebyshev polynomials. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 599–610.

CHEE FU, A. W., CHAN, P. M., CHEUNG, Y.-L., AND MOON, Y. 2000. Dynamic VP-Tree indexing for N-nearest neighbor search given pair-wise distances. *J. VLDB*, 154–173.

CHEN, Q., LI, M., AND ZHOU, M. 2007. Improving query spelling correction using Web search results. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language*. 181–189.

CHIEN, S. AND IMMORLICA, N. 2005. Semantic similarity between search engine queries using temporal correlation. In *Proceedings of the International World Wide Web Conference*. 2–11.

COVER, T. M. AND THOMAS, J. A. 1991. *Elements of Information Theory*. John Wiley and Sons.

DUBINKO, M., KUMAR, R., MAGNANI, J., NOVAK, J., RAGHAVAN, P., AND TOMKINS, A. 2006. Visualizing tags over time. In *Proceedings of the International World Wide Web Conference*. 193–202.

GUTTMAN, A. 1984. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 47–57.

HALL, E. A. M. C. K. 2009. Gazpacho and summer rash: Lexical relationships from temporal patterns of web search queries. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. 1046–1055.

KAGE, T. AND SUMIYA, K. 2006. A Web search method based on the temporal relation of query keywords. In *Proceedings of the Web Information Systems*. 4–15.

KAHVECI, T. AND SINGH, A. K. 2001. Variable length queries for time series data. In *Proceedings of the IEEE International Conference on Data Engineering*. 273–282.

KEOGH, E. 2002. Exact indexing of dynamic time warping. In *Proceedings of the International Conference on Very Large Databases*. 406–417.

KEOGH, E., CHAKRABARTI, K., MEHROTRA, S., AND PAZZANI, M. 2001. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 151–162.

KEOGH, E. J. AND KASETTY, S. 2003. On the need for time series data mining benchmarks: A survey and empirical demonstration. *Data Min. Knowl. Discov. 7*, 4, 349–371.

KLEINBERG, J. 2002. Bursty and hierarchical structure in streams. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 91–101.

LIE, B., JONES, R., AND KLINKNER, K. 2006. Measuring the meaning in time series clustering of text search queries. In *Proceedings of the International Conference on Information and Knowledge Management*. 836–837.

LIU, N., NONG, S., YAN, J., ZHANG, B., CHEN, Z., AND LI, Y. 2006. Similarity of temporal query logs based on ARIMA model. In *Proceedings of the IEEE International Conference on Data Mining*. 975–979.

OPPENHEIM, A., WILLSKY, A., AND NAWAB, S. 1997. *Signals and Systems*, 2nd Ed. Prentice Hall.

OTSUKA, S. AND KITSUREGAWA, M. 2006. Clustering of search engine keywords using access logs. In *Proceedings of the International Conference on Database and Expert Systems Applications*. 842–852.

OTSUKA, S., TOYODA, M., HIRAI, J., AND KITSUREGAWA, M. 2004. Extracting user behavior by Web communities technology on global Web logs. In *Proceedings of the International Conference on Database and Expert Systems Applications*. 957–988.

RAFIEI, D. AND MENDELZON, A. 1998. Efficient retrieval of similar time sequences using DFT. In *Proceedings of the International Conference on Foundations of Data Organization and Algorithms*. 249–257.

RATTENBURY, T., GOOD, N., AND NAAMAN, M. 2007. Towards extracting flickr tag semantics. *Proceedings of the International World Wide Web Conference*. 1287–1288.

RICHARDSON, M. 2008. Learning about the world through long-term query logs. *ACM Trans. Web, 2*, 4.

RODE, H. AND HIEMSTRA, D. 2006. Using query profiles for clarification. In *Proceedings of the European Conference on Information Retrieval*. 205–216.

SANDERSON, M. AND DUMAIS, S. 2007. Examining repetition in user search behavior. In *Proceedings of the European Conference on Information Retrieval*. 597–604.

SUN, Y., LIU, N., XIE, K., YAN, S., ZHANG, B., AND CHEN, Z. 2007. Causal relation of queries from temporal logs. In *Proceedings of the International World Wide Web Conference*. 1141–142.

VLACHOS, M., MEEK, C., VAGENA, Z., AND GUNOPULOS, D. 2004. Identification of similarities, periodicities & bursts for online search queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 131–142.

VLACHOS, M., YU, P., AND CASTELLI, V. 2005. On Periodicity detection and structural periodic similarity. In *Proceedings of the SIAM International Conference on Data Mining*.

WANG, C. AND WANG, X. S. 2000. Multilevel filtering for high dimensional nearest neighbor search. In *Proceedings of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*.

WANG, X., ZHAI, C., HU, X., AND SPROAT, R. 2007. Mining correlated bursty topic patterns from coordinated text streams. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 784–793.

WEBER, R., SCHEK, H.-J., AND BLOTT, S. 1998. A quantitative analysis and performance study for similarity search methods in high-dimensional spaces. In *Proceedings of the International Conference on Very Large Database*. 194–205.

YI, B. AND FALOUTSOS, C. 2000. Fast time sequence indexing for arbitrary Lp norms. In *Proceedings of the International Conference on Very Large Database*. 385–394.

YIANILOS, P. 1992. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithm*. 311–321.

ZHAO, Q., HOI, S., LIU, T.-Y., BHOWMICK, S. S., LYU, M. R., AND MA, W.-Y. 2006. Time-dependent semantic similarity measure of queries using historical click-through data. In *Proceedings of the International World Wide Web Conference*. 543–552.

ZIEGLER, C.-N., SIMON, K., AND LAUSEN, G. 2006. Automatic computation of semantic proximity using taxonomic knowledge. In *Proceedings of the International Conference on Information and Knowledge Management*. 465–474.

Received December 2007; revised October 2009; accepted December 2009