Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information and Computation 194 (2004) 203–241
Information and Computation
www.elsevier.com/locate/ic

Bounded similarity querying for time-series data☆

Dina Q. Goldin[a,*], Todd D. Millstein[b], Ayferi Kutlu[a]

[a] *Computer Science and Engineering Department, University of Connecticut, Storrs, CT 06269, USA*
[b] *Computer Science Department, University of California, Los Angeles, CA 90095, USA*

Received 14 October 2003
Available online 25 September 2004

Abstract

We define the problem of *bounded similarity querying* in time-series databases, which generalizes earlier notions of similarity querying. Given a (sub)sequence S, a query sequence Q, lower and upper bounds on *shifting* and *scaling* parameters, and a *tolerance* ϵ, S is considered *boundedly similar* to Q if S can be shifted and scaled within the specified bounds to produce a modified sequence S' whose distance from Q is within ϵ. We use *similarity transformation* to formalize the notion of bounded similarity. We then describe a framework that supports the resulting set of queries; it is based on a *fingerprint* method that normalizes the data and saves the *normalization parameters*. For off-line data, we provide an indexing method with a single index structure and search technique for handling all the special cases of bounded similarity querying. Experimental investigations find the performance of our method to be competitive with earlier, less general approaches.

Keywords: Bounded similarity querying; Similarity transformation; Shift; Scale; Fingerprint; Normalization parameter

☆ Supported in part by NSF Grant Nos. 9733678 and IRI-0296195.
* Corresponding author.
E-mail address: dqg@cse.uconn.edu (D.Q. Goldin).
URL: www.cse.uconn.edu/~dqg.

0890-5401/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2004.07.001

Preface

The theoretical framework for this work was shaped by Paris Kanellakis together with the first author (Goldin), who was then Paris' Ph.D. student; the preliminary draft was published in [16]. The implementation work was started by the second author (Millstein) as an undergraduate honors project under Paris' guidance, though it had to be completed independently [33]. We are glad that this special issue of *Information and Computation* can provide a venue for this work, one of Paris' last.

1. Introduction

1.1. Similarity querying of time-series data

Time-series data are sequences of real numbers representing measurements at uniformly spaced temporal instances. Time-series are the principal format of data in many applications, from financial to scientific. The next generation of database technology, with its emphasis on multimedia, is expected to provide flexible and clean interfaces to facilitate data mining of time-series. However, for very large data sets, such linguistic facilities must be supported by indexing to provide reasonable I/O efficiency.

A basic problem in these applications is *first-occurrence subsequence matching*.

First-occurrence subsequence matching. Given a query sequence Q of length n and a data sequence S of arbitrary length longer than n, find the first subsequence X of length n in S that matches Q exactly.

A wide range of algorithms has been developed for in-core versions of this question [3] for strings over an alphabet or for values over bounded discrete domains. There are particularly elegant linear-time $O(n + N)$ algorithms (by Knuth–Morris–Pratt and Boyer–Moore) and practical searching utilities for more general patterns instead of query strings (e.g., regular patterns in grep). The part of this technology that is most relevant for our paper is the Rabin–Karp randomized linear-time algorithm [23], which provides an efficient in-core solution based on *fingerprint* functions. Fingerprints are a form of sequence hashing that allow constant-time comparisons between hash values and are incrementally computable (Section 3).

For most practical applications, time-series data in continuous domains are inherently *inexact*, due to imprecision in measuring devices and clocking strategies. Therefore, a more general, *approximate* notion of matching is needed, such as for the two sequences in Fig. 1. In addition, it is often necessary to find *all* matches to the query within a longer data sequence, instead of simply the first one. These additional considerations lead to the *approximate subsequence matching* problem:

Definition 1 (*Approximate subsequence matching*)**.** Given a query sequence Q of length n, a tolerance $\epsilon \geqslant 0$ and a data sequence S of arbitrary length longer than n, find all subsequences X of length n in S satisfying the property that $D(Q,X) \leqslant \epsilon$, where D is some distance measure (not necessarily Euclidean).

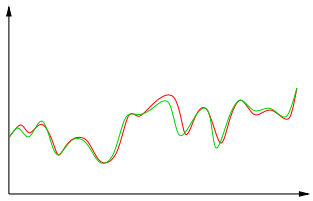

Fig. 1. Sequences which Match Approximately

In many cases, it is more natural to allow the matching of sequences that are not close to each other in an Euclidean sense. For example, two companies may have identical stock price fluctuations, but one's stock is worth twice as much as the other at all times. As another example, two sales patterns might be similar, even if the sales volumes are different. We shall refer to the difference between these sequences as *scale*. In another example, the temperature on two different days may start at different values but then go up and down in exactly the same way. We shall refer to the difference between these sequences as *shift*. Shift and scale transformations are illustrated in Fig. 2; these are instances of *similarity transformations*.

Some examples of queries which involve these transformations are:

- find months in the last 5 years with sales patterns of minivans like last month's;
- find other companies whose stock price fluctuations resembles Microsoft's;
- find months in which the temperature in Paris showed patterns similar to this month's pattern.

A good time-series indexing mechanism should be able to answer queries like those above. We formalize this notion of *similarity querying* in the next section. To avoid confusion, we shall use the term *approximate matching* when only distance measures are involved, and the term *similarity querying* where similarity transformations are involved as well.

Lastly, the size of the data in commercial applications of time-series similarity querying may be too large to be stored in memory. Therefore, the research in approximate matching, similarity querying, and other time-series query problems has concentrated on the case when storage is in secondary memory, as opposed to in-core. An indexing method must be used to minimize the number of disk accesses; it is assumed that the index is built once off-line, then used for ad hoc querying of the data.

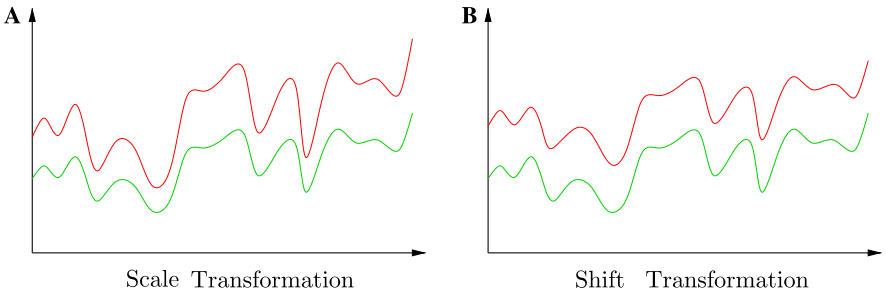

Fig. 2. Shift and scale transformations.

To summarize, following are the desirable characteristics of the similarity querying problem:

- all matching subsequences are to be returned;
- the match can be *approximate*;
- the match allows for *shift* and *scale* transformations;
- there is *indexing* support for large datasets.

Note however that, the problem of similarity querying does not take into consideration the amount of shifting and scaling between sequences. Two sequences are considered similar if there exists a way to shift or scale one to match the other; there are no bounds on the value of the shift and scale parameters. This is a limitation; for example, we may want to specify that the shift factor should be between -10 and 10, and the scale factor should be at most 2. We refer to this as *bounded similarity querying*.

1.2. Overview

In this paper, we address the problem of *bounded similarity querying*, a generalization of similarity querying. Given a subsequence S in a time-series database, a query sequence Q, lower and upper bounds on the degrees of *shifting* and *scaling* allowed, and a *tolerance* ϵ, S is considered *boundedly similar* to Q if S can be shifted and scaled within the specified bounds to produce a modified sequence S' whose distance from Q is within ϵ. The problem of bounded similarity querying is to find all subsequences in a time-series database that are boundedly similar to a query sequence Q, given user-defined shifting and scaling bounds and a tolerance.

In Section 2, we formalize the notion of bounded similarity between time-series sequences. First we extend the distance metric used in Agrawal et al. [2] with invariance under a group of transformations. We refer to the new metric as *similarity distance* and show that it is non-negative, symmetric, and effectively computable; it also obeys the triangle inequality. Then we define *bounded similarity queries* and show how these queries specialize to various simpler queries, such as approximate matching.

Next we describe a framework that supports this more expressive set of queries, while preserving the desirable characteristics of similarity querying identified above. Our fingerprint method (Section 3) is central to the bounded similarity querying framework. We define *similarity fingerprints*, which normalize the data and store the *normalization parameters*. We show that *similarity fingerprints* have several desirable properties, such as *updateability* and *accuracy*, which play an important role in enabling an efficient implementation for bounded similarity querying.

The key characteristic allowing similarity fingerprints to achieve these properties is the fact that time-series data have a *skewed energy spectrum*, to use the terminology from discrete signal processing (DSP) [12]. As a result, our work, like most of the technology of information retrieval in this area, is influenced by signal processing methods, in particular by the *discrete Fourier transform* [35].

In Section 4, we provide an efficient in-core algorithm for bounded similarity querying. A bounded similarity query is first translated into an *internal query*. While bounded similarity queries are specified as optimization problems, the internal query is a special case of a *multidimensional range query* involving no optimization, for which efficient algorithms exist [12]. We show how similarity

fingerprints help to efficiently answer the internal query and discuss how to use the output from this query to find all answers to the original user query.

In Section 5, we consider extensions that make bounded similarity querying more practical; specifically, we consider: (a) indexing methods, (b) searching for all similar subsequences in a database, and (c) handling queries of varying lengths. We provide an efficient index structure for bounded similarity querying, based on similarity fingerprints. Our indexing strategy extends the approach of [2] to handle bounded similarity querying in its full generality. For handling queries of varying lengths, we propose a new *stitching* approach and compare it with older approaches. We describe an extension to our method, for handling queries of varying lengths without rebuilding the index structure.

In Section 6, we use methods from linear algebra and mathematical statistics [32] to prove the correctness of our algorithms for obtaining the internal query from the user query, and for retrieving the answers to the user query from the answers to the internal query.

In Section 7, we present experimental results for our implementation of bounded similarity querying. We also assess the performance penalty that we pay for solving a more general problem, as compared to systems that specialize in subcases of bounded similarity querying. Experimental investigations find the performance of our method to be competitive with the less general approaches to similarity querying [13].

In Section 9, we discuss related work; we conclude in Section 10.

2. The semantics of similarity

This section provides the basic definitions for bounded similarity querying. They include *similarity transformations* and *normal forms* of *similarity classes*.

2.1. Similarity transformations

Our formalization of the notion of *similarity* is based on *transformations* between sequences of time-series data, from some set of transformations G. Two time-series sequences are defined *similar* if there exists a transformation in G which maps one to the other:

Definition 2 (*Similarity relation*)**.** Let D be a distance metric between sequences and $\epsilon \geqslant 0$ a tolerance. Query sequence Q is *approximately similar* within tolerance ϵ to data sequence S when there exists a similarity transformation T in G so that $D(Q, T(S)) \leqslant \epsilon$. When $D(Q, T(S)) = 0$, we say that Q and S are *exactly similar,* or just *similar*.

The similarity relation provides the semantics for the *similarity query*:

Similarity query. Given a query sequence Q and a database B consisting of sequences of the same length as Q, find all sequences S in B such that Q and S are (exactly) similar.

In geometry, the set G of transformations typically forms a *group*. *Shift transformations*, where the value of all members in a sequences is increased or decreased by the same constant, form a group; so do *scale transformations*, where all the values are multiplied by some constant. Combinations of

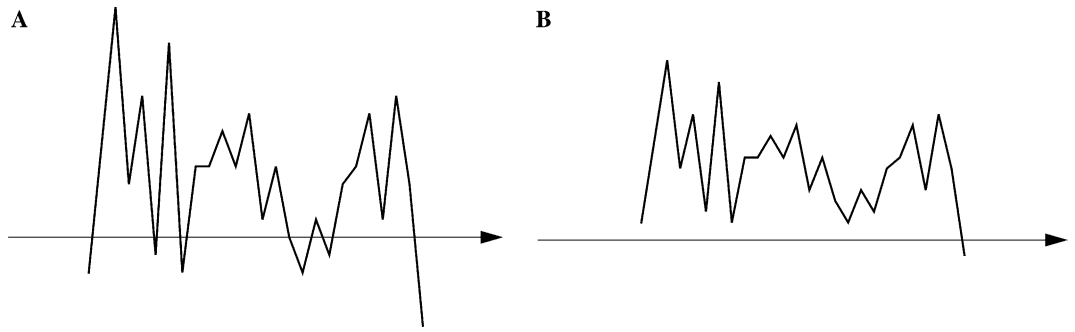

Fig. 3. Sequence (B) is a similarity transformation of (A).

scaling and shifting are shape-preserving transformations, known as *similarity transformations* in the mathematical field of transformational geometry [34]. Fig. 3 illustrates a time-series sequence before and after a similarity transformation. For the rest of the paper, we adopt this well-established geometrical perspective for our notion of similarity.

Remark 1. In the literature, the term "similarity" often refers to *approximate matching* (Definition 1). We distinguish between the two, and reserve this term for searches that involve shifting and scaling transformations.

An *n-sequence* X is a sequence $\{x_1, \ldots, x_n\}$ of real numbers. A pair of reals (a, b) defines a *similarity transformation* $T_{a,b}$ over n-sequences by mapping each element x_i to $a * x_i + b$. In a transformation $T_{a,b}$, we call a the *scale factor* and b the *shift factor*. If a is 1, the transformation is a pure *shift*; if b is 0, it is a pure *scaling*. We will assume that all similarity transformations are *non-degenerate*, i.e., that $a \neq 0$. In fact, we will further assume that $a > 0$; this restriction on a implies that a sequence symmetric to X w.r.t. the x-axis is not considered similar to it. As we describe in Section 2.4, this restriction turns out to be no loss of generality for our method.

Therefore, we take the group of transformations G to be the set of all $T_{a,b}$ where $(a, b) \in [R^+ \times R]$. Then, by Definition 2 we have that sequences X and Y are similar if there is a transformation in G mapping Y to X; that is, if there exist some $(a, b) \in [R^+ \times R]$ such that $X = T_{a,b}(Y)$. The set of all sequences similar to a given one is called its *similarity class*.

Definition 3. The *similarity class* of X consists of all n-sequences Y that are similar to it.

We shall denote the similarity class of X by X^*.

2.2. Normal forms of similarity classes

This *similarity relation* is reflexive, symmetric, and transitive:

- **Reflexivity:** for any sequence X, $X = T_{1,0}(X)$ [the *identity* transformation];
- **Symmetry:** if $X = T_{a,b}(Y)$ then $Y = T_{1/a,-b/a}(X) = T_{a,b}^{-1}(X)$ [the *inverse* of $T_{a,b}$];

- **Transitivity:** if $X = T_{a,b}(Y)$ and $Y = T_{c,d}(Z)$, then $X = T_{ac,ad+b}(Z) = (T_{a,b} * T_{c,d})(Z)$ [the non-commutative *product* of $T_{a,b}$ and $T_{c,d}$].

Therefore, it is an *equivalence relation*, partitioning all n-sequences into similarity classes. Note that the same would be true if our set of transformations were restricted to pure shifts. The identity transformation is a pure shift; the inverse of a shift is a shift; and the product of two shifts is also a shift. This allows us to conclude that the set of all shifts of a given sequence is an equivalence class. The same is true of the set of all scalings.

To be able to refer to similarity classes, we need to have a unique representative for each class. *Normal sequences* serve this purpose.

Definition 4 (*Normal sequence*)**.** Let X be an n-sequence; $X = \{x_1, x_2, \ldots, x_n\}$. X is *normal* if $\mu(X) = 0$ and $\sigma(X) = 1$, where μ and σ denote the *average* and *deviation* of X, respectively:

$$\mu(X) = (1/n) \sum_{1 \leqslant i \leqslant n} x_i; \quad \sigma(X) = \left((1/n) \sum_{1 \leqslant i \leqslant n} (x_i - \mu(X))^2 \right)^{1/2}.$$

We shall feel free to drop the argument X to μ and σ when the context is not ambiguous, treating them as constants.

Proposition 5. *Let X be normal and Y be similar to X. Then, Y is normal only if it is identical to X.*

Proof. $Y = T_{a,b}(X)$ for some $(a, b) \in [R^+ \times R]$. Then, $\mu(Y) = b$ and $\sigma(Y) = a$:

$$\begin{aligned} \mu(Y) &= (1/n) \sum_{1 \leqslant i \leqslant n} y_i = (1/n) \sum_{1 \leqslant i \leqslant n} (ax_i + b) = (a/n)\mu(X) + b = b; \\ \sigma^2(Y) &= (1/n) \sum_{1 \leqslant i \leqslant n} (y_i - \mu(Y))^2 = (1/n) \sum_{1 \leqslant i \leqslant n} (ax_i + b - b)^2 \\ &= (a^2/n) \sum_{1 \leqslant i \leqslant n} (x_i - 0)^2 = a^2 * \sigma^2(X) = a^2. \qquad \square \end{aligned}$$

This means that a similarity class has exactly one normal member; we will call it the *normal form* of all the members of the class. Thus we have found a unique representative for each equivalence class of similar sequences.

Given any n-sequence X, $\hat{X}$ denotes its normal form; $\hat{X}$ is easily computable for any sequence X. If μ is the average of X, and σ is the deviation of X, we have shown that $X = \sigma * \hat{X} + \mu$. Therefore, we can compute $\hat{X}$ from X by the inverse transformation:

$$\hat{X} = T^{-1}_{\sigma,\mu}(X) = T_{1/\sigma,-\mu/\sigma}(X).$$

We refer to this computation as *normalization* of X, and to σ, μ as its *normalization parameters*.

Note that normal forms provide a simple solution for the *similarity query* 2.1; we know that Q and S are similar if and only if $\hat{Q} = \hat{X}$. Normal forms will also be used for *similarity fingerprints*, a central part of the bounded similarity framework.

2.3. Similarity distance

Given two sequences X and Y, we define the *similarity distance* between X and Y, denoted $D_S(X, Y)$, as the normalized Euclidean distance between their normal forms.

Definition 6 (*Similarity distance D_S*)**.**

$$D_S(X, Y) = \sqrt{(1/n)} * D_E(\hat{X}, \hat{Y}) = D_N(\hat{X}, \hat{Y}),$$

where D_E is the *Euclidean distance*:

$$D_E(X, Y) = \sum_{1 \leqslant i \leqslant n} (x_i - y_i)^2$$

and D_N is the *normalized Euclidean distance*:

$$D_N(X, Y) = \sqrt{(1/n)} * D_E(X, Y).$$

The intuition behind normalized Euclidean distance is that it represents the average distance per point between two sequences. For example, the deviation of a sequence X can be thought of as the normalized Euclidean distance between X and a constant sequence A all of whose values are the average of X:

$$\sigma(X) = \sqrt{(1/n)} * D_E(X, A).$$

Euclidean distance is a standard distance metric in discrete signal processing (DSP) [12]. Since we will be using techniques from DSP for our fingerprint strategy (Section 3), we have chosen to use it here. In principle, any proper distance metric for n-sequences can be used instead of normalized Euclidean distance.

Note that, given two similarity classes, the similarity distance will be the same for any pair of sequences from these classes. Essentially, it is the distance between similarity classes rather than between individual sequences. It is easy to see that similarity distance satisfies the criteria of a distance metric:

- it is non-negative and symmetric;
- it obeys the triangle inequality;
- it is effectively computable.

Remark 2. There are other distance measures that also capture our definition of similarity (Definition 2). For example, given Q and S, one can use the minimum distance between Q and all $S' \in S^*$ (the equivalence class of S); however, this distance measure is not symmetric. Given Q and S, we can also use the minimum distance between Q' and S' for all $Q' \in Q^*$ and $S' \in S^*$. However, by choosing the members of Q^* and S^* with arbitrarily small deviations, such a distance will always approach 0. Our distance metric D_S does not suffer from these defects.

2.4. Bounded similarity query

We are now ready to define the *bounded similarity query*.

Definition 7 (*Bounded similarity query*)**.** Given a query sequence Q, a database B consisting of sequences of the same length as Q, a tolerance $\epsilon \geqslant 0$, and bounds $0 < l_a \leqslant u_a$ and $l_b \leqslant u_b$, find all tuples $[S, a, b]$ where S is a sequence in B and $(a, b) \in [R^+ \times R]$ such that $D_N(Q, aS + b) \leqslant \epsilon$, $l_a \leqslant a \leqslant u_a$, and $l_b \leqslant b \leqslant u_b$.

Remarks:

(1) Every bounded similarity query is fully specified by the following parameters:

$$\{\epsilon, l_a, u_a, l_b, u_b\}.$$

(2) We assume that D is the normalized Euclidean distance (Definition 6). Other distance metrics may be used; for example, regular Euclidean distance may be used, multiplying ϵ by a factor of $\sqrt{n}$, where n is the length of Q.
(3) We allow u_a and u_b to be ∞ and l_b to be $-\infty$; this effectively removes the corresponding bounds from a and b.
(4) The distance measure used here is the *normalized Euclidean distance* D_N, rather than the *similarity distance* D_S; see Definition 6 for both. D_S is more central to the semantics of similarity; it will be used for the so-called *internal query* (Section 4.1). However, it is not appropriate here, since it returns the same distance for all members of a given similarity class, ignoring the magnitude of the shift and scale parameters. Section 6 derives the formulas that translate between bounded similarity queries and their internal representation. It is an open problem whether other distance measures can be used for for bounded similarity querying, while preserving the desirable properties of similarity querying enunciated in Section 1.1.
(5) Note that the restriction that $a > 0$ does not preclude us from finding sequences where $a < 0$. To find them, we just reflect our query sequence with respect to the x-axis (by negating all values), then perform a query with this new sequence.
(6) The problem is not formulated in terms of subsequences, unlike approximate matching. However, by regarding every subsequence as a separate sequence in the database, we see that the case of subsequence search is included. We further address this case in Section 5.2.

The following queries can be viewed as special cases of bounded similarity queries; assume we are given a query sequence Q and a database B consisting of sequences of the same length as Q:

Exact bounded similarity: Given bounds $0 < l_a \leqslant u_a$ and $l_b \leqslant u_b$, find all $[S, a, b]$ such that $Q = aS + b$.
(Exact) Similarity: Find all $[S, a, b]$ such that $Q = aS + b$.
Approximate similarity: Given a tolerance $\epsilon \geqslant 0$, find all $[S, a, b]$ such that $D_N(Q, aS + b) \leqslant \epsilon$.
Scaling: Given a tolerance $\epsilon \geqslant 0$ and bounds $0 < l_a \leqslant u_a$, find all $[S, a]$ such that $D_N(Q, aS) \leqslant \epsilon$.
Exact scaling: Given bounds $0 < l_a \leqslant u_a$, find all $[S, a]$ such that $Q = aS$.
Shifting: Given a tolerance $\epsilon \geqslant 0$ and bounds $l_b \leqslant u_b$, find all $[S, b]$ such that $D_N(Q, S + b) \leqslant \epsilon$.

Exact shifting: Given bounds $l_b \leqslant u_b$, find all $[S, b]$ such that $Q = S + b$.
Approximate match: Given a tolerance $\epsilon \geqslant 0$, find all S such that $D_N(Q, S) \leqslant \epsilon$.
Exact match: Find all S such that $Q = S$.

When we say that some query is a special case of the bounded similarity query, it means that this query corresponds to some tuple $\{\epsilon, l_a, u_a, l_b, u_b\}$ specifying a bounded similarity query. For example, the similarity query corresponds to the tuple $\{0, 0, \infty, -\infty, \infty\}$. As another example, the approximate match query corresponds to the tuple $\{\epsilon, 1, 1, 0, 0\}$.

Our framework for bounded similarity queries can handle all these queries in a uniform fashion, with a single search technique and a single index data structure. This flexibility is due to our fingerprinting strategy, discussed in the next section.

3. Fingerprint mechanism for bounded similarity queries

In this section, we present the notion of *similarity fingerprints* for time-series sequences. These fingerprints form the basis both of the in-core and disk-based bounded similarity querying algorithms. We show that similarity fingerprints have several desirable characteristics which play an important role in enabling an efficient implementation of bounded similarity querying.

3.1. Similarity fingerprints

In general, a fingerprint of an n-sequence S is a sequence $F(S)$ consisting of one or more numbers; the size of $F(S)$ is constant and presumably small. Assuming that the *fingerprint distance metric* D_F preserves proximity between sequences, the fingerprint mechanism provides *fast rejection*, filtering out most of the sequences that are not similar.

We identify the following criteria of a good fingerprint mechanism:

- **Compactness:** The distance between the fingerprints of two n-sequences can be computed in constant time.
- **Validity:** If S is a valid answer to query Q, then the comparison of $F(S)$ and $F(Q)$ should return *TRUE*:

$$D_S(Q, S) \leqslant \epsilon \Longrightarrow D_F(F(Q), F(S)) \leqslant \epsilon.$$

- **Accuracy:** If the distance between $F(Q)$ and $F(S)$ is within ϵ, then S is highly likely to be a valid query answer.
- **Updateability:** Computing the fingerprints of all n-subsequences of an N-sequence, for N much larger than n, can be done in $O(N)$ time.
- **Continuity:** Given two adjacent subsequences, the difference between the corresponding coefficients of their indices is likely to be small.

We now define our fingerprint function F as well as the fingerprint distance function D_F. The similarity fingerprint of a data sequence consists of its average, its standard deviation, and the first few coefficients of the discrete fourier transform (DFT) of the *normal form* (Definition 4) of the sequence:

Definition 8. A *similarity fingerprint* $F(X)$ of an n-sequence $X = \{x_1, \ldots, x_n\}$ is the tuple

$$[\mu(X), \sigma(X), DFT_1(\hat{X}), \ldots, DFT_m(\hat{X})],$$

where m is a small constant and $DFT_k(S)$ is the kth coefficient of the *discrete Fourier transform* of an n-sequence S:

$$DFT_k(s_0, \ldots, s_{n-1}) = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} (s_j e^{-j(2\pi i)m/n}),$$

where $i = \sqrt{-1}$.

Note that DFT_k is a complex number: $DFT_k = a_k + b_k i$ for some $a_k, b_k \in R$. Thus, $F(X)$ is specified by a sequence of $2m + 2$ real values. Also note that we are not including $DFT_0(\hat{X})$ in the fingerprint, since its coefficients are always 0.

Our fingerprint function is similar to the one used for approximate matching in [2]. However, we: (a) normalize our sequences, and (b) include the *normalization parameters* with the fingerprint.

The distance between fingerprints is defined next.

Definition 9. The *fingerprint distance* D_F between $F(X)$ and $F(Y)$ is the Euclidean distance between the real-valued sequences for $F(X)$ and $F(Y)$ divided by $\sqrt{n}$, where $n = |X| = |Y|$.

By taking m to be a small constant, our fingerprint mechanism is *compact*. In the rest of the section, we establish its validity, accuracy, updateability, and continuity.

3.2. Validity

Validity. If S is a valid query answer, then the comparison of $F(S)$ and $F(Q)$ should return *TRUE*.

To establish the validity of fingerprinting, we need to show that

$$D_S(X, Y) \leqslant \epsilon \Longrightarrow D_F(F(X), F(Y)) \leqslant \epsilon.$$

We make use of the fact that the *DFT* is a *linear function*, i.e.,

$$DFT_k(aX + bY) = aDFT_k(X) + bDFT_k(Y)$$

for all scalars a and b. And we make use of the fact that the coefficients of DFT_0 for normal sequences are both 0. Also, we rely on *Parseval's theorem*, well-known in DSP:

$$\sum_{0 \leqslant k \leqslant n-1} |DFT_k(X)|^2 = \sum_{1 \leqslant i \leqslant n} |x_i|^2.$$

First, we show that $D_S(X, Y) \geqslant D_F(F(X), F(Y))$:

$$\begin{aligned} D_S(X, Y) &= \sqrt{1/n} * D_E(\hat{X}, \hat{Y}) = \left(\sum_{1 \leqslant i \leqslant n} |\hat{X}[i] - \hat{Y}[i]|^2 \right)^{1/2} / \sqrt{n} \\ &= \left(\sum_{0 \leqslant k \leqslant n-1} |DFT_k(\hat{X} - \hat{Y})|^2 \right)^{1/2} / \sqrt{n} \\ &\geqslant \left(\sum_{0 \leqslant k \leqslant n} |DFT_k(\hat{X} - \hat{Y})|^2 \right)^{1/2} / \sqrt{n} = D_F(F(X), F(Y)). \end{aligned}$$

It immediately follows that $D_F(F(X), F(Y)) \leqslant \epsilon$ whenever $D_S(X, Y) \leqslant \epsilon$.

3.3. Accuracy

Accuracy. If the comparison of $F(S)$ and $F(Q)$ returns *TRUE*, S is highly likely to be a valid query answer.

To establish accuracy, we want to know how likely it is that $D_S(X, Y) \leqslant \epsilon$ provided that $D_F(F(X), F(Y)) \leqslant \epsilon$. The cases when $D_F(F(X), F(Y)) \leqslant \epsilon$ but $D_S(X, Y) \leqslant \epsilon$ represent *false alarms*, and we want to minimize their occurrence. Therefore, we would like the ratio $D_F(F(X), F(Y))/D_S(X, Y)$ to be close to 1.

The actual ratio strongly depends on the nature of the data sequences. It is worst in the case of *white noise*, when

$$D_F(F(X), F(Y))/D_S(X, Y) = (m + 1)/n.$$

However, a large class of signals (the *colored noises*) have a *skewed energy spectrum*, including most time-seried data, which implies that:

The amplitude of DFT_m decreases rapidly for increasing values of m.

In fact, the decrease is as $O(f^{-b})$ [12], where:

- $b = 1$ for *pink noise*, such as musical scores;
- $b = 2$ for *brown noise*, such as stock movements and exchange rates;
- $b > 2$ for *black noise*, such as rainfall patterns.

As a result, 2–3 complex coefficients are sufficient to provide good accuracy for most applications, including the one used for our experiments. The actual number of coefficient needed for a specific amount of accuracy depends on the distribution of the sequence as well as its length. We will assume that the parameter m in the fingerprint is 3, and each fingerprint consists of $2m + 2 = 8$ real values.

3.4. Updateability

Updateability. Given the similarity fingerprint of a subsequence $\{x_k, \ldots, x_{k+n-1}\}$, it is possible to compute the fingerprint for $\{x_{k+1}, \ldots, x_{k+n}\}$ in constant time.

When computing similarity fingerprints of all subsequences of length n for a much longer sequence of length N, the efficiency of the algorithm hinges on the updateability property of the similarity fingerprints. That is, we can simply *update* the fingerprint value as we move along rather than recompute it for every subsequence.

Let X_k be the first subsequence, and X_{k+1} be the second subsequence. We show how to compute the fingerprint $F(X_{k+1})$ from the fingerprint $F(X_k)$ in constant time. As inputs to the update step, we assume that we have the values of the following expressions:

$$\sum_{k\leqslant j\leqslant k+n-1} x_j, \sum_{k\leqslant j\leqslant k+n-1} (x_j)^2, \mu(X_k), \sigma(X_k), DFT_1(X_k), \ldots, DFT_m(X_k), F(X_k).$$

We also assume that all constants (such as $1/n$) are pre-computed. During the update step, we obtain the values for the above expressions with $k+1$ instead of k; the computation proceeds as follows:

(1) Increment k to $k+1$;
(2) Look up x_{k+1}, x_{k+n};
(3) Compute $\sum_{k+1\leqslant j\leqslant k+n} x_j$. This involves one subtraction and one addition:

$$\sum_{k+1\leqslant j\leqslant k+n} x_j = \left(\sum_{k\leqslant j\leqslant k+n-1} x_j\right) - x_k + x_{k+n};$$

(4) Compute $\sum_{k+1\leqslant j\leqslant k+n} (x_j)^2$, using two multiplications, one subtraction, and one addition:

$$\sum_{k+1\leqslant j\leqslant k+n} x_j^2 = \left(\sum_{k\leqslant j\leqslant k+n-1} x_j^2\right) - x_k^2 + x_{k+n}^2;$$

(5) Compute $\mu(X_{k+1})$, using one multiplication:

$$\mu(X_{k+1}) = \mu_{k+1} = (1/n) \sum_{k+1\leqslant j\leqslant k+n} x_j;$$

(6) Compute $\sigma(X_{k+1})$, using two multiplications, one subtraction, and one square root:

$$\sigma^2(X_{k+1}) = \sigma_{k+1}^2 = (1/n) \left(\sum_{k+1\leqslant j\leqslant k+n} x_j^2\right) - \mu_{k+1}^2;$$

(7) Compute $DFT_1(X_{k+1}), \ldots, DFT_m(X_{k+1})$, using one subtraction, one addition, and two multiplications for each index:

$$DFT_m(X_{k+1}) = \frac{1}{\sqrt{n}} \sum_{0 \leqslant j \leqslant n-1} (x_{j+k+1} \mathrm{e}^{-\mathrm{j}(2\pi\mathrm{i})m/n})$$
$$= \frac{1}{\sqrt{n}} \sum_{1 \leqslant j \leqslant n} (x_{j+k} \mathrm{e}^{-(\mathrm{j}-1)(2\pi\mathrm{i})m/n}) = \mathrm{e}^{(2\pi\mathrm{i})m/n} \left(DFT_m(X_k) + \frac{x_{n+k} - x_k}{\sqrt{n}} \right);$$

(8) Compute the similarity fingerprint of X_{k+1}, using one division for each index. Here, we rely on the linearity of *DFT*s, and on the fact that, for a constant sequence of 1's (denoted **1**), $DFT_m(\mathbf{1})$ is 0 when $m > 0$:

$$DFT_m(\overline{X_{k+1}}) = DFT_m(X_{k+1}/\sigma_{k+1} - \mu_{k+1}/\sigma_{k+1})$$
$$= (1/\sigma_{k+1}) DFT_m(X_{k+1}) - (\mu_{k+1}/\sigma_{k+1}) DFT_m(\mathbf{1})$$
$$= DFT_m(X_{k+1})/\sigma_{k+1}.$$

Note that the above algorithm is *on-line*, suitable in the context of *data streams*, when the data are streaming past and we can never back over it. It therefore holds promise of being useful in a *continuous query* setting [7].

3.5. Continuity

Continuity: Given two adjacent subsequences, the difference between the corresponding coefficients of their indices is likely to be small.

The continuity property assumes *subsequence search*, discussed in Section 5.2. This property has been acknowledged in this literature before; for example, in [9], the concept of a *trivial match* is introduced specifically for matches between adjacent or near-adjacent sequences.

The argument for continuity of the similarity fingerprint function is a "heuristic" statistical argument. We denote adjacent sequences by $X_0 = \{x_0, \ldots, x_{n-1}\}$ and $X_1 = \{x_1, \ldots, x_n\}$; we assume that n is reasonably large, so that $1/n$ is considered to be a small constant. We denote the corresponding similarity fingerprints by (μ_0, σ_0, F_0) and (μ_1, σ_1, F_1), respectively. We denote the order of the expected value of an expression by $\approx$.

We proceed by considering each element of the index separately.

(1) The expected value of $|\mu_1 - \mu_0|$ is $|x_n - x_0|/n$:

$$|\mu_1 - \mu_0| = (1/n) \left(\sum_{1 \leqslant i \leqslant n} x_i - \sum_{0 \leqslant i \leqslant n-1} x_i \right) = |x_n - x_0|/n.$$

(2) The expected value of $|\sigma_1 - \sigma_0|$ is in the order of σ_0/n:

$$|\sigma_1 - \sigma_0|(\sigma_1 + \sigma_0) = |\sigma_1^2 - \sigma_0^2| = \frac{1}{n} |(x_n - \mu_1)^2 + (x_0 - \mu_0)^2$$
$$+ \sum_{1 \leqslant j \leqslant n-1} ((x_j - \mu_1)^2 - (x_j - \mu_0)^2)|$$
$$= |(x_n - x_0)(x_n - \mu_1 + x_0 - \mu_0)|/n \approx (\sigma_0/n)(\sigma_1 + \sigma_0).$$

(3) The expected value of $|DFT_m(\overline{X_1}) - DFT_m(\overline{X_0})|$ is in the order of $|DFT_m(\overline{X_0})|/n + 1/\sqrt{n}$. Here, we use the equations for *DFT*s derived in Section 3, as well as the ones derived above:

$$\begin{aligned}
|DFT_m(\overline{X_1}) - DFT_m(\overline{X_0})| &= |DFT_m(X_1)/\sigma_1 - DFT_m(\overline{X_0})| \\
&= \left| \frac{e^{(2\pi i)m/n}}{\sigma_1} \left(\sigma_0 DFT_m(\overline{X_0}) + \frac{x_n - x_0}{\sqrt{n}} \right) - DFT_m(\overline{X_0}) \right| \\
&= \left| DFT_m(\overline{X_0}) \left(\frac{\sigma_0}{\sigma_1} e^{(2\pi i)m/n} - 1 \right) + \frac{x_n - x_0}{\sigma_1 \sqrt{n}} e^{(2\pi i)m/n} \right| \\
&\approx |DFT_m(\overline{X_0})((1+1/n)(1 - 2\pi m/n) - 1) + (1 + 1/n)(1 - 2\pi m/n)/\sqrt{n}| \\
&\approx |DFT_m(\overline{X_0})|/n + O(1/\sqrt{n}).
\end{aligned}$$

4. In-core bounded similarity querying

This section presents an efficient in-core algorithm for bounded similarity querying. We only discuss the *whole-sequence* case; subsequences, as well as indexing, are discussed in Section 5. Note that the same algorithm, as well as its subsequence version, can be used for disk-based data, retrieving the desired data directly from disk; this is known as a *linear scan.*

4.1. Internal representation of similarity queries

As the first step in query evaluation, each bounded similarity query is translated to the following *internal query*:

Definition 10 (*Internal query*)**.** Given a query sequence Q of length n, an internal tolerance $\epsilon_i \geqslant 0$, bounds $l_\mu \leqslant u_\mu$ and $l_\sigma \leqslant u_\sigma$, and a database B consisting of sequences of the same length as Q, find all sequences S in B such that $D_S(Q,X) \leqslant \epsilon_i$, $l_\mu \leqslant \mu_X \leqslant u_\mu$, and $l_\sigma \leqslant \sigma_X \leqslant u_\sigma$.

While a bounded similarity query can be viewed as an optimization problem, where we seek the best shift and scale parameters for minimizing sequence distance, the internal query that forms a key step of our algorithm can be viewed as a multidimensional range query [15] over the normalization parameters of the sequences in the database, involving no optimization.

As mentioned before (Section 2.4), every bounded similarity query is fully specified by the following parameters:

$$\{\epsilon, l_a, u_a, l_b, u_b\};$$

we refer to them as the *external specification*, or the *external query*, since these parameters are set externally by the user. By contrast, an internal query is fully specified by the following tuple of parameters that are invisible to the user:

$$\{\epsilon_i, l_\mu, u_\mu, l_\sigma, u_\sigma\}.$$

ϵ_i	$\sqrt{2-2\sqrt{1-\epsilon^2/\sigma^2}}$
l_μ	$(\mu_Q - u_b - \epsilon)/u_a$
u_μ	$(\mu_Q - l_b + \epsilon)/l_a$
l_σ	$(\sigma_Q + \epsilon)/u_a$
u_σ	$(\sigma_Q + \epsilon)/l_a$

Fig. 4. Computing the parameters of the internal query.

Given an external specification, an internal specification can be obtained from it using the conversion formulas in Fig. 4. These formulas are derived in Section 6. Note that an earlier draft of this work [16] contained a bug in these formulas, omitting ϵ from the formulas for the bounds; this was discovered during the implementation process and corrected.

Internal queries allow us to determine, based on similarity fingerprints, which sequences might be a *potential match* for a bounded similarity query.

Definition 11 (*Potential match*)**.** A database (sub)sequence S, with a fingerprint $(\mu_S, \sigma_S, F(S))$, is a *potential match* for a query sequence Q, with a fingerprint $(\mu_Q, \sigma_Q, F(Q))$, if:

$$D_F(F(Q), F(S)) \leqslant \epsilon_i, \quad l_\mu \leqslant \mu_S \leqslant u_\mu, \quad \textit{and} \quad l_\sigma \leqslant \sigma_S \leqslant u_\sigma.$$

Note that a potential match does not necessarily constitute an answer to our internal query; some potential matches turn out to be *false alarms*. This is due to the inherent lossiness of fingerprints: for any sequences S_1, S_2 and ϵ, if $D_S(S_1, S_2) \leqslant \epsilon_i$, then $D_F(F(S_1), F(S_2)) \leqslant \epsilon_i$ but not vice versa. As a result, *post-processing step* needs to be performed for each potential match to determine whether it actually satisfies the internal query. Specifically, we need to check that $D_S(Q, X) \leqslant \epsilon_i$; the rest of the requirements of the internal query, in particular the bounds on μ_S and σ_S, are guaranteed to hold of all potential matches.

4.2. The filtering step

If a sequence S does not satisfy the internal query, it will not satisfy the external query, either. The reverse, however, is not true. A *filtering* step needs to be performed to check that it also satisfies the user query. This step answers the following question:

Definition 12 (*Filtering step*)**.** Given a subsequence X, do there exist a and b within the bounds specified by the external query such that $D_N(Q, aX + b) \leqslant \epsilon$?

The filtering step relies on the following proposition, proved in Section 6:

Proposition 13. *Assume we are given an external query* $\{\epsilon, l_a, u_a, l_b, u_b\}$*, a query sequence* Q*, and a data sequence* S*. The bounds* (l_a, u_a) *and* (l_b, u_b) *define an (open) rectangle* R *in the* (a, b)*-coordinate space. Let* t *be the following linear transformation of the* (a, b)*-coordinate space:*

$$t(a,b) = (a\sigma_S - \sigma_Q \hat{X}\hat{Q}, a\mu_S + b - \mu_Q),$$

where $\hat{X}\hat{Q}$ is the product of $\hat{X}$ and $\hat{Q}$, see Definition 15. *t transforms R into a parallelogram $t(R)$. Let (a_t, b_t) be the point in $t(R)$ closest to the origin; let (a_0, b_0) be the point in R-coordinate space such that $t(a_0, b_0) = (a_t, b_t)$. Then, S passes the filter if and only if $D_N(Q, a_0X + b_0) \leqslant \epsilon$.*

This filtering is one of the features unique to bounded similarity querying. It represents a potential performance disadvantage of our approach, as a trade-off to achieve the generality. We will be discussing performance issues in Section 7.

4.3. Algorithm for in-core bounded similarity querying

We now summarize and discuss the algorithm for in-core bounded similarity querying. We are given some query sequence Q and some query specification $\{\epsilon, l_a, u_a, l_b, u_b\}$. We assume that a similarity fingerprint $F(S)$ is already stored with each data sequence S.

(1) Compute a *similarity fingerprint* for the query sequence Q.
(2) Obtain the *internal specification* $\{\epsilon_i, l_\mu, u_\mu, l_\sigma, u_\sigma\}$ from the external one, using the conversion formulas in Fig. 4.
(3) Find *potential matches* to the internal query (Definition 11), by comparing the fingerprint of Q with the fingerprint of each sequence.
(4) Perform *post-processing* of potential matches; for each potential match S, check that $D_S(Q, S) \leqslant \epsilon_i$; D_S is defined in 6.
(5) Perform *filtering*; given a subsequence S, return it only if there exist a and b within the bounds specified by the external query, such that $D_N(Q, aX + b) \leqslant \epsilon$.

The compactness property of similarity fingerprints ensures that each fingerprint distance calculation can be done quickly, in time $O(m)$. The validity property of similarity fingerprints ensures that the set of potential matches of Q includes all sequences that are actual matches for our internal query; there are no *false dismissals*. In Section 6, we prove that this implies there are no false dismissals to the original user-defined query. In the same section, we also provide the necessary proofs and calculations for the filtering step.

5. Bounded similarity querying made practical

The bounded similarity querying algorithm presented in the last section is rather simplistic. It does not allow for several possibilities that are common in practical database applications:

- Data may be too large to fit in main memory. When data reside on disk, data transfer from disk to main memory is extremely time-consuming, and one cannot afford to read in all the data in order to answer each query [44]. We need to provide an indexing method.
- Data sequences may be longer than the query sequence; in this case, we are interested in *subsequences* that are similar to the query. We need to extend bounded similarity querying to the subsequence case.

- The query sequence may be longer than the length of the fingerprinted (sub)sequences. We need to be able to answer such queries without recomputing the fingerprints or rebuilding the index.

In this section, we discuss how to extend the bounded similarity querying algorithm (Section 4.3) to address these possible scenarios.

5.1. Indexing for bounded similarity querying

It is assumed that the data are too large to fit in main memory, but rather reside on disk. Because data transfer from disk to main memory is extremely time-consuming, one cannot afford to read in all the data in order to answer each query [44]. It is therefore necessary to introduce an indexing method. The idea is to do a linear scan of the data only once in order to build an index structure. Queries are then answered with a search on the index structure, that returns all sequences of interest.

This section describes our indexing technique for similarity querying. In particular, we use our index to quickly find potential matches to the internal query, as described in Section 4.1. Any index structure can be used, as long as it allows spatial access methods for range queries of multidimensional points [15,41]; our implementation uses an $R*$-tree [4,14]. The entries in the index are the sequence fingerprints, which are treated as multidimensional points.

First, a multidimensional *query rectangle* R_Q is computed from the internal query. R_Q has the same number of dimensions as $F(Q)$, the similarity fingerprint of Q (Section 3). The first dimension ranges from l_μ to u_μ, and the second dimension ranges from l_σ to u_σ. These dimensions account for the bounds on the average and standard deviation of a matching subsequence. The rest of the dimensions correspond to the DFT coefficients in $F(Q)$. If a DFT coefficient has some value u, then the corresponding dimension of R_Q ranges from $u - \epsilon_i$ to $u + \epsilon_i$.

Then, a range query is performed on the index structure, searching for all points (fingerprints) that intersect R_Q. These are the potential matches. The other steps of the index-based version of the bounded similarity query algorithm are the same as for the in-core version.

5.2. Bounded subsequence similarity queries

A *bounded subsequence similarity query* is a generalization of the bounded similarity query (Definition 7):

Definition 14 (*Bounded subsequence similarity query*)**.** Given a query sequence Q, a database B consisting of sequences of at least as long as Q, a tolerance $\epsilon \geqslant 0$, and bounds $0 < l_a \leqslant u_a$ and $l_b \leqslant u_b$, find all tuples $[S, j, a, b]$ where S is a subsequence at offset j of some sequence X in B, and $(a, b) \in [R^+ \times R]$ such that $D_N(Q, aS + b) \leqslant \epsilon$, $l_a \leqslant a \leqslant u_a$, and $l_b \leqslant b \leqslant u_b$.

Note that subsequence versions can be analogously defined for all the queries in Section 2.4.

As before, we only need to modify the *finding potential matches* step of the bounded similarity query algorithm (Section 4.3). Specifically, we need to compare the fingerprint of Q with the fingerprint of each *subsequence* S of each sequence X in the data set.

The updateability of similarity fingerprints, discussed in Section 3, ensures that this can be done efficiently. In particular, given a sequence X of length m, where $m \geqslant n$, the fingerprints for all n-subsequences of X can be computed in time $O(m)$ rather than $O(nm)$.

5.3. Indexing for the subsequence case

Above, we discussed an in-core algorithm for bounded subsequence similarity querying. Let us now consider an index-based version. An obvious idea is to use each similarity fingerprint as the index for its associated subsequence. Unfortunately, such an index is not useful, due to a simple observation: a sequence of length m contains $m - n + 1$ n-subsequences, so there would be $m - n + 1$ fingerprints to store in the index for m data values. Such space overhead renders indexing less efficient than a direct sequential scan of the data [2].

This problem is overcome with the *minimum bounding rectangle* (MBR) technique, introduced in [2]. This technique significantly reduces the size of the indexing structure, at the cost of introducing some additional false alarms that must be eliminated by later steps; it is described next.

MBR partitions the set of similarity fingerprints into sequences of successive fingerprints. Because of the continuity property of adjacent similarity fingerprints, each group of successive fingerprints can be approximated by its *minimum bounding hyper-rectangle*. Assuming that each group contains a fair number of fingerprints, there will be relatively few rectangles compared to the number of subsequence fingerprints. These rectangles are stored in the index.

Given an internal query, a *query rectangle* is computed as before (Section 5.1). To find potential matches, we perform a range query on the index structure to find all rectangles that intersect R_Q. After the range query, we need an *extraction step* which allows us to extract the similarity fingerprints for the potential matches out of the returned rectangles.

Extraction step: Given a matching rectangle, we need to check which of the fingerprints contained within this rectangle actually intersect the query rectangle. Each fingerprint that intersects the query rectangle corresponds to a potential match to the internal query.

It is important to note that the rectangle heuristic only provides a method for grouping fingerprint points, but it does not affect these fingerprint values in any way. Therefore, there will be exactly the same number of fingerprints which intersect the query rectangle regardless of the rectangle heuristic employed.

5.4. Handling queries of varying length

So far, we have taken n to be both the length of the query and the length of the fingerprinted subsequences. In this section, we discuss how to handle longer queries, without having to recompute all fingerprints or to rebuild the entire index structure every time that we want to ask a query of a different length. In this section, we present a past approach and our new approach to solving this problem. We end with a discussion of their relative merits. Experimental results for queries of varying lengths are presented in Section 8.

In the rest of this section, n denotes the length of fingerprinted subsequences, and q denotes the length of a query. n represents the smallest reasonable query length; for example, the smallest

meaningful subsequence length for a database of daily stock prices would be a quarter's worth of data. We further assume that q is an exact multiple of n; $q = pn$, for some $p \geqslant 1$. If this is not the case, we simply use our method on the longest prefix of the query which is a multiple of n, but we do our post-processing step on the entire query, in order to weed out false alarms.

Note that a subsequence search is assumed, because a whole-sequence scenario makes no sense here. If all data sequences had length n, a query of length q greater than n would return nothing.

5.4.1. Multi-piece search

Multi-piece search is a method for handling queries of varying length introduced in [13]. The query sequence Q of length q is broken up into p sub-queries $\{Q_0, Q_1, \ldots, Q_{p-1}\}$, each of length n. Similarly, all data subsequences S of length q can be viewed as p fragments $\{S_0, S_1, \ldots S_{p-1}\}$, each of length n. It can be shown that

$$\text{if } D_N(Q,S) \leqslant \epsilon, \text{ then } D_N(Q_j, S_j) \leqslant \epsilon/\sqrt{p} \text{ for some } 0 \leqslant j \leqslant p-1.$$

Therefore, we simply ask all p sub-queries $\{Q_0, Q_1, \ldots, Q_{p-1}\}$ on the index structure, with a tolerance of $\epsilon_i/\sqrt{p}$, and retrieve all fragments which are similar to at least one subquery Q_j. The, we check all subsequences of length q that contain such a frament anywhere inside, to find those that match our original query.

5.4.2. Stitching

Observe that in the above approach, each of the p sub-queries is completely independent from the others.

Our approach, on the other hand, is to break the query Q into p subqueries $\{Q_0, Q_1, \ldots, Q_{p-1}\}$ as above, but to then use matches to the sub-queries in order to "stitch" together a match to the query.

We begin by asking an internal query for Q_0 on our index structure, using $\epsilon_i, l_\mu, u_\mu, l_\sigma$, and u_σ as specified by the internal query of Q. When the potential matches are returned, we store the offset for each of these matches. We also record the minimal internal distance d_m between the fingerprint of Q_0 and any matching fingerprint.

Because we know that at least d_m of internal distance has been used up by any match to Q_0, we can update the allowable internal tolerance for Q_1:

$$\epsilon_i := \sqrt{\epsilon_i^2 - d_m^2}.$$

Thus we run our internal query for Q_1 with this internal tolerance. The bounds on the average and deviation remain the same. Furthermore, we can use the offsets stored from Q_O as a filter. Specifically, we need only retain those potential matches to Q_1 which can be "stitched" onto some potential match to Q_0. We store only those offsets which pass this test, and we can now discard the offsets for Q_0. We also compute d_m for this sub-query.

We continue in this way for all p sub-queries, using d_m of one query to decrease the tolerance value ϵ_i of the subsequent query, and using the offsets of a previous query to weed out the current offsets, by trying to stitch them together. At the end of this process, we will have a list of potential matches to Q. All that remains is a filtering step (see Section 4.2) on these stitched subsequences to see if indeed they match Q.

5.4.3. Discussion of stitching

The advantage of multi-piece search over stitching is that every sub-query has a smaller tolerance value as compared to the overall tolerance value for a match. This allows for faster querying for each sub-query. However, it has the major disadvantage that there will be a filtering step on a sequence of length q for every matching fragment found by one of the queries of size n. Since a match on a query of size n is an extremely weak criterion in testing for a match on the entire q-length sequence, this method will generate a large number of false alarms, requiring many expensive disk accesses.

Our method attempts to remedy this problem by performing the stitching at index level, thereby weeding out most of the false alarms without necessitating any disk accesses. As j goes from 0 to $p-1$, the similarity query on Q_j results in fewer and fewer false alarms.

Our method is further able to prune false alarms by using the d_m of a sub-query to decrease the internal tolerance value for the subsequent sub-query. If d_m is fairly high for each sub-query, then we will be able to prune many false alarms quickly. However, in the worst case, when a subsequence matches Q with exact similarity (i.e., a distance of 0), the d_m value of each sub-query will be 0, so the internal tolerance value will never decrease. This is potentially inefficient, because the internal tolerance value for Q_0 is the original ϵ_i value from the internal query specification of Q rather than $\epsilon_i/\sqrt{p}$.

The main problem, then, with both of these methods is that their efficiency is highly dependent on the relationship between Q and the data sequences. We explore this relationship in Section 8. Because of the high sensitivity of their efficiency to the specifics of the problem, the decision as to which method to use should be based on the nature of the data sequences and types of queries being asked. Possible future work includes the development of an algorithm which combines the strengths of each method.

6. From external to internal queries and back

In this section, we derive the formulas for the translation between the general similarity query and the internal query, and provide the proofs omitted from Section 4. As before, we denote the normal form of X by $\hat{X}$ and the average and standard deviation of X by μ_X and σ_X, respectively.

As a reminder, we reiterate the definitions of the external and internal queries (Definitions 7 and 10).

Bounded similarity query: Given $Q, \epsilon, (l_a, u_a)$, and (l_b, u_b),
find all $[S, a, b]$ such that $D_N(Q, aS + b) \leqslant \epsilon$, $l_a \leqslant a \leqslant u_a$ and $l_b \leqslant b \leqslant u_b$.
Internal query: Given $Q, \epsilon_i, (l_\mu, u_\mu)$ and (l_σ, u_σ),
find all S such that $D_N(\hat{Q}, \hat{X}) \leqslant \epsilon_i,\ l_\mu \leqslant \mu_S \leqslant u_\mu,\ l_\sigma \leqslant \sigma_S \leqslant u_\sigma$.

The translation needs to be made in both directions:

Direction I (*Computing internal query values*): Given a sequence Q and the values for $\{\epsilon, l_a, u_a, l_b, u_b\}$, find the values for $\{\epsilon_i, l_\mu, u_\mu, l_\sigma, u_\sigma\}$ such that for any sequence S, there exist (a, b) with $[S, a, b]$ satisfying the bounded similarity query *only if* S satisfies the internal query.
Direction II (*Filtering potential matches*): Given a sequence S returned by the internal query, find the values of a and b (if any) such that $[S, a, b]$ satisfies the external query.

6.1. Normalized products

To help with the proofs in the remainder of this section, we define the *normalized product* of sequences, and present some associated propositions.

Let X, Y be two n-sequences. XY, the *normalized product* of X and Y, is defined as follows:

Definition 15. Let $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$ be two sequences of length n. XY is the *normalized product* of X and Y:

$$XY = (1/n) \sum_{1 \leqslant i \leqslant n} (x_i y_i).$$

X^2 denotes the normalized product of X with itself.

It is easy to verify that normalized product operation is commutative, and that it is distributive with respect to sequence sums and differences (i.e., $X(Y + Z) = XY + XZ$). The normalized product has some other useful properties, well known in the field of mathematical statistics (see [32] for details):

Property 1: $XY = \mu_X \mu_Y + \sigma_X \sigma_Y \hat{X}\hat{Y}$
Property 2: $-1 \leqslant \hat{X}\hat{Y} \leqslant 1$
Property 3: $X^2 = \mu_X^2 + \sigma_X^2$
Property 4: $\hat{X}^2 = \mu_{\hat{X}}^2 + \sigma_{\hat{X}}^2 = 0^2 + 1^2 = 1$

In our calculations, we assume that $0 \leqslant \hat{X}\hat{Y} \leqslant 1$, because if $\hat{X}\hat{Y}$ is negative, this means that as the values in X increase, the values in Y tend to decrease, and vice versa [32]. This corresponds to a negative scale factor in our similarity relation, which we disallow; we do not wish these two sequences to be considered similar. (However, as discussed in Section 2.4, such sequences can still be found with our method).

Note that the normalized product has an intimate connection to the normalized distance D_N (Definition 6):

Proposition 16.

$$D_N^2(X, Y) = (X - Y)^2.$$

Proof. Follows from definitions. □

The following two propositions are also about D_N.

Proposition 17.

$$D_N^2(\hat{X}, \hat{Y}) \leqslant 2.$$

Proof.

$$D_N^2(\hat{X}, \hat{Y}) = (\hat{X} - \hat{Y})^2 = \hat{X}^2 + \hat{Y}^2 - 2\hat{X}\hat{Y} = 1 + 1 - 2\hat{X}\hat{Y} = 2 - 2\hat{X}\hat{Y} \leqslant 2. \qquad \square$$

Corollary 18.

$$\hat{X}\hat{Y} = (1 - D_N^2(\hat{X}, \hat{Y}))/2.$$

6.2. Computing the internal epsilon

In this section, we derive the formula for computing the value for ϵ_i, the first parameter of the *internal query* (Definition 10). First, we need a few definitions and associated facts.

We define $D_m(X, Y)$ to be the minimum normalized Euclidean distance between X and Y', for all Y' in Y^*, the set of all sequences similar to Y: $D_m(X, Y) = \min\{Y' \in Y^* : D_N(X, Y')\}$.

To compute a conservative value for ϵ_i, in order to ensure no false dismissals, we make use of the following equality;

Proposition 19.

$$D_m^2(X, Y) = \sigma_X^2(D_N^2(\hat{X}, \hat{Y}) - D_N^4(\hat{X}, \hat{Y})/4).$$

Proof. Let us denote $D_N^2(X, aY + b)$ by $f(a, b)$; $f(a, b) = (1/n)\sum(x_i - (ay_i + b))^2$. We obtain minimum of f by finding the values of a, b where $\mathrm{d}f/\mathrm{d}a = \mathrm{d}f/\mathrm{d}b = 0$. Then, we will complete the calculations by substituting these values back into $f(a, b)$. The rest of the proof provides the details for these calculations.

(1) Let us denote $(y_i - \mu_Y)$ by $\tilde{y}_i$; similarly, $(x_i - \mu_X)$ is $\tilde{x}_i$.
(2) It is easy to verify that: (a) $\sum \tilde{x}_i = \sum \tilde{y}_i = 0$; (b) $\sum \tilde{x}_i\tilde{y}_i = n\sigma_X\sigma_Y\hat{X}\hat{Y}$. From (b), it also follows that (c) $\sum \tilde{x}_i^2 = n\sigma_X^2$, $\sum \tilde{y}_i^2 = n\sigma_Y^2$.
(3) $\mathrm{d}f/\mathrm{d}b = (1/n)\sum 2(x_i - ay_i - b) = 2(\mu_X - a\mu_Y - b) = 0$. Therefore, the desired value for b is $(\mu_X - a\mu_Y)$.
(4) $\mathrm{d}f/\mathrm{d}a = (1/n)\sum 2y_i(x_i - ay_i - b) = (2/n)\sum y_i(x_i - ay_i - (\mu_X - a\mu_Y)) = (2/n)\sum y_i(\tilde{x}_i - a\tilde{y}_i) = (2/n)\sum(\mu_Y + \tilde{y}_i)(\tilde{x}_i - a\tilde{y}_i) = (2/n)(\mu_Y\sum \tilde{x}_i - a\mu_Y\sum \tilde{y}_i + \sum \tilde{y}_i(\tilde{x}_i - a\tilde{y}_i)) = (2/n)\sum \tilde{y}_i(\tilde{x}_i - a\tilde{y}_i) = 0$. Therefore, the desired value for a is $\left(\sum \tilde{x}_i\tilde{y}_i\right)/\left(\sum \tilde{y}_i^2\right)$.
(5) Now, we do the substitution into $f(a, b)$.

$$\begin{aligned} f(a,b) &= (1/n)\sum(x_i - ay_i - b)^2 = (1/n)\sum(\tilde{x}_i - a\tilde{y}_i)^2 = (1/n)\sum(\tilde{x}_i^2 + a^2\tilde{y}_i^2 - 2a\tilde{x}_i\tilde{y}_i) \\ &= (1/n)\left(\sum \tilde{x}_i^2 + \left(\sum \tilde{y}_i^2\right)\left(\sum \tilde{x}_i\tilde{y}_i / \sum \tilde{y}_i^2\right)^2 - 2\left(\sum \tilde{x}_i\tilde{y}_i\right)\left(\sum \tilde{x}_i\tilde{y}_i / \sum \tilde{y}_i^2\right)\right) \\ &= (1/n)\left(\sum \tilde{x}_i^2 - \left(\sum \tilde{x}_i\tilde{y}_i\right)^2 / \sum \tilde{y}_i^2\right) = (1/n)\left(n\sigma_X^2 - (n\sigma_X\sigma_Y\hat{X}\hat{Y})^2/(n\sigma_Y^2)\right) \\ &= \sigma_X^2(1 - (\hat{X}\hat{Y})^2). \end{aligned}$$

(6) Let us denote $D_N^2(\hat{X}, \hat{Y})$ by $\mathbb{D}$. We make use of Corollary 18 to complete the proof:

$$D_m^2(X, Y) = \sigma_X^2(1 - (\hat{X}\hat{Y})^2) = \sigma_X^2(1 - (1 - \mathbb{D}^2/2)^2) = \sigma_X^2(\mathbb{D}^2 - \mathbb{D}^4/4). \qquad \square$$

We are now ready to compute the value for ϵ_i:

$$\begin{aligned} D_N(Q, aX+b) \leqslant \epsilon \;&\Rightarrow D_m(Q,X) \leqslant \epsilon \Leftrightarrow D_m^2(Q,X) \leqslant \epsilon^2 \\ &\Leftrightarrow \sigma_Q^2(D_N^2(\hat{Q},\hat{X}) - D_N^4(\hat{Q},\hat{X})/4) \leqslant \epsilon^2 \\ &\Leftrightarrow D_N^4(\hat{Q},\hat{X}) - 4D_N^2(\hat{Q},\hat{X}) + 4\epsilon^2/\sigma_Q^2 \geqslant 0 \\ &\Leftrightarrow \left(D_N^2(\hat{Q},\hat{X}) \leqslant 2 - 2\sqrt{1-\epsilon^2/\sigma_Q^2}\right) \vee \left(D_N^2(\hat{Q},\hat{X}) \geqslant 2 + 2\sqrt{1-\epsilon^2/\sigma_Q^2}\right). \end{aligned}$$

By Proposition 17, we see that only the first inequality for $D_N^2(\hat{Q},\hat{X})$ is valid. Therefore,

$$D_N(\hat{Q},\hat{X}) \leqslant \sqrt{2 - 2\sqrt{1-\epsilon^2/\sigma_Q^2}} = \epsilon_i.$$

Notice that if a and b are unbounded, then

$$D_N(Q, aX+b) \leqslant \epsilon \Leftrightarrow D_m(Q,X) \leqslant \epsilon,$$

and this value for ϵ_i is tight.

Remark 3. The above value for ϵ_i only makes sense when $\epsilon^2 \geqslant \sigma_Q^2$. However, this does not reduce the expressibility, because $D_m^2(Q,X) \leqslant \sigma_Q^2$ for all sequences X. Therefore, any tolerance value greater than σ_Q^2 will produce no new matches.

6.3. Computing internal bounds

Now we compute bounds on the average and deviation; these bounds serve as parameters in the internal query (Definition 10). First, we prove the following proposition:

Proposition 20. *Given two sequences Q and X, let t be the following linear transformation:*

$$t(a,b) = (a\sigma_X - \sigma_Q\hat{X}\hat{Q}, a\mu_X + b - \mu_Q).$$

Then, for all (a,b), if $(a',b') = t(a,b)$ and $c = \sigma_Q^2(1-(\hat{X}\hat{Q})^2)$, we have:

$$D_N^2(Q, aX+b) = (a')^2 + (b')^2 + c.$$

Proof.

$$\begin{aligned} D_N^2(Q, aX+b) &= (Q-(aX+b))^2 = Q^2 + a^2X^2 + b^2 + 2ab\mu_X - 2b\mu_Q - 2aXQ \\ &= (\mu_Q^2 + \sigma_Q^2) + a^2(\mu_X^2 + \sigma_X^2) + b^2 + 2ab\mu_X - 2b\mu_Q - 2a(\mu_X\mu_Q + \sigma_X\sigma_Q\hat{X}\hat{Q}) \\ &= (a\mu_X + b - \mu_Q)^2 + (a\sigma_X - \sigma_Q\hat{X}\hat{Q})^2 + \sigma_Q^2(1-(\hat{X}\hat{Q})^2) = (b')^2 + (a')^2 + c. \end{aligned}$$

□

Next, we combine the fact that $D_N(Q, aX + b) \leqslant \epsilon$ with the equality derived in Proposition 20:

$$D_N(Q, aX + b) \leqslant \epsilon \Leftrightarrow D_N^2(Q, aX + b) \leqslant \epsilon^2 \Leftrightarrow (a')^2 + (b')^2 + c \leqslant \epsilon^2 \Leftrightarrow (a')^2 + (b')^2 \leqslant \epsilon^2 - c.$$

Note that σ_X (μ_X) appears only in a' (b'), and its range can be maximized by setting b' (a') to 0:

$$(a')^2 = (a\sigma_X - \sigma_Q \hat{X}\hat{Q})^2 \leqslant \epsilon^2 - c,$$

$$(b')^2 = (a\mu_X + b - \mu_Q)^2 \leqslant \epsilon^2 - c.$$

Solving the resulting inequalities for μ_X and σ_X, we obtain:

$$\left(\mu_Q - b - \sqrt{\epsilon^2 - c}\right) \Big/ a \leqslant \mu_X \leqslant \left(\mu_Q - b + \sqrt{\epsilon^2 - c}\right) \Big/ a,$$

$$\left(\sigma_Q \hat{X}\hat{Q} - \sqrt{\epsilon^2 - c}\right) \Big/ a \leqslant \sigma_X \leqslant \left(\sigma_Q \hat{X}\hat{Q} + \sqrt{\epsilon^2 - c}\right) \Big/ a.$$

Unfortunately, the above formulas involve values that are not constant at the time of computation; i.e., a, b, and $\hat{X}\hat{Q}$. We need to substitute constants for a, b, and $\hat{X}\hat{Q}$; we do this conservatively so as to ensure no false dismissals, while trying to allow for the widest range of μ_X and σ_X. Specifically, we find that taking $\hat{X}\hat{Q} = 1$ always minimizes the lower bounds and maximizes the upper bounds. Therefore, the square root expression always collapses to $\sqrt{\epsilon^2} = \epsilon$. For a and b, we choose appropriately from the external query bounds on these values in order to obtain the widest bounds. Therefore, we obtain the following bounds:

$$(\mu_Q - u_b - \epsilon)/u_a \leqslant \mu_X \leqslant (\mu_Q - l_b + \epsilon)/l_a,$$

$$(\sigma_Q - \epsilon)/u_a \leqslant \sigma_X \leqslant (\sigma_Q + \epsilon)/l_a.$$

This completes the specification of the conversion from the external to the internal query. The conversion results are summarized in Fig. 4.

6.4. Filtering potential matches

In this subsection, we derive the formulas for performing the *filtering step* of our method (Section 4.2). That is, given a *potential match* X, we need to determine there exist an a and b within the bounds specified by the external query such that $D_N(Q, aX + b) \leqslant \epsilon$. The proposition below is repeated from Section 4, where it was not proven.

Proposition 21. *Given a query Q, a potential match X, the (open) bounds $(l_a, u_a), (l_b, u_b)$ on a and b define an (open) rectangle R:*

$$(a, b) \in R \Leftrightarrow (l_a \leqslant a \leqslant u_a) \wedge (l_b \leqslant b \leqslant u_b).$$

The transformation t (defined in Proposition 20) maps R to an (open) parallelogram t(R). Let (a_t, b_t) *be the point in* $t(R)$ *closest to the origin, and let* (a_0, b_0) *be the inverse transformation* $t^{-1}(a_t, b_t)$*:*

$$a_0 = (a_t + \sigma_Q \hat{X}\hat{Q})/\sigma_X, \quad b_0 = b_t + \mu_Q - a_0\mu_X.$$

Then, $\min\{D_N(Q, aX + b) : (l_a \leqslant a \leqslant u_a) \wedge (l_b \leqslant b \leqslant u_b)\} = D_N(Q, a_0X + b_0) = \sqrt{a_t^2 + b_t^2 + c}$.

Proof.

(1) Let us consider some arbitrary $(a_1, b_1) \in R$, with $t(a_1, b_1) = (a_1', b_1')$. We know that $(a_t^2 + b_t^2) \leqslant (a_1'^2 + b_1'2)$.
(2) This means that $0 \leqslant (a_1'2 + b_1'2) - (a_t^2 + b_t^2) = (a_1'2 + b_1'2 + c) - (a_t^2 + b_t^2 + c) = D_N^2(Q, a_1X + b_1) - D_N^2(Q, a_0X + b_0)$ (by Proposition 20).
(3) We have just shown that for all (a_1, b_1) in R, $D_N(Q, a_1X + b_1) \geqslant D_N(Q, a_0X + b_0)$. Since t is linear, (a_0, b_0) is itself in R.
(4) Since we know that $D_N(Q, a_0X + b_0) = \sqrt{a_t^2 + b_t^2 + c}$, this completes the proof. □

To find the point (a_t, b_t) in $t(R)$ closest to the origin, we first check whether the origin is inside $t(R)$, in which case $(0, 0)$ is the point we are looking for. Otherwise, the closest point to the origin must be either one of the vertices of $t(R)$ or one of the points of intersection between an edge of $t(R)$ and a line perpendicular to this edge passing through the origin. So there are only eight points to check. We simply assign (a_t, b_t) to the one of these eight points which has the smallest Euclidean distance to the origin.

If $a_t^2 + b_t^2 > c$, then there are no satisfying numbers a and b, so we have a false alarm. Otherwise, we have a match, where the actual distance $D_N(Q, aX + b)$ is $\sqrt{a_R^2 + b_R^2 - \sigma_Q^2((\hat{X}\hat{Q})^2 - 1)}$.

7. Experimental results

We have implemented the index-based versions of bounded subsequence similarity querying and of approximate matching. We have also implemented a version with no index, where data are scanned for each query. We refer to the three systems as *SQ-index*, *AQ-index*, and *No-index*, respectively:

SQ-index: This is the *index-based bounded subsequence similarity querying*, described in Sections 5.1 and 5.3. The index is an 8-dimensional $R*$-tree on the set of fingerprints, grouped by rectangles.
AQ-index: The specialized index for *approximate matching* (Definition 1). It has the same number of dimensions as the SQ index, but can only perform approximate match searches. The reason for comparing the two indices is to assess the additional overhead of SQ-index due to using normalization parameters.
No-index: It is assumed that the data are on disk, but the access to it is sequential rather than index-based; this is also called *linear scan* (LS). The algorithm is very similar to the *in-core bounded*

subsequence similarity querying, described in Section 4, but each datum read generates a disk access.

Note an important difference between the fingerprint functions for the SQ-index and the AQ-index. For the SQ-index, we normalize the sequence first, which allows us to ignore the first coefficient of the *DFT*. The three *DFT* coefficients we use are DFT_1, DFT_2, DFT_3; each one is a complex number corresponding to two real values. Together with the average and the standard deviation (the normalization parameters) of a sequence, there are a total of 8 values. While the fingerprints for approximate matching (AQ) are the same size, it only uses the DFT coefficients (starting at DFT_0) and has no normalization parameters.

In this section, we discuss experiments that were performed over the three systems. We are interested in the costs of SQ-index vs. Linear Scan, as well as the costs for SQ-index vs. AQ-index. It is expected that, unless query selectivity is very low, the SQ index performs better than a linear scan. This is not to be taken for granted; [6] point out that much of the work on multidimensional indexing is less efficient than a linear scan would be. However, experiments confirm the superiority of SQ index over the linear scan.

Furthermore, it is expected that, by being specialized, the AQ-index will be more efficient for approximate matching than the more general SQ-index. Experiments confirm that SQ-index is competitive to AQ-index for approximate matching; surprisingly, in some cases it is better. It was outside the scope of our work to provide a definitive explanation for these surprising results.

Our database consists of 10,000 sequences of varying length representing 5 years of closing prices for US mutual funds; the total size of the dataset is approximately 35 MB. We chose a subsequence size of 256 for fingerprinting, thus we only use sequences that have at least 256 points. We chose 256 because it is approximately one year's worth of daily stock prices. Our experiments study the following questions about our system:

- How does the search performance scale as ϵ grows?
- How does the search performance scale as the size of the database grows?
- How does the search performance scale as the shifting and scaling bounds grow?

To answer these questions, we ran three tests—the *tolerance test*, the *DB size test*, and the *bounds test*—each varying one parameter of the system while keeping the other parameters fixed. For a fair comparison, all methods involved in any experiment executed the same queries and returned the same data. This means that, whenever the experiment involved AQ-index, the other two methods used bounds $(1, 1)$ for shifting and $(0, 0)$ for scaling.

The systems are implemented in C; the experiments were run on a 2 GHz PC running Linux, with 512 MB of memory.

7.1. Cost calculation

In all our experiments, we measured two cost parameters, *CPU* and *I/O*. The former refers to the query processing time; the latter refers to the number of disk page accesses during query processing. We discuss the latter in more detail.

There are three possible sources of disk access that serve as components in the I/O cost measurement for index-based bounded similarity querying:

(1) accessing the pages of the index to retrieve qualifying rectangles (assuming the index does not fit in memory);
(2) accessing the corresponding fingerprints pointed to by a rectangle to find potential matches;
(3) accessing the data sequences for each qualifying fingerprint.

The overall I/O costs are determined as follows:

(1) For the first component, we measure the number of *R*-tree index pages that we must search in order to find matches to the query.
(2) Note that the index does not store pointers to individual sub sequences but to the offsets of a series of subsequences from one data sequence. Thus, for the second component, we have the number of rectangles *R* retrieved by the index. as an access number.
(3) Each rectangle refers to a place in a data sequence. Then, once we checked the fingerprints for a possible match, we have another access to retrieve the real sequences. Again, this is in the order of the number of rectangles, *R*.

For the linear scan, the I/O cost corresponds to the number of page accesses to retrieve the database from disk. If the database has, e.g., 9 M data points, and if we assume 4 bytes/data point, then we have about 36 M bytes of data to fetch. Assuming a block size of 1024 bytes, this makes about 35,400 page accesses.

Note, however, that there are two types of page accesses involved in the I/O cost, *random* and *sequential*, depending on whether data are to be retrieved by an arbitrary address or if we simply retrieve the next subsequent block on disk. (For the indexed solutions, only random disk access is involved). Random disk access involves additional data transfer cost, for the *seek* operation and the *rotational delay* [11].

Of the 35,400 page accesses in the example above, all but the first will be sequential. To be able to compute the overall cost, we assume that sequential disk access is 10 times faster than random, a factor also used in [28]. Hence, the actual number in the example above would be $1 + (35,399/10)$, or about 3,540.

To compute each data point for a given experiment, we randomly chose 10% of the sequences in the database and queried them against the rest of the database. The recorded I/O and CPU costs are the average over these queries. (This explains why some measurements, such as the number of matches, do not have an integer value when it seems they should.)

The time to build the index structure is not included in any of the costs; it is assumed to happen only once, before the querying starts, and it does not cost anything during querying.

7.2. Experiment I: varying distance tolerance

With this test, we would like to see how the three methods scale with increasing distance tolerance ϵ, and therefore decreasing selectivity. We chose a range of ϵ that starts with about one match and goes up to about 100 matches. It is expected that the indexed methods (AQ-index and SQ-index)

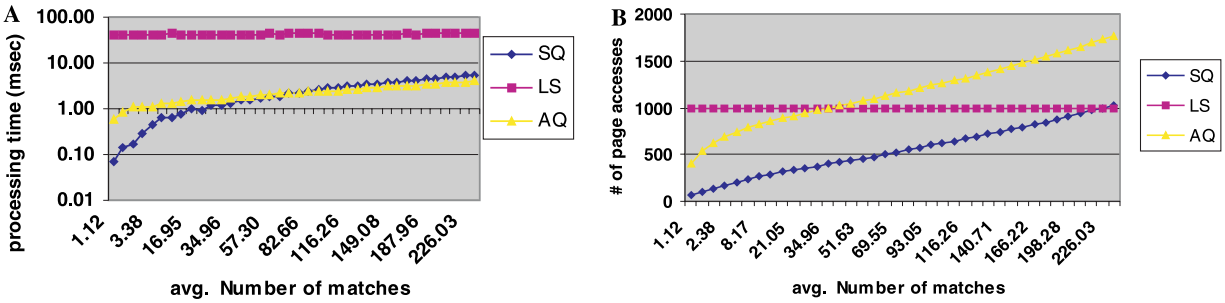

Fig. 5. (A) CPU and (B) I/O cost, plotted against the number of matches as the distance tolerance is increased.

will perform better than the no-index method (LS) for reasonable selectivities, including the ones in this test.

The results are plotted in Fig. 5. Fig. 5A, plotted on a logarithmic scale, shows that in terms of CPU time, both indexed methods easily outperform linear scan. As the search tolerance grows, both the CPU time and the number of page accesses grow for the indexed methods because there are more matches to the query. By contrast, the performance parameters stay constant for the linear scan method, because it has to load all sequences no matter what the tolerance.

We can observe that our method is at least as efficient as approximate matching for small tolerances. However, starting at about 60 matches (out of 10,000 sequences), AQ-index slightly outperforms SQ-index.

Fig. 5B is more surprising. In terms of I/O costs, linear scan starts outperforming both index methods relatively soon. More interestingly, SQ-index consistently outperforms AQ index, by a large margin. While we have no definitive explanation for this phenomenon, we assume it is due to the difference in the fingerprint method. In the case of the AQ-index, only the DFT coefficients of the sequence are used in the fingerprint. In the case of the SQ-index, the sequence is normalized prior to computing its DFT, and the fingerprint augments DFT coefficients with the normalization parameters (i.e., the average and the standard deviation); the total fingerprint size is the same. We conclude that pure DFT-based indexing performs worse than normalization-based indexing.

7.3. Experiment II: varying database size for selective queries

This time we vary the size of the database, doubling it until it reaches 10,000. The distance tolerance ϵ stays fixed at 0.5; the number of matches ranges from an average of 1.3 for the smallest database size to 4.9 for the largest. The tolerance was chosen so the queries were very selective; note however that the number of matches returned by the index search is greater (ranging from 1.7 to 14.7), because of the false matches that are subsequently filtered out. The test should show that in this setting, indexing becomes more and more attractive with growing database size.

The results are plotted in Fig. 6. As expected, as the database size grows, the benefits of using an index for selective queries become more and more pronounced. As before, SQ-index slightly outperforms AQ-index in terms of CPU cost, and greatly outperforms it in terms of I/O cost.

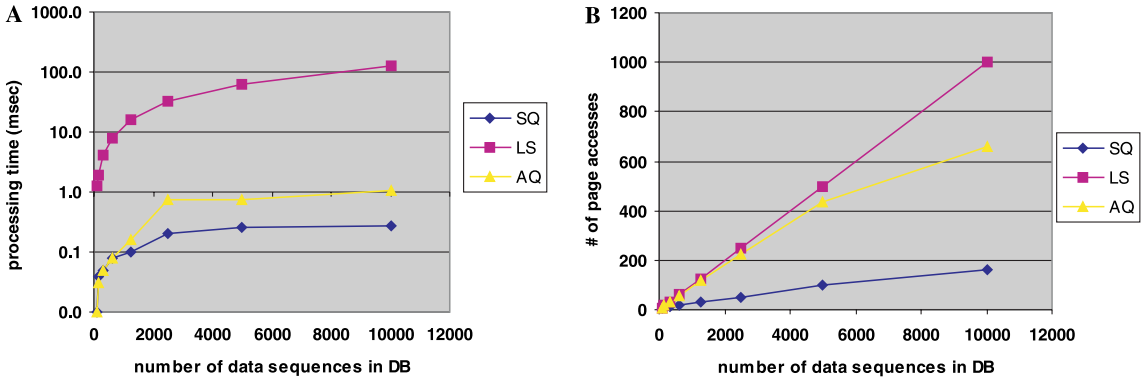

Fig. 6. (A) CPU and (B) I/O cost, plotted against the size of the database; the distance tolerance is kept constant.

7.4. Experiment III: varying shift/scale parameter

This test involves only two of the methods, *SQ-index* and *No-index* (LS); the *AQ-index* method is not applicable. We run bounded similarity queries with varying shift and scale bounds. The tolerance is kept very low; by widening the bounds on the normalization parameters, we get more and more matches to the query. We try the following shift/scale bounds, staring from no shifting and scaling to constrained and finally unbounded shifting and scaling (i.e., a similarity query). We expect the processing time to grow with the value space for the normalization parameters.

Query type	Shifting bounds	Scaling bounds
1. No shifting and scaling	(0, 0)	(1, 1)
2. Pure scaling	(0, 0)	$(-\infty, +\infty)$
3. Pure shifting	$(-\infty, +\infty)$	(1, 1)
4. Constrained shift and scale	(0.75, 1)	(−2, 2)
5. Constrained shift and scale	(0.5, 2)	(−4, 4)
6. Constrained shift and scale	(0.4, 2.5)	(−8, 8)
7. Constrained shift and scale	(0.25, 4)	(−16, 16)
8. Similarity query	$(-\infty, +\infty)$	$(0, +\infty)$

The results are plotted in Fig. 7. Each curve represents a different value for the distance tolerance ϵ. Only one curve is shown for the linear scan, since the costs of this method are not affected by either the distance tolerance or the shift/scale bounds. Again, there is a larger performance difference for the CPU cost than for the I/O cost, as in previous experiments.

As expected, the larger the tolerance and the bounds, the larger the cost of the index-based bounded similarity query, since there are more matches to be retrieved. However, there is a surprising exception: when $\epsilon = 0$, the CPU cost goes down for the largest bounds (but not the I/O cost).

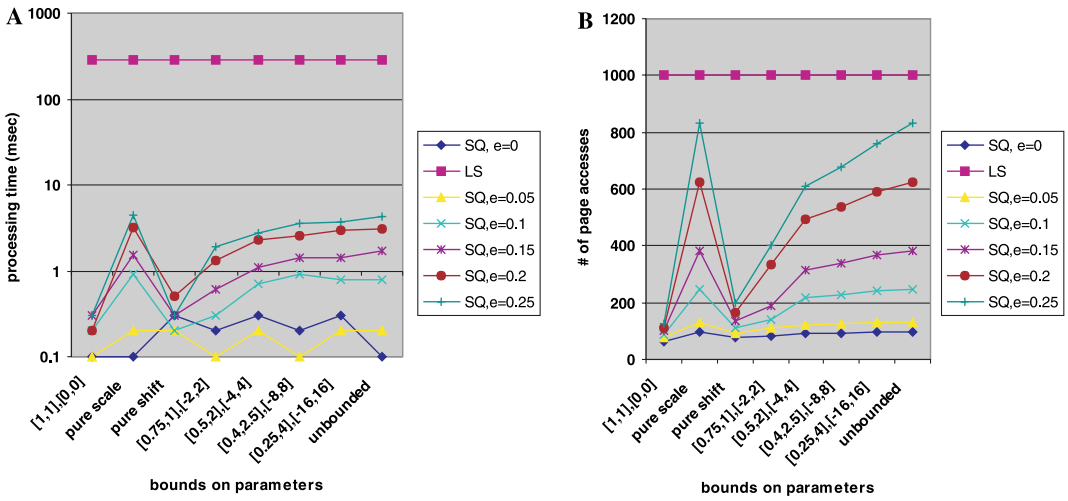

Fig. 7. (A) CPU and (B) I/O cost, plotted against different shift/scale bounds; the distance tolerance is kept constant for each curve.

Also, note the consistent difference in performance between the pure scaling and the pure shifting queries; the former is always larger. In fact, it is as large as for the unbounded case. This is because the two parameters are not symmetric; this can be seen from the formulas discussed in Section 6.

Fig. 8 shows the number of potential matches (PM) and of actual matches returned for each of the eight queries listed above. It can be easily seen that, while the actual number is about the same for pure scale as for pure shift, the number of potential matches is much larger for pure scale. This large number of false alarms explains the difference in their cost.

ε		[1,1], [0,0]	pure scale	pure shift	constrained [.75,1],[-2,2]	constrained [.5,2],[-4,4]	constrained [.4,2.5],[-8,8]	constrained [.25,4],[-16,16]	Un-bounded
0.00	PM	1.39	1.39	1.39	1.39	1.39	1.39	1.39	1.39
	MATCHES	1.39	1.39	1.39	1.39	1.39	1.39	1.39	1.39
0.05	PM	1.69	4.56	2.40	3.62	3.93	4.06	4.39	4.56
	MATCHES	1.49	1.52	1.52	1.52	1.52	1.52	1.52	1.59
0.10	PM	1.94	46.09	3.93	9.04	38.53	41.01	44.69	46.09
	MATCHES	1.68	2.33	2.27	2.45	2.46	2.46	2.52	2.53
0.15	PM	2.08	96.63	7.79	21.72	71.38	79.74	91.44	96.63
	MATCHES	1.94	2.65	2.50	2.98	3.03	3.04	3.11	3.12
0.20	PM	2.53	283.54	21.91	104.23	187.03	216.70	251.61	283.54
	MATCHES	2.18	3.87	3.36	4.73	31.27	32.91	35.09	35.57
0.25	PM	2.14	36.38	114.74	197.83	314.90	226.62	264.51	314.90
	MATCHES	1.79	5.38	7.00	13.18	15.12	15.42	21.97	45.19

Fig. 8. The number of potential matches (PM) and of actual matches returned.

It can also be seen that, as the number of actual matches goes up for any of the queries, the number of false alarms goes up even faster. While false alarms do not affect the performance of the linear scan, they directly affect the index-based methods, who need to retrieve more potential matches. Hence, index-based bounded similarity querying is most effective when query selectivity is low.

8. Experiments for varying length queries

In this section, we discuss our experiments for variable length queries, with the goal of comparing the two approaches discussed in Section 5.4—multi-piece search and stitching. Here are our predictions:

(1) When $\epsilon > 0$ and there is no exact match and many false alarms, stitching is a better option.
(2) When $\epsilon > 0$ and there is an exact match, stitching is worse.

For these experiments, we chose the fingerprint subsequence size of 128, so we can test queries with a large number of fragments, up to 9 in this case. We constructed an example for each of the predictions above and tested it, reporting average I/O cost over 10 queries.

8.1. Case 1

Consider the case where our query sequence has no exact match in the database. The benefit of stage processing is an early stop of the query processing, as the plots will show

First, we created query sequences that have no approximate match in the database. Then, we ran a variable-length version of the approximate match for these queries. We varied the length of the queries so that p, the number of fragments, varied between 1 and 9, and measured the I/O cost. The results are shown in Fig. 9. The stitching method stops searching after three subqueries, finding that there are no new rectangles that can be stitched to a rectangle from the previous stage. By contrast, the fixed length strategy runs each of its subqueries, before finding out that there is no match. The total number of disk accesses is 950 for stitching, and 5873.5 for multi-piece search.

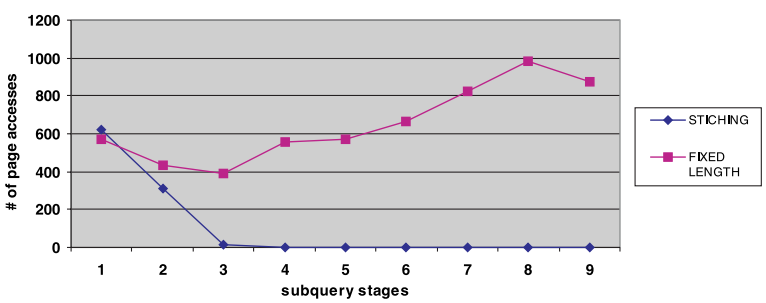

Fig. 9. The two approaches for varying length queries: case 1.

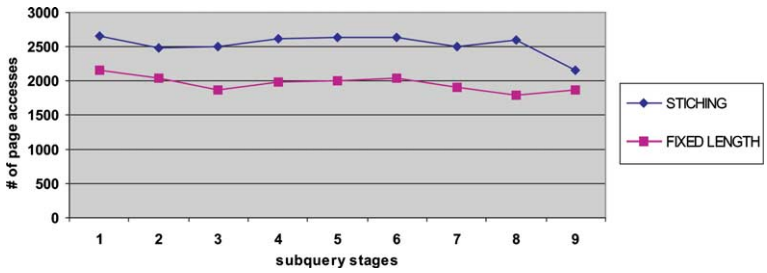

Fig. 10. The two approaches for varying length queries: case 2.

8.2. Case 2

Now we consider the case where we do have an exact match. As expected, our method performs less efficiently, see Fig. 10. Since we have at least one exact match at each state, the tolerance of stitching, which starts out higher than for multi-piece search, never decreases. As a result, the I/O cost is higher for stitching, for each of the nine subqueries.

8.3. Comparison with linear scan

The linear scan method needs no indexing and allows us to peform queries of arbitrary length on the fly. It is an alternative to both of the methods discussed in Section 5.4. For this experiment, we decided to see how the linear scan compares to stitching. We used fingerprint subsequence size 256, and allowed queries to be of length up to 1280; p, the number of fragments, ranged between 1 and 5.

A number of different queries of various lengths were run. For each value of p and ϵ, we reported the average I/O cost of all the queries, with both methods. The results are shown in Fig. 11, with a separate curve for stitching with different values of ϵ. There is only one curve for the linear scan, since its performance is not affected by the value of ϵ.

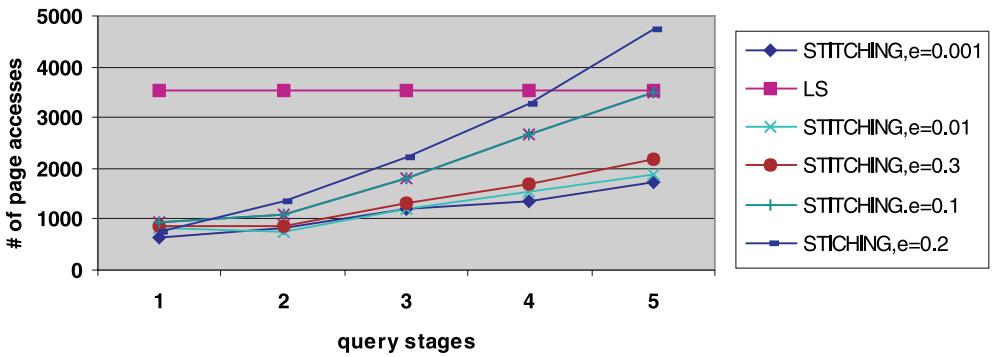

Fig. 11. I/O costs for stitching vs. linear scan.

Clearly, stitching is preferable to the linear scan for small values of p and ϵ. Once p and/or ϵ are sufficiently large, it is no longer worthwhile to perform stitching; a linear scan should be used instead.

9. Related work

We have implemented a robust and efficient framework for time-series similarity queries that allows the user to pose a wide variety of queries. It is based on a formal notion of *sequence similarity* as an equivalence relation, with *normal forms* of sequences serving as unique representatives of the equivalence classes. In this section, we compare our work with related approaches.

9.1. Background

The pioneering work of [2] enabled an efficient search mechanism for the *approximate match* problem by using the discrete Fourier transformation (DFT) to map time series to the frequency domain. Keeping only the first few DFT coefficients, the sequences became efficiently indexable using any multidimensional index structure such as the R^*-tree. The work covered what is known as the "whole match" problem. Faloutsos et al. [13] proposed a solution for the "subsequence matching" problem, where query and data sequences are no longer assumed to have the same length.

Bounded similarity querying extends approximate matching by allowing for shifting and scaling of time series. Our work on bounded similarity querying is an extended revision of [16]; the most notable additions have been the experimental component and the proofs of Section 6. The experimental component appeared in preliminary form in [33], and the proofs first appeared in [17].

Our definition of similarity is based on the notion of *transformation groups* from geometry [34]. It measures the Euclidean distance between two time series after one has been optimally shifted and scaled within a range requested by the user. In this sense, our work is a generalization of the work on approximate matching. This generalization is in the direction of [19], which discusses translation and distortion transformations but does not provide the guarantees of [2] and of our indexing scheme.

When our method was originally proposed in draft form in [16], it was the first time that fast (index-based) methods for approximate subsequence matching (Definition 1) were extended to similarity queries, as well as to bounded similarity queries. It was also the first time that normalization was used for fingerprinting, and that normalization parameters appeared in the index. Since then, there has been much work that considers shifting and scaling. Rafiei [39] proposed *moving averages* as a new similarity transformation; Vlachos et al. [45] proposed a non-metric similarity function based on subsequences.

The approach of [10] is a symmetric version of our unbounded case; the distance between two sequences is defined to be the smallest distance after scaling and shifting the query sequence to be as close as possible to the other. A similar approach can be found in [22], where it is extended to multi-attribute sequences and a new *ConeSlice* index is presented for it. It is interesting whether bounded similarity querying can be extended to subsume the approaches of [10,22], and how the performance will compare to ConeSlice.

For feature extraction, we use the discrete Fourier transform (DFT), keeping the first few Fourier coefficients for each sequence. A variety of other dimensionality reduction techniques has

been explored, including the *discrete wavelet transform* [8] and *singular value decomposition* [31]. In [29] and [46], the sequences are reduced to a set of *equi-length segments*; this is known as *piecewise aggregate approximation* (PAA). In [27] the technique is further improved to different length segments.

Our work on variable-length queries (Section 5.4) first appeared in draft form in [33]. Variable-length queries have since been considered in other contexts, such as PAA [26] and wavelet-based methods [21]. Notably, the *MR algorithm* for variable-length queries in [21] bears a strong similarity to our approach, including a similar formula for the refinement of ϵ_i. However, unlike ours, theirs is a *multiple-resolution* method, where each subquery is performed at a different resolution. It is an open question whether these techniques for feature extraction and for variable-length queries can be applied to the problem of bounded similarity querying.

An even more general framework for similarity queries is described in [20]. Our work happens to be (an efficiently solvable) special case. The [20] framework for similarity-based queries has three components: a pattern language P, an approximation language (they refer to it as transformation rule language), and a query language. In our case, P is the set of allowable transformations on the query sequence Q. An expression in P specifies a set of data objects; in our case, it is the set of all sequences exactly similar to Q. Approximations have a cost, and the distance between objects is defined as the minimal cost of reaching one object from the other via approximations. In our case, the approximations are the distortions in the time-series data (i.e., the "jiggling" of individual points); the cost is the distance between the original sequence and the distorted one. Note that membership testing in the [20] framework is at best exponential; thus this framework is too general for our purposes.

9.2. Transformations in the time domain

An obvious direction for extending our method is by enlarging the set of similarity transformations to include shifting and scaling along the time axis, for queries such as "find sequences that represent a similar trend, but slower."

The work of Rafiei and Mendelzon [40] provides a framework for this extension to similarity querying, allowing the user to specify shifts and scales in each dimension as well as moving average before comparing two sequences. Their algorithm requires the user to predefine one or more transformations that might make two sequences similar. However, this work does not allow bounds on the shift and scale parameters as we do, where the optimal transformation within these bounds is sought. Moreover, this approach is not scalable, since the index structure has to be rebuilt for each query, rather than remain unchanged throughout querying, as for our method.

Time Warping is a very flexible, but also high cost distance function that is not based on Euclidean distance, introduced in [5]. It may be useful in applications where sequences are of different lengths and sample rates, such as in biological time series. The *time warping measure* creates an alignment between the points of two sequences by allowing local accelerations and decelerations of the signal.

Dynamic Time Warping (DTW) is a dynamic programming algorithm to compute the *time warping distance*. It allows sequences to be stretched or compressed along the time axis for an optimal alignment. However, dynamic time warping is challenging when it comes to indexing, which is

crucial for large time series datasets. This is because DTW does not satisfy the triangle inequality [47], and thus cannot prevent false dismissals when using spatial access structure such as the *R*-tree. The authors of [47] propose to use a lower bounding function helping to prune false alarms. However, they trade in possible false dismissals.

Further approaches for efficient similarity querying under Time Warping Distance include [37,30,36], and finally, the work of [24,25], showing that Time Warping can be efficiently indexed under no false dismissals.

9.3. Other notions of similarity

Alternative notions of similarity have been proposed, that do not use distance as a measure of similarity.

For example, in [1], sequences are defined to be *similar* if they have enough non-overlapping subsequences (separted by *gaps*) that are *similar*, which means that one lies within an envelope of width ϵ around the other, after being appropriately scaled and shifted.

This method relies on bounding boxes to determine the amount of shifting and scaling needed. Unlike our approach, bounding boxes are very sensitive to outliers; gaps are necessary because they may contain outliers.

In fact, the unbounded search of our query system found the same matches as reported in [1]. They report two subsequence matches from the mutual fund database: for Ivy and Harbor mutual funds, and for Eck and Fidelity mutual funds. For simplicity, let I, H, V, and F denote the matching subsequences in these mutual funds. Our system found that $D_N(I, 1.02H + 2.68) = 0.187$ (Euclidean distance of 3.67) and $D_N(V, 0.84F + .77) = 0.291$ (Euclidean distance of 5.72).

Yet other notions of similarity have been proposed, that are not related to shifting and scaling. However, these methods do not naturally lend themselves to indexing structures, rendering them useless for very large databases.

In the *Landmark model* [38] distance is measured between a set of *landmarks*, or the peak points of the two sequences. This similarity model is invariant to various transformations, including shifting, scaling, non-uniform amplitude scaling and time warping. However, the time distance between those landmarks is not preserved, leading to many false alarms. In [42], sequences are approximated by piecewise linear functions; in [18], similarity is based on a probabilistic model rather than a distance metric.

9.4. Similarity: unbounded vs. normalized

With bounded similarity querying, users will sometimes not care to constrain the shift and scale parameters, instead looking for the overall optimal value of these parameters; this is the *unbounded* case (Section 2.4). In this case, there is an alternate approach based on *normalization*; one normalizes all data and queries and then performs an approximate match over the normalized sequences. An example of this normalized approach is Keogh's "cylinder-bell-funnel" test suite [28], where all data are normalized a priori.

Note that it is not hard to extend our method to handle the normalization approach, without changing the fingerprint method or the index structure. As discussed in Section 2.4, we can already

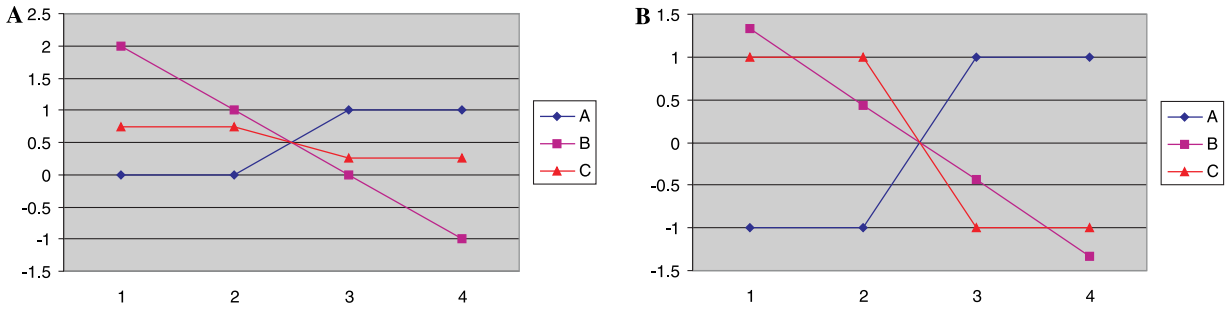

Fig. 12. The sequences after: (A) optimal transformation and (B) normalization.

perform an approximate match query, by setting the shift and scale parameters to 0 and 1, respectively. While usually our query is over unnormalized sequences, we can easily refocus on normalized sequences instead, by ignoring the normalization parameters μ and σ in the fingerprints during the search, as if they were always 0 and 1.

Furthermore, normalization does not always lead to optimal results in terms of qualitative similarity. To demonstrate the possible advantage of shifting and scaling over normalization, consider a simple example where we have three sequences, $A = \{0,0,1,1\}$, $B = \{6,4,2,0\}$, $C = \{1,1,0,0\}$; we want to know which of B and C is more similar to A.

Actually, the minimal distance to A is achieved when the scale factor is 0 for both B and C, flattening them both, so unbounded similarity querying returns both of these sequences. However, if the scale factor a is constrained to have a small positive lower bound (such as 0.5), C will be the sequence of choice for our method:

$$D(A,B) = 10, \text{ with } a = 0.5 \text{ and } b = -1 \text{ for } B;$$

$$D(A,C) = 2.25, \text{ with } a = 0.5 \text{ and } b = 0.25 \text{ for } C.$$

These scaled and shifted versions of B and C are shown in Fig. 12A. On the other hand, if the sequences are normalized (see Fig. 12B), these distances become $D(\hat{A},\hat{B}) = 15.15$ and $D(\hat{A},\hat{C}) = 16$. Therefore, the normalization approach prefers B.

We argue that A and C are in fact more similar than A and B. Although there is a short opposite trend in the middle, there are large portions where the trends in A and C are the same. In contrast, there is no obvious similarity in shape between A and B.

10. Summary

We have described a framework for approximate similarity querying of time-series data. We have formally defined the notion of similarity with respect to a group of transformations. We presented an in-core similarity querying method that avoids false dismissals. We then illustrated an indexing technique for approximate similarity querying that preserves the desirable properties of previous algorithms for approximate querying of time-series data (without similarity).

Experiments show that our system, which implements the index-based version of similarity querying, can duplicate the results of other approaches, and handle more expressive kinds of queries. At the same time, the performance of our system is competitive with earlier approaches; in fact, its I/O performance is better. As discussed earlier, we assume that this difference is due to the use of normalization in our fingerprint method; a definitive explanation of is a matter of future work.

References

[1] R. Agrawal, K. Lin, H. Sawhney, K. Shim, Fast similarity search in the presence of noise, scaling, and translation in time-series databases, in: Proceedings of the 21st VLDB Conference, 1995, pp. 490–501.
[2] R. Agrawal, C. Faloutsos, A. Swami, Efficient similarity search in sequence databases, FODO Conference, Evanston, Illinois, October 1993.
[3] A. Aho, Algorithms for Finding Patterns in Strings, in: Van Leeuwen J. (Ed.), Handbook of TCS, vol. A, Elsevier, Amsterdam, 1990, Chap. 5.
[4] N. Beckmann, H. P. Kriegel, R. Schneider, B. Seeger, The R*-Tree, an efficient and robust access method for points and rectangles, in: Proceedings of the ACM SIGMOD Conference on Management of Data, 1990, pp.322–331.
[5] D. Berndt, J. Clifford, Using dynamic time warping to find patterns in time series, in: AAAI-94 Workshop on Knowledge Discovery in Databases, 1994, pp. 229–248.
[6] K.S. Beyer, J. Goldstein, R. Ramakrishnan, U. Shaft, When is "nearest neighbor" meaningful? in: Proceedings of the 7th International Conference on Database Theory, 1999, pp. 217–235.
[7] S. Babu, J. Widom, Continuous queries over data streams, SIGMOD Record, September 2001.
[8] K. Chan, A.W. Fu, Efficient, time series matching by wavelets, in: Proceedings of the 15th IEEE International Conference on Data Engineering, Sydney, Australia, March 1999, pp. 126–133.
[9] B. Chiu, E. Keogh, S. Lonardi, Probabilistic discovery of time series motifs, in: Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August 2003, pp. 493–498.
[10] K. Chu, M. Wong, Fast time-series searching with scaling and shifting, in: in Proceedings of the 18th ACM Symposium on Principles of Database Systems, Philadelphia, PA, May 1999, pp 237–248.
[11] R. Elmasri, S. Navathe, Fundamentals of Database Systems, Addison-Wesley, New York, 2000.
[12] C. Faloutsos, Searching Multimedia Databases by Content, Kluwer Academic Publishers, Dordrecht, 1996.
[13] C. Faloutsos, M. Ranganathan, Y. Manolopoulos, Fast subsequence matching in time-series databases, in: Proceedings of the ACM SIGMOD Conference on Management of Data, May 1994, pp. 419–429.
[14] A. Guttmann, R-trees: a dynamic index structure for spatial searching, in: Proceedings of the ACM SIGMOD Conference on Management of Data, 1984, pp. 47–57.
[15] V. Gaede, O. Gunther, Multidimensional access methods, ACM Comput. Surv. 30 (2) (1998).
[16] D. Goldin, P. Kanellakis, On similarity queries for time-series data: constraint specification and implementation, in: Proceedings of CP'95, September 1995.
[17] D. Goldin, Constraint query algebras, Computer Science Doctorate Thesis, Brown University, Jan. 1997.
[18] X. Ge, P. Smyth, Deformable Markov model templates for time-series pattern matching, in: Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Boston, MA, August 2000, , pp 81–90.
[19] H.V. Jagadish. A retrieval technique for similar shapes, in: Proceedings of the ACM SIGMOD Conference, May 1991, pp. 208–217.
[20] H.V. Jagadish, A.O. Mendelzon, T. Milo, Similarity-based queries, in: Proceedings of the 14th ACM PODS, 1995.
[21] T. Kahveci, A. Singh, Variable length queries for time series data, in: Proceedings of the 17th International Conference on Data Engineering, Heidelberg, Germany, 2001.
[22] T. Kahveci, A. Singh, A. Gürel, Similarity searching for multi-attribute sequences, in: Proceedings of the 14th International Conference on Scientific and Statistical Database Management, 2002.
[23] R.M. Karp, M.O. Rabin, Efficient randomized pattern-matching algorithms, IBM J. Res. Dev. 31 (2) (1987).
[24] E. Keogh, Exact indexing of dynamic time warping, in: Proceedings of the 28th International Conference on Very Large Data Bases, Hong Kong, 2002, pp. 406–417.

[25] E. Keogh. Efficiently finding arbitrarily scaled patterns in massive time series databases, in: Proceedings of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases, September 2003, pp. 253–265.
[26] E. Keogh, K. Chakrabarti, S. Mehrotra, M. Pazzani, Locally adaptive dimensionality reduction for indexing large time series databases, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2001.
[27] E. Keogh, K. Chakrabarti, M. Pazzani, S. Mehrota, Dimensionality reduction for fast similarity search in large time series databases, J. Knowledge Inf. Syst. 3 (3) (2001).
[28] E. Keogh, S. Kasetty, On the need for time series data mining benchmarks: a survey and empirical demonstration, in: Proceedings of 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2002, pp. 102–111.
[29] E. Keogh, M. Pazzani, A simple dimensionality reduction technique for fast similarity search in large time series databases, in: Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2000, pp. 122–133.
[30] S. Kim, S. Park, W. Chu, An index-based approach for similarity search supporting time warping in large sequence databases, in: Proceedings of the 17th International Conference on Data Engineering, 2001, pp. 607–614.
[31] F. Korn, H. Jagadish, C. Faloutsos, Efficiently supporting ad hoc queries in large datasets of time sequences, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, Tucson, AZ, May 1997, pp. 289–300.
[32] W. Mendenhall, R. Scheaffer, D. Wackerly, Mathematical Statistics with Applications, Duxbury Press, Boston, MA, 1986.
[33] T.D. Millstein, Design and Implementation of Similarity Querying for Time-Series Databases, Undergraduate Honors Thesis, Computer Science Department, Brown University, May 1996.
[34] Modenov, Pakhomenko, Geometric Transformations, Academic Press, New York, 1965.
[35] A.V. Oppenheim, R.W. Schafer, Digital Signal Processing, Prentice Hall, New Jersey, 1975.
[36] S. Park, D. Lee, W. Chu, Fast retrieval of similar subsequences in long sequence databases, in: 3rd IEEE Knowledge and Data Engineering Exchange Workshop, 1999.
[37] S. Park, W. Chu, J. Yoon, C. Hsu, Efficient searches for similar subsequences of different lengths in sequence databases, in: Proceedings of the 16th IEEE International Conference on Data Engineering, 2000, pp. 23–32.
[38] Chang-Shing Perng, Haixun Wang, Sylvia R. Zhang, D. Stott Parker, Landmarks: a new model for similarity-based pattern querying in time series databases, in: Proceedings of the 16th International Conference on Data Engineering (ICDE'2000), San Diego, USA, 2000.
[39] D. Rafiei, On similarity-based queries for time series data, in: Proceedings of the 15th IEEE International Conference on Data Engineering, Sydney, Australia, March 1999.
[40] D. Rafiei, A. Mendelzon, Similarity-based queries for time series data, in: Proceedings of the ACM SIGMOD Conference, Tucson, Arizona, May 1997.
[41] H. Samet, The Design and Analysis of Spatial Data Structures, Addison-Wesley, Reading MA, 1990.
[42] H. Shatkay, S. Zdonik, Approximate queries and representations for large data sequences, in: Proceedings of the 12th IEEE International Conference on Data Engineering, New Orleans, LA, pp. 536–545, February 1996.
[43] P. Sheshadri, M. Livny, R. Ramakrishnan, Sequence query processing, Proc. ACM SIGMOD Conf. (1994) 430–441.
[44] J. Ullman, Principles of Database Systems, Computer Science Press, Rockville MD, 1982.
[45] M. Vlachos, G. Kollios, D. Gunopulos, Discovering similar multidimensional trajectories, in: Proceedings of the 18th International Conference on Data Engineering (ICDE'2002), San Jose, USA, 2002.
[46] B.-K. Yi, C. Faloutsos. Fast time sequence indexing for arbitrary *Lp* Norms, in: Proceedings of the 26th International Conference on Very Large Databases, Cairo, Egypt, 2000.
[47] B. Yi, H. Jagadish, C. Faloutsos, Efficient retrieval of similar time sequences under time warping, in: Proceedings of the 14th International Conference on Data Engineering, Orlando, FL, 1998, pp. 201–208.