On Similarity Queries for Time-Series Data: Constraint Specification and Implementation

Dina Q Goldin * and Paris C Kanellakis *

Department of Computer Science, Brown University
PO Box 1910, Providence RI 02912, USA.
Tel: 1-401-863-7647. Fax: 1-401-863-7657.
Email: dgk@cs.brown.edu and pck@cs.brown.edu.

Abstract. Constraints are a natural mechanism for the specification of similarity queries on time-series data. However, to realize the expressive power of constraint programming in this context, one must provide the matching implementation technology for efficient indexing of very large data sets. In this paper, we formalize the intuitive notions of exact and approximate similarity between time-series patterns and data. Our definition of similarity extends the distance metric used in [2, 7] with invariance under a group of transformations. Our main observation is that the resulting, more expressive, set of constraint queries can be supported by a new indexing technique, which preserves all the desirable properties of the indexing scheme proposed in [2, 7].

1 Introduction

1.1 Approximate Matching of Time-Series Data

Time-series are the principal format of data in many applications, from financial to scientific. Time-series data are sequences of real numbers representing measurements at uniformly-spaced temporal instances. The next generation of database technology, with its emphasis on multimedia, is expected to provide clean interfaces (i.e., declarative specification languages) to facilitate data mining of time-series. However, any proposal of such linguistic facilities must be supported by indexing (i.e., be implementable with reasonable I/O efficiency) for very large data sets. Examples of recent database research towards this goal include [2, 7, 18].

A most basic problem in this area is *First-Occurrence Subsequence Matching*, defined as follows: *given a query sequence Q of length n and a much longer data sequence $\overline{S}$ of length N, find the first occurrence of a contiguous subsequence within $\overline{S}$ that matches Q exactly.*

A wide range of algorithms has been developed for *internal* (i.e., in-core) versions of this question [1] for strings over an alphabet or for values over bounded discrete domains. There are particularly elegant linear-time $O(n+N)$ algorithms

* Research supported by ONR Contract N00014-94-1-1153.

(by Knuth-Morris-Pratt and Boyer-Moore) and practical searching utilities for more general patterns instead of query strings Q (e.g., regular patterns in grep). The part of this technology that is most related to our paper is the Rabin-Karp randomized linear-time algorithm [14], 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.

A variant of the above problem involves finding all occurrences; this is called the *All-Occurrences Subsequence Matching* problem.

The above two problems have variants when the data consists of many sequences of the same length as the query. The *First(All)-Occurrence(s) Whole-Sequence Matching* problem is: *given a query sequence Q of length n and a set of N/n data sequences, all of the same length n, find the first (all) of the data sequences that match Q exactly.*

Since the size of the data in commercial applications of time-series data mining is usually too large to be stored internally, the research in various time-series matching problems has concentrated on the case when storage is *external* (i.e., secondary as opposed to in-core). Here we are interested in:

> External solutions to the All-Occurrences Matching problems (either Subsequence or Whole-Sequence), with two additional characteristics:

- the match can be *approximate*;
- the match is up to *similarity*.

We define "approximate" in this subsection and "similar" in the next.

Time-series data in continuous (e.g., real-valued) domains is inherently inexact, due to the unavoidable imprecision of measuring devices and clocking strategies. This forces us to work with the approximate version of the various matching problems.

Given a tolerance $\epsilon \geq 0$ and a distance metric D between sequences, sequences S_1 and S_2 match approximately within tolerance ϵ when $D(S_1, S_2) \leq \epsilon$.

The *All-Occurrences Approximate Matching* problems (either *Subsequence* or *Whole-Sequence*) are defined as before, but with "match approximately within tolerance ϵ" instead of "match exactly".

For external solutions to the All-Occurrence Whole-Sequence Approximate version, we refer to [2, 18]; for the All-Occurrence Subsequence Approximate version, we refer to [7].

A further characteristic of time-series data that is used to advantage, is that they have a *skewed energy spectrum*, to use the terminology borrowed from Discrete Signal Processing [16]. As a result, most of the technology of information retrieval in this area is influenced by signal processing methods.

1.2 Approximately Similar Time-Series Data

The database applications of interest involve queries expressing notions of "user-perceived similarity". Here are some examples of applications for approximate time-series matching that illustrate this notion of similarity:

– find months in the last five 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.

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. For 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*. A good time-series data-mining mechanism should be able to find similar sequences, as illustrated by these examples, up to scaling and shifting.

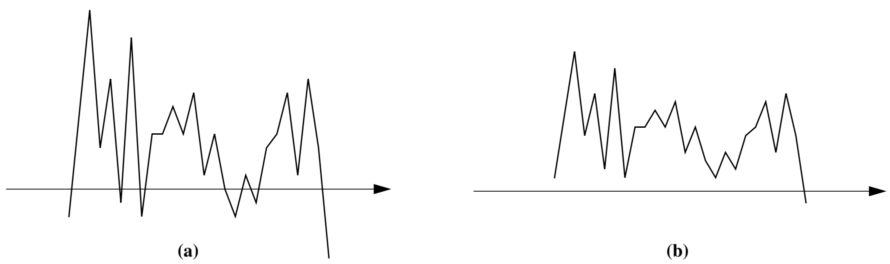

Fig. 1. Sequence (b) is a similarity transformation of (a).

Combinations of scaling and shifting are shape-preserving transformations, known as *similarity transformations* in the mathematical field of Transformational Geometry [15]. We will approach the definition of similarity from this well established geometrical perspective:

> Let G be a set of transformations then two sets of points are similar if there exists a transformation, in G, which maps one to the other.

In geometry, a transformation typically belongs to a group. Combinations of scale and shift are affine transformations. In practice, the user may restrict the allowable transformations to a set that may not be group, by imposing constraints on the scaling and shifting factors, or by fixing one or both factors.

Let D be a distance metric between sequences and $\epsilon \geq 0$ a tolerance. Query sequence Q is *approximately similar* within tolerance ϵ to data sequence S when there exists a similarity transformation T so that $D(Q, T(S)) \leq \epsilon$. When ϵ is set to 0, we have *exact similarity*.

Approximate and exact similarity queries can now be defined just like approximate and exact matching queries were defined in the last section.

The *All-Occurrences Approximate Similarity* problems (either *Subsequence* or *Whole-Sequence*) are defined as was Matching, but with "approximately similar within tolerance ϵ" instead of "match approximately within tolerance ϵ". Analogous definitions apply for *All-Occurrences Exact Similarity* problems (either *Subsequence* or *Whole-Sequence*).

When T is restricted to be the identity transformation, the various similarity problems become the matching problems of the last section. In this sense, our work is a generalization of the work of [7]. This generalization is in the direction of [8], which discusses translation and distortion transformations but does not provide the guarantees of [7] and of our indexing scheme.

A general framework for similarity queries is described in [9]. Our work happens to be (an efficiently solvable) special case. The [9] 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 [9] framework is at best exponential; thus this framework is too general for our purposes.

1.3 Contributions and Overview

Our main contribution is: *A syntax and semantics for similarity queries, that account for approximate matching, scaling and shifting, and that have efficient indexing support.* We show this using a new indexing technique, which preserves all the desirable properties of the indexing scheme proposed in [2, 7].

In Section 2, we provide a *semantics* for similarity querying where we use the *similarity distance* between Q and S (defined in Section 2.2) as the distance metric. Similarity distance constitutes a *good distance metric* because it is non-negative, symmetric, and effectively computable; it also obeys the triangle inequality. This is not true of the "naive" distance metrics that correspond more closely to the formulation of the problem in the Section 1.2. The semantics of Section 2.2 serves as the basis for the *internal representation* of the query, i.e., our *normal form*.

In Section 3, we show that the semantics of Section 2.2 has several desirable properties, such as *updateability* and *well-behaved trails*, which allow us to provide efficient implementations for similarity querying, both in the internal and external query setting. We first adapt the criteria put forth in [2, 7] (Section 3.1) and satisfy them using fingerprints of the normal form (Section 3.2). We then argue that fingerprints are incrementally computable (Section 3.3), can be used ala Rabin-Karp [14] for internal searching (Section 3.4), and most importantly external indexing (Section 3.5).

This is the implementation technology that is needed to support the internal representation of Section 2. Our new indexing technique combines the MBR structure of [7] with our internal representation. Many spatial data-structures can be used, for examples varieties of R-trees (see [17] for a comprehensive survey of the available external data-structures).

In Section 4, we provide a constraint *syntax* for similarity querying. We show how various query variations can be expressed and translated into the internal representation of Section 2. This translation also clarifies the relationship of the problem as defined in Section 1.2 with the semantics of Section 2.2.

The syntax could be embedded in most constraint logic programming languages [5, 6, 10] or constraint query languages [3, 4, 11, 12]. This completes the connection between high level specification and implementation.

The importance of combining high-level specification with efficient implementation is the common theme of constraint databases (e.g., see [4, 13]) and the main motivation for this work. In Section 5, we close with some open problems and future work.

2 The Semantics of Similarity Queries

2.1 Similarity Transformations and Normal Forms

An *n-sequence* X is a sequence $\{x_1, \ldots, x_n\}$ of real numbers. Each n-sequence X has an *average* $\alpha(X)$ and a *deviation* $\sigma(X)$:

$$\alpha(X) = (1/n) \sum_{1 \leq i \leq n} x_i; \quad \sigma(X) = ((1/n) \sum_{1 \leq i \leq n} (x_i - \alpha(X))^2)^{1/2}.$$

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

A pair of reals (a, b) defines a *similarity transformation* $T_{a,b}$ over n-sequences by mapping each element x_i to $a * x_i + b$. 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.

Definition 2.1 We say that X is *similar* to Y if there exist some $(a, b) \in [R^+ \times R]$ such that $X = T_{a,b}(Y)$.

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 $Y = T_{1/a,-b/a}(X) = T^{-1}_{a,b}(X)$ [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, the set of all sequences similar to a given one constitutes an equivalence class, which we call a *similarity class*; we shall denote the similarity class of X by X^*. The similarity relation partitions all n-sequences into similarity classes.

To be able to refer to similarity classes, we need a way to compute a unique representative for each class, given any member in it. Towards that end, we now define normal forms of sequences.

Definition 2.2 An n-sequence X is *normal* if $\alpha(X) = 0$ and $\sigma(X) = 1$.

Let X be normal and Y be similar to X, i.e., $Y = T_{a,b}(X)$ for some $(a, b) \in [R^+ \times R]$. Then, $\alpha(Y) = b$ and $\sigma(Y) = a$:

$\alpha(Y) = (1/n)\sum_{1 \le i \le n} y_i = (1/n)\sum_{1 \le i \le n}(ax_i + b) = (a/n)\alpha(X) + b = b$;
$\sigma^2(Y) = (1/n)\sum_{1 \le i \le n}(y_i - \alpha(Y))^2 = (1/n)\sum_{1 \le i \le n}(ax_i + b - b)^2 = (a^2/n)\sum_{1 \le i \le n}(x_i - 0)^2 = a^2 * \sigma^2(X) = a^2$.

Y is normal only if $\sigma(Y) = a = 1$ and $\alpha(Y) = b = 0$; this is the identity transformation. 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.

Given any n-sequence X, $\nu(X)$ denotes the normal form of X^*. If α is the average of X, and σ is the deviation of X, we've shown that $X = \sigma * \nu(X) + \alpha$. Therefore, we can compute $\nu(X)$ from X by the inverse transformation:

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

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*. 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. The normal form for these classes is defined just as for the general case.

2.2 Similarity Distance and Semantics

Given two sequences X and Y, the *similarity distance* between X and Y is the distance between the normal forms of their respective similarity classes:

Definition 2.3

$$D_S(X, Y) = D_E(\nu(X), \nu(Y)),$$

where D_E is the *Euclidean distance*, defined as follows:

Definition 2.4

$$D_E(S_1, S_2) = (\sum_{1 \le i \le l} (S_1[i] - S_2[i])^2)^{1/2}.$$

Any proper distance metric D for n-sequences can be used instead of D_E. Since we will be using techniques from Discrete Signal Processing (DSP) and the Euclidean distance is a standard distance metric in DSP, we have chosen to use it here.

Note that the similarity distance between any pair of sequences from X^* and Y^* is the same; this gives us a *distance metric for similarity classes*:

Definition 2.5

$$D_S(X^*, Y^*) = D_S(X, Y).$$

A distance metric should be non-negative and symmetric, and it should obey the triangle inequality. A good distance metric should also be effectively computable. It is easy to see that similarity distance satisfies all these criteria.

> **Remark:** Note that there are definitions of distance that correspond more closely to the naive formulation of the problem. For example, given Q and S, we could have used the minimum Euclidean distance between Q and all $S' \in S^*$(the equivalence class of S); let us denote this distance by $D_m(Q, S^*)$. However, this definition is not symmetric: $D_m(Q, S^*) \neq D_m(S, Q^*)$. Given Q and S, we could have also tried to choose the minimum Euclidean distance between Q' and S' for all $Q' \in Q^*$ (the equivalence class of S) and $S' \in S^*$ (the equivalence class of Q). However, by choosing the members of Q^* and S^* with arbitrarily small deviations, this distance will always approach 0. The normal forms provide a distance metric that does not suffer from any of these defects.

By using similarity distance, we are now ready to define a *similarity semantics* for the *All-Occurrences Subsequence Approximate Similarity* problem.

> Given a query sequence Q, a time-series $\overline{S}$, a tolerance $\epsilon \geq 0$, and a similarity relation [which partitions sequences into equivalence classes with normal forms], find all contiguous subsequences S in the time-series $\overline{S}$ such that $D_S(Q, S) \leq \epsilon$.

Note that these semantics are slightly different from the problem formulation in Section 1.2. The differences will be clarified (and bridged) in Section 4. To conclude this subsection, we want to consider the *All-Occurrences Subsequence Exact Similarity* problem (i.e., when $\epsilon = 0$).

> Given a query sequence Q, a time-series $\overline{S}$, and a similarity relation [which partitions sequences into equivalence classes with normal forms], find all contiguous subsequences S in the time-series $\overline{S}$ such that Q and S are similar [belong to the same equivalence class].

The exact case can be answered using the normal forms, because Q and S are in the same equivalence class if and only if $\nu(Q) = \nu(S)$. Finally, analogous definitions apply to Whole-Sequence problems.

3 Indexing of Similarity Queries

3.1 Sequence Fingerprints: Criteria and Definitions

Computing the similarity distance D_S between any two n-sequences requires $O(n)$ operations. An efficient implementation of similarity querying cannot afford to compute D_S every time for each sequence in the data set (for the Whole-Sequence case), or for each contiguous subsequence in the time-series (for the Subsequence case).

Following the approach of [14], which has gained wide acceptance, we introduce a *fingerprint* function F, together with a fingerprint distance metric D_F. This fingerprint mechanism provides *fast rejection*, filtering out most of the non-similar sequences.

A fingerprint mechanism needs to satisfy the following criteria:

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

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

- **Accuracy:** If the comparison of $F(S)$ and $F(Q)$ returns *TRUE*, S is highly likely to be a valid query answer.
- **Updateability:** Computing the fingerprints of all subsequences of an N-sequence for N much larger than n can be done in $O(N)$ time, by *updating* the fingerprint value as we move along rather than recomputing it for every subsequence.

We now define the fingerprint function F as well as the fingerprint distance function D_F. These definitions are similar to the ones used for Approximate Matching in [2] and [7].

Definition 3.1 A *fingerprint* $F(X)$ of an n-sequence $X = \{x_1, \ldots, x_n\}$ is the tuple

$$[DFT_1(\nu(X)), \ldots, DFT_l(\nu(X))],$$

where l is a small constant (such as 3), and DFT_m is the m'th coefficient of the *Discrete Fourier Transform* of $\nu(X)$:

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

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

Note that DFT_m is a complex number: $DFT_m = a_m + b_m i$ for some $a_m, b_m \in R$. Thus, F is specified by a sequence of $2l$ real values. Note that we are not including DFT_0 in the fingerprint, since its coefficients are both 0 for normal sequences.

Definition 3.2 The *fingerprint distance* D_F between $F(X)$ and $F(Y)$ is the Euclidean distance between the real-valued sequences for $F(X)$ and $F(Y)$.

By taking l to be a small constant our fingerprint mechanism is compact. In the subsections below, we establish its validity, accuracy, and updateability.

3.2 Validity and Accuracy of Fingerprinting

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

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

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

$$DFT_m(aX + bY) = aDFT_m(X) + bDFT_m(Y)$$

for all scalars a and b. Also, we rely on *Parseval's theorem*, well-known in DSP:

$$\sum_{0 \leq m \leq n-1} |DFT_m(X)|^2 = \sum_{0 \leq i \leq n-1} |x_i|^2.$$

And we make use of the fact that the coefficients of DFT_0 for normal sequences are both 0. First, we show that $D_S(X,Y) \geq D_F(F(X),F(Y))$:

$$\begin{aligned} &D_S(X,Y) = D_E(\nu(X),\nu(Y)) = (\textstyle\sum_{0 \leq i \leq n-1} |\nu(x_i) - \nu(y_i)|^2)^{1/2} = \\ &= (\textstyle\sum_{0 \leq m \leq n-1} |DFT_m(\nu(X) - \nu(Y))|^2)^{1/2} \geq \\ &\geq (\textstyle\sum_{0 \leq m \leq l} |DFT_m(\nu(X) - \nu(Y))|^2)^{1/2} = D_F(F(X),F(Y)). \end{aligned}$$

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

To establish accuracy, we want to know how likely it is that $D_S(X,Y) \leq \epsilon$ provided that $D_F(F(X),F(Y)) \leq \epsilon$. The cases when $D_F(F(X),F(Y)) \leq \epsilon$ but $D_S(X,Y) \leq \epsilon$ 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

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

As we mentioned in Section 1.1, time-series data have a *skewed energy spectrum*, which implies that

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

As shown in [2], 2-3 coefficients are usually sufficient to provide good accuracy. Therefore, we may assume that the length l of the fingerprint is ≤ 3. (Note that, the randomization ala Rabin-Karp [14] makes no assumptions about the spectrum.)

3.3 Updateability of Fingerprinting

When computing fingerprints of all subsequences of length n for a much longer sequence of length N, the efficiency of the algorithm hinges on a property of the fingerprint that we call *updateability*:

> Given the 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.

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:

$$\sum_{k \le j \le k+n-1} x_j, \sum_{k \le j \le k+n-1} (x_j)^2, \alpha(X_k), \sigma(X_k), DFT_1(X_k), \ldots, DFT_l(X_k), F(X_k).$$

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 $\sum_{k+1 \le j \le k+n} x_j$. This involves one subtraction and one addition:
$$\sum_{k+1 \le j \le k+n} x_j = (\sum_{k \le j \le k+n-1} x_j) - x_k + x_{k+n};$$
4. Compute $\sum_{k+1 \le j \le k+n} (x_j)^2$, using two multiplications, one subtraction and one addition:
$$\sum_{k+1 \le j \le k+n} x_j^2 = (\sum_{k \le j \le k+n-1} x_j^2) - x_k^2 + x_{k+n}^2;$$
5. Compute $\alpha(X_{k+1})$, using one multiplication:
$$\alpha(X_{k+1}) = \alpha_{k+1} = (1/n) \sum_{k+1 \le j \le k+n} x_j;$$
6. Compute $\sigma(X_{k+1})$, using two multiplications, one subtraction and one square root:
$$\sigma^2(X_{k+1}) = \sigma_{k+1}^2 = (1/n)(\sum_{k+1 \le j \le k+n} x_j^2) - \alpha_{k+1}^2;$$
7. Compute $DFT_1(X_{k+1}), \ldots, DFT_l(X_{k+1})$, using one subtraction, one addition and two multiplications for each index:
$$DFT_m(X_{k+1}) = \frac{1}{\sqrt{n}} \sum_{0 \le j \le n-1} (x_{j+k+1} e^{-j(2\pi i)m/n}) =$$
$$= \frac{1}{\sqrt{n}} \sum_{1 \le j \le n} (x_{j+k} e^{-(j-1)(2\pi i)m/n}) = e^{(2\pi i)m/n} (DFT_m(X_k) + \frac{x_{n+k} - x_k}{\sqrt{n}});$$

8. Compute the 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 $DFT_m(\mathbf{1})$ is 0 when $m > 0$:

$$DFT_m(\nu(X_{k+1})) = DFT_m(X_{k+1}/\sigma_{k+1} - \alpha_{k+1}/\sigma_{k+1}) =$$

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

Note that the above algorithm is *on-line*, suitable in a situation when the data are streaming past and we can never back over it. In addition, we have shown that the fingerprint of [2] for time-series approximate matching (without similarity) is also updateable.

3.4 Internal Algorithms

In this section, we sketch out the internal implementation of similarity queries, omitting the details that can be found in the works we reference.

The implementation is based on a *unified internal representation* of a similarity query; its definition is given below. We will show in Section 4 how to translate the constraint-based syntax of a user into to the unified internal representation. For the rest of this section, we assume that we have already obtained this internal representation.

Definition 3.3 The *internal representation* of a similarity query consists of: (query sequence Q, the values $\{\epsilon_i, l_\alpha, u_\alpha, l_\sigma, u_\sigma\}$).

This corresponds to the Section 2.2 semantics of All-Occurrences Subsequence Approximate Similarity:

Find all S such that $D_S(Q,S) \leq \epsilon_i$, $l_\alpha \leq \alpha(S) \leq u_\alpha$, $l_\sigma \leq \sigma(S) \leq u_\sigma$.

The updateability results established in Section 3.3 allow us to answer the query for the *in-core case* with an algorithm much like that of [14]. We proceed through the sequences and the subsequences, comparing $D_F(F(Q), F(S))$ to ϵ_i and checking $\alpha(S)$ and $\sigma(S)$ against the bounds. In the case of a potential match, we use the user formulation of the problem (Section 4) to determine if we have a valid query answer, or if it is a *false alarm.*

When we want to avoid run-time linear scanning of the data, we need to create an *index structure* for it. This is the case for any database application of similarity querying. A naive index structure is a list of tuples of the following form:

$[F(X), \alpha(X), \sigma(X)$, location of $X]$.

The algorithm is very similar to the in-core case; we scan the index and look for potential matches:

Is $D_F(F(Q), F(S)) \leq \epsilon_i$, $l_\alpha \leq \alpha(S) \leq u_\alpha$, and $l_\sigma \leq \sigma(S) \leq u_\sigma$?

Whenever one is found, we retrieve X and do the final determination.

3.5 External Indexing

To speed up index searching, we can instead build an indexing mechanism that allows spatial access methods for range queries of multidimensional points. The query point P_Q is computed from the internal query representation:

$$P_Q = (F(Q), m_\alpha, m_\sigma), \text{ where } m_\alpha = (u_\alpha + l_\alpha)/2,\ m_\sigma = (u_\sigma + l_\sigma)/2.$$

The answer to the similarity query is the set of all points $(F(X), \alpha(X), \sigma(X))$ such that:

$F(X)$ is within ϵ of $F(Q)$, $\alpha(X)$ is within $(u_\alpha - l_\alpha)/2$ of m_α, and
$\sigma(X)$ is within $(u_\sigma - l_\sigma)/2$ of m_σ.

Though the index structure described above performs well for the whole-sequence case, it is unsuitable in the subsequence setting. This is due to a very simple observation: for a real sequence of length m, the there would be $m-n+1$ indices with $2l$ reals each. Such space overhead renders indexing less efficient than a direct sequential scan of the data [7].

This problem is overcome with the Minimum Bounding Rectangle (MBR) technique, introduced in [7]. This technique significantly reduces the size of the indexing structure, though introducing some false alarms in the process. Our final indexing method consists of combining the MBR technique with the spatial access approach described above.

The MBR technique relies of the *continuity* of subsequence indices:

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

We conclude the discussion of indexing by verifying that our indexing possesses the continuity property. This is a "heuristic" statistical argument, that also applies to [7] (where continuity was assumed, but not shown).

Denote adjacent sequences by $X_0 = \{x_0, \ldots, x_{n-1}\}$ and $X_1 = \{x_1, \ldots, x_n\}$; denote the corresponding indices by $(\alpha_0, \sigma_0, F_0)$ and $(\alpha_1, \sigma_1, F_1)$; and Denote the order of the expected value of an expression by $\approx$. Assume that n is reasonably large, so that $1/n$ is considered to be a small constant. Proceed by considering each element of the index separately.

1. The expected value of $|\alpha_1 - \alpha_0|$ is on the order of σ_0/n:

$$|\alpha_1 - \alpha_0| = |x_n - x_0|/n \approx \sigma_0/n.$$

2. The expected value of $|\sigma_1 - \sigma_0|$ is on the order of σ_0/n:

$$|\sigma_1 - \sigma_0|(\sigma_1 + \sigma_0) = |\sigma_1^2 - \sigma_0^2| =$$

$$= \frac{1}{n}|(x_n - \alpha_1)^2 + (x_0 - \alpha_0)^2 + \sum_{1 \le j \le n-1} ((x_j - \alpha_1)^2 - (x_j - \alpha_0)^2)| =$$

$$= |(x_n - x_0)(x_n - \alpha_1 + x_0 - \alpha_0)|/n \approx (\sigma_0/n)(\sigma_1 + \sigma_0).$$

3. The expected value of $|DFT_m(\nu(X_1)) - DFT_m(\nu(X_0))|$ is on the order of $|DFT_m(\nu(X_0))|/n + 1/\sqrt{n}$. Here, we use the equations for DFT's derived in Section 3.3, as well as the ones derived above:

$$|DFT_m(\nu(X_1)) - DFT_m(\nu(X_0))| = |DFT_m(X_1)/\sigma_1 - DFT_m(\nu(X_0))| =$$

$$= |\frac{e^{(2\pi i)m/n}}{\sigma_1}(\sigma_0 DFT_m(\nu(X_0)) + \frac{x_n - x_0}{\sqrt{n}}) - DFT_m(\nu(X_0))| =$$

$$= |DFT_m(\nu(X_0))(\frac{\sigma_0}{\sigma_1}e^{(2\pi i)m/n} - 1) + \frac{x_n - x_0}{\sigma_1\sqrt{n}}e^{(2\pi i)m/n}| \approx$$

$$\approx |DFT_m(\nu(X_0))((1+1/n)(1-2\pi m/n) - 1) + (1+1/n)(1-2\pi m/n)/\sqrt{n}| \approx$$

$$\approx |DFT_m(\nu(X_0))|/n + O(1/\sqrt{n})$$

4 Constraint Specification of Similarity Queries

In the previous section, we have shown how to provide an efficient database indexing mechanism for queries with similarity semantics. This enables us to answer the following question:

Given Q, ϵ_i, (l_α, u_α) and (l_σ, u_σ),
find all S such that
$D_S(Q,S) \leq \epsilon,\ l_\alpha \leq \alpha(S) \leq u_\alpha,\ l_\sigma \leq \sigma(S) \leq u_\sigma$.

This is based on our *internal representation* of the actual user query, whose syntax may be very different. In particular, the user querying a database of sequences should probably not be expected to refer to normal forms, similarity distance, or even to averages and deviations.

In this section, we provide a *syntax* for similarity queries, based on constraints. Then, we show how to translate from this syntactic formulation to the internal representation.

It is important to note that the translation is not always *bi-directional*. There may be additional *false alarms* retrieved by the indexing mechanism. These will be filtered out via a brute-force comparison prior to returning the query result. The additional filtering indicates a potential performance disadvantage of our approach and is a trade-off to achieve generality.

4.1 Constraint-Based Syntax of Similarity Queries

For the general similarity query, we propose the following constraint-based syntax, which expresses the queries described in Section 1.2:

- **General Case:** `Given` Q`,` ϵ`,` (l_a, u_a)`, and` (l_b, u_b)`,`
  `find all` $[S, a, b]$ `such that`
  $D(Q, aS+b) \leq \epsilon$`,` $l_a \leq a \leq u_a$ `and` $l_b \leq b \leq u_b$`.`

We assume that the sequence distance D is the Euclidean distance D_E between sequences. Of course, the user may choose to omit the bounds on a and/or b:

- **Unbounded Case:** `Given` Q `and` ϵ`,`
  `find all` $[S, a, b]$ `such that` $D(Q, aS + b) \leq \epsilon$.

In all the following queries, bounds either a or b are optional.

The user may want to query for scaling transformations only, or for shift transformations only:

- **Scaling:** `Find all` $[S, a]$ `such that` $D(Q, aS) \leq \epsilon$.
- **Shifting:** `Find all` $[S, b]$ `such that` $D(Q, S + b) \leq \epsilon$.

Finally, the user may ask approximate matching queries or exact similarity queries:

- **Approximate Match:** `Find all` S `such that` $D(Q, S) \leq \epsilon$.
- **Exact Similarity:** `Find all` $[S, a, b]$ `such that` $Q = aS + b$.

For all these variations on similarity queries, it is possible to efficiently find the corresponding values for the variables used in the internal query representation:

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

We shall show how to do that in the next subsection.

4.2 From Constraint-Based Syntax to Internal Representation

We first reduce the cases of scaling, shifting, alternate matching, and exact similarity to the general case:

Scaling: force b to have the value 0 by specifing an equality constraint $b = 0$:

$$D(Q, aS) \leq \epsilon \Leftrightarrow D(Q, aS + 0) \leq \epsilon \Leftrightarrow D(Q, aS + b) \leq \epsilon \wedge b = 0.$$

Shifting: force a to have the value 1 by specifing an equality constraint $a = 1$:

$$D(Q, S + b) \leq \epsilon \Leftrightarrow D(Q, 1 * S + b) \leq \epsilon.$$

Approximate Matching: by similar reasoning, set a to 1 and b to 0.
Exact Similarity: set ϵ to 0:

$$Q = aS + b \Leftrightarrow D(Q, aS + 0) = 0.$$

Then, we convert the general case to the internal representation. We need to compute the values for $\{\epsilon_i, l_\alpha, u_\alpha, l_\sigma, u_\sigma\}$, so that S is a valid answer to the general similarity query only if

$$D_S(Q, S) \leq \epsilon_i \text{ and } l_\alpha \leq \alpha(S) \leq u_\alpha \text{ and } l_\sigma \leq \sigma(S) \leq u_\sigma.$$

To bridge the difference between Section 1.2 and Section 2.2, we proceed as follows.

Let Q be the given pattern and S an answer for the general case, such that there exist a, b with $D(Q, aS+b) \leq \epsilon$, $l_a \leq a \leq u_a$ and $l_b \leq b \leq u_b$. Let us denote the minimum distance between Q and S', for all $S' \in S^*$, by $D_m(Q, S^*)$, and $\sigma(Q)$ by σ. It can be shown that:

Lemma: $D_m^2(Q, S^*) = \sigma^2(D_S^2 - D_S^4/4)$, where $D_S = D(\nu(Q), \nu(S))$.

If $D(Q, aS+b) \leq \epsilon$ then,

$$D_m(Q, S^*) \leq \epsilon \Leftrightarrow D_m^2(Q, S^*) \leq \epsilon^2 \Leftrightarrow \sigma^2(D_S^2 - D_S^4/4) \leq \epsilon^2 \Leftrightarrow$$
$$\Leftrightarrow D_S^4 - 4D_S^2 + 4\epsilon^2/\sigma^2 \geq 0 \Leftrightarrow (D_S^2 \leq 2 - 2\sqrt{1 - \epsilon^2/\sigma^2}) \vee (D_S^2 \geq 2 + 2\sqrt{1 - \epsilon^2/\sigma^2}).$$

We know that $D_S^2 \leq 2$, since our similarity transformations do not allow scaling with a negative factor. Therefore, the square root of the first inequality is the desired expression for ϵ_i. Note that ϵ/σ must be less than 1; this does not reduce the expressibility, since the minimal distance $D_m^2(Q, S^*)$ is never greater than $\sigma(Q)$. The above calculation of ϵ_i is bi-directional (iff) when there are no bounds on a, b.

Otherwise, we also need to find "good" bounds for $\sigma(S)$ and $\alpha(S)$. We know that under exact similarity

$$Q = aS + b,$$
$$Q = \sigma(Q) * \nu(Q) + \alpha(Q),$$
$$S = \sigma(S) * \nu(S) + \alpha(S).$$

We solve these equations for a and b:

$$a = \sigma(Q)/\sigma(S),$$
$$b = \alpha(Q) - \alpha(S)\sigma(Q)/\sigma(S).$$

We combine the equation for a with the bounds $l_a \leq a \leq u_a$ to obtain the bounds for $\sigma(S)$, without any loss in accuracy:

$$l_a \leq a \leq u_a \Leftrightarrow l_a \leq \sigma(Q)/\sigma(S) \leq u_a \Leftrightarrow 1/u_a \leq \sigma(S)/\sigma(Q) \leq 1/l_a \Leftrightarrow$$
$$\Leftrightarrow \sigma(Q)/u_a \leq \sigma(S) \leq \sigma(Q)/l_a.$$

We combine the equations for b with the bounds $l_b \leq b \leq u_b$ to obtain the bounds for $\alpha(S)$:

$$l_b \leq b \leq u_b \Leftrightarrow l_b \leq \alpha(Q) - \alpha(S)\sigma(Q)/\sigma(S) \leq u_b \Leftrightarrow$$
$$\Leftrightarrow \alpha(Q) - u_b \leq \alpha(S)\sigma(Q)/\sigma(S) \leq \alpha(Q) - l_b \Leftrightarrow$$
$$\Leftrightarrow \sigma(S)(\alpha(Q) - u_b)/\sigma(Q) \leq \alpha(S) \leq \sigma(S)(\alpha(Q) - l_b)/\sigma(Q).$$

Since $\sigma(S)$ is not a constant value, we have to eliminate it from the bounds. We accomplish this goal by substituting the lower and upper bounds for $\sigma(S)$, obtained earlier, into the above inequalities:

$$(\sigma(Q)/u_a \leq \sigma(S)) \wedge (\sigma(S)(\alpha(Q) - u_b)/\sigma(Q) \leq \alpha(S)) \Rightarrow (\alpha(Q) - u_b)/u_a \leq \alpha(S);$$
$$(\sigma(S) \leq \sigma(Q)/l_a) \wedge (\alpha(S) \leq \sigma(S)(\alpha(Q) - l_b)/\sigma(Q)) \Rightarrow \alpha(S) \leq (\alpha(Q) - l_b)/l_a.$$

Combining the resulting inequalities, we obtain:

$(\alpha(Q) - u_b)/u_a \leq \alpha(S) \leq (\alpha(Q) - l_b)/l_a$.

The last reduction is not bi-directional. Though we have not forced any false dismissals, we may have generated some false alarms. The inaccuracy introduced when computing the bounds for $\alpha(S)$ is proportional to $u_a - l_a$. Therefore, the expected speed-up from bounding b prior to indexing vs. performing the filtering of b's after the indexing is maximized when there is a very tight bound on a. An ideal candidate is a pure shift query, where $a = 1$.

We summarize the results of this section in a table, providing the values for the internal representation in terms of the user-specified external values.

ϵ_i	$\sqrt{2 - 2\sqrt{1 - \epsilon^2/\sigma^2}}$
l_σ	$\sigma(Q)/u_a$
u_σ	$\sigma(Q)/l_a$
l_α	$(\alpha(Q) - u_b)/u_a$
u_α	$(\alpha(Q) - l_b)/l_a$

Fig. 2. Computing the values for the Internal Representation

5 Conclusion and Open Questions

We have proposed a framework for Time-Series Approximate Similarity Queries that allows the user to pose a wide variety of queries and that preserves desirable indexing properties of Time-Series Approximate Matching. Posssible extensions involve more powerful similarity queries and different distance functions; also indexing of time-series data that is represented using constraints (see [3, 11]).

As with a generalized version of any problem, there is a potential trade-off between a gain in expressibility and a decrease in performance. There should be some slow-down due to the extra keys in the new indexing scheme (as opposed to [7]) and additional slow-down may come from the false alarms generated by our fingerprinting as compared to the specialized cases (without a similarity transformation). We are examining this trade-off through performance evaluation of several versions of similarity querying. The first version is the most general one described here; other versions involve tailoring the internal representation and the corresponding indexing scheme for specialized subsets of similarity queries. This experimental work in progress.

The existence of a fingerprint function for internal similarity querying that is a real hash function but is also distance-preserving and updateable is an interesting open question. *Is there a fingerprint method that gives a provably linear performance for the Rabin-Karp algorithm [14], either for approximate matching or for similarity querying? Can it be truly randomized for any adversary?*

References

1. A. Aho. Algorithms for Finding Patterns in Strings. *Handbook of TCS.*, J. Van Leeuwen editor, volume A, chapter 5, Elsevier, 1990.
2. R. Agrawal, C. Faloutsos, A. Swami. Efficient Similarity Search in Sequence Databases. *FODO Conf.*, Evanston, Ill., Oct. 1993
3. M. Baudinet, M. Niezette, P. Wolper. On the Representation of Infinite Temporal Data and Queries. *Proc. 10th ACM PODS*, 280–290, 1991.
4. A. Brodsky, J. Jaffar, M.J. Maher. Toward Practical Constraint Databases. *Proc. 19th VLDB*, 322–331, 1993.
5. A. Colmerauer. An Introduction to Prolog III. *CACM*, 33:7:69–90, 1990.
6. M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, F. Berthier. The Constraint Logic Programming Language CHIP. *Proc. Fifth Generation Computer Systems*, Tokyo Japan, 1988.
7. C. Faloutsos, M. Ranganathan, Y. Manolopoulos. Fast Subsequence Matching in Time-Series Databases. *Proc. ACM SIGMOD Conf.*, pp. 419–429, May 1994
8. H.V. Jagadish. A Retrieval Technique for Similar Shapes. *Proc. ACM SIGMOD Conf.*, pp. 208–217, May 1991
9. H. V. Jagadish, A. O. Mendelzon, T. Milo. Similarity-Based Queries. to appear in *Proc. 14th ACM PODS*, 1995
10. J. Jaffar, J.L. Lassez. Constraint Logic Programming. *Proc. 14th ACM POPL*, 111–119, 1987.
11. F. Kabanza, J-M. Stevenne, P. Wolper. Handling Infinite Temporal Data. *Proc. 9th ACM PODS*, 392–403, 1990.
12. P. C. Kanellakis, G. M. Kuper, P. Z. Revesz. Constraint Query Languages. *Proc. 9th ACM PODS*, 299–313, 1990. Full version available as Brown Univ. Tech. Rep. CS-92-50. To appear in JCSS.
13. P. C. Kanellakis, S. Ramaswamy, D. E. Vengroff, J. S. Vitter. Indexing for Data Models with Constraints and Classes. *Proc. 12th ACM PODS*, 233–243, 1993.
14. R. M. Karp and M. O. Rabin. Efficient Randomized Pattern-Matching Algorithms. *IBM J. Res. Develop.*, 31(2), 1987
15. Modenov and Pakhomenko. *Geometric Transformations*, Academic Press, 1965
16. A.V. Oppenheim and R.W. Schafer. *Digital Signal Processing*, Prentice Hall, 1975
17. H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, Reading MA, 1990.
18. P. Sheshadri, M. Livny, R. Ramakrishnan. Sequence Query Processing *Proc. ACM SIGMOD Conf.*, pp. 430–441, May 1994

This article was processed using the LaTeX macro package with LLNCS style