A Signature Technique for Similarity-Based Queries

(Extended Abstract)

*C. Faloutsos**
Univ. of Maryland
christos@cs.umd.edu

H. V. Jagadish
AT&T Labs
jag@research.att.com

A. O. Mendelzon
Univ. of Toronto
mendel@db.toronto.edu

T. Milo
Tel Aviv Univ.
milo@math.tau.ac.il

1 Introduction

Sequences of real-valued data arise in many applications ranging from the stock market to electro-cardiograms. Often, it is of interest to locate sequences that are similar to a specified query sequence. The notion of similarity is application dependent, and even within a single application, may vary from one query to the next.

Work in this area is usually specific to one particular domain and uses one specific notion of similarity. For example, Faloutsos *et al* [7, 1] studied the problem of searching a database of time sequences for sequences similar to one given. They reduced sequences to points in a low-dimensional space by using Fourier transforms and used the Euclidean distance in this space to measure similarity. This notion of similarity is extended in [21] by allowing a class of transformations that includes moving average and time warping to be applied to sequences before computing the Euclidean distance. Retrieval by similarity has also been studied in the context of image retrieval [15], genome/protein matching [3, 12] and text string searching [31]. Other authors have recently studied models and languages for databases containing sequences, (*e.g.*, [5], [13], [10], [22], [26]), but without taking into account notions of similarity or approximation.

In a previous paper [16], Jagadish *et al* developed a general framework for posing queries based on similarity. The framework enables a formal definition of the notion of similarity for an application domain of choice, and then its use in queries to perform

*. This work was partially supported by the Institute of Systems Research and by the National Science Foundation under Grants No. EEC-94-02384, IRI-9205273 and IRI-9625428

similarity-based search. In this paper, we adapt this framework to the specialized domain of real-valued sequences. (Although some of the ideas we present are applicable to other types of data as well). In particular we focus on *whole-match* queries. By *whole-match* query we mean the case where the user has to specify the whole sequence (*e.g.*, in a collection of 2-second voice clips with the phrase "good-morning" *find the ones that are similar to my own utterance*).

Similarity-based search can be computationally very expensive. The computation cost depends heavily on the length of sequences being compared. To make such similarity testing feasible on large data sets, we propose the use of a signature based technique. In a nutshell, our approach is to "shrink" the data sequences into *signatures*, and search the signatures instead of the real sequences, with further comparison being required only when a possible match is indicated. Being shorter, signatures can usually be compared much faster than the original sequences. In addition, signatures are usually easier to index. For such a signature-based technique to be effective one has to assure that

(1) the signature comparison is fast, and

(2) the signature comparison gives few false alarms, and no false dismissals.

We study these issues below, and present conditions under which these requirements are satisfied. Our goal is to show that this general framework fits many real-life applications and leads to efficient searching. The techniques we suggest have been implemented and tested. At least in one application of interest, these techniques did lead to a significant improvement in performance.

2 Basics

In this section we present the basic framework on sequences, similarity measurements, and signature extraction.

2.1 Sequences, Distance Functions, and Transformation Languages

Real-valued sequences, like stock-market or electro-cardiogram data, can be viewed as strings of numbers. For example, a possible data sequence could be the string $\vec{x} = \{10.2, 12.5, 3.0\}$.

We use the following notational conventions:

- x_i denotes the i-th entry of the sequence $\vec{x}$
- $x_{i:j}$ denotes the sub-sequence $\{x_i, x_{i+1}, \ldots, x_j\}$ of the sequence $\vec{x}$

Following the framework of [16], the dissimilarity between two objects can be measured as the cost of transforming one into another by means of a transformation sequence selected from a *transformation language* $\mathcal{T}$. Thus the distance between two sequences measures the cost to transform the first sequence to the second, or both to a common, third sequence, given an application dependent set of allowable transformations and their associated costs. Given a set of transformations $\mathcal{T}$ and a transformation $T \in \mathcal{T}$, and a sequence $\vec{x}$ in some set of sequences $\mathcal{S}$ (*e.g.*, $\mathcal{S} = \Re^n$, $n = 1, 2, \ldots$), $T(\vec{x})$ is the sequence in $\mathcal{S}$ that results from applying transformation T to $\vec{x}$. The cost of this transformation is $cost(T)$.

We extend [16] by allowing the possibility that, after all allowable transformations are exhausted, the two sequences are still different, in which case we measure the distance between the transformed strings using a traditional distance function, denoted by $\mathcal{D}_0()$, such as the Euclidean distance or the city-block (Manhattan or L_0) distance function. The $\mathcal{D}_0()$ distance will be called the *base distance*. The *distance* between two strings $\vec{x}, \vec{y}$ is defined as the cost of transforming each of the strings to two strings that are as close as possible in base distance, plus the base distance between the transformed strings. Formally,

$$\mathcal{D}(\vec{s}, \vec{t}) = \min_{T_1, T_2 \in \mathcal{T}} (cost(T_1) + cost(T_2) + \mathcal{D}_0(T_1(\vec{x}), T_2(\vec{y}))) \tag{1}$$

Often, the allowable transformations consist of a sequence of basic building blocks. In this case, let $\mathcal{T}_0$ be the set of these basic, *atomic* transformations. For example, for the string-editing distance, the set of atomic transformations could be $\mathcal{T}_0 = \{$ 'insert', 'delete', 'substitute' $\}$. A composite transformation is an allowed sequence of such atomic transformations with cost that is the sum of the individual costs. We can express the distance function recursively as follows:

$$\mathcal{D}(\vec{x}, \vec{y}) = \min \left\{ \begin{array}{l} \min_{T_1, T_2 \in \mathcal{T}_0} (cost(T_1) + cost(T_2) + \mathcal{D}(T_1(\vec{x}), T_2(\vec{y}))) \\ \mathcal{D}_0(\vec{x}, \vec{y}) \end{array} \right. \tag{2}$$

As we show in Section 7, several practical distance functions follow this model. The Euclidean distance readily obeys the model, if no transformations are allowed, and $\mathcal{D}_0()$ is the Euclidean distance.

Definition 2.1 *A base distance function $\mathcal{D}_0()$ is said to be* additive *if for sequences $\vec{x}, \vec{y}$ of equal length l*
$\mathcal{D}_0(\vec{x}, \vec{y}) = \Sigma_{i=1 \ldots l} d(x_i, y_i)$, where $d()$ is some non-negative function, and $\mathcal{D}_0(\vec{x}, \vec{y})$ is undefined otherwise.

We require in this paper that the base distance function used be additive. This is not an onerous requirement since every example we are aware of in practice does satisfy this requirement. Observe that one cannot compute the base distance between two sequences of unequal length.

2.2 Signatures

Given a database containing sequences and a query sequence to be matched within a certain distance, a naive evaluation strategy is to iterate over the sequences in the database and for each one compute the distance from the given sequence. The complexity of each such test is determined by the length of the sequence and the notion of similarity being used (that is, the class of transformations allowed). Since individual sequences in the comparison can often be large, approximate matching can be computationally intensive.

We wish to reduce the computation cost by using short representative *signatures* to perform the matching instead of the real sequences. Signatures are scanned sequentially and matched against the signature of the given query. Due to their small size, this scan can take place orders of magnitude faster than a full scan and match on the entire database. In some applications, if the signatures are short enough, it may even be possible to build index structures on the signatures.

A signature is a word in a selected *description language*. We associate a deterministic Turing machine $T_{\mathcal{L}}$ with a given description language $\mathcal{L}$. We say that a sequence $\vec{x}$ is *(exactly) represented* by a word w in a description language $\mathcal{L}$, if $\vec{x}$ is the output of T_L on the input w. Note that no two sequences are represented by the same word; we often refer to the sequence represented by word w as $seq(w)$. We extend the seq mapping from words to sub-words by saying that a subsequence $\vec{x}'$ of $\vec{x}$ is represented by a sub-word w' of w when $\vec{x}'$ is the output of T_L on w'.

A sequence may be represented in many different ways in a given description language. Even when a sequence does not have a compact signature, it may be possible to use a compact signature that represents a "similar" sequence. An example of such an approximate signature is the representation of a sequence by its first few Fourier coefficients [1].

Definition 2.2 *Given a sequence $\vec{x}$, a base-distance measure $\mathcal{D}_0(.)$, and a description language $\mathcal{L}$, the ϵ-*complexity *of $\vec{x}$ is the smallest integer K such that there exists a word $w_x \in \mathcal{L}$ with $|w_x| = K$ and $\mathcal{D}_0(\vec{x}, seq(w_x)) \leq \epsilon$. If there is no such word in $\mathcal{L}$, the ϵ-complexity of $\vec{x}$ is undefined. Such a word w_x, which is not necessarily unique, is called an ϵ-signature of $\vec{x}$ in $\mathcal{L}$.*

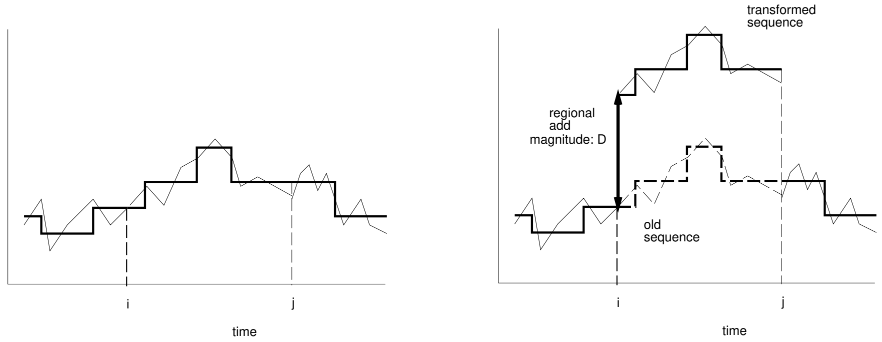
Although not every sequence will have an exact representation in the description language $\mathcal{L}$, we will choose $\mathcal{L}$ and ϵ in each application to ensure that every sequence has an ϵ-signature. For brevity we will omit the ϵ and just use the word signature. A signature of a sequence $\vec{x}$ will be denoted by w_x. Sequences that have an exact representation in a description language $\mathcal{L}$ will be called *canonical.* For a signature w, the sequence $seq(w)$, is the canonical sequence represented by w. If w_x is a signature of $\vec{x}$, the canonical sequence represented by w_x will be called a *canonical form* of $\vec{x}$ and denoted $\tilde{x}$, which is shorthand for $seq(w_x)$.

In general, finding a good representation for a sequence is difficult. Given a description language $\mathcal{L}$ (with associated Turing machine T_L), a base distance function $\mathcal{D}_0$, a set of transformations $\mathcal{T}$, a distance bound ϵ, a sequence s, and a number k, we call the problem of testing whether s has ϵ-complexity of k, the *signature testing problem.* Note that the problem of determining the Kolmogorov complexity of a sequence [17] is a special case of this problem, from which it follows that:

Theorem 2.1 *The signature testing problem is undecidable.*

The undecidability comes from the power of having an arbitrary Turing machine T_L to compute sequences from descriptions; in practice, people do not use arbitrary description languages. They use Fourier transform, piece-wise linear approximations, regular expressions, *etc.* For such languages it is easy to devise a simple grammar to determine whether a description word is valid in that it describes some sequence. And for such words the mapping usually takes time that is at most linear in the length of the sequence.

Figure 1 gives an example, where the transformation T is what we call "regional add": A regional add transformation $R < i, j, \Delta >$ of magnitude Δ at positions i through j of a sequence adds Δ to every entry of the sequence, starting from position i until position j, included. We assume that the description language $\mathcal{L}$ uses piece-wise constant (*i.e.*, zero-th order polynomial) approximations to obtain canonical sequences. Specifically, Figure 1 shows (a) a sequence (light line) and (the canonical representation of) its signature (bold line), and (b) the effect of a transformation ('regional add') on the sequence and the signature. The heavy line with the double arrows stand for the 'jump' of magnitude Δ at position i.



(a) sequence and canonical form (b) a 'regional add' transformation

Figure 1: (a) Illustration of a sample sequence, and its signature (piece-wise flat approximation (b) example of a transformation ('regional add', of magnitude Δ at interval i-j and its effect on a sequence and its signature

3 Similarity Retrieval - Conditions for Efficiency and Correctness

3.1 Lower Bounding

We need to ensure that the process of matching signatures of the database strings against the query signature does not lead to false dismissals. For this we show that, under realistic conditions, the distance between two signatures provides a lower bound on the distance between the pairs of strings that map to them. Thus, if two signatures are far apart, we know the corresponding strings must also be far apart.

Definition 3.1 *Let $\mathcal{C}_{\mathcal{L}}$ be the set of all canonical sequences with respect to language $\mathcal{L}$:*

$$\mathcal{C}_{\mathcal{L}} = \{seq(w) \mid w \in \mathcal{L}\} \tag{3}$$

We need to know how the transformations in $\mathcal{T}$ distort distances, and to find bounds on these distortions. First, we define the maximum distortion of the distance between a sequence $\vec{x}$ and any of its canonical forms $\tilde{x}$ that can be introduced by a transformation $T \in \mathcal{T}$.

Definition 3.2 *Given a set $\mathcal{S}$ of sequences, $\mathcal{T}$ a transformation language, $\mathcal{L}$ a description language and $\mathcal{D}_0()$ a base distance function, let $\vec{x} \in \mathcal{S}$ and $T \in \mathcal{T}$. We define the* transformed signature error *for $\vec{x}$ and T to be*

$$K_{x,T} \equiv \max\{\mathcal{D}_0(T(\vec{x}), T(\tilde{x})) \mid \tilde{x} \text{ is a canonical form of } \vec{x}\ \} \tag{4}$$

Let

$$K_x \equiv \max_{T \in \mathcal{T}}(\mathcal{D}_0(T(\vec{x}), T(\tilde{x})) \tag{5}$$

be the maximum distortion that any transformation can introduce between a sequence $\vec{x}$ and any of its canonical forms $\tilde{x}$. Finally, we maximize over all sequences in $\mathcal{S}$:

$$K \equiv \max_{\vec{x} \in \mathcal{S}}(K_x) \tag{6}$$

Theorem 3.1 *If the base distance measure $\mathcal{D}_0()$ satisfies the triangular inequality, then*

$$\mathcal{D}(\vec{x}, \vec{y}) \geq \mathcal{D}(\tilde{x}, \tilde{y}) - K_x - K_y \geq \mathcal{D}(\tilde{x}, \tilde{y}) - 2K \tag{7}$$

The proof is omitted.

In other words, one can find the distance between (canonical representations of) signatures and use it to bound the distance between the original sequences, even if the transforms to be applied in the matching process are different in the two cases.

An issue that now arises is that $T(\vec{x})$ might not be a canonical form, that is $T(\vec{x}) \notin \mathcal{C}_{\mathcal{L}}$. For example, if $\mathcal{L}$ keeps the first few DFT coefficients of the sequence $\vec{x}$, no canonical form $\tilde{x}$ can have high frequencies; thus, if we apply to a canonical form $\tilde{x}$ a transformation T that introduces high frequencies (*e.g.*, a regional add with a large "jump"), the result $T(\tilde{x})$ cannot possibly belong to $\mathcal{C}_{\mathcal{L}}$. It is desirable to find a related transform T', so that $T'(\vec{x}) \in \mathcal{C}_{\mathcal{L}}$, while not too far away from $T(\vec{x})$, and with cost similar to the cost of T. When this holds, we say the description language is *correspondence-bounded* with respect to the transformation language. More precisely:

Definition 3.3 *A description language $\mathcal{L}$ is said to be χ-correspondence-bounded with respect to a transformation language $\mathcal{T}$ if there is a constant χ such that for every pair of transformations T_1 and T_2 in $\mathcal{L}$ and for every pair of canonical sequences $\tilde{x}$ and $\tilde{y}$*

in $\mathcal{C}_\mathcal{L}$ there exist two other transformations T_1' and T_2' in $\mathcal{T}$ such that $T_1'(\tilde{x}), T_2'(\tilde{y})$ are canonical sequences and

$$\mathcal{D}_0(T_1(\tilde{x}), T_2(\tilde{y})) + cost(T_1) + cost(T_2) \leq \mathcal{D}_0(T_1'(\tilde{x}), T_2'(\tilde{y})) + cost(T_1') + cost(T_2') + \chi \tag{8}$$

The quantity χ is called the correspondence error bound.

Definition 3.4 *A description language $\mathcal{L}$ is said to be* closed *with respect to a transformation language $\mathcal{T}$ if for all $T \in \mathcal{T}$ and for all $w \in \mathcal{L}$ we have that $T(seq(w)) \in \mathcal{C}_\mathcal{L}$. That is, all transformations in $\mathcal{L}$ map canonical sequences to canonical sequences.*

So, we have bounds on the two sources of error: the error introduced by matching signatures instead of the original sequences, bounded by K, and the error introduced by using transformations that preserve canonical forms, instead of the transformations that we would use on the original strings; this one is bounded by χ.

In the rest of this extended abstract we consider only cases where χ is zero. The case of arbitrary χ is considered in the full version of the paper [6]. For lack of space we omit it here. This leaves us with two tasks — one is to compute the distance between (the canonical representations of) two signatures, by looking only at the signatures. The other is compute the bound K, for specified transformation and description languages. We pursue both in turn.

3.2 Match Effort

The reason to use signatures is that the comparisons of query and data can proceed rapidly – much faster than if the longer actual sequences were to be compared. Is this always true?

All that one can say in general is that it is asymptotically no more expensive to compute the distance between two sequences represented as signatures than to compute the distance between the original sequences themselves. The reason is that the complexity of obtaining the distance between two sequences is at least linear in the length of the sequences, since an additive distance function will at least require reading each point in the sequence once. The complexity of expanding a signature into a full sequence is also typically proportional to the length of the full sequence.

Of course, the whole point of using signatures is that these comparisons be significantly faster. Ideally, we would like comparisons to require time that is a function only of the length of the signature, independent of the length of the original sequence and of the canonical representation of the signature.

Definition 3.5 *We say that a description language $\mathcal{L}$, closed w.r.t. $\mathcal{T}$, is $\mathcal{T}$-*compare-polynomial *if the distance between (the canonical sequences of) two signatures can be computed in time polynomial in the length of the signatures, that is, for all w_x, w_y in $\mathcal{L}$, $D(\tilde{x}, \tilde{y})$ can be computed in time polynomial in the lengths of w_x and w_y.*

Note that, in the presence of transformations, the distance between two sequences may be hard to compute, even in the case of fully expanded sequences. For carefully chosen transformation languages, this computation can be done in polynomial time. Consequently, achieving polynomial time computation of the distance between two signatures is a good objective. We present such a case below. We need the following auxiliary definitions.

It is often the case that a signature w_x of a sequence $\vec{x}$ is a list of numbers $w_1 w_2 \ldots w_\lambda$. For example, a signature extraction algorithm would be to replace every 10 samples of $\vec{x}$ with their average: $w_1 = avg(x_1, x_2, \ldots, x_{10})$, $w_2 = avg(x_{11}, \ldots, x_{20})$ *etc.* The value of each w_i in this example depends only on 10 contiguous symbols of $\vec{x}$. In general, if every symbol w_i of the signature depends exclusively on a small subsequence of $\vec{x}$, then the description language $\mathcal{L}$ is called *modular*. The formal definition is as follows:

Definition 3.6 *A description $\mathcal{L}$ is said to be* modular *if there is a function $h_{\mathcal{L}}$ such that for every sequence $\vec{x}$ and every signature w_x of $\vec{x}$, there exist subsequences $\vec{x}_i$ of $\vec{x}$, with $\vec{x} = \vec{x}_1 \ldots \vec{x}_m$, such that for each symbol w_{xi} of w_x, $h_{\mathcal{L}}(\vec{x}_i) = w_{xi}$. We define the* module bound μ *to be the length of the longest such substring of $\vec{x}$.*

Definition 3.7 *A transformation language $\mathcal{T}$ is said to be* local *if for any two sequences $\vec{s}$ and $\vec{s}'$ that agree on the i-th symbol, $T(\vec{s})$ and $T(\vec{s}')$ also agree on the i-th symbol. That is, the value of $T(\vec{s})_i$ only depends on the value of $\vec{s}_i$. Furthermore, the cost $cost(T)$ is the sum of the costs of all transformations T_i that transform the i-th symbol of their input as T does and leave the rest of the input unchanged.*

We are now ready to present the theorem. The proof (omitted) relies on the ability to compute the distance between pairs of (transformed) subsequences represented in the modular description language in time independent of sequence length, and then uses dynamic programming to deal with overlaps of subsequences and constraints on transformations.

Theorem 3.2 *If $\mathcal{T}$ is a local transformation language, and $\mathcal{L}$ is a modular description language closed w.r.t. $\mathcal{T}$, then $\mathcal{L}$ is $\mathcal{T}$-compare polynomial.*

A special case of particular interest is when the transformation language specified is empty. In this case, we use the name *simple*-compare-linear/polynomial/exponential, *etc.* For example, a Fourier series description of a sequence, with an inverse Fourier transform as the signature inverting function, is simple-compare-linear for the Euclidean (L_2) distance measure (because "energy" is preserved in the transform domain), but is not simple-compare-polynomial for other distance measures. A piecewise linear description of a sequence, with a zero order or first order interpolation as the inverse, is simple-compare-linear for all L_p distance measures. In fact we can show the following:

Theorem 3.3 *Every modular description language is simple-compare-linear.*

3.3 Finding the Bounds

On the basis of the previous section, the basic question to ask now is how do we find the bound K. (Recall that we foccus in this extended abstract on the case were $\chi = 0$). We show in this section that for selected classes of transformations and description languages, such bound K can indeed be found.

It is often the case that a transform T cannot amplify an existing difference too much. For example, it may be the case that, if two sequences $\vec{x}$ and $\vec{y}$ differ by δ, *any* transform $T \in \mathcal{T}$ might amplify this difference by a predictable amount. Formally, for a given transformation language $\mathcal{T}$, let $f()$ be a function such that if

$$\mathcal{D}_0(\vec{x}, \vec{y}) \;=\; \delta \tag{9}$$

then

$$\mathcal{D}_0(T(\vec{x}), T(\vec{y})) \;\leq\; f(\delta) \;\;\forall\, T \in \mathcal{T} \tag{10}$$

It is easy to see the following.

Theorem 3.4 *Let $f()$ be a function such that for every two sequences $\vec{x}, \vec{y}$ and every transformation $T \in \mathcal{T}$, $\mathcal{D}_0(T(\vec{x}), T(\vec{y})) \leq f(\mathcal{D}_0(\vec{x}, \vec{y}))$. Then the transformed signature error bound is*

$$K = f(\epsilon) \tag{11}$$

In particular, if $f()$ is the identity function, that is, the base distance is invariant under the same transformation, then $K = \epsilon$. This is the case for all *add* (to y axis) transformations and L_p distance measures. On the other hand, for a uniform scaling transformation, $f()$ is clearly the scaling factor.

4 A Comprehensive Example

To place all the concepts of the preceding sections in perspective, we work through an example in this section. We consider a transformation language that allows "Regional Adds". In other words, we permit the sequence level to shift abruptly. There is a cost $CostOfTransform$ associated with each such shift in level. Such distance functions with regional adds have been used in the past [27]; other distance functions go even further, including time-shifts, scaling *etc.* [2]. Note that no straightforward base distance functions can accommodate such changes. Therefore most of the currently published retrieval techniques cannot be used effectively.

Such "regional adds" often occur in sequences as a result of sudden changes in environment or other catastrophes. One is often interested in finding sequences that are similar, modulo a few such level shifts. For example, consider companies X and Y, whose stocks move similarly because the companies belong to the same market segment. Suppose that something unexpected happens to company X only (*e.g.*, it wins a major contract) - this unexpected change boosts the stock price by, say, Δ. If we could factor-out this "catastrophe", the two stock prices would look very similar. Based on this example, we show how our approach works.

4.1 Problem definition - our input

Suppose that a domain expert, trying to take these "catastrophe" events into account, furnishes us with the following distance function $\mathcal{D}()$: For two sequences $\vec{x}$ and $\vec{y}$, their (squared) distance is the sum of squared errors plus the cost of "catastrophes", *after* the optimal number of "catastrophes" has been placed at the optimal points, with cost Δ^2 for each "catastrophe" of magnitude Δ.

This is the only input to us – it is up to us to decide how to bring this problem within our framework, which description language $\mathcal{L}$ to choose, how to obtain signatures, and which signature-to-sequence function $seq()$ to choose. However, if we manage to do all that, we will have (a) a potentially fast access method ("shrink-and-search") and (b) the guarantee that our method will not have false dismissals.

4.2 Customization of our framework

Transformation language To match the given function, we only need the "regional add" R transformation:
Such a transformation $R < p1, p2, \Delta > (\vec{x})$, gives $< x_1, x_2, ..., x_{p1} + \Delta, x_{p1+1} + \Delta, ..., x_{p2-1} + \Delta, x_{p2}, x_{p2+1} ... >$.

Thus, our set of atomic transformations is $\mathcal{T}_0 = \{R < i, j, \Delta > | 1 \leq i \leq j \leq n; \Delta \in \Re\}$.

According to the specification of the problem, the cost is $cost(R < p1, p2, \Delta >) = (\Delta)^2$

Formalization of the Distance function The distance function $\mathcal{D}()$ can be defined recursively as follows:

$$\mathcal{D}^2(\vec{x}, \vec{y}) \;=\; \min_{p_e \geq 1, \Delta \in \Re} \left(CostOfTransform \;+\; \sum_{i=1}^{p_e} (x_i + \Delta - y_i)^2 \;+\; \mathcal{D}^2(x_{p_e+1:n}, y_{p_e+1:n}) \right) \quad (12)$$

Description language We choose as the description language $\mathcal{L}$ the list of pairs (value, duration) or $(v,\ d)$, for short. Thus, a valid word w in this language would be $w = \{(3.5, 2), (5.25, 4)\}$.

As the $seq(w)$ function, that operates on a word w and generates the canonical representation, we select piece-wise constant interpolation. Thus, for each pair $(v,\ d)$ of the word w, $seq()$ will "stutter" d times the value v. For example, the canonical representation of the previous example word would be

$$\begin{aligned} seq(w) &= seq(\{(3.5, 2), (5.25, 4)\}) \\ &= \{3.5, 3.5, 5.25, 5.25, 5.25, 5.25\} \end{aligned}$$

Transformations Over Signatures Before we can compute distances between signatures, we have to agree upon what the equivalent of our transformation is in the signature domain. If we choose to have steps permitted to take place at any arbitrary position, including points in the middle of substrings represented by single symbols in the signature, then the resulting transformed sequence will have a different number of piecewise constant regions, and will not be a canonical sequence of the λ-signature language selected as our description language $\mathcal{L}$.

Instead, we require steps to be added only at substring boundaries. With this restriction to the transformation language, our description language is closed w.r.t. the transformation language.

Bounding Error and Match Effort It is not hard to show [6] that the correspondence error χ introduced here is 0. (Omitted here for lack of space) Our framework also satisfies the requirements of Thm 3.4. Therefore the transformed signature error bound, K, is simply ϵ.

Since χ is zero, and K is ϵ, performing the match with signatures is no worse than performing the match with the canonical sequences represented by the signatures.

The description language we have here is modular, the transformation language is local, and the description language is closed with respect to the (restricted) transformation language. Therefore, all conditions of Thm 3.2 are satisfied. Thus we know that we can match in the signature domain, in time that is polynomial in signature length.

4.3 The Steps To Follow

Signature Extraction We choose to work with fixed-length signatures of length λ, that is λ-signatures. With the above choices of $\mathcal{L}$ and $seq()$, the problem of finding the λ-signature of a sequence $\vec{x}$ is the classic problem of piece-wise constant sequence approximation, constrained on the number of segments λ, where the cost function is the sum of squared errors. The solution is based on Dynamic Programming; the details are omitted for brevity.

We can show that the running time of the algorithm is $O(n^2 \times \lambda)$.

Comparing Full Sequences A dynamic programming algorithm suggests itself, and indeed one has been proposed for a variant of this problem in [27]. Eq. 12 is already in a recursive form - the dynamic programming works as follows:

There is one "stage" in the algorithm for each discrete sample point. At each stage, a step has to be taken, and its duration p_e has to be selected. No additional steps are permitted until the end of this duration. The magnitude of the step can then be obtained by solving a very simple optimization: for our setting (the Euclidean distance and the cost of a step being Δ^2), we have to maximize a quadratic polynomial on Δ. Further transitions then compute the distance between sequences, taking into account the chosen step.

Thus, there are n stages in a dynamic programming algorithm to compute the distance between two sequences of length n. At each stage, there are $O(n)$ possible states. Computing the cost of transition (based on optimization performed for step size) is $O(1)$, by careful book-keeping: we can keep the partial sum $\sum_{i=1}^{p_e-1}$ from the previous step, and update it in constant time. Multiplying these, this algorithm has complexity $O(n^2)$.

Computing Distance Between Signatures For this specific case, one can work through the details of the dynamic programming formulation to show that the time required is actually $O({\lambda_2}^2)$, where λ_2 is the sum of the lengths of the query and data signatures.

5 Another Example

Consider uniform scaling as the similarity transform of interest. That is $\mathcal{T} = \{T_a | a \in \Re\}$ where $T_a(\vec{x}) = a\vec{x}$. That is, a constant scale factor a can be used to multiply all observations in one sequence to better make it match the other. Simple calculus will show the desired scale factor a to be $\sum_i x_i y_i / \sum_i x_i^2$. The cost $cost(T_a)$ is determined by the application: it could be zero, or a^2, or $|\log |a||$.

Consider a description language $\mathcal{L}$, obtained as the Fourier coefficients of the original sequence. A short signature is obtained by discarding the higher frequency coefficients and retaining only the first few. The canonical representation of the signature is obtained by applying the inverse Fourier transform to the coefficients retained, assuming that the remaining coefficients are all zero. Obtaining the signature of a sequence is easy – standard Fourier transform techniques are used.

The error in the signature is the sum of squares of the truncated coefficients. This error can be bounded by retaining enough coefficients.

The equivalent transformation in the signature domain is also multiplication by the same scaling factor as in the original sequence domain. The optimum scaling factor can be computed in the signature domain using the same calculus formulation as in the original domain. The error is bounded by the product of the error before scaling and the scaling coefficient. Therefore $K = scaling \times \epsilon$. Since the transformations in the signature domain are identical to those in the full sequence domain, we know that the correspondence error is identically zero.

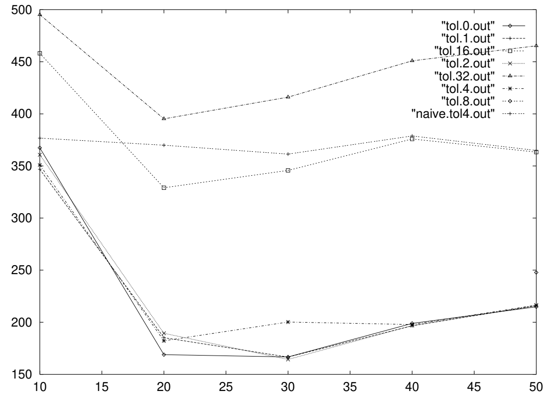
This is not a modular description language, yet signature matching is possible in polynomial time, through a parametric optimization.

6 Experimental Verification

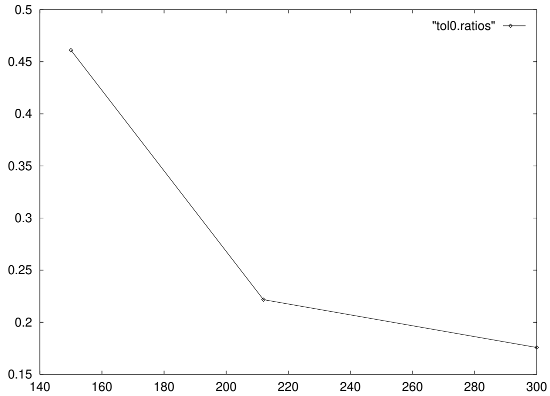
We implemented our searching method on a database of stock price movements, from `ftp://ftp.ai.mit.edu/pub/stocks/results`. We used 7 stocks; for each stock, we took its first n days (n=150-300), and considered the closing prices only. As queries we used the very same stocks.

We used λ-signatures. and implemented the searching algorithms in `nawk`. We always '`diff`'ed the output of the two methods, to verify that there were no false dismissals.

Our first set of experiments was to try to find a good value for λ. Figure 2(a) shows the results: It gives the logarithm of the response time of our method, as a function of the λ parameter, for several values of the tolerance ϵ, for n=150 days-long stocks. We varied λ from 10-50. For a careful choice of λ, the proposed method achieves 50% savings in response time. These savings increase with the length n of the sequences. Notice that (a) the higher the tolerance, the less useful the signatures, as expected and (b) for small tolerances, the optimal value of λ is approximately 30. We conjecture that the optimal value of λ will be related to the square-root of the length n. We also plot the response time of the 'naive' method, which was around 370 seconds. The specific plot corresponds to ϵ=4. As expected, it is roughly constant, the λ and the tolerance ϵ have no effect on its strategy.



(a)



(b)

Figure 2: (a) sum of response times in seconds, versus λ, for tolerance ϵ=0,1,2,4,8,16,32 (b) ratio of response time (ours over naive) vs sequence length n

Figure 2(b) examines how the speedup will scale for longer sequences. It gives the fraction of response times (our method over the naive) as a function of the sequence length n. We have chosen the tolerance to be ϵ=0 and λ to be 30, 40 and 50, respectively, for n=150, 212 and 300. Notice that the gains of our method increase drastically with the sequence length.

7 Related Work

7.1 Approximate Matching

Approximate matching for numerical time sequences ('signals') include the work on voice matching (see [20] for a textbook), where time-warping is considered. When the distance is the Euclidean metric, we have proposed an indexing method using the first few Discrete Fourier Transform (DFT) coefficients, for matching full sequences [1], as well as for sub-pattern matching [7]. This technique has been extended by Goldin and Kanellakis [11] for matching time sequences, so that it allows for shifts and scalings.

Approximate matching of signals in general are discussed in [30], [28], with a recent survey in [4]. There, the idea is to allow some elastic deformations (ie., space warpings), before matching the two signals. Signals can be, eg., 2-d gray-scale images, or 3-d MRI brain scans.

Closely related is the work on string matching. An excellent starting point is the book by Sankoff and Kruskal [25], which examines strings, signals and DNA molecules, along with popular distance functions. The survey by Hall and Dowling [14] examines matching of typed English strings, along with the basic, dynamic programming algorithm, that computes the editing distance. The book by Frakes and Baeza-Yates [9] examines Information Retrieval applications, including approximate matching there.

7.2 Distance Metrics

We list some popular distance functions. They are all encompassed within our framework, and they use zero or more of the following transformations. The transformations accept a sequence $\vec{s}$ as input and return another sequence. They also take some parameters, within angle-brackets ($<>$)

- $drop < p > (\vec{s})$: drops $\vec{s}[p]$ and shifts the elements left, to close the gap.
- $stutter < p > (\vec{s})$: repeats $\vec{s}[p]$ once, and shifts the elements to the right.

L_p Metrics and Euclidean distance: For two sequences $\mathbf{x} = x_1 \ldots x_n$, $\mathbf{y} = y_1, \ldots y_n$ this distance is defined by the formula

$$\mathcal{D}^p(\mathbf{x},\ \mathbf{y}) = \sum_{i=1 \ldots n} |x[i] - y[i]|^p \tag{13}$$

For $p = 1$ the L_p metric reduces to the 'Manhattan' or 'city-block' distance; for $p = 2$ it becomes the popular Euclidean distance.

Editing distance in strings: This is the minimum number of insertions, deletions and substitutions that are needed to transform a string s into another string t [25, 18].

$$\mathcal{D}(s,t) = \min \begin{cases} cost(Del(t[1])) + \mathcal{D}(s, Rest(t)) \\ cost(Del(s[1])) + \mathcal{D}(Rest(s), t) \\ cost(Sub(s[1], t[1])) + \mathcal{D}(Rest(s), Rest(t)) \end{cases} \tag{14}$$

where $Del(t[1])$ ($Del(s[1])$) stands for deleting the first character of t (s), and ($Sub(s[1], t[1])$) for substituting the first character of s by the first character of t, $cost$ is the cost of the deletion/substitution, and $Rest(t)$ ($Rest(s)$) is the string t (s) without its first character.

As shown in Table 1, our framework includes the string editing distance, by choosing: $\mathcal{T}_0$ to have only one transformation, the $drop < p >$ transformation, with cost=1 and by setting $\mathcal{D}_0()$ to be the Hamming distance.

operator	Euclidean	string-edit	time-warping
$drop < p >$	∞	1	∞
$stutter < p >$	∞	∞	0
$\mathcal{D}_0$	Euclidean	Hamming	city-block

Table 1: Cost of operators of our framework, for popular distance functions

The distance can be computed in time $O(N_x N_y)$, where N_x, N_y are the number of samples in each string [14], [18].

Distance functions with time-warping: Such functions are used for example in digitized voice signals, where there are fluctuations in the rate of speech.

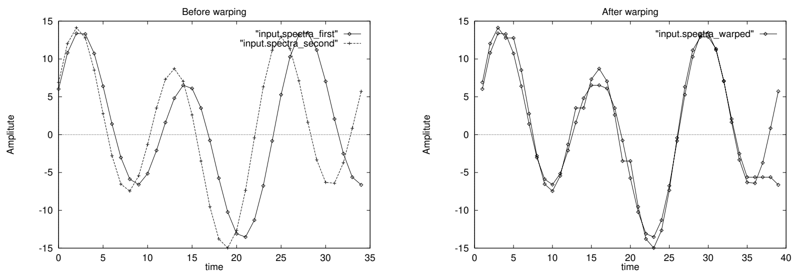
A typical distance function is [20]:

$$\mathcal{D}(x, y) = \mathcal{D}_0(Head(x), Head(y)) + \min \begin{cases} \mathcal{D}(x, Rest(y)) & /* \; x - stutter \; */ \\ \mathcal{D}(Rest(x), y) & /* \; y - stutter \; */ \\ \mathcal{D}(Rest(x), Rest(y)) & /* \; no \; stutter \; */ \end{cases} \tag{15}$$

where $Head(x)$ returns the first element of x, and $Rest(x)$ returns the remainder.

Table 1 shows that the above distance is a special case of our framework, by setting (a) the 'basic' transformation language $\mathcal{T}_0$ to consist of only the $stutter$ transformation, with cost $= 0$ and (b) the $\mathcal{D}_0()$ distance function to be the L_1 metric, that is, the city-block distance.

Figure 3 shows two time sequences, before and after the time-warping. The sequences are mixtures of similar harmonics: $x(t) = 10\sin(0.5t) + 5sin(.25t)$ and $y(t) = 11\sin(.55t) + 4.5\sin(.26t)$ respectively.



(a) before (b) after

Figure 3: Illustration of two similar sequences, before and after time-warping

7.3 Alternative Signature languages

In our main example, we used sub-sampling with piece-wise constant interpolation to depict a sequence. Additional signature languages include any piece-wise polynomial functions (see Sidiropoulos [27] for a survey of optimal algorithms to achieve such approximations), as well as techniques from Digital Signal Processing (DSP) [19], for optimal function approximation. The most popular techniques from there include the Discrete Fourier Transform (DFT), the Discrete Cosine Transform (DCT) (which is the basis of the JPEG image compression standard [29]), and, recently, the very promising Discrete Wavelet Transform (DWT) [23, 24].

The DWT is, in principle, a 'short window' Fourier transform; the major difference is that the length of the window takes several values (typically, powers of 2). Thus, it leads to multi-resolution analysis, which seems to be promising for real signals.

Natural signals seem to require few wavelet coefficients to be described with small error [8]; this is exactly the reason that a wavelet decomposition is useful for compression, feature extraction and searching. Figure 4 shows some of the basis functions for the very well-known Daubechies-4 DWT. The basis functions are translations or dilations of each other: Eg., #5 is a dilation of #9, which is a dilaton of #17 *etc.*

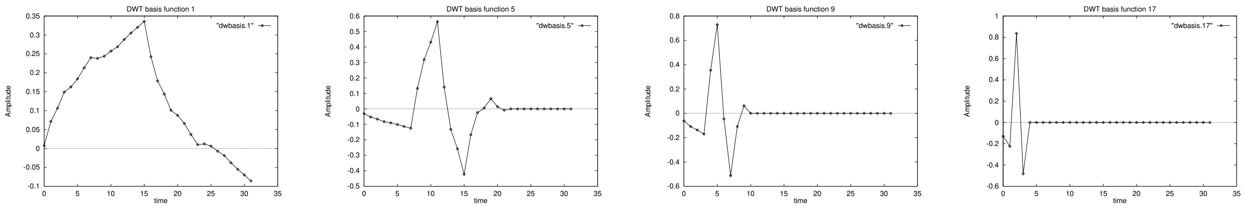


Figure 4: DWT basis functions (1, 5, 9, 17)

8 Conclusion

In this paper we described a generic signature-based technique that can be used effectively for retrieval based on many different, application-specific, notions of similarity. For a variety of general conditions, we obtained measures of goodness for our technique. We illustrated our technique with a couple of very different examples.

While the work in this paper focused on sequence data, we believe that the basic framework developed here is equally applicable to other contexts such as image, video, or text data.

References

[1] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. Efficient similarity search in sequence databases. In *Fourth Int. Conf. on Foundations of Data Organization and Algorithms (FODO)*, pages 69–84, Evanston, Illinois, October 1993. also available through anonymous ftp, from olympos.cs.umd.edu: ftp/pub/TechReports/fodo.ps.

[2] Rakesh Agrawal, King-Ip Lin, Harpreet S. Sawney, and Kyuseok Shim. Fast similarity search in the presence of noise, scaling and translation in time-series databases. *Proc. of VLDB*, pages 490–501, September 1995.

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

[4] Lisa Gottesfeld Brown. A survey of image registration techniques. *ACM Computing Surveys*, 24(4):325–376, December 1992.

[5] L.S. Colby, E.L. Robertson, L.V. Saxton, and D. Van Gucht. A query language for list based complex-objects. *Proc. 13th ACM Symp. on Principles of Database Systems*, pages 179–189, May 1994.

[6] Christos Faloutsos, H.V. Jagadish, Alberto O. Mendelzon, and Tova Milo. Signature technique for similarity-based queries. Technical Report 112530-951110-16TM, AT&T Bell Labs, Murray Hill, NJ, November 1995.

[7] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. *Proc. ACM SIGMOD*, pages 419–429, May 1994. 'Best Paper' award; also available as CS-TR-3190, UMIACS-TR-93-131, ISR TR-93-86.

[8] D.J. Field. Scale-invariance and self-similar 'wavelet' transforms: an analysis fo natural scenes and mammalian visual systems. In M. Farge, J.C.R. Hunt, and J.C. Vassilicos, editors, *Wavelets, Fractals, and Fourier Transforms*, pages 151–193. Clarendon Press, Oxford, 1993.

[9] W. Frakes and R. Baeza-Yates. *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall, 1992.

[10] S. Ginsburg and X. Wang. Towards a unified approach to querying sequenced data. *Proc. 11th ACM Symp. on Principles of Database Systems*, pages 293–300, 1992.

[11] Dina Q. Goldin and Paris C. Kanellakis. On similarity queries for time-series data: Constraint specification and implementation. *Int. Conf. on Principles and Practice of Constraint Programming (CP95)*, September 1995.

[12] G.H Gonnet, M.A. Cohen, and S.A. Benner. Exhaustive matching of the entire protein sequence database. *Science*, 256(5), June 1992.

[13] S. Grumbach and T. Milo. An algebra for pomsets. *Proc. of ICDT*, 1995.

[14] P.A.V. Hall and G.R. Dowling. Approximate string matching. *ACM Computing Surveys*, 12(4):381–402, December 1980.

[15] H.V. Jagadish. A retrieval technique for similar shapes. *Proc. ACM SIGMOD Conf.*, pages 208–217, May 1991.

[16] H.V. Jagadish, Alberto O. Mendelzon, and Tova Milo. Similarity-based queries. *Proc. ACM SIGACT-SIGMOD-SIGART PODS*, pages 36–45, May 1995.

[17] M. Li and P.M.B. Vitányi. Kolmogorov complexity and its applications. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A*, pages 188–254. MIT Press/Elsevier, 1990.

[18] R. Lowerance and R.A. Wagner. An extension of the string-to-string correction problem. *JACM*, 22(2):3–14, April 1975.

[19] Alan Victor Oppenheim and Ronald W. Schafer. *Digital Signal Processing.* Prentice-Hall, Englewood Cliffs, N.J., 1975.

[20] Lawrence Rabiner and Biing-Hwang Juang. *Fundamentals of Speech Recognition.* Prentice Hall, 1993.

[21] Davood Rafiei and Alberto O. Mendelzon. Similarity-based queries for time series data. *to appear in Proc. ACM SIGMOD*, 1997.

[22] J. Richardson. Supporting lists in a data model (a timely approach). *Proc. 18th Intl. Conf. on Very Large Databases*, August 1992.

[23] Oliver Rioul and Martin Vetterli. Wavelets and signal processing. *IEEE SP Magazine*, pages 14–38, October 1991.

[24] Mary Beth Ruskai, Gregory Beylkin, Ronald Coifman, Ingrid Daubechies, Stephane Mallat, Yves Meyer, and Louise Raphael. *Wavelets and Their Applications.* Jones and Bartlett Publishers, Boston, MA, 1992.

[25] David Sankoff and Joseph B. Kruskal. *Time Warps, String Edits and Macromolecules: the Theory and Practice of Sequence Comparisons.* Addison-Wesley Publishing Company, Inc., Reading, MA, 1983.

[26] Praveen Seshadri, Miron Livny, and Raghu Ramakrishnan. Sequence query processing. *Proc. Int. Conf. on Management of Data SIGMOD*, pages 430–441, May 1994.

[27] Nikolaos Sidiropoulos. The viterbi optimal runlength-constrained approximation nonlinear filter. *IEEE Trans. on Signal Processing*, 1996. to appear.

[28] Demetri Terzopoulos. Image analysis using multigrid relaxation methods. *IEEE PAMI*, 8(2):129–139, March 1986.

[29] Gregory K. Wallace. The jpeg still picture compression standard. *CACM*, 34(4):31–44, April 1991.

[30] A. Witkin, D. Terzopoulos, and M. Kaas. Signal matching through scale space. *Proc. am. Assoc. Artif. Intel.*, pages 714–719, 1986.

[31] Sun Wu and Udi Manber. Text searching allowing errors. *Comm. of ACM (CACM)*, 35(10):83–91, October 1992.