Efficient Subsequence Matching in Time Series Databases Under Time and Amplitude Transformations

Tassos Argyros
School of Electrical and Computer Engineering
National Technical University of Athens
Athens, Greece
targyros@cs.ntua.gr

Charis Ermopoulos
School of Electrical and Computer Engineering
National Technical University of Athens
Athens, Greece
ermop@cs.ntua.gr

ABSTRACT

Subsequence matching in large time series databases has attracted a lot of interest and many methods have been proposed that cope with this problem in an adequate extend. However, locating subsequence matches of arbitrary length, under time and amplitude transformations, has received far less attention and is still an open problem. The importance of this problem lies in the fact that many time series exhibit self-similarity, i.e., are invariant under time and amplitude rescaling. In fact, it is now known that a truly enormous and varied collection of man-made and natural phenomena exhibit self-similarity, including financial, geophysical and human speech data. The proposed algorithm is ideal for analyzing large self-similar time sequences as the ones mentioned, while relevant existing methods are either computationally inefficient or use approximation techniques that allow false dismissals. In this paper we first motivate and define the above problem. We present an efficient algorithm for variable-length subsequence matching under transformations and we provide a proof that our algorithm has no false dismissals. Further, this algorithm uses a novel similarity criterion for determining similarity under amplitude transformations in a most efficient way. Finally, our algorithm has been tested in various experiments on real data, resulting in a running time improvement of one order of magnitude compared to the naive approach.

1. INTRODUCTION

The problem of efficiently locating matches of a given time sequence in a large temporal database has recently attracted a lot of interest in the research community and many methods have been proposed that cope with this problem in an adequate extend [1, 8, 19, 31, 16]. However, most of the proposed methods locate matches of the same length. Finding matches under time scaling of a given time sequence within a large database, efficiently and with no false dismissals, is still an open problem.

The specific problem is very important, since it is profoundly related with self-similarity in time series. The term self-similarity was coined by Mandelbrot [23] and is now standard. In a nutshell, self-similarity simply means reproducing oneself on different time scales. Moreover, self-similarity is closely related to power-law distributions [29, 23]. It is now known that a widely extended range of man-made and natural phenomena exhibit self-similarity and power-law behavior. Some examples include financial time series [24], geophysical data [29], network traffic data [22] and human speech signals [25]. The proposed algorithm is ideal for analyzing large self-similar time sequences as the ones mentioned, while most of the relevant existing methods are either computationally inefficient or use approximation methods that allow false dismissals. Therefore, such methods are inappropriate for demanding applications where reliability and efficiency are of equal importance.

In this paper we present a method that can locate any scaled occurrence (under scaling in both time and amplitude axes) of a query sequence in a large time series database; in fact, the method guarantees no false dismissals. An example of a relevant case is shown in Figure 1, where we present a 30-point query subsequence (shown in its context for clarity) along with a 60-point match. Notice how similar they are both shifted and scaled, while, initially, their time and amplitude scales and offsets differ.

Figure 1: A Dow Jones index match, under different scales and offsets: (a) represents a 30-point query subsequence (shown in its context for clarity) and (b) a 60-point match. In (c) both are shown scaled and shifted.

We attack the problem in two directions. First, we need to perform efficient similarity checks under amplitude shifting
and scaling. Concerning this problem, instead of defining and applying a similarity measure, we propose a similarity criterion. We demonstrate why this criterion is (in quality) better than simply comparing normalized sequences. Furthermore, the criterion we propose is very efficient in sequential subsequence matching, since we show that when performing “sliding window” or “growing window” similarity checks, its time complexity is nearly constant (independent on the size of the window).

The other problem is time scaling. From an intuitive point of view, in order to scale (compress\(^3\)) a time sequence we group sequential points of the initial time sequence and use their averages in order to form a scaled (smaller) sequence. This approach has important advantages over other methods, mainly because it is robust to small scaling variations and to noise. Additionally, it is very close to the human intuitive notion of scaling.

Note that the inherent complexity of the problem we deal with is rather high. Consider the case where we are interested in finding all scaled occurrences of size between \(w_{\text{min}}\) and \(w_{\text{max}}\) (that is windows of \(l = w_{\text{max}} - w_{\text{min}} + 1\) different sizes) in a database of \(n\) points. The naive approach has to check \(O(l \cdot n)\) cases (given that \(n \gg l\)), let alone the time complexity of determining whether each window is a match or not. For example, in real applications (similar to the experiments conducted in this paper) we may have \(n = 25,000\) and \(l = 1000\), which results in approximately \(24,500,000\) different candidate windows that need to be compared with the query sequence. In order to cope with such high complexity, we present a method that performs a pruning on the sizes of the windows that have to be checked, resulting not only in checking much fewer windows but also those who have to be checked are the smallest in size.

All the above result in an algorithm that is efficient (outperforms the naive approach for over an order of magnitude), reliable (we provide a proof that it has no false dismissals) and its applications extend in a broad range of scientific and financial areas.

2. SIMILARITY UNDER V-AXIS TRANSFORMATIONS

As mentioned previously, in order to decide on the similarity of time series that appear under different scale and offset in both axes, we need to define time series similarity in a more flexible manner. Such flexibility can be achieved by allowing transformations on the amplitude axis (“V-axis”) and on the time axis. This section introduces similarity of time series under V-axis transformations. We first give the appropriate definitions and we then introduce a criterion that is computationally efficient for determining whether two time series are similar under such transformations.

2.1 Definitions

We start by defining time series and introducing some notational conventions.

**Definition 1.** A time sequence \(T\) of length \(n\) is an or-dered set \((t_1, t_2, \ldots, t_n)\) with \(t_i \in \mathbb{R}, \ 1 \leq i \leq n\). Concerning the notation, we denote:

- \(T[1 \ldots n]\) the sequence \(T\) of length \(n\).
- \(T[i]\) the \(i\)-th position of the sequence \(T\), \(t_i\).
- \(T[i \ldots j]\) the subsequence of \(T\) that is composed of the entries of \(T\) from position \(i\) to position \(j\) (including), i.e. the ordered set \((t_i, \ldots, t_j)\). If the subsequence includes the first point of the initial sequence, i.e. \(T[1 \ldots j]\), then we call it a prefix subsequence.
- \([T]\) the length of \(T\).

When examining time series, we often observe that while two time series exhibit similar behavior, their offsets (i.e. their average values) are very different. This fact is widely observed in stock analysis \[2\]. It is obvious that we are interested in such similarity, and thus we must allow the vertical offset of a time series to vary. We therefore need to define the V-Shift transformation as follows.

**Definition 2.** Given a time series \(T[1 \ldots n]\), a V-Shift transformation of \(T\) is a time series \(T'[1 \ldots n]\) satisfying

\[
T'[i] = T[i] + d \quad (i = 1 \ldots n)
\]

where \(d\) is a real constant.

Furthermore, in the same manner, we are interested in the similarity of two time series that appear under different vertical scale. In the stock market example, it is a fact that, in the same manner that the variation of a stock price can be similar in different offsets, it can also be similar in different vertical scaling \[28\]. Here we define V-scale transformation, which alters the amplitude scaling of a time sequence.

**Definition 3.** Given a time series \(T[1 \ldots n]\), a V-Scale transformation of \(T\) is a time series \(T'[1 \ldots n]\) satisfying

\[
T'[i] = c \cdot T[i] \quad (i = 1 \ldots n)
\]

where \(c\) is a real constant.

In the above definition, we allow \(c\) to take negative values. The reason for this is that, apart from locating similar time sequences that behave similarly but in different scale, we are also interested in time series that behave in the opposite way (hedging). The importance of opposite movement in time series, and specifically in a stock price movement, is also noted in \[28\].

The above transformations enable us to create a more flexible similarity criterion. In our approach we will expand the Euclidean distance (\(L_2\) norm)—a popular similarity measure that its variations have been extensively used in comparing time sequences \[1, 8, 4, 31, 26\]. We define here the “Euclidean similarity” which is based on the Euclidean distance.
DEFINITION 4. Given two time series, $T[1 \ldots n]$ and $Q[1 \ldots n]$, and a positive real constant $\epsilon$, $T$ and $Q$ are similar under the euclidean similarity if
\[
\sum_{i=1}^{n} (T[i] - Q[i])^2 < \epsilon.
\]

We now want to create a similarity definition that includes the V-Scale and V-Shift transformations; that is, we would like to consider a time series similar to another if there exists any combination of the above transformations that when applied to the first time series can bring them “close enough” to each other (according to the Euclidean distance measure). The following definition of $\epsilon$-similarity fulfills the above requirements.

DEFINITION 5. Given two time series, $T[1 \ldots n]$ and $Q[1 \ldots n]$ and a positive real constant $\epsilon$, we define $T$ to be $\epsilon$-similar to $Q$ if there exist real constants $c$ and $d$ such that
\[
\sum_{i=1}^{n} (c \cdot T[i] + d - Q[i])^2 < \epsilon.
\]

In other words, $T$ is $\epsilon$-similar to $Q$ if there exist $c$, $d$ such that if we multiply (scale) $T$ by $c$ and add (shift by) $d$, $T$ will come $\epsilon$-close (according to the Euclidean similarity) to $Q$. In our implementation, we normalize\(^2\) the one of the two sequences (in particular the query sequence) and then try to determine if any other sequence is $\epsilon$-similar to the normalized $Q$.

An important observation (also mentioned in [13]) is that, having $Q$ normalized, normalizing $T$ does not result in an optimal combination of V-shift and V-scale transformations in $T$ (such that the Euclidean distance of $Q$ and $T$ is minimized). This means that the common approach of simply normalizing all sequences (and then determining their Euclidean distance) may result in missing some matches. A characteristic example is shown in Figure 2.

Figure 2: An example where normalization does not result in the minimum distance. The one time sequence is pre-normalized. When the other is scaled and shifted appropriately (a) it yields a distance of 0.422, while normalized (b) results in a larger distance of 0.519.

\(^2\)Normalizing a sequence equals to offsetting by the mean and dividing (amplitude scaling) by its variance.

Note that $\epsilon$-similarity does not hold the symmetrical property. We could make $\epsilon$-similarity symmetrical by altering the above definition in the way that two sequences $T[1 \ldots n]$, $Q[1 \ldots n]$ would be $\epsilon$-similar (symmetrically) if either $T$ is $\epsilon$-similar to $Q$ or $Q$ is $\epsilon$-similar to $T$. However, as we were able to verify through our experiments, the practical applications of $\epsilon$-similarity do not require such symmetry.

A simple, yet important, property that $\epsilon$-similarity holds is described by the following lemma. This property will be useful in the analysis of the algorithm in section 3.2. Simply put, the following lemma states that if two sequences are $\epsilon$-similar, then all their prefix subsequences of equal length are also $\epsilon$-similar.

**Lemma 2.1.** If a sequence $T[1 \ldots n]$ is $\epsilon$-similar to $Q[1 \ldots n]$ then for any $1 \leq k \leq n$, $T[1 \ldots k]$ is $\epsilon$-similar to $Q[1 \ldots k]$.

**Proof.** By contradiction. If for a particular $k$, $1 \leq k \leq n$, $T[1 \ldots k]$ is not $\epsilon$-similar to $Q[1 \ldots k]$, then (for any $c$, $d$) it holds
\[
\sum_{i=1}^{k} (c \cdot T[i] + d - Q[i])^2 \geq \epsilon.
\]

However,
\[
\sum_{i=1}^{n} (c \cdot T[i] + d - Q[i])^2 \geq \sum_{i=1}^{k} (c \cdot T[i] + d - Q[i])^2,
\]
and therefore for any $c$, $d$
\[
\sum_{i=1}^{n} (c \cdot T[i] + d - Q[i])^2 \geq \epsilon,
\]
a contradiction, since we assumed that $T[1 \ldots n]$ is $\epsilon$-similar to $Q[1 \ldots n]$. \(\square\)

It is obvious, however, that without an efficient way to determine $\epsilon$-similarity, the above definitions would be of limited use. The following section deals with this subject.

### 2.2 Efficient decision of $\epsilon$-similarity— the CD-criterion

In this section we present an efficient method for determining whether two time series are $\epsilon$-similar. For this purpose, we propose the **CD-criterion** which enables us to decide (without error) if a time series is $\epsilon$-similar to another—that is, similar under V-scale and V-shift transformations. In order to present how the method works, consider that we are given two time series $T[1 \ldots n]$ and $Q[1 \ldots n]$ and that we would like to determine whether $T$ is $\epsilon$-similar to $Q$ or not\(^3\).

By definition, the above question equals to
\[
\text{Given } \epsilon > 0, \text{ find if there exist real constants } c, d \text{ such that }
\sum_{i=1}^{n} (c \cdot T[i] + d - Q[i])^2 < \epsilon.
\]

\(^3\)As explained previously, we consider $Q$ to be normalized.
The above problem is computationally hard to solve in the case that we are interested in the values of $c$, $d$ that force the inequality to hold—or we need to know the exact minimum distance. However, in the manner that we formed the above problem, the only thing needed to be examined is the existence or not of these two numbers, $c$ and $d$, such that the inequality is satisfied. This fact simplifies the problem, since, by the following theorem, the only thing that needs to be examined is whether a single inequality holds. Before presenting the actual theorem, we need some additional notation: Given two time series $T[1\ldots n]$ and $Q[1\ldots n]$, we denote:

- $\|T\|$ the Euclidean norm of $T$: $\sqrt{\sum_{i=1}^n T[i]^2}$.
- $T\cdot Q$ the inner product of $T$ and $Q$: $\sum_{i=1}^n (T[i] \cdot Q[i])$.
- $\Sigma(T)$ the sum of the points of $T$: $\sum_{i=1}^n T[i]$.

The theorem follows.

**Theorem 2.1.** Given two time series, $T[1\ldots n]$ and $Q[1\ldots n]$, $T$ is $\epsilon$-similar to $Q$ if and only if

$$(\Sigma(T) \cdot (T \cdot Q) - \|T\|^2 \cdot \Sigma(Q))^2 - \left((\Sigma(T)^2 - n \cdot \|T\|^2) \cdot (\epsilon \cdot \|T\|^2 + (T \cdot Q)^2 - \|T\|^2 \cdot \|Q\|^2)\right) > 0. (1)$$

**Proof.** We will prove the above theorem starting from the definition of $\epsilon$-similarity. If $T$ is $\epsilon$-similar to $Q$, by definition,

$$\sum_{i=1}^n (c \cdot T[i] + d - Q[i])^2 < \epsilon. (2)$$

Expanding the sum in (2) yields

$$c^2 \cdot \|T\|^2 + c \cdot (2 \cdot d \cdot \Sigma(T) - 2 \cdot T \cdot Q) + ((|Q|^2 + d^2 \cdot N - 2 \cdot d \cdot \Sigma(Q) - \epsilon) < 0. (3)$$

The left side of the above inequality is a quadratic equation of $c$, and therefore the inequality has a solution (concerning $c$ and consequently $d$) if and only if the discriminant is positive:

$$\Delta = (2 \cdot d \cdot \Sigma(T) - 2 \cdot T \cdot Q)^2 - 4 \cdot \|T\|^2 \cdot (|Q|^2 + d^2 \cdot N - 2 \cdot d \cdot \Sigma(Q) - \epsilon) > 0. (4)$$

Expanding (4), we get an inequality of the following form (we omit the details for brevity)

$$\Delta = A \cdot d^2 + B \cdot d + C > 0 (5)$$

with $A < 0$. Like previously, an appropriate $d \in \mathbb{R}$ exists if and only if

$$\Delta' = B^2 - 4 \cdot A \cdot C > 0 (6)$$

which, expanded, becomes

$$\Delta' = (\Sigma(T) \cdot (T \cdot Q) - \|T\|^2 \cdot \Sigma(Q))^2 - \left((\Sigma(T)^2 - n \cdot \|T\|^2) \cdot (\epsilon \cdot \|T\|^2 + (T \cdot Q)^2 - \|T\|^2 \cdot \|Q\|^2)\right) > 0. (7)$$

Note that inequality (7) is the same as (1). Therefore, we showed that the inequality (1) is equivalent to inequality (2), and thus the proof is complete. $\square$

Notice that inequality (1) includes only the arguments $\Sigma(T)$, $\|T\|$, $\Sigma(Q)$, $\|Q\|$, $T \cdot Q$ and $\epsilon$. Thus, in case we are interested in comparing two time series $T$, $Q$ in a “growing window” or “sliding window” scan, the evaluation of the sums $\Sigma(T)$, $\|T\|$, $T \cdot Q$ etc. involved in (1) can be done recursively by adding to the previously calculated sums the remaining terms and thus reducing the amount of computations needed. For example, in case we have computed $\|T[1\ldots 200]\|$, then $\|T[1\ldots 201]\|$ can be computed efficiently based on the relation

$$\|T[1\ldots 201]\| = \sqrt{\|T[1\ldots 200]\|^2 + T[201]^2}$$

and similarly we can calculate $\|T[2\ldots 201]\|$, since

$$\|T[2\ldots 201]\| = \sqrt{\|T[1\ldots 200]\|^2 - T[1]^2 + T[201]^2}.$$

Actually, similar relations hold for every argument in (1). This allows us to perform efficient “growing window” or “sequential scan” algorithms. For example, if we compute the arguments in (1) for a window of size $m$ in $T$, i.e. $T[i\ldots i+m-1]$, we can compute the same arguments for the “growing” window $T[i\ldots i+m]$ or for the “sliding” window $T[i+1\ldots i+m]$ in $O(1)$ time.

### 2.3 An extension of the CD-criterion—the BCD-criterion

Although the CD-criterion includes only vertical shift and scaling, the concept on which it is based allows us to extended it to various other transformations, maintaining the same levels of efficiency. As an example of the CD-criterion potential, we present an extension of the CD-criterion that introduces similarity under an additional transformation, namely skew. Skewing a time series can be thought as making the time series climb or decline (see Figure 3). Below we define the corresponding “V-Skew” transformation.

![Figure 3: Skewing a time sequence: (a) represents a 160-point time sequence and (b) the same time sequence skewed.](image)

**Definition 6.** Given a time sequence $T[1\ldots n]$, a V-Skew transformation of $T$ is a time series $T'[1\ldots n]$ satisfying

$$T'[i] = T[i] + b \cdot (i - 1)$$

where $b$ a real number.

Having defined V-Skew transformation, we present the extended $\epsilon_S$-similarity that determines whether a time series
is similar to another under V-Shift, V-Scale and V-Skew transformations.

**Definition 7.** Given two time series, \( T[1...n] \) and \( Q[1...n] \), and a positive real constant \( \epsilon \), we define \( T \) to be \( \epsilon \)-similar to \( Q \) if there exist real numbers \( b, c \) and \( d \) such that

\[
\sum_{i=1}^{n} (c \cdot T[i] + b \cdot (i-1) + d - Q[i])^2 < \epsilon.
\]

In a similar way to the CD-criterion, there exists a single inequality based upon which we can decide whether or not two time series are \( \epsilon \)-similar. However, for reasons of brevity, we will not present a proof of this here. Note that in this paper we only implement and work on the CD-criterion.

### 3. Efficient Subsequence Matching

In this section we present an algorithm that locates all scaled occurrences of a query time series \( Q \) in a much larger time series (database) \( D \). Before describing the method, we need to define time scaling.

#### 3.1 Time scaling

Here we give the basic definitions on time scaling. We will only consider compressing a time series, since this should be adequate for our algorithm in the next section. The reason for this is that given two time sequences of different length, there are two options in order to compare them: either expand the smaller or compress the largest. We prefer the second approach, since—as verified in the experiments we conducted—it is more robust to noise, accurate and can be implemented more efficiently.

Let \( T[1...n] \) be a time series and \( m \) an integer with \( 0 < m < n \). There are many ways to define the scaling of \( T \) from \( n \) to \( m \) points. Our approach scales a large time sequence \( T \) into a smaller length by grouping sequential points of \( T \) and taking their averages in order to form the smaller sequence.

By intuition, we can understand that each scaled point should “correspond” (in some way) to \( \frac{n}{m} \) points approximately. If \( m \) divides \( n \) and \( \frac{n}{m} = k \), then we use \( k \) sequential samples for each grouping. Therefore, the scaled sequence is in fact the sequence of \( k \)-points averages of the sequence \( T \). If, however, \( \frac{n}{m} \) is not an integer, some scaled points should correspond to \( \lfloor \frac{n}{m} \rfloor \) points and some to \( \lceil \frac{n}{m} \rceil \).

How exactly this correspondence is materialized is of no big importance, since for the most practical applications, a one-point difference in an average should not make a big difference. However we need to decide on a specific method in order to produce specific results. Therefore, if \( \frac{n}{m} \) is not an integer, we take the first \( n \mod m \) points of the scaled sequence to be averages of \( \lfloor \frac{n}{m} \rfloor \) points, and the rest to be averages of \( \lceil \frac{n}{m} \rceil \) points of \( T \). In an illustrative example, Figure 4 presents a scaling of a 90-point time sequence into 40 points. We now strictly define the above.

**Definition 8.** Let \( T[1...n] \) be a time series, \( m \) be an integer such that \( 1 \leq m < n \) and \( l = n \mod m \). We define the scaling of \( T \) to size \( m \), and denote by \( S_m(T) \), a time sequence \( S_m[1...m] \) where

\[
T_m[i] = \begin{cases} \frac{1}{\lceil \frac{n}{m} \rceil} \sum_{j=(i-1) \mod m + 1}^{i \mod m} T[j], & \text{if } 1 \leq i \leq l, \\ \frac{1}{\lfloor \frac{n}{m} \rfloor} \sum_{j=(i-1) \mod m + 1}^{i \mod m} T[j], & \text{if } l < i \leq m. \end{cases}
\]

The above definition simply formalizes what explained in the previous paragraph. Note that if \( l = n \mod m = 0 \), then the scaling will consist of \( \lfloor \frac{n}{m} \rfloor = \frac{n}{m} \) point averages, as expected.

Taking averages has been used successfully as an approximation and dimensionality reduction technique in time series [31, 19]. This type of scaling that we use has the critical advantage that it is robust to noise, meaning that small variances of the time sequence do not alter considerably the scaled sequence. At the same time, it can obviously be implemented in a computationally efficient way. Finally, the scaling produced by this method is very close to the human intuition of “scaling”.

Let us now focus on the scaled matching of sequences. Let \( Q[1...m] \), \( D[1...n] \) be two time sequences with \( m < n \). Suppose that \( D \) represents a very large time sequence, (i.e. stock data for the last 25 years) and \( Q \) is a small sample time sequence, (i.e. stock data for the last 30 days).\footnote{Such a proof would be almost similar to the proof of Theorem 2.1.} We would like to find the scaled matches of \( Q \) in \( D \). To become more specific, we would consider a subsequence of \( D \) a scaled match if, scaled to \( |Q| \) points and under some transformations (as described in the previous section), the scaled subsequence can become “similar” to \( Q \) (in the way that \( \epsilon \)-similarity defines). Note that for brevity and clarity of presentation we are interested only in matches of length larger that \( |Q| \) (actually, for most practical applications this is adequate). The following definition gives a formal description of the above.

**Definition 9.** Let \( Q[1...m] \), \( D[1...n] \) be two time sequences with \( m < n \), and \( \epsilon \) be a positive real constant. A scaled match or occurrence of \( Q \) in \( D \) is a subsequence \( D_M \) of \( D \), with \( |D_M| \geq m \), such that the scaling of \( D_M \) to size \( m \) is \( \epsilon \)-similar to \( Q \).
Note that we would probably want to set a lower and upper bound on the size of $D_m$ for a given $Q$. The need for an upper bound comes from the fact that if we scale very large sequences into small sizes we lose so much information from scaling\(^5\) that virtually any use of the scaled sequence becomes meaningless. In the stock example for instance, we would not consider interesting the fact that the last month’s variation of a stock price movement is similar to the variation of the last five years. Obviously though, this upper bound is subjective\(^6\). In a similar manner, we would probably not be interested in the scaling (expansion) of, e.g. two points to thirty. Therefore a lower bound, subjectively defined, should also exist\(^6\).

3.2 The algorithm

Before presenting a detailed description of the algorithm, we give a description of the naive algorithm along with the comparative advantages of our algorithm.

3.2.1 General discussion

We now focus to our problem. Let us again consider a large time sequence $D[1 \ldots n]$ and a much smaller sequence $Q[1 \ldots m]$. The task that we want to perform is “find all scaled matches—within some given scale bounds—of $Q$ in $D$”. As mentioned before, the problem itself is rather complex. The naive approach is to begin at position 1 of $D$, get a window of minimum size $w_{\text{min}}$ (where $w_{\text{min}}$ is the lower scaling bound as described in the previous section), scale $D[1 \ldots w_{\text{min}}]$ to $m$ points and see whether it matches $Q$ under some similarity measure. With the left side of the window anchored in $D[1]$, check in a similar manner if $D[1 \ldots i]$ (scaled to $m$ points) matches $Q$, for all $i$, $w_{\text{min}} < i \leq w_{\text{max}}$ (where $w_{\text{max}}$ is the upper scaling bound). After this step, repeat the same procedure with the window anchored at position $D[2]$, then $D[3]$ etc. If we let $s = w_{\text{max}} - w_{\text{min}} + 1$ (the number of possible window lengths at each anchor point) then the number of windows to be checked by the above algorithm is approximately $n \cdot s$. Moreover, considering that in each step the algorithm must perform several calculations in order to examine if the scaled window and $Q$ do actually match, it is evident that the actual time complexity of the naive approach is rather high.

Our method outperforms the naive approach in two ways:

- It performs a pruning on the sizes of the windows that we should check for a scaled match resulting in checking fewer windows.
- It is implemented in a way that takes advantage of the efficiency of the CD-criterion that allows the calculations involved in “growing window” sequential scan to be done very efficiently.

The pruning is based on the following observation (here presented rather intuitively): in order for a sequence $D'[1 \ldots r]$ to be similar to a smaller sequence $Q[1 \ldots m]$, the first \(\frac{r}{m}\) points of $D'$ must also be (in a way) similar to the first \(\frac{r}{m}\) points of $Q$ for some specific $k \leq m$. In other words, it is impossible for two sequences to have very dissimilar initial parts (prefixes) but, at the same time, be similar in whole.

3.2.2 Description of the algorithm

For the description that follows, let again $Q[1 \ldots m]$ the query sequence, with $m \geq 3$, and $D[1 \ldots n]$ the database sequence. Let $w_{\text{min}}$ and $w_{\text{max}}$, $w_{\text{min}} < w_{\text{max}}$, be the lower and upper window lengths, meaning that we are interested in finding scaled matches of $Q$ in $D$ with length ranging from $w_{\text{min}}$ to $w_{\text{max}}$. In order to enhance the clarity of presentation, we will consider $w_{\text{min}}$ and $w_{\text{max}}$ to be multiples of $|Q| = m$. The implementation of the more general case is straightforward. Let thus, $w_{\text{min}} = k_{\text{min}} \cdot m$ and $w_{\text{max}} = k_{\text{max}} \cdot m$ for appropriate $k_{\text{min}}$ and $k_{\text{max}}$.

The algorithm is an iterative procedure. In each iteration (function \texttt{find\_matching\_windows()} at Figure 5\(^7\), the algorithm decides whether or not there exist any scaled matches of windows whose left side is anchored at a specific point of $D$.

In more detail, let us consider the iteration that anchors the windows on $D[i]$. If there exists a scaled match of $Q$ in $D[i]$, then the window anchored at $D[i]$ will be a candidate window. The algorithm starts (initialization step) by checking only whether the windows $D[i \ldots i + 3 \cdot k_{\text{min}} - 1]$, $D[i \ldots i + 3 \cdot (k_{\text{min}} + 1) - 1]$, $D[i \ldots i + 3 \cdot k_{\text{max}} - 1]$, and $D[i \ldots i + 3]$, are scaled matches of $Q[1 \ldots 3]$ (lines 2–5 of function \texttt{find\_matching\_windows()}).

By performing this check, we have actually checked all the three-point-scaled prefix subsequences of all candidate windows at the specific anchoring. This is because, if we scale a window of size $l$ to $m$ points, then due to the fact that $k_{\text{min}} \leq \frac{l}{m} \leq k_{\text{max}}$, each scaled point of the window will be an average of at least $k_{\text{min}}$ and at most $k_{\text{max}}$ points.

In other words, if we consider a candidate window $W$, and scale it to $m$ points, $W[1 \ldots m]$ (remember that $m = |Q|$) then $W[1 \ldots 3]$ will have been checked whether or not it matches $Q[1 \ldots 3]$ in the initialization step. Therefore, after performing this step, if we have no matches, then we can be sure (see Lemma 2.1) that there are no scaled matches of $Q$ that start in $D[i]$ and thus the iteration ends.

However, if there are some matches in the initialization step then we take the set $M_3$ that contains all of the above windows (scaled to three points) that are scaled matches of $Q[1 \ldots 3]$. $M_3$ is created in the loop of lines 2–5. We would like now to check whether or not there exist scale matches of size four. The key observation is that in order for a window $M$ (scaled to $M[1 \ldots 4]$) to be a scale match of $Q[1 \ldots 4]$, then the prefix subsequence of $M$, i.e. $M[1 \ldots 3]$, should belong to the set $M_3$. Therefore, we only need to examine the windows that are “extensions” of the windows in the set $M_3$. More formally, we define extensions as following.

\textbf{Definition 10.} Let a time sequence $D[1 \ldots d]$ and a window (subsequence) $W[1 \ldots n]$ of $D$ anchored at $D[i]$. Let

\(^\text{5}\)This happens inevitably for every possible scaling method.

\(^\text{6}\)Our algorithm uses both lower and upper scaling bounds as input parameters.

\(^\text{7}\)Remember that the notation $S_m(D)$ that appears in the pseudocode stands for the scaling of the time sequence $D$ to $m$ points.
It can be easily proved that if \( m \) does not divide \( n \), the extension of a window \( W[i..j] \) scaled to \( W_s[i..j] \) points is \( W[i..n] + \left\lfloor \frac{n}{m} \right\rfloor \); and if \( m \) divides \( n \) we have two extensions, \( W[i..n] \) and \( W[i..n + \frac{n}{m} - 1] \).

Returning to our algorithm, the extensions of the set \( M_4 \) consists of the set of candidate windows (of size four) \( C_4 \) (lines 12–23). The windows that, scaled to four points, actually match (under the CD-criterion) \( Q[i..j] \) compose the set \( M_4 \) (lines 26–28). Obviously, it holds \( M_4 \subseteq C_4 \). In the same manner, we continue by constructing \( C_5, M_5, \) etc. until, for a number \( t \), the set \( M_t \) is empty (check is performed in line 8). It is obvious that the scaled matches that we are interested in are the elements of the set \( M_m \). Therefore, if \( t < m \) we have no matches at that offset and therefore the algorithm continues at the next offset without returning any matches (lines 31–32).

As one would expect, experiments verified that in most real datasets, if there are no actual matches starting at the point \( D[i] \), the set \( M_t \) becomes empty at a relatively small \( t \). Therefore, the algorithm can proceed in the next iteration having checked much fewer windows compared to the naive approach. However, in the case that a subsequence of \( D \) (starting at \( D[i] \)) is a match of \( Q \), the above procedure tends to make somewhat more checks than the naive method. However, in most practical applications, where we are looking for close matches of \( Q \) in a massive database, the percentage of such cases is extremely small. On the other hand, in the rest of the database our algorithm checks only a small fraction of the windows that the naive algorithm would. The result of the above is an important performance overhead of the algorithm (more clearly shown in the experiments—see section 4).

As mentioned before, our algorithm has no false dismissals. The respective theorem is presented below.

**Theorem 3.1.** The proposed algorithm has no false dismissals. More formally, given a query sequence \( Q[1..m] \) and a database sequence \( D[1..n] \), let \( W \) be a subsequence (window) of \( D \), i.e. \( W[1..w] \equiv D[i..i + w - 1] \) (with \( k_{\min} \cdot m \leq w \leq k_{\max} \cdot m \)), such that the scaling of \( W \) to \( m \) points, \( W_S[1..m] \), matches \( Q \). Then the algorithm will return \( W \) as a match.

The proof of this theorem is provided in the appendix.

### 3.2.3 Implementation of the sets \( M \) and \( C \)

For clarity we have omitted from the algorithm’s pseudocode in Figure 5 the implementation of the sets \( M_4 \) and \( C_4 \). As mentioned in Section 2.2, the CD-criterion gives us the ability to perform very efficient “growing window” matches. In order to implement this feature in this algorithm, when we insert a sequence \( D[i..j] \) into a set, e.g. \( C_{k+1} \), we actually store the attributes needed to compute the CD-criterion (\( ||D|| \), \( \Sigma(D) \) etc.). In fact, these attributes can be computed recursively; e.g. the attributes of the set \( C_{k+1} \) can be computed efficiently using (recursively) the respective attributes of the set \( M_{k-1} \) in a similar way as it was demonstrated in Section 2.2. The reason why we are able to do this is that for all new window “extensions” that we insert in the set \( C_{k+1} \) we have the attributes of their prefixes stored in the set \( M_k \). For example, if we insert a scaled window \( W_S[i..j + 1] \) in \( C_{k+1} \), we would already have the attributes of \( W_S[i..j + k] \) stored in \( M_k \). Therefore, we only need to compute the value of \( W_S[j + 1] \) and then compute recursively the new attribute values, instead of recalculating the attributes from the beginning.
Having stored these attributes in \( C_{k+1} \) we can perform the matching check at lines 26–28 using the stored information of \( C_{k+1} \). The attributes of the matching windows are then copied in \( M_{k+1} \) (actually, for the sake of efficiency, the non-matching entries of \( C_{k+1} \) are deleted and the remaining are used as the new set \( M_{k+1} \)). Therefore, we proceed in the next iteration of the loop and we are able to make (similarly) efficient computation of \( C_{k+2} \).

Naturally, in order to apply the CD-criterion we also need the attributes of \( Q \). Since \(|Q|\) is relatively small, we assume that the attributes of all prefix subsequences of \( Q \) can be stored in main memory and, in fact, in the majority of practical applications this is feasible. On the other hand, concerning the sets \( M \) and \( C \), we only need space for two set instances at any case (either \( C_k \) and \( M_k \), or \( M_k \) and \( C_{k+1} \), at any \( k \)). The storage of these sets can either be in primary or secondary memory, where in the latter case we will have an I/O overhead\(^8\).

We performed multiple experiments on these representative time series. While experimenting on the mentioned time series, we came across several subsequences that matched the query sequence. A very interesting fact was that the query subsequences (e.g., 20 points—i.e. one month—period in the stock example) matched almost perfectly much larger subsequences (in the same example, ranging from 1 to 5 months—see Figure 6 for examples of some matches on the Dow Jones index). This fact justifies the hypothesis that the timeseries we experimented upon appear to be self-similar in a great extend (as it has been discussed in the introduction) and it is indicative of the algorithm’s potential. We concentrate now on the efficiency of the proposed algorithm. For our experiments we used a Pentium IV 2.0GHz, with 512 Mbytes RAM, running Linux in single user mode. The naive algorithm has been described in section 3.2.1. In Table 1 we present the relative average running times of the experiments conducted. It is obvious that the pruning performed by our algorithm, together with the efficient implementation that was described in section 3.2.3, speeds up the process of extracting matching subsequences up to over an order of magnitude.

![Figure 6: Examples of real matches from the Dow Jones stock index, here presented rescaled to 20 points and appropriately shifted and scaled in the V-axis. Note that, initially, all these matches where at different scales and offsets in both time and amplitude axes.](image)

![Figure 7: A scan of a 16,000-point sequence. At each offset, the maximum windows size that our algorithm had to check for matches is displayed. The horizontal line indicates the maximum window length that the naive algorithm checks.](image)

\[\text{Table 1: Comparison of the proposed algorithm versus the naive.}\]

<table>
<thead>
<tr>
<th>Time Series</th>
<th>Running time of the proposed algorithm</th>
<th>Running time of the naive algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stock Data</td>
<td>0.143</td>
<td>0.091</td>
</tr>
<tr>
<td>Network Traffic Data</td>
<td>0.091</td>
<td></td>
</tr>
</tbody>
</table>

\(^8\)Actually, because the access of the sets \( M \) and \( C \) can be done sequentially, the disk seeks needed are limited and thus the overhead is relatively small.

4. EXPERIMENTS

In this section we present the experimental results of the algorithm that we propose. More specifically, we illustrate both the quality of the results we came across as well as the performance of the algorithm, compared to the naive approach. For our experiments we used the following time series:

3. Network traffic data (approx. one million points) [22, 9].

sequence. In all the other cases (where there where actually no matches) our algorithm performed an extensive pruning. The horizontal line on the top of the distribution indicates the query length that the naive approach reaches at each starting location (100 points). It is worth noticing that instead of starting by checking all the three-point candidate (scaled) windows, as described in section 3.2, we can start by any number desired; the theoretical results remain unchanged. The optimal setting depends on the nature of the time series examined, but in our experiments we found that $|Q|/4$ is an adequate heuristic that yields good results in most cases (in Figure 7 this number is 5 so $5 \cdot k_{\text{max}} = 25$).

5. RELATED WORK

The problem of efficiently discovering matches of a small query sequence in a large database has attracted a lot of attention and a number of approaches have been proposed. A large fraction of them is based on indexing, which we will first briefly describe and afterwards discuss the reasons why it is improper for our problem.

Feature indexing is a technique where we first extract some features from every subsequence of the database (and therefore perform dimensionality reduction), store them in a spatial data structure, like a variant of an R tree [11], and then use operations like range queries [10] on the index in order to find candidate matches. Postprocessing is used in order to find the actual matches. Pioneer work on this approach was done by Agrawal et al. in [1] and subsequent research has produced significant results in diverse variations of the initial idea [4, 8, 5, 17, 16, 12].

Although indexing is used very extensively, creating an index that can handle queries of arbitrary length along with time and amplitude scaling is non-trivial, mostly due to the vast number of candidate subsequences. In [28, 31] time scaling is allowed in the sense that it supports queries of different lengths. However, given a specific query, arbitrary-length subsequence matching is not supported since the index is build on fixed-length time series (either whole time series or subsequences).

In the same path, Perng et al. propose in [27] Landmarks, a similarity model that extracts the most important points (features) of a time series and defines some similarity measures that are invariant to certain transformations. However, since all these approaches approximate time series, it follows that they cannot be applied to our problem without false dismissals.

As far as time scaling is concerned, a similarity model that has attracted much interest (mostly as an alternative of the Euclidean distance) is Dynamic Time Warping (DTW). DTW is a distance measure that allows flexible manipulation of the rate of the signals (time axis). Therefore, it implies a different definition of scaling than the one used on the similarity model that of this paper. This technique was introduced to the data mining community in [3] and there has been a lot of recent interest [30, 18, 21, 6, 15]. However, DTW is inappropriate for our problem. The first reason is the different definition of time scaling. Further, DTW is either used in a form of sequential scanning, where its complexity is quadratic on the mean length of the sequences compared and therefore highly inefficient, or it is implemented as part of an index—an approach that, as discussed before, appears difficult to adopt in our problem (especially with support for amplitude scaling).

Regarding amplitude shift and scaling, Chu and Wong [5] have proposed a transformation of time sequences that is shift invariant. After transforming time sequences, one can apply special operations on the transformed sequences in order to determine their minimum distance. However, this approach requires more computations than the CD-criterion we propose. Furthermore, it seems hard to extend the shift invariant transformation so as to include more operations (other than shifting and scaling) like skew which is included in the BCD-criterion. A variation of the above idea is also presented in [13].

Concerning the problem of finding scaled matches of different lengths, there have been proposed a number of techniques that solve the problem approximately. In [20] Keogh and Smyth propose using a piecewise linear approximation method, in order to extract the most “characteristic” points in a time sequence. This idea is also used by Keogh in [14] achieving to compress time sequences as well as to perform faster search of similar patterns under scaling and shifting. In the same path, Perng et al. propose in [27] Landmarks, a similarity model that extracts the most important points (features) of a time series and defines some similarity measures that are invariant to certain transformations. However, since all these approaches approximate time series, it follows that they cannot be applied to our problem without false dismissals.

As far as time scaling is concerned, a similarity model that has attracted much interest (mostly as an alternative of the Euclidean distance) is Dynamic Time Warping (DTW). DTW is a distance measure that allows flexible manipulation of the rate of the signals (time axis). Therefore, it implies a different definition of scaling than the one used on the similarity model that of this paper. This technique was introduced to the data mining community in [3] and there has been a lot of recent interest [30, 18, 21, 6, 15]. However, DTW is inappropriate for our problem. The first reason is the different definition of time scaling. Further, DTW is either used in a form of sequential scanning, where its complexity is quadratic on the mean length of the sequences compared and therefore highly inefficient, or it is implemented as part of an index—an approach that, as discussed before, appears difficult to adopt in our problem (especially with support for amplitude scaling).

In the same path, Perng et al. propose in [27] Landmarks, a method for identifying similar time series under some linear amplitude transformations. In their work, they define two time series to be similar if they have similar long subsequences. However, in order to find whether two time series are similar or not, they propose an approximation algorithm that (due to approximation) may yield false results. Moreover, the distance measure that is used is less flexible than the euclidean distance, thus reducing the robustness of the algorithm against noise or slight variations of the time series, while it is not obvious how non-linear transformations could be implemented in this framework.

Finally, there has been some effort to overcome the inflexibility of time axes in ways other than scaling. Agrawal et al. [2] propose a similarity model that implements gap removing, offset translation and amplitude scaling and propose an efficient algorithm that implements it. However, there is no ability of time scaling.

6. CONCLUSION AND FUTURE WORK

We have presented an algorithm that copes with the important problem of finding scaled occurrences of a query sequence in a large time sequence database. The main contributions of this paper are:

- We have proposed a novel and efficient algorithm for variable-length subsequence matching under time and amplitude transformations, and verified its efficiency in various experiments on real data.
- We have provided a proof that the algorithm does not sacrifice its reliability (i.e., it has no false dismissals) in order to achieve the high efficiency it has.
- Instead of using a similarity measure, we have proposed a criterion for determining whether one sequence
matches another under V-axis transformations. We have shown why this criterion is computationally efficient and we believe that it could be of independent interest to researchers.

- We introduce the significance of the problem that we analyze, based on the nature of self-similar data, in diverse fields of science and finance.

Concerning future work, we believe that the CD-criterion can be easily expanded so as to include more transformations (e.g. in the way that the BC-D-criterion included the "skew" transformation). Thus, efficient similarity criteria under multiple transformations can be implemented that would expand the range of applications of the proposed algorithm.

7. ACKNOWLEDGMENTS

We would like to thank Foto Afrati for invaluable comments on a draft of this paper. Also, Alexander Dimakis for insightful discussions on some aspects of this paper.

8. REFERENCES


THEOREM 3.1. The proposed algorithm has no false dismissals. More formally, given a query sequence $Q[1 \ldots m]$ and a database sequence $D[1 \ldots n]$, let $W$ be a subsequence (window) of $D$, i.e. $W[1 \ldots w] \equiv D[i \ldots i + w - 1]$ (with $k_{\min} \cdot m \leq w \leq k_{\max} \cdot m$), such that the scaling of $W$ to $m$ points, $W_S[1 \ldots m]$, matches $Q$. Then the algorithm will return $W$ as a match.

PROOF. The proof is based on showing that, for each $i \geq 3$, the prefix subsequence of $W_S$, $W_S[1 \ldots i]$ will be in the set $M_i$. In other words, we will show that

$$\forall i, \quad 3 \leq i \leq m, \quad W_S[1 \ldots i] \in M_i. \quad (8)$$

We shall prove (8) by induction. By the discussion in section 3.2.2 it follows that for $i = 3$ equation (8) holds. For the induction step, assume that (8) holds for $i = k$, i.e. $W_S[1 \ldots k] \in M_k$. We want to show that it holds for $i = k + 1$. We will show that $W_S[1 \ldots k + 1] \in M_{k+1}$.

Note first that we only have to show that $W_S[1 \ldots k + 1] \in C_{k+1}$. This is because by Lemma 2.1., since $W_S[1 \ldots m]$ matches $Q[1 \ldots m]$ then $W_S[1 \ldots k + 1]$ should match $Q[1 \ldots k + 1]$ (since $k + 1 \leq m$) and therefore it will be added to $M_{k+1}$ by the algorithm (at lines 26–28). For convenience, name $D_M$ the subsequence of $W$ that its scaling to $k$ points is $W_S[1 \ldots k]$. In order to prove that $W_S[1 \ldots k + 1] \in C_{k+1}$ we examine two cases.

- First case: $|D_M| \mod k \neq 0$.
  In this case, the scaled window $W_S[1 \ldots k]$ consists of \(\left\lfloor \frac{|D_M|}{k} \right\rfloor\) averages (for the first $k$ mod $i$ points) and \(\left\lfloor \frac{|D_M|}{k} \right\rfloor\) points for the rest. Obviously, if we add \(\left\lfloor \frac{|D_M|}{k} \right\rfloor\) to $D_M$ and scale it to $k + 1$ points we will get $W_S[1 \ldots k + 1]$. But that is exactly what happens at lines 16–17 of the algorithm. Therefore, $W_S[1 \ldots k + 1] \in C_{k+1}$.

- Second case: $|D_M| \mod k = 0$.
  In this case, $D_M$ scaled to $W_S[1 \ldots k]$ consists of averages of \(\frac{|D_M|}{k}\) points (exactly). Therefore, the last point of $W_S[1 \ldots k + 1]$ can either consist of \(\frac{|D_M|}{k} + \left\lfloor \frac{|D_M|}{k+1} - 1 \right\rfloor = \frac{|D_M|}{k} - 1\) points (see comment on Definition 10). In both cases $W_S[1 \ldots k+1]$ is inserted to $C_{k+1}$ at lines 19–22 of the algorithm. Therefore, in any case it holds $W_S[1 \ldots k + 1] \in C_{k+1}$.

Thus, in each case we have proven the induction step. Therefore, if we have a matching window then it will be added, eventually, to the set $M_m$ and therefore it will be returned as a match by the algorithm at line 31. \(\square\)