The Move-Split-Merge Metric for Time Series

Alexandra Stefan, Vassilis Athitsos, and Gautam Das

Abstract—A novel metric for time series, called MSM (move-split-merge), is proposed. This metric uses as building blocks three fundamental operations: Move, Split, and Merge, which can be applied in sequence to transform any time series into any other time series. A Move operation changes the value of a single element, a Split operation converts a single element into two consecutive elements, and a Merge operation merges two consecutive elements into one. Each operation has an associated cost, and the MSM distance between two time series is defined to be the cost of the cheapest sequence of operations that transforms the first time series into the second one. An efficient, quadratic-time algorithm is provided for computing the MSM distance. MSM has the desirable properties of being metric, in contrast to the dynamic time warping (DTW) distance, and invariant to the choice of origin, in contrast to the Edit Distance with Real Penalty (ERP) metric. At the same time, experiments with public time series datasets demonstrate that MSM is a meaningful distance measure, that oftentimes leads to lower nearest neighbor classification error rate compared to DTW and ERP.

Index Terms—Time series, similarity measures, similarity search, distance metrics.

1 Introduction

Time series data naturally appear in a wide variety of domains, including financial data (e.g. stock values), scientific measurements (e.g. temperature, humidity, earthquakes), medical data (e.g. electrocardiograms), audio, video, and human activity representations. Large time series databases can serve as repositories of knowledge in such domains, especially when the time series stored in the database are annotated with additional information such as class labels, place and time of occurrence, causes and consequences, etc.

A key design issue in searching time series databases is the choice of a similarity/distance measure for comparing time series. In this paper, we introduce a novel metric for time series, called MSM (move-split-merge). The key idea behind the proposed MSM metric is to define a set of operations that can be used to transform any time series into any other time series. Each operation incurs a cost, and the distance between two time series X and Y is the cost of the cheapest sequence of operations that transforms X into Y.

The MSM metric uses as building blocks three fundamental operations: Move, Split, and Merge. A Move operation changes the value of a single point of the time series. A Split operation splits a single point of the time series into two consecutive points that have the same value as the original point. A Merge operation merges two consecutive points that have the same value into a single point that has that value. Each operation has an associated cost. The cost of the Move operation is the absolute difference between the old value and the new value. The cost of each Split and Merge operation is equal and set to a constant.

Our main motivation in formulating MSM has been to satisfy, with a single distance measure, a set of certain desirable properties that no existing method satisfies fully. One such property is robustness to temporal misalignments. Such robustness is entirely lacking in methods where time series similarity is measured using the Euclidean distance [1], [2], [3], [4] or variants [5], [6], [7]. Such methods cannot handle even the smallest misalignment caused by time warps, insertions, or deletions.

Another desired property is metricity. As detailed in Section 3.1.1, metricity allows the use of an extensive arsenal of generic methods for indexing, clustering, and visualization, that have been designed to work in any metric space. Several distance measures based on dynamic programming (DP), while robust to temporal misalignments, are not metric. Such methods include dynamic time warping (DTW) [8], constrained dynamic time warping (cDTW) [9], Longest Common Subsequence (LCSS) [10], Minimal Variance Matching (MVM) [11], and Edit Distance on Real Sequence (EDR) [12]. All those measures are non-metric, and in particular do not satisfy the triangle inequality.

Edit Distance with Real Penalty (ERP) [13] is a distance measure for time series that is actually a metric. Inspired by the edit distance [14], ERP uses a sequence of "edit" operations, namely insertions, deletions, and substitutions, to match two time series to each other. However, ERP has some behaviors that, in our opinion, are counterintuitive. First, ERP is not translation-invariant: changing the origin of the coordinate system changes the distances between time series, and can radically alter similarity rankings. Second, the cost of inserting or deleting a value depends exclusively on the absolute magnitude of that value. Thus, ERP does not treat all values equally; it explicitly prefers inserting and

- *A. Stefan, V. Athitsos and G. Das are with the Department of Computer Science and Engineering, University of Texas at Arlington, Arlington, TX, 76019.*

deleting values close to zero compared to other values. In our formulation, we aimed to ensure both translation invariance and equal treatment of all values.

A desired property of any similarity measure is computational efficiency. Measuring the DTW or ERP distance between two time series takes time quadratic to the sum of lengths of the two time series, whereas linear complexity is achieved by the Euclidean distance and, arguably, cDTW (if we treat the warping window width, a free parameter of cDTW, as a constant). One of our goals in designing a new distance measure was to not significantly exceed the running time of DTW and ERP, and to stay within quadratic complexity.

The proposed MSM metric is our solution to the problem of satisfying, with a single measure, the desired properties listed above: robustness to misalignments, metricity, translation invariance, treating all values equally, and quadratic time complexity. The MSM formulation deviates significantly from existing approaches, such as ERP and DTW, and has proven quite challenging to analyze. While the proposed algorithm is easy to implement in a few lines of code (see Figure 10), proving that these few lines of code indeed compute the correct thing turned out to be a non-trivial task, as shown in Sections 4 and 5. We consider the novelty of the formulation and the associated theoretical analysis to be one of the main contributions of this paper.

For real-world applications, satisfying all the above-mentioned properties would be of little value, unless the distance measure actually provides meaningful results in practice. Different notions of what is meaningful may be appropriate for different domains. At the same time, a commonly used measure of meaningfulness is the nearest neighbor classification error rate attained in a variety of time series datasets. We have conducted such experiments using the UCR repository of time series datasets [15]. The results that we have obtained illustrate that MSM performs quite well compared to existing competitors, such as DTW and ERP, yielding the lowest error rate in several UCR datasets.

In summary, the contributions of this paper are the following:

- We introduce the MSM distance, a novel metric for time series data.
- We provide a quadratic-time algorithm for computing MSM. The algorithm is short and simple to implement, but the proof of correctness is non-trivial.
- In the experiments, MSM produces the lowest classification error rate, compared to DTW and ERP, in ten out of the 20 public UCR time series datasets. These results show that, in domains where classification accuracy is important, MSM is worth considering as an alternative to existing methods.

2 Defining the MSM Distance

Similar to the edit distance and ERP, MSM uses a set of operations that can transform any time series to any other time series. The basic operations in the edit distance and ERP are Insert, Delete, $\mathrm{Substitute}$. MSM also uses the $\mathrm{Substitute}$ operation, we just have renamed it and call it the Move operation. This operation is used to change one value into another.

Our point of departure from the edit distance and ERP is in handling insertions and deletions. In the edit distance, all insertions and deletions cost the same. In ERP, insertions and deletions cost the absolute magnitude of the value that was inserted or deleted. Instead, we aimed for a cost model where inserting or deleting a value depends on both that value and the adjacent values. For example, inserting a 10 between two 10s should cost the same as inserting a 0 between two 0s, and should cost less than inserting a 10 between two 0s.

Our solution is to not use standalone Insert and Delete operations, and instead to use Split and Merge operations. A Split repeats a value twice, and a Merge merges two successive equal values into one. In MSM, an Insert is decomposed to a Split (to create a new element) followed by a Move (to set the value of the new element). Similarly, a delete is decomposed to a Move (to make an element equal in value to either the preceding or the following element) followed by a Merge (to delete the element we just moved). This way, the cost of insertions and deletions depends on the similarity between the inserted or deleted value and its neighbors.

We now proceed to formally define the three basic operations and the MSM distance. Let time series $X = (x_1, \ldots, x_m)$ be a finite sequence of real numbers x_i. The **Move operation**, and its cost, are defined as follows:

$$\mathrm{Move}_{i,v}(X) = (x_1, \ldots, x_{i-1}, x_i + v, x_{i+1}, \ldots, x_m) \ . \quad (1)$$

$$\mathrm{Cost}(\mathrm{Move}_{i,v}) = |v|. \quad (2)$$

In words, operation $\mathrm{Move}_{i,v}(X)$ creates a new time series X', that is identical to X, except that the i-th element is *moved* from value x_i to value $x_i + v$. The cost of this move is the absolute value of v.

The **Split operation**, and its cost, are defined as:

$$\mathrm{Split}_i(X) = (x_1, \ldots, x_{i-1}, x_i, x_i, x_{i+1}, \ldots, x_m) \ . \quad (3)$$

$$\mathrm{Cost}(\mathrm{Split}_i) = c. \quad (4)$$

Operation $\mathrm{Split}_i(X)$ creates a new time series X', obtained by taking X and splitting the i-th element of X into two consecutive elements. The cost of this split is a nonnegative constant c, which is a system parameter.

The **Merge operation** acts as the inverse of the Split operation. The Merge operation is invoked as $\mathrm{Merge}_i(X)$, and is only applicable if $x_i = x_{i+1}$. Given a time series $X = (x_1, \ldots, x_m)$, and assuming $x_i = x_{i+1}$:

$$\mathrm{Merge}_i(X) = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_m) \ . \quad (5)$$

$$\mathrm{Cost}(\mathrm{Merge}_i) = c \ . \quad (6)$$

Operation $\mathrm{Merge}_i(X)$ creates a new time series X′, that is identical to X, except that elements x_i and x_{i+1} (which are equal in value) are *merged* into a single element. The

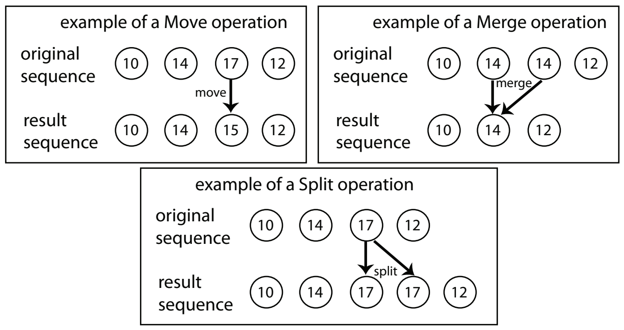


Fig. 1. Examples of the Move, Split, Merge operations.

cost of a Merge operation is equal to the cost of a Split operation. This is necessary, as we explain in Section 3.1, in order for MSM to be metric (otherwise, symmetry would be violated).

Figures 1 and 4 show example applications of Move, Split, and Merge operations.

We define a **transformation** $\mathbb{S} = (S_1, \ldots, S_{|\mathbb{S}|})$ to be a sequence of operations, where $|\mathbb{S}|$ indicates the number of elements of $\mathbb{S}$. Each S_k in the transformation $\mathbb{S}$ is some Move_{i_k,v_k}, Split_{i_k}, or Merge_{i_k} operation, for some appropriate values for i_k and v_k. The result of applying transformation $\mathbb{S}$ to time series X is the result of consecutively applying operations $S_1, \ldots, S_{|\mathbb{S}|}$ to X:

$$\mathrm{Transform}(X, \mathbb{S}) = \mathrm{Transform}(S_1(X), (S_2, ..., S_{||\mathbb{S}||})) \ . \quad (7)$$

In the trivial case where $\mathbb{S}$ is the empty sequence (), we can define $\mathrm{Transform}(X, ()) = X$.

The cost of a sequence of operations $\mathbb{S}$ on X is simply the sum of costs of the individual operations:

$$\mathrm{Cost}(\mathbb{S}) = \sum_{k=1}^{|\mathbb{S}|} \mathrm{Cost}(S_k) \ . \quad (8)$$

Given two time series X and Y, there are infinite transformations $\mathbb{S}$ that transform X into Y. An example of such a transformation is illustrated in Figure 4.

Using the above terminology, we are now ready to formally define the MSM distance. The **MSM distance** $D(X, Y)$ between two time series X and Y is defined to be the cost of the lowest-cost transformation $\mathbb{S}$ such that $\mathrm{Transform}(X, \mathbb{S}) = Y$. We note that this definition does not provide a direct algorithm for computing $D(X, Y)$. Section 5 provides an algorithm for computing the MSM distance between two time series.

3 Motivation for MSM: Metricity and Invariance to the Choice of Origin

In this section we show that the MSM distance satisfies two properties: metricity and invariance to the choice of origin. Satisfying those two properties was a key motivation for our formulation. We also discuss simple examples highlighting how MSM differs from DTW and ERP with respect to these properties.

3.1 Metricity

The MSM distance satisfies reflexivity, symmetry, and the triangle inequality, and thus MSM satisfies the criteria for a metric distance. In more detail:

Reflexivity: Clearly, $D(X, X) = 0$, as an empty sequence of operations, incurring zero cost, converts X into itself. If $c > 0$, then any transformation $\mathbb{S}$ that converts X into Y must incur some non-zero cost. If, for some domain-specific reason, it is desirable to set c to 0, an infinitesimal value of c can be used instead, to guarantee reflexivity, while producing results that are practically identical to the $c = 0$ setting.

Symmetry: Let S be a Move, Split, or Merge operation. For any such S there exists an operation S^{-1} such that, for any time series X, $S^{-1}(S(X))$ = X. In particular:

- The inverse of $\mathrm{Move}_{i,v}$ is $\mathrm{Move}_{i,-v}$.
- Split_i and Merge_i are inverses of each other.

Any sequence of operations $\mathbb{S}$ is also reversible: if $\mathbb{S} = (S_1, \ldots, S_{|\mathbb{S}|})$, then the inverse of $\mathbb{S}$ is $\mathbb{S}^{-1} = (S_{|\mathbb{S}|}^{-1}, \ldots, S_1^{-1})$. $\mathrm{Transform}(X, \mathbb{S}) = Y$ if and only if $\mathrm{Transform}(Y, \mathbb{S}^{-1}) = X$.

It is easy to see that, for any operation S, $\mathrm{Cost}(S) = \mathrm{Cost}(S^{-1})$. Consequently, if $\mathbb{S}$ is the cheapest (or a tie for the cheapest) transformation that converts X into Y, then $\mathbb{S}^{-1}$ is the cheapest (or a tie for the cheapest) transformation that converts Y into X. It readily follows that $D(X, Y) = D(Y, X)$.

Triangle inequality: Let X, Y, and Z be three time series. We need to show that $D(X, Z) \leq D(X, Y) + D(Y, Z)$. Let $\mathbb{S}_1$ be an optimal (i.e., lowest-cost) transformation of X into Y, so that $\mathrm{Cost}(\mathbb{S}_1) = D(X, Y)$. Similarly, let $\mathbb{S}_2$ be an optimal transformation of Y into Z, so that $\mathrm{Cost}(\mathbb{S}_2) = D(Y, Z)$. Let's define $\mathbb{S}_3$ to be the concatenation of $\mathbb{S}_1$ and $\mathbb{S}_2$, that first applies the sequence of operations in $\mathbb{S}_1$, and then applies the sequence of operations in $\mathbb{S}_2$. Then, $\mathrm{Transform}(X, \mathbb{S}_3) = Z$ and $\mathrm{Cost}(\mathbb{S}_3) = D(X, Y) + D(Y, Z)$.

If $\mathbb{S}_3$ is the cheapest (or a tie for the cheapest) transformation converting X into Z, then, $D(X, Z) = D(X, Y) + D(Y, Z)$, and the triangle inequality holds. If $\mathbb{S}_3$ is *not* the cheapest (or a tie for the cheapest) transformation converting X into Z, then $D(X, Z) < D(X, Y) + D(Y, Z)$, and the triangle inequality still holds.

3.1.1 Advantages of Metricity

Metricity distinguishes MSM from several alternatives, such as DTW [8], LCSS [10], MVM [11], and EDR [12]. Metricity allows MSM to be combined with an extensive arsenal of off-the-shelf, generic methods for indexing, clustering, and visualization, that have been designed to work in any metric space.

With respect to indexing, metricity allows the use of generic indexing methods designed for arbitrary metrics (see [16] for a review). Examples of such methods include VP-trees [17] and Lipschitz embeddings [18]. In fairness to competing non-metric alternatives, we should mention that several custom-made indexing methods have been demonstrated to lead to efficient retrieval

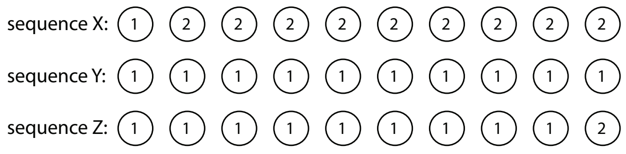

sequence X: 1 2 2 2 2 2 2 2 2 2
sequence Y: 1 1 1 1 1 1 1 1 1 1
sequence Z: 1 1 1 1 1 1 1 1 1 2

Fig. 2. An example where DTW violates the triangle inequality: $\text{DTW}(X,Y) = 9$, $\text{DTW}(X,Z) = 0$, $\text{DTW}(Z,Y) = 1$. Thus, $\text{DTW}(X,Z) + \text{DTW}(Z,Y) < \text{DTW}(X,Y)$.

using non-metric time series distance measures [9], [19], [20].

Another common operation in data mining systems is clustering. Metricity allows the use of clustering methods that have been designed for general metric spaces. Examples of such methods include [21], [22], [23].

Metricity also allows for better data visualization in time series datasets. Visualization typically involves an approximate projection of the data to two or three Euclidean dimensions, using projection methods such as, e.g., MDS [24], GTM [25], or FastMap [26]. In general, projections of non-Euclidean spaces to a Euclidean space, and especially to a low-dimensional Euclidean space, can introduce significant distortion [18]. However, non-metricity of the original space introduces an additional source of approximation error, which is not present if the original space is metric.

As an example, suppose that we want to project to 2D the three time series shown in Figure 2, so as to visualize the DTW distances among those three series. Any projection to a Euclidean space (which is metric) will significantly distort the non-metric relationship of those three time series. On the other hand, since MSM is metric, the three MSM distances between the three time series of Figure 2 can be captured exactly in a 2D projection.

3.1.2 An Example of Non-Metricity in DTW

To highlight the difference between MSM and DTW, Figure 2 illustrates an example case where DTW violates the triangle inequality. In that example, the only difference between Y and Z is in the last value, as $y_{10} = 1$ and $z_{10} = 2$. However, this small change causes the DTW distance from X to drop dramatically, from 9 to 0: $DTW(X,Y) = 9$, and $DTW(X,Z) = 0$.

In contrast, in MSM, to transform X into Y, we perform 8 Merge operations, to collapse the last 9 elements of X into a single value of 2, then a single Move operation that changes the 2 into a 1, and 8 Split operations to create 8 new values of 1. The cost of those operations is $16c + 1$. To transform X into Z, the only difference is that x_{10} does not need to change, and thus we only need 7 Merge operations, one Move operation, and 7 Split operations. The cost of those operations is $14c+1$. Thus, the small difference between Y and Z causes a small difference in the MSM distance values: $\text{MSM}(Y,Z) = 1$, $\text{MSM}(X,Y) = 16c + 1$, $\text{MSM}(X,Z) = 14c + 1$.

We should note that, in the above example, constrained DTW (cDTW) would not exhibit the extreme behavior of DTW. However, cDTW is also non-metric, and the more we allow the warping path to deviate from the diagonal, the more cDTW deviates from metric behavior. DTW itself is a special case of cDTW, where the diagonality constraint has been maximally relaxed.

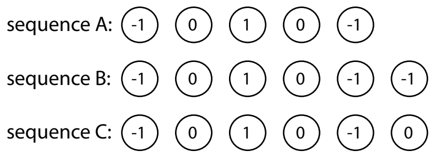


Fig. 3. An example illustrating the different behavior of MSM and ERP. Both sequences B and C are obtained by inserting one value at the end of A. According to ERP, A is closer to C than to B: $\text{ERP}(A,B) = 1$, $\text{ERP}(A,C) = 0$. According to MSM, A is closer to B than to C: $\text{MSM}(A,B) = c$, $\text{MSM}(A,C) = 1+c$.

3.2 Invariance to the Choice of Origin

Let $X = (x_1, \ldots, x_m)$ be a time series where each x_i is a real number. A translation of X by t, where t is also a real number, is a transformation that adds t to each element of the time series, to produce $X+t = (x_1+t, \ldots, x_m+t)$. If distance measure D is invariant to the choice of origin, then for any time series X, Y, and any translation t, $D(X,Y) = D(X+t, Y+t)$. The MSM distance is invariant to the choice of origin, because any transformation S that converts X to Y also converts $X + t$ to $Y + t$.

3.2.1 Contrast to ERP: Translation Invariance and Equal Treatment of All Values

Invariance to the choice of origin is oftentimes a desirable property, as in many domains the origin of the coordinate system is an arbitrary point, and we do not want this choice to impact distances and similarity rankings. In contrast, the ERP metric [13] is not invariant to the choice of origin. For the full definition of ERP we refer readers to [13].

To contrast MSM with ERP, consider a time series X consisting of 1000 consecutive values of v, for some real number v, and let Y be a time series of length 1, whose only value is a v as well. In ERP, to transform X into Y, we need to delete v 999 times. However, the cost of these deletions depends on the value of v: the cost is 0 if $v = 0$, and is $999v$ otherwise. In contrast, in MSM, the cost of deleting v 999 times (by applying 999 Merge operations) is independent of v. Thus, the MSM distance is translation-invariant (does not change if we add the same constant to both time series), whereas ERP is not.

A simple remedy for making ERP translation-invariant is to normalize each time series so that it has a mean value of 0. However, even in that case, the special treatment of the origin by ERP leads to insertion and deletion costs that are, in our opinion, counterintuitive in some cases. Such a case is illustrated in Figure 3. In that example, we define sequence $A = (-1, 0, 1, 0, -1)$. Then, we define sequences B and C, by copying A and inserting respectively a value of -1 and a value of 0 at

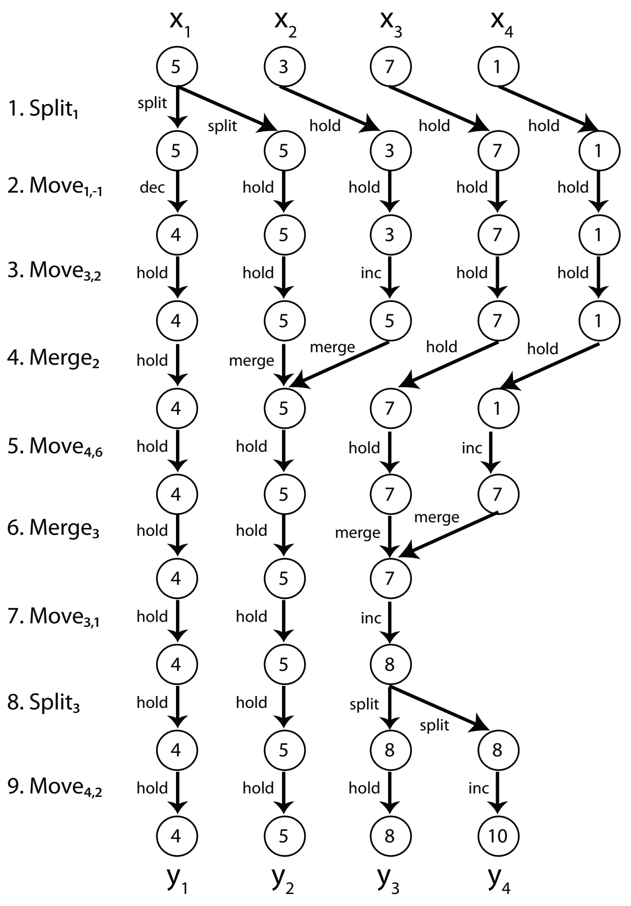


Fig. 4. An example of a (non-optimal, but easy-to-visualize) transformation that converts input time series $(5, 3, 7, 1)$ into output time series $(4, 5, 8, 10)$. We see the effects of each individual operation in the transformation, and we also see the step-by-step graph defined by applying this transformation to the input time series.

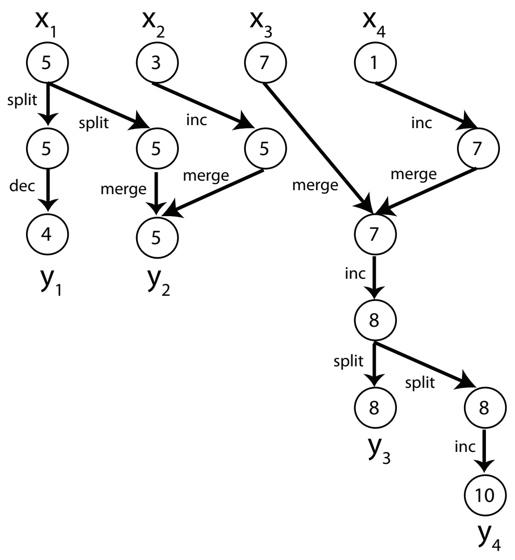


Fig. 5. The transformation graph corresponding to the step-by-step graph of Figure 4.

the end. According to ERP, A is closer to C than to B, and actually $ERP(A, C) = 0$, because ERP treats 0 (the origin) as a special value that can be inserted anywhere with no cost. In contrast, according to MSM, A is closer to B, as a single Split operation of cost c transforms A to B. Transforming A to C requires a Split and a Move, and costs $c + 1$.

This difference between MSM and ERP stems from the fact that, in MSM, the cost of inserting or deleting a value v only depends on the difference between v and its adjacent values in the time series. Thus, in MSM, inserting a 10 between two 10's is cheaper (cost = c) than inserting a 10 between two zeros (cost = $10+c$), and inserting a 10 between two zeros is as expensive (cost = $10 + c$) as inserting a 0 between two 10s. On the other hand, ERP treats values differently depending on how close they are to the origin: inserting a 10 between two 10's costs the same (cost = 10) as inserting a 10 between two zeros, and inserting a 10 between two zeros (cost = 10) is more expensive than inserting a 0 between two 10's (cost = 0).

4 Transformation Graphs and the Monotonicity Lemma

In Section 5 we describe an algorithm that computes the MSM distance between two time series. However, the correctness of that algorithm derives from certain theoretical observations. In this section we lay the theoretical groundwork for explaining the algorithm of Section 5.

4.1 Step-By-Step Graphs and Transformation Graphs

For any time series X and any transformation $\mathbb{S}$ we can draw what we call a **step-by-step graph**, that illustrates the intermediate results and the final result that we obtain, starting with X, and applying in sequence the operations of transformation $\mathbb{S}$. An example of such a graph is shown in Figure 4. In that figure, $X = (5, 3, 7, 1)$, and transformation $\mathbb{S}$ consists of nine operations, which are shown in detail. The final result of $\mathrm{Transform}(X, \mathbb{S})$ is time series $Y = (4, 5, 8, 10)$.

The step-by-step graph is a directed graph, that is divided into layers. The first layer corresponds to the input sequence X. Layer $k+1$, for $k > 0$ corresponds to the result of applying the first k operations of $\mathbb{S}$ on X. In intermediate layers, every node is connected to one or two parent nodes, and one or two children nodes. Every directed edge has a label that shows how the child node was obtained from the parent node. There are five types of edge labels:

- **HOLD:** A HOLD edge indicates that no Move, Split, or Merge operation was applied to the parent node.
- **INC:** An INC edge indicates that a Move operation was applied to the parent node, and that a positive value was added as a result of the move.
- **DEC:** A DEC edge indicates that a Move operation

was applied to the parent node, and that a a negative value was added as a result of the move.

- **SPLIT:** A Split operation generates two SPLIT edges, going from a parent node to two children nodes.
- **MERGE:** A Merge operation generates two MERGE edges, going from two parents to a common child.

In a step-by-step graph, most edges are typically HOLD edges. Given a step-by-step graph, we can obtain a significantly more concise graph, called a **transformation graph**, by applying the following process:

- Copy the original step-by-step graph.
- Delete all HOLD edges.
- Collapse into a single node any set of nodes that, in the original step-by-step graph, were connected by a path consisting exclusively of HOLD edges.

Figure 5 shows the transformation graph obtained from the step-by-step graph of Figure 4.

The **cost of a transformation graph** is defined to be the sum of the costs of the operations appearing in that graph. If a transformation $\mathbb{S}$ has G as its transformation graph, then $\mathbb{S}$ and G have the same cost. Similarly, the **cost of a path** in a transformation graph is defined to be the sum of the costs of the operations associated with the edges of the path.

Any step-by-step graph corresponds to one and only one sequence of operations, because the step-by-step graph imposes a full order on the set of operations appearing in that graph. On the other hand, a transformation graph imposes only a partial order on the set of operations appearing in that graph. Given a transformation graph G, a sequence of operations $\mathbb{S}$ has G as its transformation graph if:

- $\mathbb{S}$ contains all the operations appearing in G.
- $\mathbb{S}$ contains no operation that does not appear in G.
- The order of operations in $\mathbb{S}$ respects the partial order defined by G.

For example, in the graph of Figure 5, consider the move of the "3" node of the top layer to a "5", and the move of the "1" node of the top layer to a "7". The order of those two moves is interchangeable. On the other hand, the move of the "1" node to a "7" must occur before the move of the "7" to an "8".

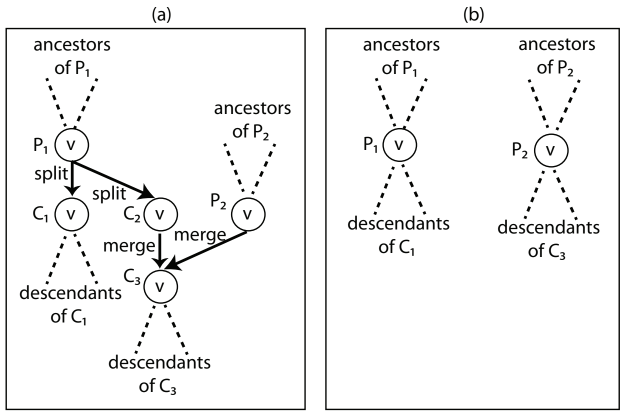


Fig. 6. Editing a transformation graph to delete consecutive SPLIT-MERGE edges. (a) A local region of a transformation graph, that includes consecutive SPLIT-MERGE edges. The numerical values stored in nodes P_1, P_2, C_1, C_2, C_3 must all be equal to the same real number v. (b) An edited but equivalent version of the region shown in (a). We note that nodes C_1, C_2 and C_3 have been deleted, P_1 is directly connected to what were the descendants of C_1 in (a), and P_2 is directly connected to what were the descendants of C_3 in (a).

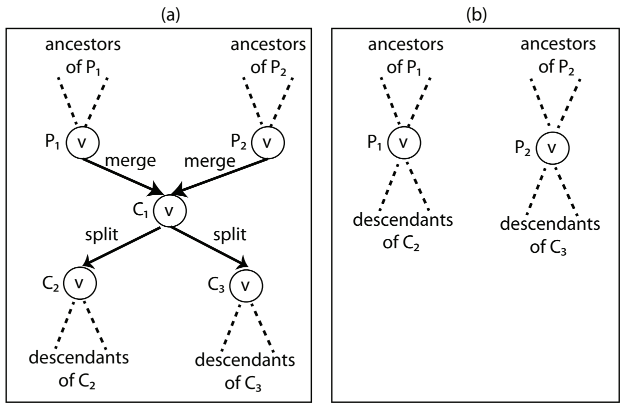


Fig. 7. Editing a transformation graph to delete consecutive MERGE-SPLIT edges. (a) A local region of a transformation graph, that includes consecutive MERGE-SPLIT edges. The numerical values stored in nodes P_1, P_2, C_1, C_2, C_3 must all be equal to the same real number v. (b) An edited but equivalent version of the region shown in (a). We note that nodes C_1, C_2 and C_3 have been deleted, P_1 is directly connected to what were the descendants of C_2 in (a), and P_2 is directly connected to what were the descendants of C_3 in (a).

4.2 The Monotonicity Lemma

Using transformation graphs, we can derive certain claims about transformations of a time series X into a time series Y. We will use these claims to derive an efficient algorithm for computing MSM distances.

We define two transformation graphs to be **equivalent transformation graphs** if they have the same input time series and output time series. Note that any transformation graph fully specifies an input time series X, an output time series Y, and a partially ordered set of operations that converts X into Y.

Proposition 1: Let G be a transformation graph that converts time series X into time series Y. If G includes any consecutive SPLIT-MERGE edges, we can convert G into an equivalent transformation graph G', such that G' is at least as cheap as G, and G' contains no consecutive SPLIT-MERGE edges.

Proof: There are two possible local topologies corresponding to consecutive SPLIT-MERGE edges. The first is the case where the Merge operation directly undoes the effects of the preceding Split operation. In that case, clearly these two operations cancel each other out and can be deleted without changing the output of the transformation.

Figure 6 illustrates the local topology corresponding to the second case. In that figure, the numerical values

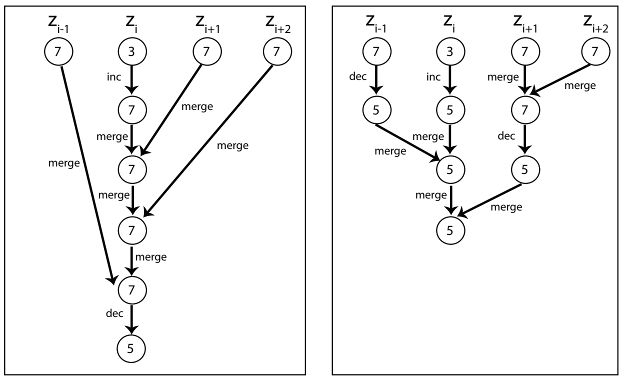


Fig. 8. Left: a local region of a transformation graph that includes a non-monotonic path, of the form INC-MERGE-MERGE-MERGE-DEC. This region transforms series $(7, 3, 7, 7)$ into single-element series (5). The cost is $6 + 3c$. Right: The result of converting the region shown on the left into an equivalent but monotonic region, with the same cost $6 + 3c$, following the description of Case 1 in the proof of Proposition 3.

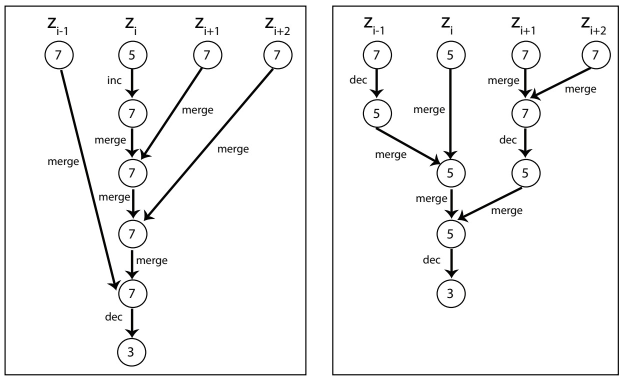


Fig. 9. Left: a local region of a transformation graph that includes a non-monotonic path, of the form INC-MERGE-MERGE-MERGE-DEC. This region transforms series $(7, 5, 7, 7)$ into single-element series (3). The cost is $6 + 3c$. Right: The result of converting the region shown on the left into an equivalent but monotonic region, with the same cost $6 + 3c$, following the description of Case 2 in the proof of Proposition 3. We note that no INC edges appear in the region on the right.

stored in nodes P_1, P_2, C_1, C_2, C_3 are all equal to the same value v, because of the definition of the Split and Merge operations. The consecutive Split and Merge operations have the net effect of converting two consecutive v values (stored in nodes P_1 and P_2) into two consecutive v values (of nodes C_1 and C_3), and thus they can be deleted without changing the output of the graph. By deleting those two operations and editing the local topology as shown in the figure, we obtain an equivalent transformation graph, that is cheaper than the original transformation graph by a difference of $2c$. $\square$

Proposition 2: Let G be a transformation graph that converts time series X into time series Y. If G includes any consecutive MERGE-SPLIT edges, we can convert G into an equivalent transformation graph G', such that G' is at least as cheap as G, and G' contains no consecutive MERGE-SPLIT edges.

Proof: Figure 7 illustrates the local topology corresponding to consecutive MERGE-SPLIT edges. The Merge operation merges two values of v into one, and the Split operation directly undoes the effects of the preceding Merge operation, by recreating two values of v. Thus, we can delete both the Merge and the Split operation without changing the final output of the transformation graph. $\square$

We define a path of a transformation graph to be a **monotonic path** if it does not contain both INC and DEC edges. We define a **monotonic transformation graph** to be a transformation graph that only contains monotonic paths. We define a **monotonic transformation** to be a transformation whose transformation graph is monotonic.

Proposition 3: Let X and Y be two time series. Let $\mathbb{S}$ be a transformation that converts X into Y. If $\mathbb{S}$ is not monotonic, we can convert $\mathbb{S}$ into another transformation $\mathbb{S}'$, that also converts X into Y, is as cheap or cheaper than $\mathbb{S}$, and is monotonic.

Proof: This proposition has a long proof, because we have to cover several different cases. We can assume that transformation $\mathbb{S}$ has already been processed as described in Propositions 1 and 2, so that there are no consecutive SPLIT-MERGE edges in the transformation graph. Also, any consecutive INC-DEC edges or DEC-INC edges are clearly suboptimal, and can be replaced with a single INC or DEC edge. So, we can ignore such cases from here on.

If the transformation graph is not monotonic, it must have a non-monotonic path. Then, the path must have a subpath, whose one end is an INC edge, the other end is a DEC edge, and the intermediate edges are either all of type MERGE or all of type SPLIT (based on Propositions 1 and 2). We will primarily consider the case where the path is of the form INC-MERGE-...-MERGE-DEC, because once we prove the proposition for that case, the proof for the other cases is straightforward. Two examples of an INC-MERGE-...-MERGE-DEC path and its surrounding local topology are illustrated in Figures 8 and 9. We advise the reader to refer to these examples while reading the remainder of this proof.

Since we can re-order operations in $\mathbb{S}$ into any ordering compatible with the partial order imposed by the transformation graph, we choose to use an order in which the operations specified by the INC-MERGE-...-MERGE-DEC path are applied consecutively. Let $Z = (z_1, \ldots, z_t)$ be the time series to which the first operation of the path is applied. In that case, the INC edge corresponds to some operation $\mathrm{Move}_{i,v}$, for some i and some positive v. This operation moves the i-th element of Z from value z_i to value $z_i + v$. Then, there is a sequence of Merge operations, that merge the i-th element with l elements $z_{i-l}, \ldots, z_{i-1}$, and with r elements $z_{i+1}, \ldots, z_{i+r}$, which all have the same value $z_i + v$. It is possible for either l or r to be equal to 0. In Figure 8, $l = 1$, $r = 2$, $z_i = 3$,

$v = 4$, and $z_{i-1} = z_{i+1} = z_{i+2} = 7$.

After all the Merge operations have been applied, elements $z_{i-l}, \ldots, z_{i+r}$ have been converted into a single element, with value $z_i + v$. The final DEC edge corresponds to changing value $z_i + v$ to a new value $z_i + v - v'$, where v' is a positive real number ($v' = 2$ in Figure 8). The net result of all those operations is merging elements $z_{i-l}, \ldots, z_{i+r}$ of time series Z into a single value $z_i + v - v'$. The overall cost of all these operations is $v + v' + (l + r) * c$, since we do two Move operations of magnitude v and v' respectively, and $l + r$ Merge operations. Our task is now to show that we can convert all elements $z_{i-l}, \ldots, z_{i+r}$ into a single element with value $z_i + v - v'$, with less than or equal cost, and without having a non-monotonic path.

We will consider two cases: $v \geq v'$, and $v < v'$.

Case 1: $v \geq v'$. Figure 8 illustrates an example of this case. Consider replacing the sequence of operations specified by the INC-MERGE-…-MERGE-DEC path with the following combination of operations:

1) We move z_i up to $z_i + v - v'$, with cost $v - v'$.
2) If $l > 0$, we merge elements $z_{i-l}, \ldots, z_{i-1}$ into a single element whose value is $z_i + v$, and we move that single element down to $z_i + v - v'$. The cost of these operations is $(l - 1) * c + v'$.
3) If $r > 0$, we merge elements $z_{i+1}, \ldots, z_{i+r}$ into a single element whose value is $z_i + v$, and we move that single element down to $z_i + v - v'$. The cost of these operations is $(r - 1) * c + v'$.
4) We merge the results of steps 1, 2, and 3 into a single element. The cost here is at most 2 * c, it can be less if $l = 0$ or $r = 0$.

Step 4 must take place after steps 1, 2, and 3, whereas the order of steps 1, 2, and 3 is not important. Overall, the total cost of the above four steps is $(l + r) * c + v + v'$, which is equal to the cost of the original INC-MERGE-…-MERGE-DEC path. In the special case where $l = 0$ or $r = 0$, the total cost becomes $(l + r) * c + v$, which is better than the original cost. At the same time, the local topology resulting from these changes to the transformation graph includes only monotonic paths. Furthermore, the resulting transformation is at least as cheap as the original transformation. Figure 8 shows an example of this process, the local topology corresponding to the original INC-MERGE-…-MERGE-DEC, and the local topology corresponding to the new combination of operations.

Case 2: $v < v'$. Figure 9 illustrates an example of this case. Consider replacing the sequence of operations specified by the INC-MERGE-…-MERGE-DEC path with the following combination of operations:

1) If $l > 0$, we merge elements $z_{i-l}, \ldots, z_{i-1}$ into a single element whose value is $z_i + v$, and we move that single element down to value z_i. The cost of these operations is $(l - 1) * c + v$.
2) If $r > 0$, we merge elements $z_{i+1}, \ldots, z_{i+r}$ into a single element whose value is $z_i + v$, and we move that single element down to value z_i. The cost of these operations is $(r - 1) * c + v$.
3) We merge z_i and the results of steps 1 and 2 into a single element, with value z_i. The cost here is 2 * c, or less if $l = 0$ or $r = 0$.
4) We move the result of step 3 down to final value $z_i + v - v'$, with cost $v' - v$.

Steps 1 and 2 can take place in any order, but step 3 must be taken after steps 1 and 2, and step 4 after step 3. The cost of these four steps is at most $(l + r) * c + v + v'$, so it is not greater than the cost of the original sequence of operations. At the same time, the local topology resulting from these changes to the transformation graph includes only monotonic paths. Furthermore, the resulting transformation is at least as cheap as the original transformation. Figure 9 shows an example of this process, the local topology corresponding to the original INC-MERGE-…-MERGE-DEC, and the local topology corresponding to the new combination of operations.

We can now briefly consider the remaining cases of non-monotonic paths. The proof for paths of the form form DEC-MERGE-…-MERGE-INC is a direct adaptation of the proof we provided for paths of the form INC-MERGE-…-MERGE-DEC. For paths of the form INC-SPLIT-…-SPLIT-DEC or DEC-SPLIT-…-SPLIT-INC, we use the fact that, as discussed in Section 3.1 (when demonstrating that MSM is symmetric), any transformation of X into Y can be inverted, to produce an equal-cost transformation of Y into X. Thus, if, for some transformation $\mathbb{S}$ of X into Y, the corresponding transformation graph contains a path of the form INC-SPLIT-…-SPLIT-DEC or DEC-SPLIT-…-SPLIT-INC, then for the inverse transformation $\mathbb{S}^{-1}$ of Y into X the transformation graph contains a path of the form INC-MERGE-…-MERGE-DEC or DEC-MERGE-…-MERGE-INC. We can edit $\mathbb{S}^{-1}$ to remove such paths, and then invert it again, to obtain a transformation that changes X into Y and that does not include paths of the form INC-SPLIT-…-SPLIT-DEC or DEC-SPLIT-…-SPLIT-INC.

At this point, we have shown that, for any type of non-monotonic path in a transformation graph, we can edit the graph so that the non-monotonic path is replaced with an arrangement of monotonic paths, and we have shown that the edited graph is equivalent to G and at least as cheap as G. By repeating such edits, we can convert any transformation graph G into an equivalent, monotonic, and at least as cheap transformation graph G', and thus we have concluded the proof of Proposition 3. □

We are now ready to state and prove the monotonicity lemma, which is a key lemma for describing, in Section 5, the algorithm for computing MSM distances.

Proposition 4: (Monotonicity lemma) For any two time series X and Y, there exists an optimal transformation that converts X into Y and that is monotonic.

Proof: Let $\mathbb{S}$ be an optimal transformation that converts X into Y. Let G be the transformation graph corresponding to applying $\mathbb{S}$ to X. If G is not monotonic, we can convert G to a monotonic graph G' that is at least as cheap as G (and thus also optimal), by editing G as described in the proofs of Propositions 1, 2, and 3. Then, any transformation S compatible with G' is an optimal and monotonic transformation of X into Y. $\square$

5 Computing the MSM Distance

Let $X = (x_1, \ldots, x_m)$ and $Y = (y_1, \ldots, y_n)$ be two time series. Figure 10 describes a simple dynamic programming algorithm for computing the MSM distance between X and Y. For each (i, j) such that $1 \leq i \leq m$ and $1 \leq j \leq n$, we define $\text{Cost}(i, j)$ to be the MSM distance between the first i elements of X and the first j elements of Y. This way, the MSM distance between X and Y is simply $\text{Cost}(m, n)$.

As the algorithm on Figure 10 shows, for $i > 1$ and $j > 1$, $\text{Cost}(i, j)$ can be computed recursively based on $\text{Cost}(i, j-1)$, $\text{Cost}(i-1, j)$, and $\text{Cost}(i-1, j-1)$. In this section we explain why it is correct to define the Cost function in this recursive manner, and we fully specify how to actually compute the Cost function.

First, we note that $\text{Cost}(1, 1)$ is simply the cost of moving x_1 to y_1, so this is a trivial case. The interesting case is when $i > 1$ or $j > 1$. In that case, we know from the monotonicity lemma that there exists an optimal monotonic transformation $\mathbb{S}_{i,j}$ converting $(x_1, \ldots, x_i)$ into $(y_1, \ldots, y_j)$. We use notation $G_{i,j}$ for the transformation graph corresponding to applying $\mathbb{S}_{i,j}$ to X. In $G_{i,j}$ there is a monotonic path moving x_i to y_j. There can be three cases for that path, that we need to analyze separately.

Case 1 (applicable if $i > 1$ and $j > 1$): the monotonic path taking x_i to y_j does not include any SPLIT or MERGE edges. In that case, without loss of generality, we can assume that the monotonic path taking x_i to y_j contains a single INC or DEC edge. We refer the reader to Figure 11 for an example.

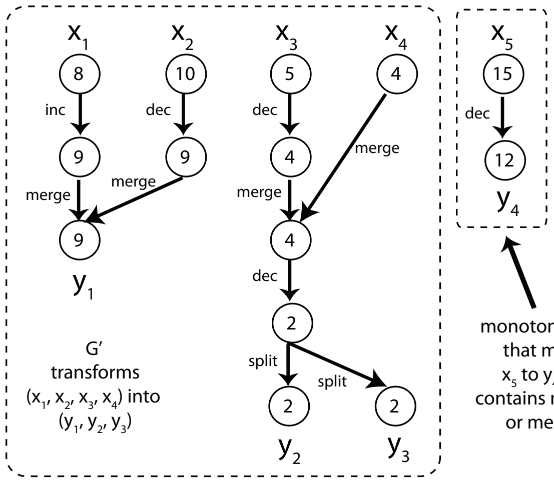
Consider the transformation graph G' that we obtain by removing the INC or DEC edge connecting x_i to y_j from transformation graph $G_{i,j}$. We show by contradiction that G' defines an optimal transformation of $(x_1, \ldots, x_{i-1})$ into $(y_1, \ldots, y_{j-1})$. If G' is not optimal, then there exists an optimal transformation $\mathbb{S}_1$ that has a smaller cost than G'. If we add a Move operation to the end of $\mathbb{S}_1$, that moves x_i to y_j, we obtain a transformation that converts $(x_1, \ldots, x_i)$ into $(y_1, \ldots, y_j)$ and that is cheaper than $\mathbb{S}_{i,j}$, which was assumed to be optimal. Therefore, we have reached a contradiction.

Consequently, if Case 1 holds, we obtain an optimal transformation $\mathbb{S}_{i,j}$ by adding a move operation (moving x_i to y_j) to an optimal transformation converting $(x_1, \ldots, x_{i-1})$ into $(y_1, \ldots, y_{j-1})$. It follows readily that, if Case 1 holds, $\text{Cost}(i, j) = \text{Cost}(i-1, j-1) + |x_i - y_j|$.

Case 2 (applicable if $i > 1$): in the monotonic path moving x_i to y_j, the first non-move operation is a Merge. In the transformation graph $G_{i,j}$, that first Merge operation creates a node M with two parents. One of those

```
function MSM_Distance($X$, $Y$)

Inputs:
  Time series $X = (x_1, \ldots, x_m)$
  Time series $Y = (y_1, \ldots, y_n)$

Initialization:
  $\text{Cost}(1,1) = |x_1 - y_1|$.
  For $i = 2, \ldots, m$:
    $\text{Cost}(i,1) = \text{Cost}(i-1,1) + C(x_i, x_{i-1}, y_1)$
  For $j = 2, \ldots, n$:
    $\text{Cost}(1,j) = \text{Cost}(1,j-1) + C(y_j, x_1, y_{j-1})$

Main Loop:
  For $i = 2, \ldots, m$:
    For $j = 2, \ldots, n$:
      $\text{Cost}(i,j) = \min\{ \text{Cost}(i-1,j-1) + |x_i - y_j|,$
                      $\text{Cost}(i-1,j) + C(x_i, x_{i-1}, y_j),$
                      $\text{Cost}(i,j-1) + C(y_j, x_i, y_{j-1})\}$

Output: The MSM distance $D(X,Y)$ is $\text{Cost}(m,n)$.
```

Fig. 10. A simple, quadratic-time algorithm for computing the MSM distance between two time series $X = (x_1, \ldots, x_m)$ and $Y = (y_1, \ldots, y_n)$. Function C, used in computing values for the Cost array, is defined in Equation 9.



x1
x2
x3
x4
x5
8
10
5
4
15
inc
dec
dec
dec
9
9
4
12
merge
merge
merge
merge
y4
9
4
y1
dec
2
G′ transforms (x1, x2, x3, x4) into (y1, y2, y3)
split
split
2
2
y2
y3
monotonic path that moves x5 to y4 and contains no splits or merges

Fig. 11. An example of Case 1 for an optimal monotonic transformation graph $G_{i,j}$. $G_{i,j}$ maps $(x_1, \ldots, x_i)$ to $(y_1, \ldots, y_j)$. In Case 1, $G_{i,j}$ is obtained from an optimal transformation graph G' mapping $(x_1, \ldots, x_{i-1})$ to $(y_1, \ldots, y_{j-1})$, by adding to G' a Move operation that moves x_i to y_j. In the example shown here, $i = 5$ and $j = 4$.

parents, that we call P_i, has x_i as an ancestor. The other parent, that we call P_{i-1}, has x_{i-1} as an ancestor. There is a path passing through P_{i-1} and M that connects x_{i-1} to y_j. There is another path passing through P_i and M that connects x_i to y_j. Since the transformation is monotonic, the value v stored at node M must be between x_{i-1} and y_j, and also between x_i and y_j. Figure 12 illustrates three examples, with the position of node M indicated.

For Case 2, there are three subcases that we need to

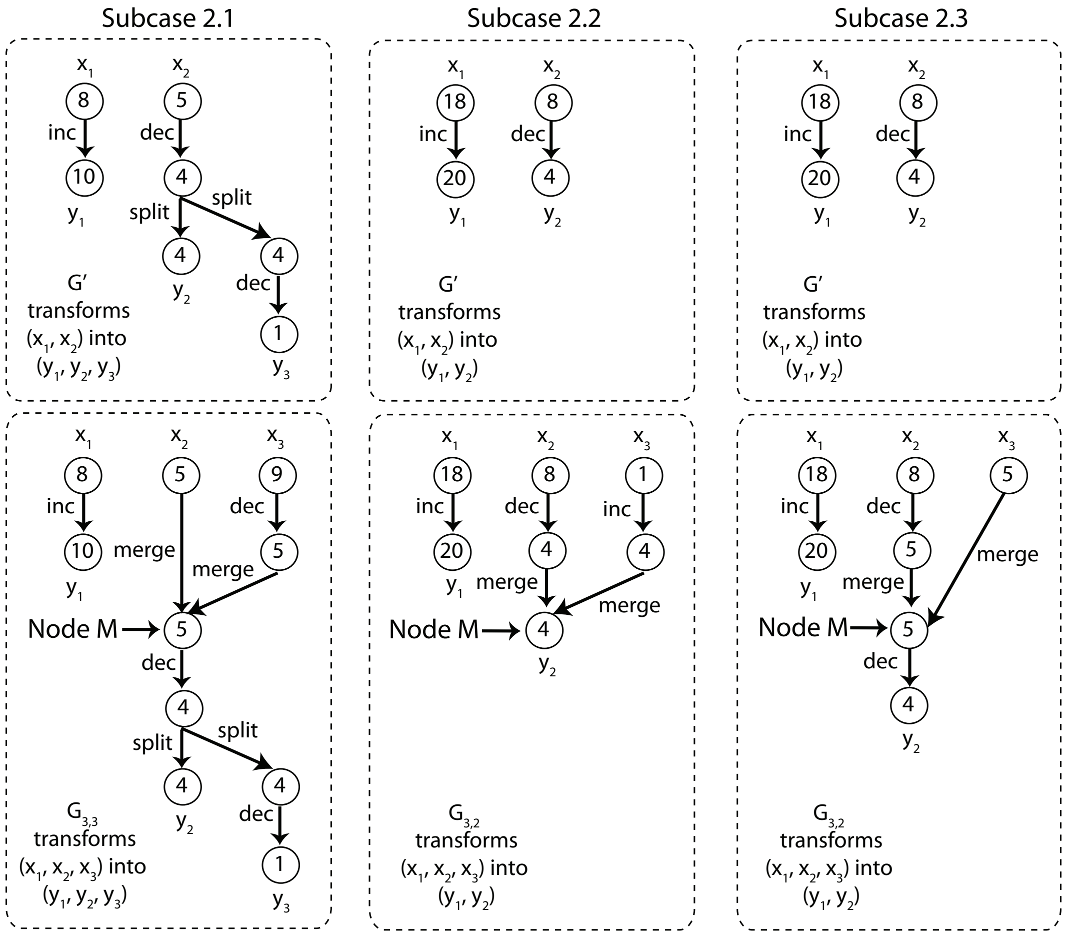

Subcase 2.1
Subcase 2.2
Subcase 2.3
x1 x2 8 5 inc dec 10 4 y1 split split 4 4 y2 dec 1 y3
G′ transforms (x1, x2) into (y1, y2, y3)
x1 x2 18 8 inc dec 20 4 y1 y2
G′ transforms (x1, x2) into (y1, y2)
G′ transforms (x1, x2) into (y1, y2)
x1 x2 x3 8 5 9 inc dec 10 merge merge 5 y1
Node M 5 dec 4 split split 4 4 y2 dec 1 y3
G3,3 transforms (x1, x2, x3) into (y1, y2, y3)
x1 x2 x3 18 8 1 inc dec inc 20 4 4 y1 merge merge Node M 4 y2
G3,2 transforms (x1, x2, x3) into (y1, y2)
x1 x2 x3 18 8 5 inc dec 20 5 merge y1 merge Node M 5 dec 4 y2
G3,2 transforms (x1, x2, x3) into (y1, y2)

Fig. 12. Examples of the three subcases of Case 2 for an optimal monotonic transformation graph $G_{i,j}$. $G_{i,j}$ maps $(x_1, \ldots, x_i)$ to $(y_1, \ldots, y_j)$. In Case 2, $G_{i,j}$ is obtained from an optimal transformation graph G' mapping $(x_1, \ldots, x_{i-1})$ to $(y_1, \ldots, y_j)$. Subcase 2.1: the value of x_i is not between the value of x_{i-1} and the value of y_j, and x_i is closer to x_{i-1} than to y_j. In the example for Subcase 2.1, $i = 3$ and $j = 3$. In Subcase 2.2, the value of x_i is not between the value of x_{i-1} and the value of y_j, and x_i is closer to y_j than to x_{i-1}. In the example for Subcase 2.2, $i = 3$ and $j = 2$. In Subcase 2.3, the value of x_i is between the value of x_{i-1} and the value of y_j. In the example for Subcase 2.3, $i = 3$ and $j = 2$. Note that, in this example, in the optimal transformation from (x_1, x_2) to (y_1, y_2), x_2 moves directly from value 8 to value 4. In the optimal transformation from (x_1, x_2, x_3) to (y_1, y_2), x_2 moves first to an intermediate value of 5, that allows a merge with x_3, and then to value 4.

address. An example for each subcase is shown in Figure 12.

- **Subcase 2.1:** the value of x_i is not between the value of x_{i-1} and the value of y_j, and x_i is closer to x_{i-1} than to y_j. Then, x_i first moves to value x_{i-1}, and then merges.
- **Subcase 2.2:** the value of x_i is not between the value of x_{i-1} and the value of y_j, and x_i is closer to y_j than to x_{i-1}. Then, x_i first moves to value y_j, and then merges.
- **Subcase 2.3:** the value of x_i is between the value of x_{i-1} and the value of y_j. In that case, x_i merges immediately with a value along the monotonic path that moves x_{i-1} to y_j.

In all three subcases, by removing the one or two operations linking x_i with node M from the transformation graph $G_{i,j}$, we obtain a transformation graph G' that converts $(x_1, \ldots, x_{i-1})$ into $(y_1, \ldots, y_j)$. As in Case 1, we can show that if G' is suboptimal, then $G_{i,j}$ is suboptimal (which is a contradiction). Consequently, G' is optimal, and if Case 2 holds then $\text{Cost}(i, j) = \text{Cost}(i-1, j) + C(x_i, x_{i-1}, y_j)$, where $C(x_i, x_{i-1}, y_j)$ is defined as follows:

$$C(x_i, x_{i-1}, y_j) = \left\{ \begin{array}{l} c \;\text{ if } x_{i-1} \leq x_i \leq y_j \text{ or } x_{i-1} \geq x_i \geq y_j \\ c + \min(|x_i - x_{i-1}|, |x_i - y_j|) \text{ otherwise} \end{array} \right. \tag{9}$$

In Figure 12, for Subcase 2.3 in particular, we should note that the transformation graph obtained by removing the Merge operation from the bottom graph is *not* identical to the top graph. However, both graphs have equal cost. The only difference is that in the top graph x_{i-1} moves directly from a value of 8 to a value of 4, and in the bottom graph x_{i-1} moves first to an intermediate value of 5, and then to the final value of 4.

Case 3 (applicable if $j > 1$): in the monotonic path moving x_i to y_j, the first non-move operation is a Split. We omit the details here, but the analysis for this case is a direct adaptation of the analysis for Case 2. In summary, in Case 3 we can obtain from transformation graph $G_{i,j}$ an optimal transformation graph G' that converts $(x_1, \ldots, x_i)$ into $(y_1, \ldots, y_{j-1})$, so that $\mathrm{Cost}(i, j) = \mathrm{Cost}(i, j-1) + C(y_j, x_i, y_{j-1})$.

Based on the above considerations, the algorithm on Figure 10 checks which of the three cases leads to a cheaper transformation of $(x_1, \ldots, x_i)$ into $(y_1, \ldots, y_j)$. The cost of the transformation corresponding to each case is computed in a few operations, using the already computed values for $\mathrm{Cost}(i, j-1)$, $\mathrm{Cost}(i-1, j)$, and $\mathrm{Cost}(i-1, j-1)$. The algorithm for computing MSM distances is fairly simple, and can be implemented in a few lines of code. We have posted Matlab, Java, and C++ impelementations of the $\mathrm{MSM\_Distance}$ function on the web, at two mirrored sites:

- `http://omega.uta.edu/~athitsos/msm/`
- `http://vlm1.uta.edu/~alex/msm/`

Computing $\mathrm{Cost}(i, j)$ takes constant time for each (i, j). Therefore, the time complexity of computing the MSM distance is $O(mn)$. The $O(mn)$ complexity is the same as the time complexity of DTW (without the diagonality constraint [9]) and ERP. The Euclidean distance, in contrast, has linear time complexity $O(m)$, and $n = m$ in that case. Constrained DTW [9], that utilizes the diagonality constraint, also has linear time complexity if we consider that the radius around the diagonal does not depend on the length of the time series.

6 EXPERIMENTS

We compare MSM to cDTW, DTW, ERP, and the Euclidean distance, based on the 1-nearest neighbor classification error rate attained on the 20 time series datasets available on the UCR time series archive [15]. We should note that, while the UCR time series website shows results on 22 datasets, only 20 of those datasets are publicly available, and those are the 20 datasets that we have used. A note on the website indicates that two of those datasets (namely, the "Car" and "Plane" datasets) are still not publicly available.

The MSM algorithm has one free parameter, namely c, the cost of every Split and Merge operation. For each of the 20 datasets, the value for c was chosen from the set $\{0.01, 0.1, 1, 10, 100\}$, using leave-one-out cross-validation on the training set. It is important to emphasize that c was *not* optimized based on results on the test data. Overall we have found it fairly straightforward to pick a value for c by simply trying those five values on the training data.

We should note that considering a lot of possible values for c could slow down the training phase significantly, as a separate cross-validation measurement must be obtained for each individual value. In our experiments, MSM produced competitive error rates while considering only five widely-spaced values (differing by factors of 10) for c. Considering only five widely-spaced values demonstrates that no careful finetuning of c was needed to obtain good results.

Table 1 shows the error rate for each method on each dataset. The table also shows characteristics of each dataset, the parameter values used by MSM and cDTW for that dataset, and the statistical significance (p-value) of the results. The p-value specifically measures the statistical significance of the difference between the top two methods for each dataset.

We note that for each method there are some datasets where that method is at least as accurate as the other four methods. MSM produces lower error rates than its competitors in 10 datasets. Each of DTW and ERP produces the lowest error rate in two datasets. In the remaining six datasets, two or more methods tie for lowest error rate. Table 2 shows, for each competitor of MSM, the number of datasets where MSM produces respectively better accuracy, equal accuracy, and worse accuracy compared to the competitor.

Our primary goal in these experiments has been to demonstrate that MSM has competitive performance on 1-nearest neighbor classification, compared to cDTW, DTW, and ERP. We are *not* making a claim that MSM is a fundamentally more accurate measure than cDTW, DTW, or ERP. Our interpretation of the results is that all these methods are valuable, and any one of them may outperform the other methods in a new dataset. At the same time, MSM has some attractive theoretical properties that DTW or ERP do not have.

A natural question to ask is how to determine which of these methods to use on a new dataset. A simple answer to that question is to evaluate all methods on the training set (using leave-one-out cross-validation), and choose the method with the lowest error rate. We have tried that approach, and we show the results on the rightmost two columns of Table 1. If two or more methods tied on the training set, we show the average test error of those methods. We tried two variants: in the CV+MSM variant, we chose for each dataset the best out of all five methods. In the CV-MSM variant we excluded MSM from consideration.

In those results, CV+MSM matched the best error rate in 12 datasets and CV-MSM matched the best error rate (excluding MSM) in 11 datasets. In head-to-head comparison with each of the individual methods they included, both CV+MSM and CV-MSM gave better results in more datasets than they gave worse results. Both CV+MSM and CV-MSM had lower average error rates than any of the individual methods that they included. Thus, these results demonstrate that cross-validation is a good way to choose automatically which method to use in each dataset. Furthermore, we note that CV+MSM had a lower error rate than CV-MSM in 10 datasets, and higher error rate in only three datasets. This result further illustrates the advantages of considering MSM as an alternative to DTW and ERP in practical applications.

In Figures 13, 14 and 15 we illustrate some specific examples where MSM gives better or worse accuracy

TABLE 1
1-nearest neighbor classification error rates attained by MSM, constrained DTW (denoted as cDTW), unconstrained DTW (denoted as DTW), ERP, and the Euclidean distance, on each of the 20 datasets in the UCR repository of time series datasets [15]. The last row indicates the average error rate over all 20 datasets. In addition, the table shows for each dataset: the number of classes, the number of training objects, the number of test objects, the length of each sequence in the dataset, the value of c used by MSM on that dataset, and the length of the warping window (as specified in [15]) used by cDTW on that dataset. We also show, for each dataset, the statistical significance(p-value) of the difference between the two best-performing methods for that dataset. The last two columns show the results of the CV+MSM and CV-MSM hybrid methods, described in the text, where the distance measure used for each dataset is the one that minimizes training error.

Dataset	class num.	train. size	test size	seq. length	MSM	cDTW	DTW	ERP	Euclid	p value	MSM c	cDTW param.	CV +MSM	CV -MSM
Cofee	2	28	28	286	0.236	**0.179**	**0.179**	0.25	0.25	0.5	0.01	3	0.179	0.179
CBF	3	30	900	128	0.012	0.004	**0.003**	**0.003**	0.148	0.5	0.1	11	0.006	**0.003**
ECG	2	100	100	96	**0.11**	0.12	0.23	0.13	0.12	0.3285	1	0	**0.117**	0.12
Synthetic	6	300	300	60	0.027	0.017	**0.007**	0.037	0.12	0.2076	0.1	6	0.007	0.007
Gun Point	2	50	150	150	0.06	0.087	0.093	**0.04**	0.087	0.1595	0.01	0	**0.078**	0.087
FaceFour	4	24	88	350	**0.057**	0.114	0.17	0.102	0.216	0.0224	1	2	**0.057**	0.114
Lightning-7	7	70	73	319	**0.233**	0.288	0.274	0.301	0.425	0.2476	1	5	0.288	0.288
Trace	4	100	100	275	0.07	0.01	**0**	0.17	0.24	0.1599	0.01	3	0	0
Adiac	37	390	391	176	0.384	0.391	0.396	**0.379**	0.389	0.3276	1	3	0.384	**0.379**
Beef	5	30	30	30	0.5	**0.467**	0.5	0.5	**0.467**	0.5	0.1	0	0.467	0.467
Lightning-2	2	60	61	637	0.164	**0.131**	**0.131**	0.148	0.246	0.5	0.01	6	0.131	0.131
OliveOil	4	30	30	570	0.167	0.167	**0.133**	0.167	**0.133**	0.5	0.01	1	0.167	0.167
OSU Leaf	6	200	242	427	**0.198**	0.384	0.409	0.397	0.483	< 0.0001	0.1	7	**0.198**	0.384
SwedishLeaf	15	500	625	128	**0.104**	0.157	0.21	0.12	0.213	0.0703	1	2	**0.104**	0.157
Fish	7	175	175	463	**0.08**	0.16	0.167	0.12	0.217	0.0448	0.1	4	**0.08**	0.16
FaceAll	14	560	1690	131	**0.189**	0.192	0.192	0.202	0.286	0.2243	1	3	**0.189**	0.197
50words	50	450	455	270	**0.196**	0.242	0.31	0.281	0.369	0.01	1	6	**0.196**	0.242
Two Patterns	4	1000	4000	128	0.001	0.0015	**0**	**0**	0.09	0.5	1	4	0.0003	**0**
Wafer	2	1000	6174	152	**0.004**	0.005	0.02	0.011	0.005	0.2249	1	1	**0.008**	0.011
Yoga	2	300	3000	426	**0.143**	0.155	0.164	0.147	0.17	0.2207	0.1	2	**0.143**	0.155
average					0.147	0.164	0.179	0.175	0.234				0.140	0.162

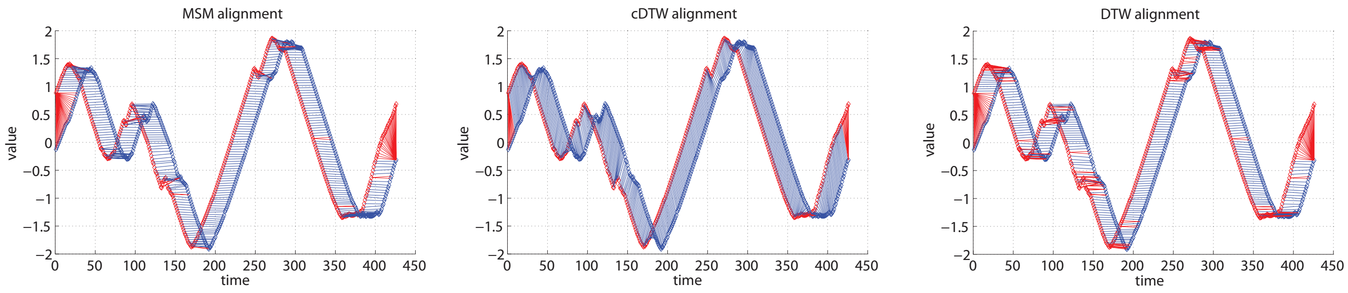


Fig. 13. An example, from the Yoga dataset, of a query that MSM classifies correctly whereas cDTW and DTW classify incorrectly, due to time shift. The query series is shown in blue. Its nearest neighbor according to MSM (which belongs to the same class) is shown in red. The alignments computed by MSM (left), cDTW (middle), and DTW (right) are shown via links connecting corresponding elements.

compared to its competitors. These examples help build some intuition about how the behavior of different methods can influence classification results.

Figure 13 shows an example where MSM classifies the query correctly, whereas cDTW and DTW give the wrong answer. The main difference between the query and its MSM-based nearest neighbor is time shift, which causes mismatches at the beginning and the end of the sequences. MSM erases (via small moves and merges) the mismatched points with relatively low cost. In DTW, the cost of matching the extra points prevents this training object from being the nearest neighbor of the query. The time shift affects cDTW even more severely, as the warping window is too small to compensate for the shift.

Figure 14 shows another example where the query is classified correctly by MSM, and incorrectly by cDTW and DTW. Here, the query contains a valley between times 80 and 100, and that valley is not matched well by the query's MSM-based nearest neighbor. MSM "collapses" the mismatched valley to a single point with relatively low cost. In DTW, the cost of matching elements of the training object to points in that valley is large enough to prevent this training object from being the nearest neighbor of the query.

Figure 15 shows a case where MSM gives the wrong answer, whereas cDTW, DTW and ERP give the right

TABLE 2
We indicate the number of UCR datasets for which MSM produced better, equal, or worse accuracy compared to ERP, and also compared to DTW.

	MSM better	Tie	MSM worse
MSM vs. cDTW	13	1	6
MSM vs. DTW	12	1	7
MSM vs. ERP	13	2	5
MSM vs. Euclidean	18	0	2

TABLE 3
Runtime efficiency comparisons. For each dataset, in the MSM time column, the time it took in seconds to compute *all* distances from the entire test set to the entire training set. In the rightmost four columns we show the factor by which MSM was *slower* than each of cDTW, DTW, ERP, and the Euclidean distance.

Dataset	MSM time (sec)	cDTW factor	DTW factor	ERP factor	Euclidean factor
Cofee	1.59	13.25	1.49	1.42	159
CBF	16.05	3.77	1.51	1.61	94.41
ECG	2.88	3.56	1.55	1.46	48
Synthetic	10.89	8.71	1.51	1.35	36.3
Gun Point	4.14	10.89	1.48	1.18	103.5
FaceFour	9.59	17.76	1.99	0.97	479.5
Lightning-7	14.17	14.61	1.56	1.15	472.33
Trace	30.89	1.98	1.38	1.74	514.83
Adiac	103.33	2.52	1.28	1.03	178.16
Beef	6.11	2.54	1.14	0.97	611
Lightning-2	51.62	2.76	1.28	1.23	1720.67
OliveOil	8.83	1.89	1.04	1.03	883
OSU Leaf	259.16	2.94	1.27	1.09	959.85
SwedishLeaf	125.31	10.55	1.38	1.16	113.92
Fish	183.3	2.28	1.17	1.06	1018.33
FaceAll	491.56	12.89	1.25	1.28	111.46
50words	323.13	3.13	1.3	1.07	359.03
Two Patterns	2348.1	9.33	1.6	1.56	157.91
Wafer	3281.24	11.05	1.51	1.18	148
Yoga	4606.48	2.43	1.25	1.07	988.52
min		1.890	1.040	0.970	36.300
max		17.760	1.990	1.740	1,720.670
median		3.665	1.380	1.170	268.595
average		6.942	1.397	1.231	457.886

answer. For that query, we show both its MSM-based nearest neighbor (denoted as D), which belongs to the wrong class, as well as its MSM-based nearest neighbor (denoted as S) among training examples of the same class as the query. The main difference between the query and D is a peak and a valley that the query exhibits between time 200 and time 250. This difference gets penalized by DTW, cDTW, and ERP, and thus, according to those measures the query is closer to S than to D. On the other hand, the MSM distance between the query and D is not affected much by the extra peak and valley of the query. Thus, according to MSM, the query is closer to D than to S.

Figures 14 and 15 indicate that MSM penalizes extra peaks and valleys less severely than cDTW, DTW, and ERP. This may be a desirable property in data where such extra peaks and valleys appear due to outlier ob-

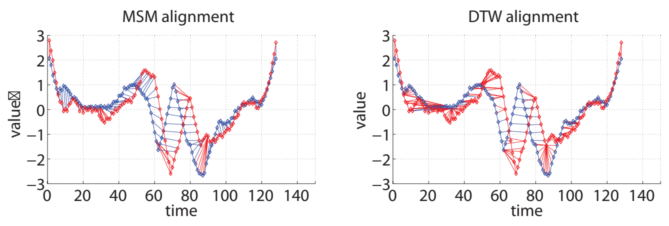


Fig. 14. An example from the Swedish Leaf dataset, where MSM does better than DTW. The query series is shown in blue. Its nearest neighbor according to MSM is shown in red, and belongs to the same class as the query. For MSM (left) and and DTW (right), the alignment between the red and the blue series is shown via links connecting corresponding elements.

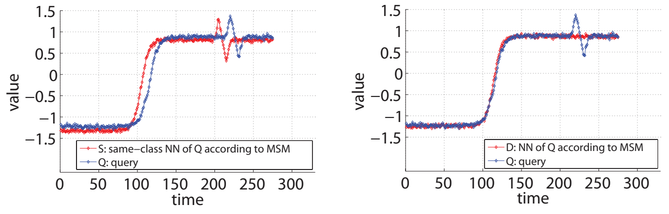


Fig. 15. An example from the Trace dataset where MSM does worse than DTW and ERP. On the left, we show in blue Q, a query series, and in red S, the nearest neighbor (according to MSM) of Q among training examples of the same class as Q. On the right, we show in blue the same query Q, and in red we show D, the overall nearest neighbor (according to MSM), which belongs to a different class.

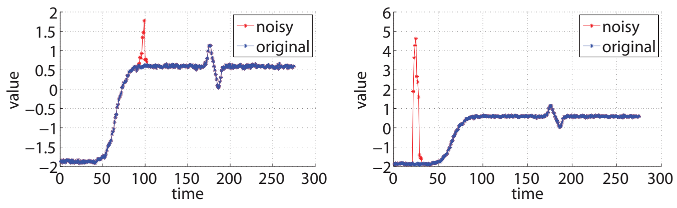


Fig. 16. Two examples of a peak added to time series. In blue we show the original time series. The modified version is the same as the original time series, except for a small region (shown in red) of 10 values, where we have added a peak.

servations. We simulated this situation in the following experiment: for each test example of each of the 20 UCR datasets, we modified that example by adding an extra peak. The width of the peak was 10 elements, and the height of the peak was chosen randomly and uniformly between 0 and 80. Two examples of this modification are shown on Figure 16. We measured the error rates of MSM and its competitors on this modified dataset. We note that the training examples were not modified, and thus the free parameters chosen via cross-validation for MSM and cDTW remained the same.

Due to lack of space, the table of error rates for this experiment is provided as supplementary material. The summary of those results is that, while the average error rates of all methods increase, MSM suffers significantly less than its competitors. MSM gives lower error rate than cDTW, DTW, and the Euclidean distance on all 20 datasets. Compared to ERP, MSM does better on 16 datasets, worse in 3 datasets, and ties ERP in 1 dataset.

Finally, Table 3 compares the efficiency of MSM to that of its competitors. As expected, the Euclidean distance and cDTW are significantly faster than MSM, DTW, and ERP. In all datasets the running time for MSM was between 0.97 and 2 times the running time of DTW and ERP. Running times were measured on a PC with 64-bit Windows 7, an Intel Xeon CPU running at 2GHz, 4GB of RAM, and using a single-threaded implementation.

7 Conclusions

We have described MSM, a novel metric for time series, that is based on the cost of transforming one time series into another using a sequence of individual Move, Split, and Merge operations. MSM has the attractive property of being both metric and invariant to the choice of origin, whereas DTW is not metric, and ERP is not invariant to the choice of origin. These properties may make MSM a more appealing choice, compared to existing alternatives, in various domains. Metricity, in particular, allows the use of a large number of existing tools for indexing, clustering and visualization, that have been designed to work in arbitrary metric spaces.

We have presented a quadratic-time algorithm for computing the MSM distance between two time series. A large part of the paper has been dedicated to explaining the algorithm and proving its correctness. At the same time, despite the relatively complex proof, the actual algorithm is quite short and easy to implement, as shown on Figure 10, and on the implementations we have posted online.

Experiments on all 20 datasets available at the UCR time series archive [15] demonstrate that, in ten of the 20 datasets, MSM produces lower nearest neighbor classification error rate than constrained DTW, unconstrained DTW, ERP, and the Euclidean distance. The fact that MSM gave the best accuracy in several datasets supports the conclusion that MSM is a method worth being aware of and experimenting with, in domains where practitioners currently use DTW or ERP. The attractive theoretical properties of MSM are an additional factor that can make MSM an appealing choice, compared to existing alternatives.

Acknowledgements

The research reported here has been partially funded by grants from the National Science Foundation: IIS-0705749, IIS-0812601, IIS-1055062, CNS-0923494, and CNS-1035913. This work has also been supported by a UTA startup grant to Professor Athitsos, and UTA STARS awards to Professors Chris Ding and Fillia Makedon.

References

[1] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, "Fast subsequence matching in time-series databases." in *ACM International Conference on Management of Data (SIGMOD)*, 1994, pp. 419–429.

[2] K.-P. Chan and A. W.-C. Fu, "Efficient time series matching by wavelets," in *IEEE International Conference on Data Engineearing (ICDE)*, 1999, pp. 126–133.

[3] Y. Moon, K. Whang, and W. Han, "General match: a subsequence matching method in time-series databases based on generalized windows." in *ACM International Conference on Management of Data (SIGMOD)*, 2002, pp. 382–393.

[4] Y. Moon, K. Whang, and W. Loh, "Duality-based subsequence matching in time-series databases." in *IEEE International Conference on Data Engineering (ICDE)*, 2001, pp. 263–272.

[5] T. Argyros and C. Ermopoulos, "Efficient subsequence matching in time series databases under time and amplitude transformations." in *International Conference on Data Mining*, 2003, pp. 481–484.

[6] D. Rafiei and A. O. Mendelzon, "Similarity-based queries for time series data." in *ACM International Conference on Management of Data (SIGMOD)*, 1997, pp. 13–25.

[7] H. Wu, B. Salzberg, G. C. Sharp, S. B. Jiang, H. Shirato, and D. R. Kaeli, "Subsequence matching on structured time series data." in *ACM International Conference on Management of Data (SIGMOD)*, 2005, pp. 682–693.

[8] J. B. Kruskal and M. Liberman, "The symmetric time warping algorithm: From continuous to discrete," in *Time Warps*. Addison-Wesley, 1983.

[9] E. Keogh, "Exact indexing of dynamic time warping," in *International Conference on Very Large Databases (VLDB)*, 2002, pp. 406–417.

[10] M. Vlachos, D. Gunopulos, and G. Kollios, "Discovering similar multidimensional trajectories," in *IEEE International Conference on Data Engineering (ICDE)*, 2002, pp. 673–684.

[11] L. Latecki, V. Megalooikonomou, Q. Wang, R. Lakämper, C. Ratanamahatana, and E. Keogh, "Elastic partial matching of time series," in *European Conference on Principles of Data Mining and Knowledge Discovery (PKDD)*, 2005, pp. 577–584.

[12] L. Chen, M. T. Özsu, and V. Oria, "Robust and fast similarity search for moving object trajectories," in *ACM International Conference on Management of Data (SIGMOD)*, 2005, pp. 491–502.

[13] L. Chen and R. T. Ng, "On the marriage of lp-norms and edit distance," in *International Conference on Very Large Databases (VLDB)*, 2004, pp. 792–803.

[14] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics*, vol. 10, no. 8, pp. 707–710, 1966.

[15] E. Keogh, "The UCR time series data mining archive. http://www.cs.ucr.edu/ eamonn/tsdma/index.html," 2006. [Online]. Available: http://www.cs.ucr.edu/ eamonn/TSDMA/index.html

[16] G. R. Hjaltason and H. Samet, "Index-driven similarity search in metric spaces," *ACM Transactions on Database Systems (TODS)*, vol. 28, no. 4, pp. 517–580, 2003.

[17] P. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in *ACM-SIAM Symposium on Discrete Algorithms*, 1993, pp. 311–321.

[18] G. Hjaltason and H. Samet, "Properties of embedding methods for similarity searching in metric spaces," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 25, no. 5, pp. 530–549, 2003.

[19] M. Morse and J. Patel, "An efficient and accurate method for evaluating time series similarity," in *ACM International Conference on Management of Data (SIGMOD)*, 2007, pp. 569–580.

[20] Y. Sakurai, M. Yoshikawa, and C. Faloutsos, "FTW: fast similarity search under the time warping distance," in *Principles of Database Systems (PODS)*, 2005, pp. 326–337.

[21] N. Brisaboa, O. Pedreira, D. Seco, R. Solar, and R. Uribe, "Clustering-based similarity search in metric spaces with sparse spatial centers," in *SOFSEM 2008: Theory and Practice of Computer Science*, ser. Lecture Notes in Computer Science, V. Geffert, J. Karhumaki, A. Bertoni, B. Preneel, P. Navrat, and M. Bielikova, Eds. Springer Berlin / Heidelberg, 2008, vol. 4910, pp. 186–197.

[22] V. Ganti, R. Ramakrishnan, J. Gehrke, A. L. Powell, and J. C. French, "Clustering large datasets in arbitrary metric spaces," in *IEEE International Conference on Data Engineering (ICDE)*, 1999, pp. 502–511.

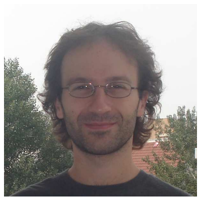
[23] P. Indyk, "A sublinear time approximation scheme for clustering in metric spaces," in *Proceedings of Annual Symposium on Foundations of Computer Science (FOCS)*, 1999, pp. 154–159.

[24] I. Borg and P. Groenen, *Modern Multidimensional Scaling: Theory and Applications*. Springer, 2005.

[25] C. M. Bishop, M. Svensén, and C. K. I. Williams, "GTM: The generative topographic mapping," *Neural Computation*, vol. 10, no. 1, pp. 215–234, 1998.

[26] C. Faloutsos and K. I. Lin, "FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets," in *ACM International Conference on Management of Data (SIGMOD)*, 1995, pp. 163–174.

Alexandra Stefan received the BS degree in mathematics and computer science from the University of Bucharest in 2002, and the MS degree in computer science from Boston University in 2008. She is currently a Ph.D. candidate at Computer Science and Engineering Department of the University of Texas at Arlington, and a member of the Vision-Learning-Mining Lab. Her research interests include computer vision and data mining. Her recent work has focused on efficient similarity-based retrieval in multimedia databases, with applications to sign language recognition, time series analysis, and recognition of a large number of classes.



Vassilis Athitsos received the BS degree in mathematics from the University of Chicago in 1995, the MS degree in computer science from the University of Chicago in 1997, and the PhD degree in computer science from Boston University in 2006. In 2005-2006 he worked as a researcher at Siemens Corporate Research, developing methods for database-guided medical image analysis. In 2006-2007 he was a postdoctoral research associate at the Computer Science department at Boston University. Since August 2007 he is an assistant professor at the Computer Science and Engineering department at the University of Texas at Arlington. His research interests include computer vision, machine learning, and data mining. His recent work has focused on gesture and sign language recognition, detection and tracking of humans using computer vision, efficient similarity-based retrieval in multimedia databases, shape modeling and detection, and medical image analysis. His research has been supported by the National Science Foundation, including an NSF CAREER award.

Gautam Das is Professor at the CSE department of UT-Arlington. Prior to joining UTA in Fall 2004, Dr. Das has held positions at Microsoft Research, Compaq Corporation and the University of Memphis. He graduated with a B.Tech in computer science from IIT Kanpur, India, and with a Ph.D in computer science from the University of Wisconsin, Madison. Dr. Das's research interests span data mining, information retrieval, databases, algorithms and computational geometry. He is currently interested in ranking, top-k query processing, and sampling problems in databases, as well as data management problems in the deep web, P2P and sensor networks, social networks, blogs and web communities. His research has resulted in over 130 papers, receiving several awards, including the IEEE ICDE 2012 Influential Paper award, VLDB Journal special issue on Best Papers of VLDB 2007, Best Paper of ECML/PKDD 2006, and Best Paper (runner up) of ACM SIGKDD 1998. He is on the Editorial Board of the journals ACM TODS and IEEE TKDE. He has served as General Chair of ICIT 2009, Program Chair of COMAD 2008, CIT 2004 and SIGMOD-DMKD 2004, and Best Paper Awards Chair of ACM SIGKDD 2006. Dr. Das's research has been supported by grants from National Science Foundation, Office of Naval Research, Department of Education, Texas Higher Education Coordinating Board, Microsoft Research, Nokia Research, Cadence Design Systems and Apollo Data Technologies.