Data Mining and Knowledge Discovery (2023) 37:595–626
https://doi.org/10.1007/s10618-022-00908-2

On computing exact means of time series using the move-split-merge metric

Jana Holznigenkemper[1] · Christian Komusiewicz[1] · Bernhard Seeger[1]

Received: 22 July 2022 / Accepted: 11 December 2022 / Published online: 9 January 2023

Abstract

Computing an accurate mean of a set of time series is a critical task in applications like nearest-neighbor classification and clustering of time series. While there are many distance functions for time series, the most popular distance function used for the computation of time series means is the non-metric dynamic time warping (DTW) distance. A recent algorithm for the exact computation of a DTW- Mean has a running time of $\mathcal{O}(n^{2k+1}2^k k)$, where k denotes the number of time series and n their maximum length. In this paper, we study the mean problem for the move-split-merge (MSM) metric that not only offers high practical accuracy for time series classification but also carries of the advantages of the metric properties that enable further diverse applications. The main contribution of this paper is an exact and efficient algorithm for the MSM- Mean problem of time series. The running time of our algorithm is $\mathcal{O}(n^{k+3}2^k k^3)$, and thus better than the previous DTW-based algorithm. The results of an experimental comparison confirm the running time superiority of our algorithm in comparison to the DTW- Mean competitor. Moreover, we introduce a heuristic to improve the running time significantly without sacrificing much accuracy.

Keywords Time series means · Time series metrics · Dynamic programming · Exact algorithm

Responsible editor: Eamonn Keogh

✉ Jana Holznigenkemper
holznigenkemper@mathematik.uni-marburg.de

Christian Komusiewicz
komusiewicz@mathematik.uni-marburg.de

Bernhard Seeger
seeger@mathematik.uni-marburg.de

[1] Mathematics and Computer Science, University of Marburg, Marburg, Germany

1 Introduction

Time series databases have gained much attention in academia and industry due to demands in many new challenging applications like Internet of Things (IoT), bioinformatics, social and system monitoring. In particular, because of the emergence of IoT, the requirement for developing dedicated systems (Garcia-Arellano et al. 2020) supporting time series as a first-class citizen has increased recently. In addition to supporting fundamental database operations like filters and joins, analytical operations like clustering and classification are highly relevant in time series databases.

The analysis of time series like clustering largely depends on the underlying distance functions. In a recent study, Paparrizos et al. (2020) re-examined the impact of 71 distance functions on classification for many data sets. While *dynamic time warping (DTW)* and related functions (Berndt et al. 1994) had the reputation of being the best choice,

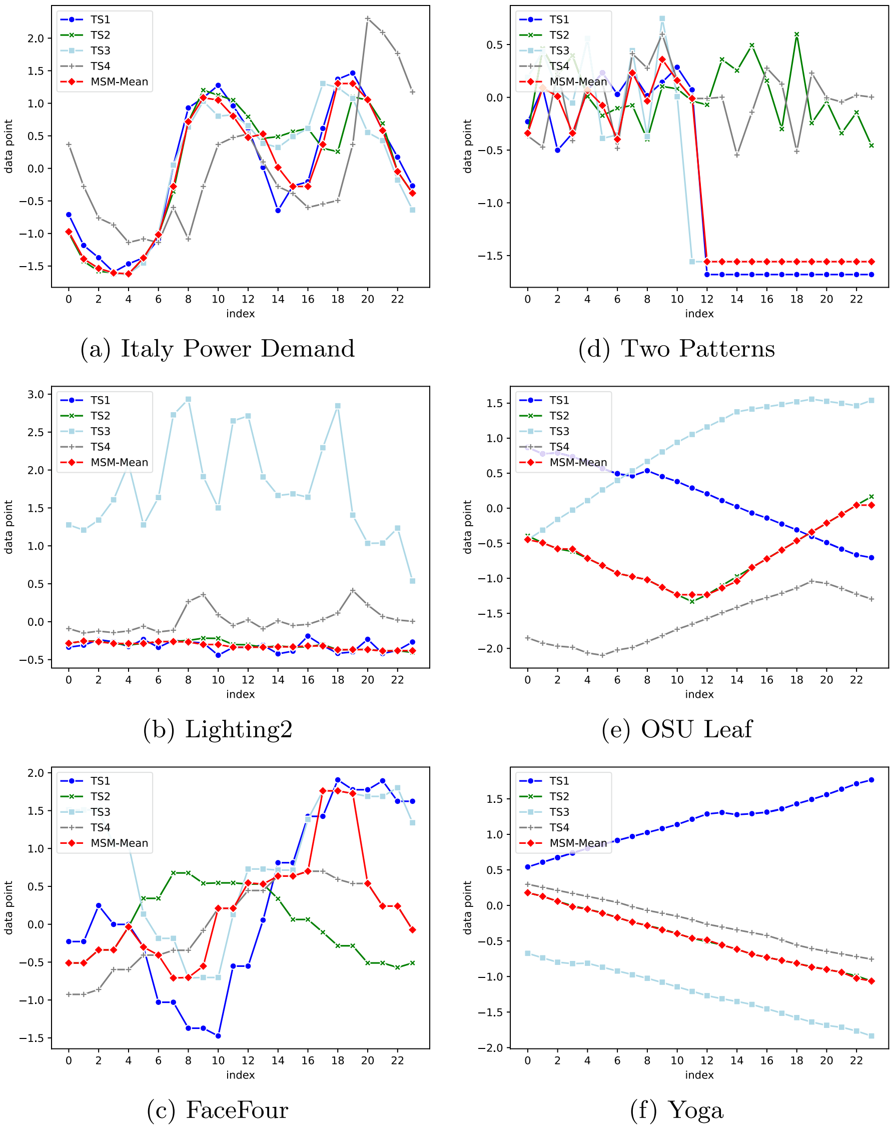
Paparrizos et al. (2020) found DTW performing inferior for time series classification in comparison to many other elastic distance functions. Among those is the *move-split-merge (MSM) metric* (Stefan et al. 2012). It works similarly to the Levensthein distance (Levenshtein 1996) by transforming one time series into another using three types of operations. A move operation changes the value of a data point, a merge operation fuses two consecutive points with equal values into one, and a split operation splits a point into two adjacent points with the same value. In addition to its superiority to DTW, MSM offers another significant advantage: it satisfies the properties of a mathematical metric, and thus it is ready-to-use for metric indexing (Novak et al. 2011) and algorithms that presume the triangle inequality.

Partition-based algorithms such as k-means clustering are among the best methods for clustering time series (Paparrizos and Gravano 2017). One of the fundamental problems of k-means clustering for time series is how to compute a mean for a set of time series. Brill et al. (2019) studied the problem for DTW and developed an algorithm computing an exact mean of k time series in $\mathcal{O}(n^{2k+1}2^k k)$ time, where n is the maximum length of an input time series. To the best of our knowledge, the mean problem of time series has not been addressed for other distance functions like the MSM metric so far.

In this paper, we examine the mean problem of time series for the MSM metric. The mean m of a set X of input time series is a time series that minimizes the sum of the distances to the time series in X regarding the MSM metric. In the following, we use MSM- Mean and DTW- Mean to denote the problem of computing means with respect to the MSM metric and DTW distance function, respectively. The actual means are referred to as MSM means and DTW means.

Some examples of MSM means are depicted in Fig. 1. Each comprises four sample time series from a data set of the UCR time series archive (Chen et al. 2015) with their respective MSM mean.

In contrast to DTW means, we show that each set of input time series has an MSM mean consisting only of values present in the input time series. This observation is crucial for the design and efficiency of our algorithm. We prove that the running time of our algorithm is $\mathcal{O}(n^{k+3}2^k k^3)$, thus faster than the DTW-based competitor (Brill et al. 2019).



(a) Italy Power Demand

(d) Two Patterns

(b) Lighting2

(e) OSU Leaf

(c) FaceFour

(f) Yoga

Fig. 1 MSM means each of four time series from different data sets of the UCR Archive containing time series samples of length $n = 24$. The respective values of the parameter c are shown in Table 1. For the data set *Italy Power Demand* c is set to 0.1

In summary, our contributions are:

- We give new essential characteristics of the MSM metric. We first prove that there always exists an optimal transformation graph that is a forest to further specify the values of some crucial nodes within this forest.
- We show that there is always an MSM mean consisting of data points that are present in at least one time series of the input set.

- We develop a dynamic program computing the (optimal) MSM mean of k input time series achieving a better theoretical running time than its competitor DTW-Mean.
- In experiments on samples of real-world time series, we show that our algorithm for solving MSM- Mean is faster than DTW- Mean in practice as well.
- We present preliminary heuristics for computing the MSM mean which are significantly faster without sacrificing much accuracy.

The remainder of the paper is structured as follows. Section 2 reviews related work. In Sect. 3, we give some important preliminaries for the MSM metric and formulate the MSM- Mean problem. Then, in Sect. 4, we introduce some new properties of the MSM metric to prove at the end of the section that there always exists a mean consisting of data points of the input time series. The dynamic program for the exact MSM- Mean algorithm is given in Sect. 5. We experimentally evaluate our approach, discuss various heuristics, and compare it to the DTW- Mean algorithm in Sect. 6, and conclude in Sect. 7.

2 Related work

For the exploratory analysis of time series, clustering is used to discover interesting patterns in time series data sets (Das et al. 1998). Much research has been done in this area (Liao 2005). The surveys (Aghabozorgi et al. 2015; Rani and Sikka 2012) give a recent overview of many methods. The problem of mean computation is discussed for Euclidean distance and for the DTW distance, but not for the MSM metric.

Moreover, the use of classification methods is indispensable for accurate time series analysis (Jiang 2020). The temporal aspect of time series has to be taken into account for clustering and classification, though finding a representation of a set of time series is a challenging task. Determining accurate means of time series is crucial for partitioning clustering approaches (Niennattrakul and Ratanamahatana 2007) like k-means (MacQueen 1967), where the prototype of a cluster is a mean of its objects, and for nearest-neighbor classification (Petitjean et al. 2016). These methods are based on the choice of the underlying distance function. Among the existing time series distance measures (Paparrizos et al. 2020), the DTW distance (Sakoe and Chiba 1978) is a very important measure with application in, e.g., similarity search (Sakurai et al. 2005), speech recognition (Sankoff and Kruskal 1983), or gene expression (Aach and Church 2001). We now give an overview of mean computation methods using the DTW distance.

Besides the exact DTW- Mean algorithm (Brill et al. 2019) that minimizes the Fréchet function (Fréchet 1948), there are many heuristics trying to address this problem. Some approaches first compute a multiple alignment of k input time series and then average the aligned time series column-wise (Hautamäki et al. 2008; Petitjean et al. 2012). *DTW barycenter averaging (DBA)* (Petitjean et al. 2011) is a heuristic strategy that iteratively refines an initially average sequence in order to minimize its square DTW-distance to the input time series. Other approaches exploit the properties of the Fréchet function (Cuturi and Blondel 2017; Schultz and Jain 2018). Their methods are based on the observation that the Fréchet function is Lipschitz continuous and thus differentiable almost everywhere. Cuturi et al. (2017) use a smoothed version of the DTW-distance to obtain a differentiable Fréchet DTW-distance. Brill et al. (2019) showed that none of the aforementioned approaches is sufficiently accurate compared to the exact method. Since clustering methods based on partitioning rely on cluster prototype determination (Aghabozorgi et al. 2015), it is necessary to compute an accurate mean. All these observations make it indispensable to consider the problem for other distance functions, like the MSM metric.

The MSM metric is already investigated for classification. One of the first studies of the MSM metric concerning its application for classification problems was by Stefan et al. (2012). They perform their tests on 20 data sets of the UCR archive (Chen et al. 2015). The MSM distance is tested against the DTW distance, the constrained DTW distance, the edit distance on real sequence and the Euclidean distance. For a majority of the tests, the MSM distance performs better than the compared measures. There have been further studies of the accuracy of different time series distance measures regarding 1-NN classification problems (Bagnall et al. 2017; Lines and Bagnall 2015). Bagnall et al. (2017) also conclude that the MSM distance leads to results with higher accuracy than DTW at the cost of a higher running time. All these studies come to a similar result as the most recent study of Paparrizos et al. (2020). To the best of our knowledge, there are no studies that investigate and extend the theoretical concepts and applications of the MSM distance.

The subject of time series, also known as data series, has attracted attention within the database research domain recently, see (Jensen et al. 20117) for a recent survey. There are time series data bases, also known as event stores, that are specially designed for the analysis of time series (Bader et al. 2017; Garcia-Arellano et al. 2020). These systems rarely support clustering, but focus on supporting the basic building blocks for query processing.

Since the MSM distance obeys all properties of a mathematical metric, especially the triangle inequality, it also applies to problems like metric indexing (Chen et al. 2017; Novak et al. 2011). In fact, metric indexing also requires the computation of pivots that is closely related to the mean. However, pivots belong to the underlying data set, while a mean (of a time series) is generally a newly generated object.

3 Preliminaries

Let us first introduce our notation and problem definition. For $k \in \mathbb{N}$, let $[k] := \{1, \ldots, k\}$. A *time series* of length n is a sequence $x = (x_1, \ldots, x_n)$, where each *data point*, in short *point*, x_i is a real number. Let $V(x) = \{x_i \mid x \in x\}$ be the set of all values of points of x. For $i < j$, the point x_i is a *predecessor* of the point x_j and the point x_j is a *successor* of the point x_i. For a set of time series $X = \{x^{(1)}, \ldots, x^{(k)}\}$, the ith point of the jth time series of X is denoted as $x_i^{(j)}$; time series $x^{(j)}$ has length n_j. Further let $V(X) = \cup_{j \in [k]} V(x^{(j)}) = \{v_1, \ldots, v_r\}$ be the set of the values of all points of all time series in X.

3.1 Move-split-merge operations

We now define the MSM metric, following the notation of Stefan et al. (2012), and the MSM- Mean problem. The MSM metric allows three transformation operations to transfer one time series into another: *move, split*, and *merge* operations. For time series $x = (x_1, \dots, x_n)$ a move transforms a point x_i into $x_i + w$ for some $w \in \mathbb{R}$, that is, $\text{Move}_{i,w}(x) := (x_1, \dots, x_{i-1}, x_i + w, x_{i+1}, \dots, x_n)$, with cost $\text{Cost}(\text{Move}_{i,w}) = |w|$. Informally, we say that there is a *move at point x_i to another point $x_i + w$*. The split operation splits the ith element of x into two consecutive points. A split at point x_i is defined as $\text{Split}_i(x) := (x_1, \dots, x_{i-1}, x_i, x_i, x_{i+1}, \dots, x_n)$.

A merge operation may be applied to two consecutive points of equal value. For $x_i = x_{i+1}$, it is given by $\text{Merge}_i(x) := (x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n)$. We say that x_i and x_{i+1} *merge to a point z*. Split and merge operations are inverse operations. Their costs are assumed to be equal and determined by a given nonnegative constant $c = \text{Cost}(\text{Split}_i) = \text{Cost}(\text{Merge}_i)$. A *sequence of transformation operations* is given by $\mathbb{S} = (S_1, \dots, S_s)$, where $S_j \in \{\text{Move}_{i_j,w_j}, \text{Split}_{i_j}, \text{Merge}_{i_j}\}$. A *transformation $T(x, \mathbb{S})$* of a time series x for a given sequence of transformation operations $\mathbb{S}$ is defined as $T(x, \mathbb{S}) := T(S_1(x), (S_2, \dots, S_s))$. If $\mathbb{S}$ is empty, we define $T(x, \emptyset) := x$. The cost of a sequence of transformation operations $\mathbb{S}$ is given by the sum of all individual operations cost, that is, $\text{Cost}(\mathbb{S}) := \sum_{S \in \mathbb{S}} \text{Cost}(S)$. We say that $\mathbb{S}$ transforms x to y if $T(x, \mathbb{S}) = y$. We call a transformation an *optimal transformation* if it has minimal cost transforming x to y. The MSM *distance $d(x, y)$* between two time series x and y is defined as the cost of an optimal transformation. The distance $D(X, y)$ of multiple time series $X = \{x^{(1)}, \dots, x^{(k)}\}$ to a time series y is given by $D(X, y) = \sum_{x \in X} d(x, y)$. A *mean m* of a set of time series X is defined as a time series with minimum distance to X, that is, $m = \arg\min_{z \in \mathcal{Z}} D(X, z)$, where $\mathcal{Z}$ is the set of all finite time series. The problem of computing a mean is thus defined as follows:

MSM- Mean
Input: A set of time series $X = \{x^{(1)}, \dots, x^{(k)}\}$.
Output: A time series m such that $m = \arg\min_{z \in \mathcal{Z}} D(X, z)$.

Before we regard the MSM- Mean problem in more detail, we will introduce the concept of *transformation graphs* to describe the structure of a transformation $T(x, \mathbb{S}) = y$.

3.2 Transformation graphs

The transformation $T(x, \mathbb{S}) = y$ can be described by a directed acyclic graph $G_{\mathbb{S}}(x, y)$, the *transformation graph*, with *source nodes* $N(x) = \{u_1, \dots, u_m\}$ and *sink nodes* $N(y) = \{v_1, \dots, v_n\}$, where a node u_i represents the point x_i and the node v_j represents the point y_j. All nodes which are neither source nor sink nodes are called *intermediate nodes*. If the time series and operation sequence are clear from context, we may write G instead of $G_{\mathbb{S}}(x, y)$. Each node in the node set V of G is associated with a value given by a function $\text{val} : V \rightarrow \mathbb{R}$. For source and sink nodes we have

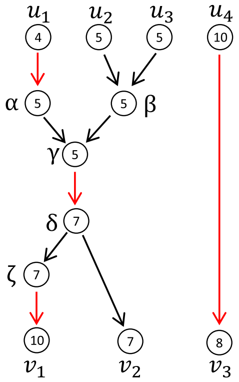


Fig. 2 Optimal transformation graph of $x = (4, 5, 5, 10)$ to $y = (10, 7, 8)$ for $c = 0.1$. Move edges are colored in red in this work. The cost of a move edge is the difference between the source and the target point. We have total cost merge and split cost $3c$ and move cost of 8. Hence, the distance between x and y is $d(x, y) = 8.3$ (Color figure online)

$\mathrm{val}(u_i) = x_i$ and $\mathrm{val}(v_j) = y_j$. Each intermediate node is also associated with a value. The edges represent the transformation operations of $\mathbb{S}$. To create a transformation graph, for each operation in $\mathbb{S}$ a respective *move edge* or two *split*, or *merge edges* are added to the graph. A move edge can be further specified as an *increasing (inc-)* or *decreasing (dec-)* edge if the move operation adds a positive or negative value to the value of the parent node, respectively. An edge can be either a move, split or merge edge. If a node α is connected to a node β by a split edge and β is a child of α, then there exists a node $\gamma \neq \beta$ to which α is connected by a split edge and which is a child of α. If the nodes α and β are connected by a merge edge and α is a parent of β, then there exists a node $\gamma \neq \alpha$ which is connected to β by a merge edge and is a parent of β. Moreover, for the split and the merge case, it holds that $\mathrm{val}(\alpha) = \mathrm{val}(\beta) = \mathrm{val}(\gamma)$.

Given a sequence of operations $\mathbb{S}$, the transformation graph $G_{\mathbb{S}}(x, y)$ is unique. Given a transformation graph G, it may be derived from different sequences of operations since a sequence $\mathbb{S}$ is only partially ordered. Taking the example of a transformation graph (see Fig. 2), that means, that for example the move operation between the node u_1 and α and the move operation between u_4 and v_3 are interchangeable.

A *transformation path*, in short *path*, in $G_{\mathbb{S}}(x, y)$ is a directed path from a source node $u_i \in N(x)$ to a sink node $v_j \in N(y)$. We say that u_i is *aligned* to v_j. A path can be further characterized by its sequence of edge labels. For example, in Fig. 2, the path from u_1 to v_2 is an inc-merge-inc-split path. Analogously, we say that the path consists of consecutive inc-merge-inc-split edges.

A transformation path is *monotonic* if the move edges on this path are only inc- or only dec-edges. A monotonic path may be specified as *increasing* or *decreasing*. A transformation is *monotonic* if the corresponding transformation graph only contains monotonic paths. A transformation graph is *optimal* if it belongs to an optimal transformation. Two transformation graphs are *equivalent* if they have the same sink and source nodes.

In the next section, we recap some known properties about the transformation graph and extend them proving some new essential characteristics.

3.3 Properties of transformation graphs

In the following, we summarize some important known properties about the transformation graph by Stefan et al. (2012). The first lemma states that there exists an optional transformation graph without split and merge edges that occur directly after another on a path.

Lemma 1 (Proposition 2 (Stefan et al. 2012)) *For any two time series x and y, there exists an optimal sequence of transformation operations $\mathbb{S}$ such that $G_{\mathbb{S}}(x, y)$ contains no consecutive merge-split or split-merge edges.*

By construction, two consecutive move-move edges are not useful, since they can be combined into one move edge. We extend Lemma 1 to further path restrictions in an optimal transformation graph. That is, that there exists an optimal transformation graph without paths containing consecutive split-move-merge edges.

Lemma 2 *For any two time series x and y, there exists an optimal sequence of transformation operations $\mathbb{S}$ such that $G_{\mathbb{S}}(x, y)$ contains no consecutive split-move-merge edges.*

Proof Assume an optimal transformation graph $G_{\mathbb{S}}(x, y)$ including split-move-merge edges. Since the underlying set of transformation operations $\mathbb{S}$ of G is partially ordered, we can reorder the operations in $\mathbb{S}$, choosing an order, where split, move and merge operations are directly applied after one another. Figure 3 shows two different possibilities how consecutive split-move-merge edges may be contained in a transformation graph.

Case I: We consider a split at node α to the nodes α' and α'' where $\text{val}(\alpha) = \text{val}(\alpha') = \text{val}(\alpha'')$. It is followed by two move edges from α' to β' and from α'' to β'' and a merge of β' and β'' (see Fig. 3a). Therefore, the values added to $\text{val}(\alpha)$ have to be equal on both move edges, that is a value $w \in \mathbb{R}$. The cost of these transformation operations are $2c+2|w|$. Consider replacing the two split-move-merge edges with one direct move edge from α to β adding w to $\text{val}(\alpha)$ (see Fig. 3b). This replacement leads to an equivalent transformation with cost $|w| < 2c + 2|w|$. This is a contradiction to our assumption that G is optimal.

Case II: Consider the part of a transformation graph in Fig. 3c. There is a split at α to α' and α''. The node α'' is connected by a move edge to β' adding a value w to $\text{val}(\alpha'')$. The node β' merges with β'' to β. Deleting the split-move-merge edge and editing the part of the graph as shown in Fig. 3d leads to an equivalent transformation graph, saving cost of $2c + |w|$. This is a contradiction to our assumption that G is optimal. □

The next lemma states that there is always an optimal monotonic transformation.

Lemma 3 (Monotonicity lemma (Stefan et al. 2012)) *For any two time series x and y, there exists an optimal transformation that converts x into y and that is monotonic.*

Summarizing the above properties, there always exists an optimal transformation graph only containing paths from source to sink nodes of the following consecutive edge types:

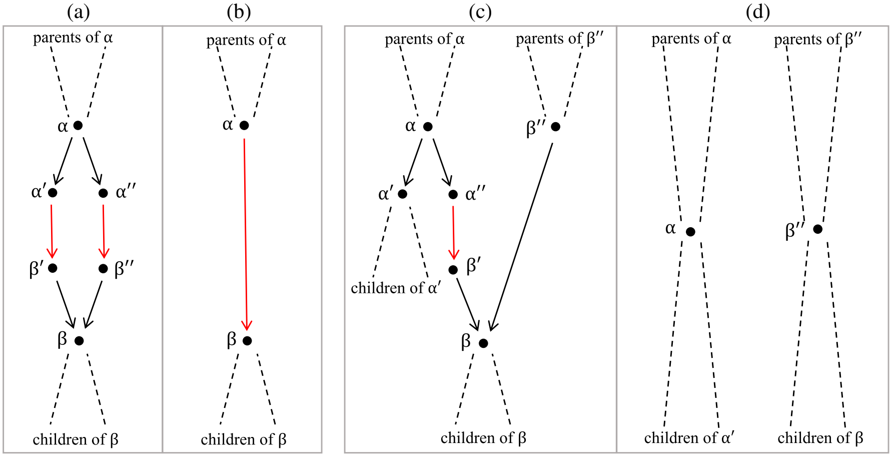


Fig. 3 (**a**) First possibility of a transformation graph including consecutive split-move-merge edges. (**b**) Equivalent transformation graph to (**a**). (**c**) Second possibility of a transformation graph including consecutive split-move-merge edges. (**d**) Equivalent transformation graph to (**c**)

Type 1: move - move - · · · - move - move
Type 2: split/move - split/move - · · · - split/move -split/move
Type 3: merge/move - merge/move - · · · - merge/move - merge/move
Type 4: Type 3 - merge - move - split - Type 2

Note, that paths of Type 2 and Type 3 contain at least one split or merge edge, respectively. In the following, we consider only transformation graphs that contain only paths of Type 1–4. To identify independent transformation operations, we decompose an optimal transformation graph into its weakly connected components. A weakly connected component is a tree if its underlying subgraph is a tree.

In the following, we give a more substantive view on those weakly connected components which are trees (see Fig. 4).

The first ones are *trees of Type 1*. These trees contain only paths of Type 1, that is, there is only one move edge in the tree connecting one source and sink node (see Fig. 4a). A weakly connected component containing only paths of Type 2 has only one source node and at least two paths of Type 2, that is, it has at least two sink nodes. It is a tree since all all nodes have indegree 1. We call these trees *trees of Type 2* (see Fig. 4b). *Trees of Type 3* contain only paths of merge or move operations (Type 3). These trees have at least two source nodes whose paths reach the same sink node. All nodes have outdegree 1 (see Fig. 4c). The last weakly connected component which is a tree is a *tree of Type 4*. These trees include only paths of Type 4. They contain only and at least two paths of Type 4, that is, that they have at least two source and two sink nodes (see Fig. 4d). For the following sections, we need a more detailed description of Type-4-Trees. All source nodes merge and move to some intermediate node σ. After σ, there is a move to σ^* with subsequent split and move edges leading to the source nodes. All source and sink nodes of this tree are connected by one single path which we call a *bottleneck* with σ as the *first bottleneck node* and σ^* as the *second bottleneck node*. All nodes above and including the first bottleneck node have outdegree 1. We call this

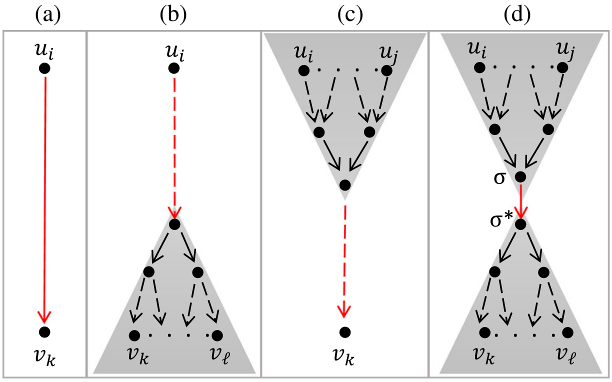


Fig. 4 All red edges are move operations. Black arrows are merge or split edges. The dashed lines represent paths from one node to another without specifying how many intermediate node are on them. **(a)** Tree of Type 1. **(b)** Tree of Type 2. **(c)** Tree of Type 3. **(d)** Tree of Type 4 (Color figure online)

subgraph the *upper tree* of σ. All nodes below and including the second bottleneck node have indegree 1. We call this subgraph a *lower tree* of σ^*.

The following lemma states that there always exists an optimal transformation graph, where every weakly connected component is a tree of Type 1–4.

Lemma 4 *Let x and y be two time series. Then there exists an optimal transformation graph $G_{\mathbb{S}}(x, y)$ such that its weakly connected components are only trees of Type 1–4.*

Proof We show that if a weakly connected component of a given optimal transformation graph G is not a tree of Type 1–3, then it has to be a tree of Type 4. We consider a path of Type 4. In a path of Type 4 there is one part with consecutive merge-move-split edges. Let σ be the node between this merge and move operation and σ^* be the node between this move and split operation. We add further move and merge operations to the part above σ. We still have outdegree 1 of each node above σ. The same applies for the part below σ^*: Adding further move and split operations still leads to indegree 1 of each node below σ^*. Hence, the subgraph of G consisting of the edge from σ to σ^* and all move or merge edges above σ and all move or split edges below σ^* is a tree of Type 4. We now show that this tree structure cannot be extended without violating our assumptions of the above Lemmas. Let α be a node in the upper tree that is connected to a node α' that is neither in the upper nor in the lower tree. If α is a source node or an intermediate node except of σ, the first operation is a split, where one split edge is on the path to σ and the other is on the path to α'. We get a contradiction to Lemma 2, because the first path includes consecutive split-move-merge edges. If $\alpha = \sigma$, we have again a split at α, which is a contradiction to Lemma 1 because we have a consecutive merge-split edge. The same argumentation is applied for an extension of the lower tree, because it is the symmetric case of the one we described. □

It follows that we can decompose an optimal transformation graph $G_{\mathbb{S}}(x, y)$ into a sequence of distinct trees $(\mathcal{T}_1, \ldots, \mathcal{T}_t)$. Each tree $\mathcal{T}_i$ has a set of sink nodes $N_{\mathcal{T}_i}(x)$ and a set of source nodes $N_{\mathcal{T}_i}(y)$. All nodes of $N_{\mathcal{T}_i}(x)$ and $N_{\mathcal{T}_i}(y)$ are successors of $N_{\mathcal{T}_{i-1}}(x)$ and $N_{\mathcal{T}_{i-1}}(y)$, respectively. We call a tree *monotonic* if all paths in the tree are monotonic. Further a tree may be specified as *increasing* or *decreasing*. Two trees are *equivalent* if they have the same set of source and sink nodes. The *cost of a tree* $\mathcal{T}$ is the sum of the cost of all edges in the tree.

In the following, we denote an optimal transformation graph fulfilling all the above properties as an *optimal transformation forest*.

4 Properties of the MSM metric

As a main result of this section, we prove that for a set of time series X there exists a mean m such that all points of m are points of at least one time series of X. To this end, we first analyze the structure of trees of optimal transformation forests. Some of the following results are only proven for trees of Type 4 since these trees include all types of possible paths; as a consequence the proofs for other tree types are simpler versions of the ones for Type 4.

4.1 Properties of alignment trees

We first regard some properties of so-called *subtrees*, which are substructures of trees of Type 4.

4.1.1 Subtrees

Let $G_{\mathbb{S}}(x, y)$ be an optimal transformation forest. For an intermediate node δ in G, that has two parent nodes connected to it by a merge edge each, let $\mathcal{S}(\delta)$ be the *subtree of* δ consisting of all source nodes of G that have a path to δ and of all nodes and edges on these paths. Each subtree has a set of source nodes $N_{\mathcal{S}(\delta)}(x)$. Let $N_{\mathcal{S}(\delta)}(x) = \{u_i, \ldots, u_j\}$ be the source nodes of $\mathcal{S}(\delta)$; we call u_i the *start node of* $\mathcal{S}(\delta)$ and u_j the *end node of* $\mathcal{S}(\delta)$. A subtree is *increasing* (*decreasing*) if all paths to δ are increasing (decreasing). In the following, we will give some properties of subtrees. If there are two move edges to some nodes α_2 and β_2 that merges to another node γ (see Fig. 5a), we first observe that these two move edges cannot be both increasing or decreasing.

Lemma 5 *Let $G_{\mathbb{S}}(x, y)$ be an optimal transformation forest with nodes $\alpha_1, \alpha_2, \beta_1, \beta_2$ and γ and move edges between α_1, α_2 and β_1, β_2. If α_2 and β_2 merge to γ, then the edges between α_1, α_2 and β_1, β_2 cannot be both increasing or decreasing.*

Proof Without loss of generality, we prove this assumption for increasing paths. Assume towards a contradiction two inc-edges between α_1 and α_2 and between β_1 and β_2 with $\text{val}(\alpha_1) < \text{val}(\beta_1)$ (see Fig. 5a). Since $\text{val}(\alpha_2) = \text{val}(\beta_2) = \text{val}(\gamma)$ the cost for

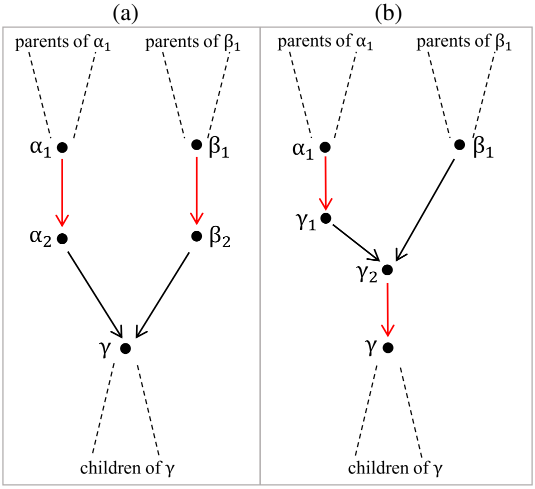


Fig. 5 **(a)** Merge structure of the (intermediate) nodes $\alpha_1, \alpha_2, \beta_1, \beta_2$ and γ. **(b)** Equivalent tree to the tree in (a)

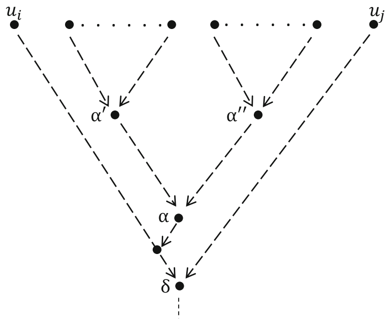


Fig. 6 Structure of the subtree of δ explained in the proof of Lemma 6. Note, that this is only a schematic representation and that there may be further intermediate nodes which are not marked

these move operations are $2\,\mathrm{val}(\gamma) - \mathrm{val}(\beta_1) - \mathrm{val}(\alpha_1)$. We now consider a modified merge structure with an additional intermediate node γ_1 with $\mathrm{val}(\gamma_1) = \mathrm{val}(\beta_1)$ (see Fig. 5b). We now have an inc-edge from α_1 to γ_1, which merges with the new node β_1 to a new node γ_2. At γ_2, there is an inc-edge to γ. The modified transformation forest is equivalent to the old one, since the parent and children nodes of the regarded nodes stay the same (see Fig. 5). The cost for the modified move operations are $\mathrm{val}(\gamma) - \mathrm{val}(\beta_1) + \mathrm{val}(\beta_1) - \mathrm{val}(\alpha_1) < 2\,\mathrm{val}(\gamma) - \mathrm{val}(\beta_1) - \mathrm{val}(\alpha_1)$ since $\mathrm{val}(\beta_1) < \mathrm{val}(\gamma)$. This is a contradiction to G being optimal. □

We now make two observations about the value of the node δ in a subtree $\mathcal{S}(\delta)$. The first lemma states, that the value of δ is equal to one value of the source nodes of $\mathcal{S}(\delta)$. Recall that $u_1, \ldots, u_m$ are the source nodes of $G_{\mathbb{S}}(x, y)$ with values $x_1, \ldots, x_m$.

Lemma 6 *Let $\mathcal{S}(\delta)$ be an increasing (decreasing) subtree of δ in an optimal transformation forest $G_{\mathbb{S}}(x, y)$ with $N_{\mathcal{S}(\delta)}(x) = \{u_i, \ldots, u_j\}$. Then,* $\mathrm{val}(\delta) = \max(x_i, \ldots, x_j)$ *for increasing subtrees and* $\mathrm{val}(\delta) = \min(x_i, \ldots, x_j)$ *for decreasing subtrees.*

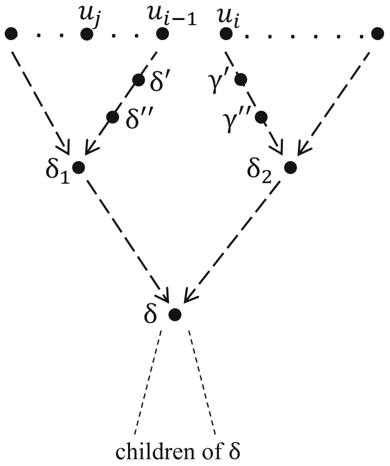


Fig. 7 Structure of the subtree of δ described in the proof of Lemma 7. Note that this is only a schematic representation and that there may be further intermediate nodes which are not marked

Proof of Lemma 6 Without loss of generality, we prove this assumption for increasing subtrees. Figure 6 depicts this subgraph with all mentioned intermediate nodes. Assume towards a contradiction, that $\mathrm{val}(\delta) \neq \max(x_i, \dots, x_j)$. Therefore, there exists an $u_\ell \in \{u_i, \dots, u_j\}$ such that $\mathrm{val}(\delta) \neq x_\ell$. For $\mathrm{val}(\delta) < x_\ell$, it follows that we have a decreasing edge between u_ℓ and δ, which is a contradiction. For $\mathrm{val}(\delta) > x_\ell$, let α be the first intermediate node below $\{u_i, \dots, u_j\}$ such that $\mathrm{val}(\alpha) \neq \max_{u \in \mathcal{N}_{\mathcal{S}(\alpha)}(x)}(u)$, where $\mathcal{N}_{\mathcal{S}(\alpha)}(x) \subseteq \mathcal{N}_{\mathcal{S}(\delta)}(x)$ are the source nodes of the subtree $\mathcal{S}(\alpha)$ of α. There exist two intermediate nodes α' and α'' such that for the source nodes of their subtrees $\mathcal{S}(\alpha')$ and $\mathcal{S}(\alpha'')$, respectively, it holds that $\mathcal{N}_{\mathcal{S}(\alpha')}(x) \cup \mathcal{N}_{\mathcal{S}(\alpha'')}(x) = \mathcal{N}_{\mathcal{S}(\alpha)}(x)$. It follows that there exists a path from α' to α and from α'' to α. Since α is the first intermediate node below $\mathcal{N}_{\mathcal{S}(\delta)}(x)$ such that $\mathrm{val}(\alpha) \neq \max_{u \in \mathcal{N}_{\mathcal{S}(\delta)}(x)}(u)$, it holds that $\mathrm{val}(\alpha') = \max_{u \in \mathcal{N}_{\mathcal{S}(\alpha')}(x)}(u)$ and $\mathrm{val}(\alpha'') = \max_{u \in \mathcal{N}_{\mathcal{S}(\alpha'')}(x)}(u)$. Consequently, there is an inc-edge on the path from α' to α and on the path from α'' to α. Applying Lemma 5, this is a contradiction to G being optimal. □

In the next lemma, we specify the value of δ in a subtree $\mathcal{S}(\delta)$, stating that it is always equal to the value of a specific source node of $\mathcal{N}_{\mathcal{S}(\delta)}(x)$.

Lemma 7 *Let the nodes δ_1 and δ_2 merge to a node δ in an optimal transformation forest G. Let $\mathcal{S}(\delta)$ be the subtree of δ, $\mathcal{S}(\delta_1)$ be the subtree of δ_1 with the end node u_{i-1}, and $\mathcal{S}(\delta_2)$ be the subtree of δ_2 with the start node u_i. If the subtree $\mathcal{S}(\delta)$ is increasing (decreasing) and $x_{i-1} > x_i$, then* $\mathrm{val}(\delta) = x_{i-1}$ ($\mathrm{val}(\delta) = x_i$)*. If $x_{i-1} < x_i$, then* $\mathrm{val}(\delta) = x_i$ ($\mathrm{val}(\delta) = x_{i-1}$ *for decreasing subtrees).*

Proof Without loss of generality, $\mathcal{S}(\delta)$ is increasing. Figure 7 depicts this subgraph with all mentioned intermediate points. We will only show the case that $x_{i-1} > x_i$ since the other case is analogous. By Lemma 6, it holds that $\mathrm{val}(\delta) = \max(\mathrm{val}(\delta_1), \mathrm{val}(\delta_2))$. We first show that $\mathrm{val}(\delta) = \mathrm{val}(\delta_1)$. Assume towards a contradiction that $\mathrm{val}(\delta_1) < \mathrm{val}(\delta_2) = \mathrm{val}(\delta)$. Since $x_{i-1} > x_i$, there exists intermediate nodes γ' and γ'' on the path between u_i and δ_2 such that $x_{i-1} \geq \text{val}(\gamma')$ and $x_{i-1} < \text{val}(\gamma'')$. Let $\mathcal{S}'(\delta)$ be the modified subtree of $\mathcal{S}(\delta)$. The only difference between $\mathcal{S}$ and $\mathcal{S}'$ is that γ' merges to some intermediate node on the path between u_{i-1} and δ_1. The cost of $\mathcal{S}'(\delta)$ is $\text{Cost}(\mathcal{S}'(\delta)) = \text{Cost}(\mathcal{S}(\delta)) - \text{val}(\gamma'') - \text{val}(\gamma') < \text{Cost}(\mathcal{S}(\delta))$. This is a contradiction to G being optimal. Applying Lemmas 5 and 6, we get $\text{val}(\delta) = \text{val}(\delta_1)$.

In a second step, we prove that $\text{val}(\delta_1) = x_{i-1} = \max_{u \in \mathcal{N}_{\mathcal{S}(\delta_1)}(x)}(\text{val}(u))$. Assume towards a contradiction, that there exists a $u_j \in N_{\mathcal{S}((\delta_1)}(x) \backslash u_{i-1}$ such that $\text{val}(\delta_1) = x_j > x_{i-1} > x_i$. Then, it holds that there exist two intermediate nodes δ' and δ'' on the path between u_{i-1} and δ_1 such that $\text{val}(\delta') < \text{val}(\delta_1)$ and $\text{val}(\delta'') = \text{val}(\delta_1)$. We consider the modified subtree $\mathcal{S}''(\delta)$, which is almost equal to $\mathcal{S}(\delta)$, the only difference being that δ_2 merges to some intermediate node on the path between δ' and δ''. The cost of $\mathcal{S}''(\delta)$ are $\text{Cost}(\mathcal{S}''(\delta)) = \text{Cost}(\mathcal{S}(\delta)) - \text{val}(\delta_1) - \text{val}(\delta_2) < \text{Cost}(\mathcal{S}(\delta))$. This is a contradiction to G being optimal. □

We will now apply the above properties to the bottleneck nodes in a tree of Type 4 stating that the first and second bottleneck nodes always have values of the input time series x and y, respectively. Recall, that the first bottleneck node σ is the intermediate node where all source nodes in the tree of Type 4 merge to, followed by a move edge to the second bottleneck node σ^*.

Corollary 1 *Let $G_{\mathbb{S}}(x, y)$ be an optimal transformation forest. In a tree $\mathcal{T}$ of Type 4* $\text{val}(\sigma) \in V_{\mathcal{T}}(x)$ *and* $\text{val}(\sigma^*) \in V_{\mathcal{T}}(y)$.

Proof To prove that $\sigma \in V_{\mathcal{T}}(x)$ we apply Lemma 6 since the upper part of the tree $\mathcal{T}$ is the subtree of σ. By symmetry reasons, it follows that $\sigma^* \in V_{\mathcal{T}}(y)$. □

4.2 The effect of perturbing single values

We aim to show that there exists a mean of a set of time series that only consists of points of the input set. To this end, we make observations on the effect of shifting points of a time series that are not from $V(X)$. The first proof step is to analyze for two time series x and y, how the distance between x and y may be affected by shifting one point of x by $\varepsilon \in \mathbb{R}$. We let $x_{\varepsilon,i}$ denote the new time series that is equal to x except at the position i, where it has the new point $x_i + \varepsilon$. The change of the node u_i in the transformation forest is denoted by u_i^{ε}. In the following we say that if the distance between $x_{\varepsilon,i}$ and y is shorter than between x and y, the replacement of x by $x_{\varepsilon,i}$ is *beneficial*. If it leads to a longer distance, it is *detrimental*, and if the distance does not change it is *neutral*. Assume that $x_i \notin V(y)$, the next lemma states that if the replacement of x by $x_{\varepsilon,i}$ is not neutral, it is beneficial for either ε or $-\varepsilon$.

Lemma 8 *Let x and y be two time series with distance $d(x, y)$. If $x_{i-1} \neq x_i \neq x_{i+1}$ and $x_i \notin V(y)$, there either exists an $\varepsilon' > 0$ such that for all $\varepsilon \in [0, \varepsilon']$ one of the following equations holds:*

(1) $d(x_{\varepsilon,i}, y) + \varepsilon = d(x, y) = d(x_{-\varepsilon,i}, y) - \varepsilon$ *(beneficial increase),*
(2) $d(x_{-\varepsilon,i}, y) + \varepsilon = d(x, y) = d(x_{\varepsilon,i}, y) - \varepsilon$ *(beneficial decrease),*

or there exist $\varepsilon_I, \varepsilon_D > 0$ such that

(3.1) *$d(x, y) = d(x_{\varepsilon,i}, y)$ for all $\varepsilon \in [0, \varepsilon_I]$ (neutral increase), and*
(3.2) *$d(x, y) = d(x_{-\varepsilon,i}, y)$ for all $\varepsilon \in [0, \varepsilon_D]$ (neutral decrease).*

Moreover, for beneficial increases $x_i + \varepsilon' \in (V(y) \cup \{x_{i-1}, x_{i+1}\})$, for beneficial decreases $x_i - \varepsilon' \in (V(y) \cup \{x_{i-1}, x_{i+1}\})$, and for neutral increases and decreases $x_i + \varepsilon_I, x_i - \varepsilon_D \in (V(y) \cup \{x_{i-1}, x_{i+1}\})$.

Proof We show the lemma for trees of Type 4. All other cases are simpler versions of this proof. Let $\mathcal{T}$ be a tree of Type 4 in $G_{\mathbb{S}}(x, y)$. By Lemma 3, the tree $\mathcal{T}$ is monotonic. We assume, without loss of generality, that all monotonic paths in $\mathcal{T}$ are increasing. We distinguish whether u_i has only predecessors or only successors (*Case 1*) or both (*Case 2*) in $\mathcal{T}$. We denote the predecessors of u_i as $\mathcal{P}$ and the successors of u_i as $\mathcal{F}$.

Case 1: u_i has only predecessors or successors in $\mathcal{T}$. We prove the case that u_i has only predecessors, the other case is analogous. We first describe the possible structures of the upper tree in $\mathcal{T}$ for this case, which are depicted in Fig. 8. There is a potential move at u_i to a node γ. The node γ merges to δ with a node α^*, which is the node resulting from a move at α. The nodes $\{u_{i-\ell}, \dots, u_{i-1}\} \subseteq \mathcal{P}$ are the source nodes of the subtree of α. Below δ there may be further subsequent merge and move operations to the first bottleneck node σ. Since $x_{i-1} \neq x_i$ there has to be an inc-edge either between u_i and γ, if $x_{i-1} > x_i$, or between u_{i-1} and α^*, if $x_{i-1} < x_i$ because in the first case $\text{val}(\delta) = x_{i-1}$ and in the second case $\text{val}(\delta) = x_i$ (see Lemma 7).

Case 1.1: $x_{i-1} > x_i$. There is an inc-edge between u_i and γ (see Fig. 8a). The replacement of x by $x_{\varepsilon,i}$ is a beneficial increase for all $\varepsilon \in [0, \varepsilon']$ with $\varepsilon' = \text{val}(\gamma) - x_i$ because the node u_i^{ε} approaches the node γ and the cost of the adapted move decrease by ε. Thus, we get the left side of Equation (1), $d(x_{\varepsilon,i}, y) + \varepsilon = d(x, y)$. Since the subtree of δ is increasing and $x_{i-1} > x_i$, it holds by Lemma 7 that $\text{val}(\delta) = x_{i-1}$. We get that $x_{i-1} = \text{val}(\gamma) = x_i + \varepsilon'$. For the right side of Equation (1), the argumentation is similar: After replacing x by $x_{-\varepsilon,i}$ for $\varepsilon \leq \varepsilon'$, the cost for the move between $x_i - \varepsilon$ and γ are $\text{val}(\gamma) - x_i + \varepsilon$. Therefore, they increase by ε.

Case 1.2: $x_{i-1} < x_i$. There is an inc-edge between u_{i-1} and α^* (see Fig. 8b). We modify the structure of $\mathcal{T}$ for the replacement of x by $x_{\varepsilon,i}$ for $\varepsilon \in [0, \varepsilon_I]$, $\varepsilon_I > 0$. Let $\mathcal{T}'$ be the modified tree with a new node u_i^{ε} instead of u_i. In $\mathcal{T}'$, the nodes α^* and δ does not exist but $\mathcal{T}'$ contains a new node δ' such that $\text{val}(\delta') \in [\text{val}(\alpha^*), \text{val}(\sigma^*)]$. The node u_i^{ε} merges to δ'. The rest of the tree stays unchanged. For all $\varepsilon \in [0, \varepsilon_I]$ with $\varepsilon_I = \text{val}(\sigma^*) - x_i$ the cost of $\mathcal{T}'$ is equal to the cost of $\mathcal{T}$ because we only shifted a merge operation to another position in the tree (see Fig. 8c). This is a neutral increase for all $\varepsilon \in [0, \varepsilon_I]$. It holds that $x_i + \varepsilon_I = \sigma^* \in V(y)$ (see Corollary 1). Let $\mathcal{T}''$ be another modified tree of $\mathcal{T}$ with a new node $u_i^{-\varepsilon}$ instead of u_i for $\varepsilon \in [0, \varepsilon_D]$, $\varepsilon_D > 0$. The tree $\mathcal{T}''$ does not contain the node δ but contains a new node δ'' such that $\text{val}(\delta'') \in [x_{i-1}, \text{val}(\alpha^*)]$ and $u_i^{-\varepsilon}$ merges to δ'' (see Fig. 8d). For all $\varepsilon \in [0, \varepsilon_D]$ with $\varepsilon_D = \text{val}(\alpha^*) - x_{i-1}$ we get equal cost of $\mathcal{T}$ and $\mathcal{T}''$ since we only shifted a merge operation. From Lemma 7 we get that $\text{val}(\alpha^*) = \text{val}(\delta) = x_i$ and hence $x_i - \varepsilon_D = x_{i-1}$.

Case 2: u_i has predecessors $\mathcal{P}$ and successors $\mathcal{F}$. Again, we first describe the upper Type-4-Tree $\mathcal{T}$ (see Fig. 9). Let $\{u_{i-\ell}, \dots, u_{i-1}\} \subseteq \mathcal{P}$ be the source nodes of the subtree of α. At α there is a potential move to α^*. Let $\{u_{i+1}, \dots, u_{i+r}\} \subseteq \mathcal{F}$ be the source nodes of the subtree of ζ. At ζ there is a potential move to ζ^* After a potential

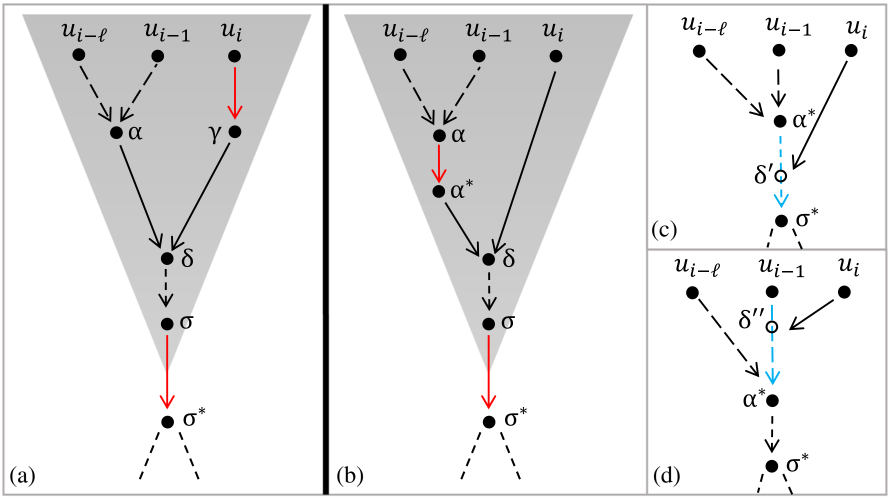


Fig. 8 Schematic representation of the trees discussed for Case 1 in the proof of Lemma 8. The node u_i has only predecessors. The dashed red edges show potential move operations. **(a)** Case 1.1: $x_{i-1} > x_i$. **(b)** Case 2.1: $x_{i-1} < x_i$. **(c)** Proof mechanism introducing the modified tree $\mathcal{T}'$, where the path on which the new node δ' can be shifted on is marked in blue. **(d)** Proof mechanism introducing the modified tree $\mathcal{T}''$ following the same mechanism as in (**c**) (Color figure online)

move from u_i to γ there is a merge with α^*, which is afterwards merged with ζ^* to an intermediate node δ. Without loss of generality, we assume this order of merge to δ. What follows are potential move and merge operations until all nodes in $N_{\mathcal{T}}(x)$ merge to the first bottleneck node σ. Since $\mathcal{T}$ is increasing, the subtree of δ is increasing. We further analyze the relation between x_i to its direct predecessor x_{i-1} and its direct successor x_{i+1}.

Case 2.1: $x_{i-1} < x_i < x_{i+1}$. By Lemma 7, it follows that $\mathrm{val}(\delta) = x_{i+1}$. Furthermore, there is no inc-edge between u_i and γ because u_i merges with α^* to β with a subsequent inc-edge to β^* (see Fig. 9a). We modify the tree structure of $\mathcal{T}$ for the replacement of x by $x_{\varepsilon,i}$. Let $\mathcal{T}'$ be the modified tree of $\mathcal{T}$, where we have the new node u_i^{ε} instead of u_i for $\varepsilon \in [0, \varepsilon_I], \varepsilon_I > 0$. In $\mathcal{T}'$ the node β does not exist anymore but $\mathcal{T}'$ includes a new node β', such that $\mathrm{val}(\beta') \in [\mathrm{val}(\alpha^*), \mathrm{val}(\beta^*)]$, where u_i^{ε} merges with α^*. The rest of the tree stays unchanged. For $\varepsilon \in [0, \varepsilon_I]$ with $\varepsilon_I = \mathrm{val}(\beta^*) - x_i$ the cost of $\mathcal{T}'$ is equal to the cost of $\mathcal{T}$ because we only shifted a merge operation to another position in the tree. This is a neutral increase for all $\varepsilon \in [0, \varepsilon_I]$. It holds that $x_i + \varepsilon_I = \mathrm{val}(\beta^*) = x_{i+1}$. Let further be $\mathcal{T}''$ another modified tree of $\mathcal{T}$. The tree $\mathcal{T}''$ does not contain the node β, instead it contains a new node β'', such that $\mathrm{val}(\beta'') \in [u_{i-1}, \mathrm{val}(\alpha^*)]$, where $u_i^{-\varepsilon}$ merges to. Again, we only shifted a merge operation, that leads to equal cost of $\mathcal{T}$ and $\mathcal{T}''$ for all $\varepsilon \in [0, \varepsilon_D]$ with $\varepsilon_D = \mathrm{val}(\alpha^*) - x_{i-1}$. We have $\mathrm{val}(\alpha^*) = x_i$ and hence $x_i - \varepsilon_D = x_{i-1}$.

Case 2.2: $x_{i-1} > x_i > x_{i+1}$. This case is analogous to Case 2.1.

Case 2.3: $x_{i-1} < x_i > x_{i+1}$. We further assume, without loss of generality, that $x_{i-1} < x_{i+1}$. By Lemma 7 it holds that $x_i = \mathrm{val}(\delta)$. We have inc-edges between u_{i-1} and α^* and between u_{i+1} and ζ^* (see Fig. 9b). The replacement of x by $x_{-\varepsilon,i}$ for an

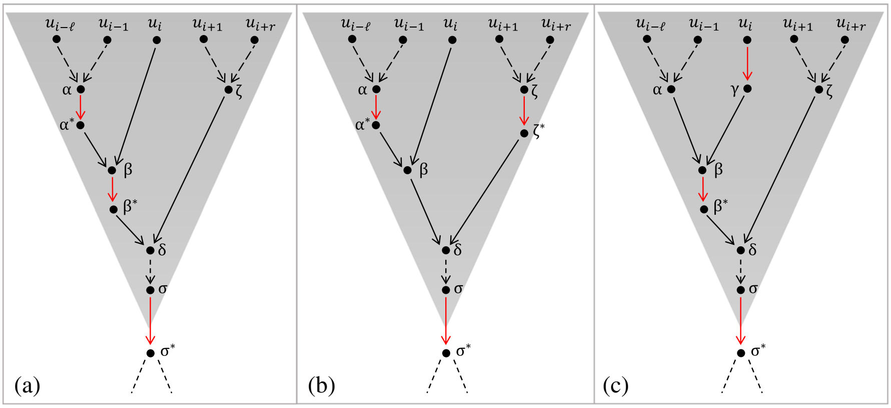


Fig. 9 Schematic representation of the trees discussed for Case 2 in the proof of Lemma 8. The node u_i has predecessors and successors. **(a)** Case 2.1: $x_{i-1} < x_i < x_{i+1}$. **(b)** Case 2.3: $x_{i-1} < x_i > x_{i+1}$. **(c)** Case 2.4: $x_{i-1} > x_i < x_{i+1}$

$\varepsilon \in [0, \varepsilon_D]$ is a beneficial decrease because the merge points β and δ are shifted by $-\varepsilon$: The move cost are $\text{val}(\alpha^*) - \varepsilon - x_{i-1}$ and $\text{val}(\zeta^*) - \varepsilon - x_{i+1}$ for the two move operations, that is a decrease of 2ε. The new merge node of β^* and ζ^* is denoted by δ'. For the new path between δ' and σ^* we have cost of $|\sigma^* - \delta' + \varepsilon|$, that is an increase of cost by ε. We get the left side of Equation (2), that is, $d(x_{-\varepsilon,i}, y) + \varepsilon = d(x, y)$ for all $\varepsilon \in [0, \varepsilon']$ with $\varepsilon_D = x_i - x_{i+1}$. It holds that $x_i - \varepsilon_D = x_{i+1}$. The argumentation of the detrimental replacement of x by $x_{\varepsilon,i}$ is analogous to Case 1.1.

Case 2.4: $x_{i-1} > x_i < x_{i+1}$. There is an inc-edge between u_i and γ (see Fig. 9c). Again, we further assume, without loss of generality, that $x_{i-1} < x_{i+1}$. Following the same argumentation as in Case 1.1, the replacement of x by $x_{\varepsilon,i}$ is a beneficial increase for all $\varepsilon \in [0, \varepsilon_I]$ with $\varepsilon_I = \text{val}(\gamma) - x_i$. By Lemma 7, it holds that $\text{val}(\beta) = x_{i-1}$ and $\text{val}(\delta) = x_{i+1}$. Note, that there are no increasing paths between u_{i-1} and β^* and u_{i+1} and ζ^* because otherwise there is no move between u_i and γ (see Lemma 5). It holds that $\text{val}(\gamma) = x_{i-1} = x_i + \varepsilon'$. The detrimental replacement of x by $x_{-\varepsilon,i}$ is analogous to Case 1.1. □

Let us briefly discuss the trees of Type 1–3. If the tree $\mathcal{T}$ is of Type 3, then $\text{val}(\sigma^*)$ is already in $\mathcal{N}_{\mathcal{T}}(y)$. The same proof as for trees of Type 4 can be applied. Since the symmetries properties hold for the MSM metric, the Lemma holds for trees of Type 2 as well. For a tree containing only a move edge, the argumentation is the same as in Case 1.1.

In the following, we regard a *block* $\mathcal{B}$ of adjacent source nodes $N_{\mathcal{B}}(x) = \{u_i, \ldots, u_\ell\}$ representing points of equal value of a time series x. A block is a maximal contiguous sequence of nodes with the same value. Our aim is to show a generalization of Lemma 8 shifting all points of a block $\mathcal{B}$ by some $\varepsilon \in \mathbb{R}$. We show that shifting a block is either beneficial for one direction or neutral for both directions. Let $x_{\varepsilon,i,\ell}$, $i < \ell$, denote the time series that is equal to x except at the positions $i, \ldots, \ell$, where the points x_i of x are replaced by $x_i + \varepsilon$. The definitions of beneficial, detrimental or neutral replacements of x by $x_{\varepsilon,i,\ell}$ are analogous to the previous one. A block may be

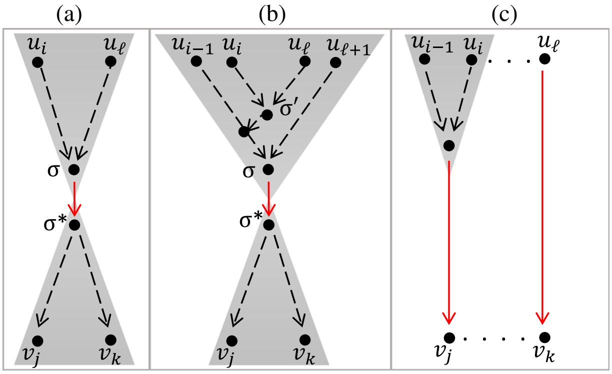


Fig. 10 Schematic representation of the three cases for proving Lemma 9, depending on the structure of a block $\mathcal{B}$. **(a)** Case 1.1: $N_{\mathcal{T}}(x) = N_{\mathcal{B}}(x)$. **(b)** Case 1.2: $|N_{\mathcal{T}}(x)| > |N_{\mathcal{B}}(x)|$. **(c)** Case 2: The nodes in $N_{\mathcal{B}}(x)$ belong to different trees

contained in several trees, hence shifting a block affects the cost of all these trees. To count the number of trees with beneficial or detrimental replacement, we introduce two further parameters $\rho_I, \rho_D \in \mathbb{N}$.

Lemma 9 *Let $x = (x_1, \ldots, x_m)$ and $y = (y_1, \ldots, y_n)$ be two time series with a distance $d(x, y)$. If we consider a block $\mathcal{B}$ of similar points $N_{\mathcal{B}}(x) = \{u_i, \ldots, u_\ell\}$ with $x_i \notin V(y)$, then there either exists an $\varepsilon' > 0$ and $\rho_I, \rho_D \in \mathbb{N}$ such that for all $\varepsilon \in [0, \varepsilon']$ one of the following equations holds:*

(1) $d(x_{\varepsilon,i,\ell}, y) + \rho_I \cdot \varepsilon = d(x, y) = d(x_{-\varepsilon,i,\ell}, y) - \rho_D \cdot \varepsilon$ *(b. increase),*
(2) $d(x_{-\varepsilon,i,\ell}, y) + \rho_D \cdot \varepsilon = d(x, y) = d(x_{\varepsilon,i,\ell}, y) - \rho_I \cdot \varepsilon$ *(b. decrease),*

or there exist $\varepsilon_I, \varepsilon_D > 0$ such that

(3.1) $d(x, y) = d(x_{\varepsilon,i}, y)$ *for all $\varepsilon \in [0, \varepsilon_I]$ (neutral increase), and*
(3.2) $d(x, y) = d(x_{-\varepsilon,i}, y)$ *for all $\varepsilon \in [0, \varepsilon_D]$ (neutral decrease).*

Moreover, for beneficial increases $x_i + \varepsilon' \in (V(y) \cup \{x_{i-1}, x_{i+1}\})$, for beneficial decreases $x_i - \varepsilon' \in (V(y) \cup \{x_{i-1}, x_{i+1}\})$, and for neutral increases and decreases $x_i + \varepsilon_I, x_i - \varepsilon_D \in (V(y) \cup \{x_{i-1}, x_{i+1}\})$.

Proof We distinguish whether all nodes of a block $\mathcal{B}$ belong to the same tree or if they are in different trees. Without loss of generality, we specify monotonic paths and trees to be increasing.

Case 1: All nodes $N_{\mathcal{B}}(x)$ are in one tree $\mathcal{T}$ (see Fig. 10a,b). Without loss of generality, $\mathcal{T}$ is considered to be a tree of Type 4, since all other cases follows the same or a simpler argumentation. We further distinguish whether the nodes of $N_{\mathcal{B}}(x)$ are the only nodes in $\mathcal{T}$.

Case 1.1: $N_{\mathcal{T}}(x) = N_{\mathcal{B}}(x)$. For the bottleneck nodes it holds that $\mathrm{val}(\sigma) = x_i$ and $\mathrm{val}(\sigma^*) \in V(y)$ (see Corollary 1). The replacement of x by $x_{\varepsilon,i,\ell}$ is a beneficial increase for all $\varepsilon \in [0, \varepsilon']$ with $\varepsilon' = \sigma^* - x_i$ because the intermediate node is also shifted to $\sigma + \varepsilon$ that leads to lower move cost of $\mathrm{val}(\sigma^*) - \mathrm{val}(\sigma) - \varepsilon$, that is, a decrease by ε. Therefore, we get the left side of the first equation $d(x_{\varepsilon,i,\ell}, y) + \varepsilon = d(x, y)$ for $\rho_I = 1$. It holds that $x_i + \varepsilon' = \text{val}(\sigma^*) \in V(y)$. For the right side of the equation with $\rho_D = 1$, the argumentation is similar. Replacing x by $x_{-\varepsilon,i,\ell}$ we get new move cost of $\text{val}(\sigma^*) - \text{val}(\sigma) + \varepsilon$, that is, an increase by ε.

Case 1.2: $|N_{\mathcal{T}}(x)| > |N_{\mathcal{B}}(x)|$. Since $\mathcal{T}$ is a tree of Type 4, all nodes in $N_{\mathcal{T}}(x)$ merge to the intermediate node σ. Moreover, it is evident that the merge of adjacent points in $\mathcal{T}$ that are equal creates lower cost than merging two points that are different. Therefore, there exists an intermediate node σ' where all nodes in $N_{\mathcal{B}}(x)$ merge to (see Fig. 10b). Then Lemma 8 can be applied for $u_i = \sigma'$ with $\rho_I = \rho_D = 1$. Depending on the case, an ε_I is specified such that we get one of the above equations for an $\varepsilon \in [0, \varepsilon_I]$. For $\varepsilon = \varepsilon_I$, the block is shifted until it reaches a value of the adjacent points of the block, that is x_{i-1} or $x_{\ell+1}$, or it reaches a point in $V(y)$.

Case 2: The nodes in $N_{\mathcal{B}}(x)$ belong to different trees (see Fig. 10c). In this case, we need to count how many trees we have beneficial increases and decreases. To decide whether a replacement is beneficial or detrimental, there are two possible cases of trees belonging to the block $\mathcal{B}$. The first case is that all nodes in a tree in $\mathcal{B}$ belong to $N_{\mathcal{B}}(x)$. Then we can apply Case 1.1. The second case concerns the boundary values of $N_{\mathcal{B}}(x)$ merging with the predecessors or successors of the block $\mathcal{B}$. Following the argumentation of Case 1.2, we determine if the replacement of x by $x_{\varepsilon,i,\ell}$ is beneficial, neutral, or detrimental. Ignoring neutral replacements, we set $x^+_{i,\ell}$ as the number of trees for which we have a beneficial increase and $x^-_{i,\ell}$ as the number of trees for which we have a beneficial decrease. Shifting a whole block may therefore lead to a reduction of distance of more than ε. We get the above statement for $\rho_I = x^+_{i,\ell}$ and $\rho_D = x^-_{i,\ell}$. By Lemma 8, we get an $\varepsilon_{\mathcal{T}}$ for all trees $\mathcal{T}$ in a block $\mathcal{B}$ restricting a beneficial or neutral replacement. Let ε^+_{min} be the minimum of all $\varepsilon_{\mathcal{T}}$ for which we have a beneficial increase. Analogously, ε^-_{min} is defined for beneficial decreases. Without loss of generality, we assume $x^+_{i,\ell} > x^-_{i,\ell}$. Hence, it holds that for $x_i + \varepsilon^+_{min}$ is in $\{x_{i-1}, x_{\ell+1}\}$ or in $V(y)$. □

4.3 MSM mean values

We now use beneficial and neutral replacements to prove that for any set X there exists a mean m such that all points of m are points of at least one time series of X.

Lemma 10 *Let $X = \{x^{(1)}, \ldots, x^{(k)}\}$ be a set of k time series. Then there exists a mean $m = (m_1, \ldots, m_N)$ of X such that $m_i \in V(X)$ for all $m_i, i \in [N]$.*

Proof Assume towards a contradiction that every mean has at least one point that is not in $V(X)$. Among all means, choose a mean m such that 1) n_V, the number of points of m that are in $V(X)$, is maximum and 2) among all means with n_V points from $V(X)$, the number of transitions from $m^{(i)}_j$ to $m^{(i)}_{j+1}$ where $m^{(i)}_j \neq m^{(i)}_{j+1}$ is minimum. In other words, m has a minimal number of blocks. Let $\mathcal{B}$ be a block in m whose points are not in $V(X)$. We apply Lemma 9 to show that there exists an $\varepsilon \in \mathbb{R}$ such that $m_{\varepsilon,i,\ell}$ is a mean where the points of the shifted block $\mathcal{B}$ reach a point of a predecessor or successor of $\mathcal{B}$ or a point in $V(X)$. We now specify ε. First, we determine whether ε is positive or negative. For each sequence, one of the Cases (1) to (3) of Lemma 9 applies. For each time series in X, we introduce two variables to count how many beneficial increases and decreases we have. Neutral replacements are not counted. Let x^+ be the sum of ρ_I for beneficial increases and x^- be the sum of ρ_D for beneficial decreases of all time series. If $x^+ \geq x^-$, we set ε as the minimum of the specified ε' for beneficial increases and all ε_I (see Lemma 9). If $x^+ < x^-$ we set ε as the maximum of the specified $-\varepsilon'$ for beneficial decreases and all $-\varepsilon_D$. Compared to the mean m, all values of $m_{\varepsilon,i,\ell}$ are the same except the values of the shifted block. By Lemma 9 the points of the new mean $m_{\varepsilon,i,\ell}$ are shifted for the specified ε until they reach a point of the right or left neighbor block or a point in $V(X)$. If they reach a point of the right or left neighbor block, we have a contradiction to the selection of a mean with a minimal number of transitions. If they reach a point in $V(X)$, we have the contradiction to the selection of a mean with minimal number of values that are not in $V(X)$. □

5 Computing an MSM mean

Based on Lemma 10, we now give a dynamic program computing a mean m of k time series $X = \{x^{(1)}, \ldots, x^{(k)}\}$. The transformation operations are described for the direction transforming X to m.

5.1 Dynamic program

We fill a $(k+2)$-dimensional table D with entries $D[(p_1, \ldots, p_k), \ell, s]$, where

- $p_i \in [n_i]$ indicates the *current position* in time series $x^{(i)}$,
- the index $\ell \in [N]$ indicates the *current position of* m, and
- s is the index of a point $v_s \in V(X)$.

We also say that $(p_1, \ldots, p_k)$ are the *current positions of* X. For clarity, we write $p = (p_1, \ldots, p_k)$. The entry $D[p, \ell, s]$ represents the cost of the partial time series $\{(x_1^{(1)}, \ldots, x_{p_1}^{(1)}), \ldots, (x_1^{(k)}, \ldots, x_{p_k}^{(k)})\}$ transforming to a mean $(m_1, \ldots, m_\ell)$ assuming that $m_\ell = v_s$. Giving a recursive formula filling table D we have two transformation cases. The case distinction is based on the computation of the MSM distance for two time series $x = (x_1, \ldots, x_m)$ and $(y_1, \ldots, y_n)$. This computation fills a two-dimensional table D^*. An entry $D^*[i, j]$ represents the cost of transforming the partial time series $(x_1, \ldots, x_i)$ to the partial time series $(y_1, \ldots, y_j)$. The distance $d(x, y)$ is given by $D^*[m, n]$. Stefan et al. (2012) give the recursive formulation of the MSM metric as the minimum of the cost for the three transformation operations.

$$\begin{aligned}
D^*[i, j] &= \min\{A_{MO}[i, j], A_M[i, j], A_{SP}[i, j]\}, \text{ where} \\
A_{MO}[i, j] &= D^*[i-1, j-1] + |x_i - y_i| && (move) \\
A_M[i, j] &= D^*[i-1, j] + C(x_i, x_{i-1}, y_j) && (merge) \\
A_{SP}[i, j] &= D^*[i, j-1] + C(y_j, x_i, y_{j-1}) && (split)
\end{aligned}$$

for

$$C(x_i, x_{i-1}, y_j) = \begin{cases} c & \text{if } x_{i-1} \leq x_i \leq y_j \text{ or } x_{i-1} \geq x_i \geq y_j \\ c + \min(|x_i - x_{i-1}|, |x_i - y_j|) & \text{otherwise.} \end{cases}$$

When the recursion reaches a border of D^*, we have the special cases $D^*[i, 1] = D^*[i-1, 1] + C(x_i, x_{i-1}, y_1)$ (only merge operation may be further applied) and $D^*[1, j] = D^*[1, j-1] + C(y_j, x_1, y_{j-1})$ (only split operation may be further applied). The base case is reached for $D^*[1, 1] = |x_1 - y_1|$ where only a move operation is applied.

For the recursion formula for the MSM- Mean problem, we distinguish between applying moves and splits (A_{MS}) and only merges (A_{ME}):

$$D[p, \ell, s] = \min\{A_{MS}[p, \ell, s], A_{ME}[p, \ell, s]\}.$$

To distinguish between these cases, we introduce index sets I_{MO}, I_{SP}, and I_{ME} for move, split, and merge operations, respectively. They represent the indices for those time series which either move, split, or merge. All index sets are subsets of $I = [k]$. Let $\overline{p}_{I_{MO}}$ be the tuple obtained from p by setting $\overline{p}_i = p_i - 1$ for all $i \in I_{MO}$. The tuple $\overline{p}_{I_{ME}}$ is defined analogously. The first case considers that at some current positions of X there are move and at all other positions there are split operations. It holds that $I_{MO} \cup I_{SP} = I$. For these operations, the recursive call of the function decreases the current position of m:

$$\begin{aligned} A_{MS}[p, \ell, s] = \min_{v_{s'} \in V(X)} \{ \min_{I_{MO}, I_{SP}} \big(& D[\overline{p}_{I_{MO}}, \ell - 1, s'] \\ & + \sum_{i \in I_{MO}} |x_{p_i}^{(i)} - v_s| + \sum_{i \in I_{SP}} C(v_s, x_{p_i}^{(i)}, v_{s'}) \big) \}. \end{aligned}$$

The second case treats merge operations of at least one current position of X. If a merge is applied, all other time series pause since the recursive call does not decrease the position of m:

$$A_{ME}[p, \ell, s] = \min_{I_{ME}} \big(D[\overline{p}_{I_{ME}}, \ell, s] + \sum_{i \in I_{ME}} C(x_{p_i}^{(i)}, x_{p_i - 1}^{(i)}, v_s) \}.$$

For the last step in the recursion, the entries $D[(1, \ldots, 1), 1, s]$ for all $v_s \in V(X)$ are calculated by

$$D[(1, \ldots, 1), 1, s] = \sum_{i \in I} |x_1^{(i)} - v_s|.$$

All entries $D[p, \ell, s]$ for which $p_i < 1$ for some $i \in [k]$ is set to $+\infty$. If $\ell = 1$ and all $p_i > 1$, only merge operations may be applied:

$$D[p, 1, s] = \min_{I_{ME} \subseteq I} \big(D[\overline{p}_{I_{ME}}, 1, s] + \sum_{i \in I_{ME}} C(x^{(i)}_{p_i}, x^{(i)}_{\overline{p}_i}, v_s).$$

The correctness of the dynamic program hinges on the fact that in the recursive definition of the pairwise distance, the value of $D^*[i, j]$ depends only on the values of $D^*[i, j-1]$, $D^*[i-1, j]$, x_i, y_j, x_{i-1}, and y_{j-1}; we omit the formal correctness proof.

5.2 Running time bound

We now show an upper bound on the maximum mean length in terms of the total length of X. To this end, we first make the observation that the index set I_{MO} is never empty. That is, it is not optimal to apply only split operations in one recursion step.

Lemma 11 *Let m be a mean of a set of k time series X. It holds that $D[p, \ell, s] < \min_{v_{s'} \in V(X)} \{D[p, \ell - 1, s'] + \sum_{i \in I} C(v_s, x^{(i)}_{p_i}, v_{s'})\}$.*

Proof Let $D(X, m)$ be the distance of a mean m to X. Assume towards a contradiction, that there exists a recursion step, where $I_{MO} = \emptyset$. That is, in each time series in X there is a split at a point $x^{(i)}_{p_i}$, $i \in I$ to the points m_ℓ and $m_{\ell-1}$. We regard the cost for the transformation up to the positions $(p_1, \dots, p_k)$ of X and ℓ of m. Applying the recursion formula for $I_{SP} = I$, we get $D[p, \ell, s] = \min_{v_{s'} \in V(X)} \{D[p, \ell - 1, s'] + \sum_{i \in I} C(v_s, x^{(i)}_{p_i}, v_{s'})\}$. Let m' be a mean of X equal to m but where m_ℓ is deleted. For the mean m', we save the cost for splitting $\sum_{i \in I} C(v_s, x^{(i)}_{p_i}, v_{s'})$, without changing the alignment of all other points in X. It follows that $D(X, m') < D(X, m)$. This is a contradiction to m being a mean. □

Lemma 11 now leads to the following upper bound for the MSM mean length.

Lemma 12 *Let $X = \{x^{(1)}, \dots, x^{(k)}\}$ be a set of time series with maximum length $max_{j \in [k]} |x^{(j)}| = n$. Then, every mean m has length at most $(n-1)k + 1$.*

Proof Towards a contradiction, let m be a mean of X with length $N > (n-1)k + 1$. The entry of the first recursion call is $D[(n_1, \dots, n_k), N, s]$. Consider any sequence of recursion steps from $D[(n_1, \dots, n_k), N, s]$ to $D[(1, \dots, 1), \cdot, \cdot]$; each step is associated with index sets I_{MO} and I_{SP}, or I_{ME}. By Lemma 11, it holds that $I_{MO} \neq \emptyset$ in each step. That is, at least one current position of X is reduced by one in each recursion step until the entry $D[(1, \dots, 1), \ell', s']$ is reached. These are at most $(n-1)k + 1$ recursion steps. Since $N > (n-1)k + 1$, it holds that $\ell' > 1$. The only possibility for a further recursion step for $D[(1, \dots, 1), \ell', s']$ is to set $I_{SP} = I$, since $D[p, \ell, s] = +\infty$ whenever $p_i < 1$ for some i. By Lemma 11, we get a contradiction to m being a mean. □

We now bound the running time of our algorithm.

Lemma 13 *The* MSM- MEAN *problem for k input time series of length at most n can be solved in time $\mathcal{O}(n^{k+3}2^k k^3)$.*

Proof In the dynamic programming table D at most $n^{k+2}k^2$ entries have to be computed. This number is the dimension of k time series with maximum length n, the maximum length of the mean $(n-1)k+1 \leq kn$ and the size of $V(X)$ which is at most kn. For each table entry, the minimum over the set $V(X)$ is taken, which includes again kn data points. For each minimum over $V(X)$ all subsets of $[k]$ are considered which are at most 2^k sets. All subsets of $[k]$ are only generated once for both I_{MO} and I_{ME}. Thus, filling the table iteratively takes time $\mathcal{O}(n^{k+3}2^k k^3)$.

For the traceback, the start entry of D is any of the position $(n_1, \ldots, n_k)$ with minimal cost, that is,

$$D[(n_1, \ldots, n_k), \ell_{start}, s_{start}] = \min_{\ell,s} D[(n_1, \ldots, n_k), \ell, s].$$

The length of the mean m is ℓ_{start} with $m_{start} = v_{s_{start}}$. In each traceback step, the *predecessors* of the current entry are determined, that are the entries leading to the cost of the current entry. A predecessor of an entry is not unique. For setting the mean data point we consider the current entry $D[(p_1, \ldots, p_k), \ell, s]$ and the entry of the predecessor $D[(q_1, \ldots, q_k), \ell', s']$. If $\ell' = \ell - 1$, the point $v_{s'}$ is assigned to the mean point $m_{\ell'}$ and we continue with the next traceback step. Otherwise, the next traceback step is directly applied without assigning a mean point. We repeat this procedure until we reach the entry $D[(1, \ldots, 1), 1, s^*]$. The running time of filling the table clearly dominates the linear time for the traceback. □

5.3 Implementation & window heuristic

We fill the table D iteratively and apply the above described traceback mechanism afterwards. Since the running time of MSM- MEAN will often be too high for moderate problem sizes, we introduce the *window heuristic* to avoid computing all entries of table D. Similar to a heuristic of the Levenshtein distance (Ukkonen 1985), the key idea is to introduce a parameter d called the window size representing the maximum difference between the current positions of the time series within the recursion. All entries whose current positions are not within distance d will be discarded. For example, an entry with current position (6, 3, 4) of X will not be computed for $d = 2$. In the case of a set of time series with unequal lengths, where n_{min} and n_{max} denotes the minimum length and the maximum lengths, respectively, of all time series, d has to be greater than $n_{max} - n_{min}$.

6 Experimental evaluation

This section provides important results from a selection of experiments using implementations of mean algorithms.[1] After a description of the experimental setup, we

[1] All code is available on GitHub: https://github.com/JanaHolznigenkemper/msm_mean.

Table 1 List of 21 UCR time series data sets

Data Set	#Classes	#TS Training	TS Length	MSM c
50words	50	450	270	1
Adiac	37	390	176	1
Beef	5	30	470	0.1
CBF	3	30	128	0.1
Coffee	2	28	286	0.01
ECG	2	100	96	1
FaceAll	14	560	131	1
Face (four)	4	24	350	1
Fish	7	175	463	0.1
Gun Point	2	50	150	0.01
Italy Power*	2	67	24	*
Lightning-2	2	60	367	0.01
Lightning-7	7	70	319	1
OliveOil	4	30	470	0.01
OSU Leaf	6	200	427	0.1
Swedish Leaf	15	500	128	1
Synthetic C	6	300	60	0.1
Trace	4	100	275	0.01
Two Pattern	4	1000	128	1
Wafer	2	1000	152	1
Yoga	2	300	426	0.1

For our running time experiment, we did not take the Italy Power Demand data set (*) since the time series are too short. The quality analysis of MSM- MEAN using this data set was conducted for $c \in \{0.01, 0.1, 0.2, 0.5\}$

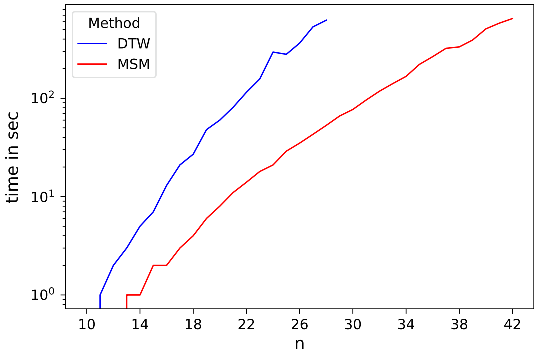
first provide a running time comparison of the DTW- MEAN algorithm (Brill et al. 2019) and our MSM- MEAN algorithm. Furthermore, we examine accuracy and running times of MSM- MEAN for various heuristics.

6.1 Experimental setup

The running times of our Java implementations are measured on a server with Ubuntu Linux 20.04 LTS, two AMD EPYC 7742 CPUs at 2.25 Ghz (2.8 Ghz boost), 1TB of RAM, Java version 15.0.2. Our implementations are single-threaded. For our results at most 26GB of RAM were occupied.

The experiments are conducted on 20 UCR data sets (Chen et al. 2015) that Stefan et al. (2012) already used (see Table 1). The UCR data sets were collected for the use of time series classification. Each set consists of a training and a testing set. The data sets consists of time series of different classes and different lengths. Since we are not using the sets for classification use cases yet, we just take the training sets of each data set for our experimental setup. The parameter c is set constant for every data set following the suggestions of Stefan et al. (2012).

Fig. 11 Running time comparison of MSM- MEAN and DTW- MEAN for $k = 3$ as a function of n



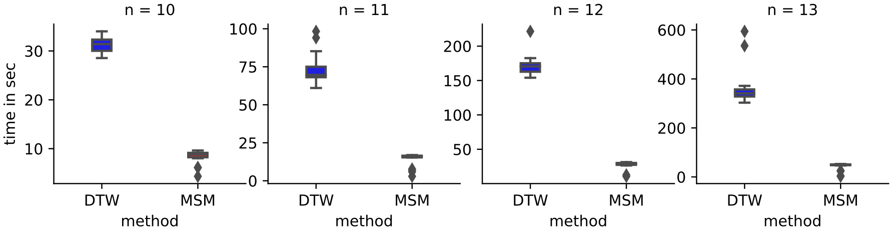


Fig. 12 Running time comparison of MSM- MEAN and DTW- MEAN for $k = 4$

Due to the complexity of the algorithms, we draw time series samples from the training sets obtained from the UCR archive in the following way. For each class of the training sets, we randomly pick k time series, $k \in \{3, 4, 5\}$, and for each of them, we cut out a contiguous subsequence of length n starting at a random data point, $n \in \{10, \dots, 50\}$. In addition, we limit the length of the mean time series to at most n.

6.2 Running time comparisons

In our first experiment, we consider the running times for $k = 3, 4$. Figure 11 shows the average running time over all 20 data sets as a function of n. Our MSM- MEAN algorithm is substantially faster than the DTW counterpart. The outlier in the DTW graph is due to a data set where the implementation does not complete within 10 min. Figure 12 in the appendix provides box plots depicting the running times of both algorithms for $k = 4$ and $n = 10, \dots, 13$. They reveal that the MSM- MEAN algorithm has smaller medians and interquartile ranges and fewer outliers of high running times compared to the DTW- MEAN algorithm. The MSM- MEAN implementation was able to compute any instance for $k = 3$, $n < 43$, $k = 4$, $n < 19$, and $k = 5$, $n < 11$ within 10 min. For DTW- MEAN, this was only the case for $k = 3$, $n < 29$ and $k = 4$, $n < 14$.

Table 2 Distance of the mean to time series of the data set *ItalyPowerDemand* of one class and to time series of mixed classes for $k = 3$ and $n = 24$ for varying c

c	0.01	0.1	0.2	0.5
One class	5.55	7.05	8.02	9.72
Two classes	11.66	15.41	18.46	26.1

6.2.1 MSM mean quality

To evaluate the quality of the computed MSM mean, we use the algorithm on the *ItalyPowerDemand* data set (Chen et al. 2015) where each time series has length 24. The data set contains two classes. For different values of $c \in \{0.01, 0.1, 0.2, 0.5\}$, Table 2 shows the distance of the MSM mean to three other time series. The first row reports the distance when the time series belong to one class, while the second row provides the distance when taking them from both classes. The results confirm for all c that distances of the MSM mean are lower when the time series belong to one class.

6.2.2 Length of the mean

We implemented two versions of the MSM- Mean algorithm, one with a fixed length n of the mean and one without length constraints. As shown in Lemma 12, the length of an MSM mean is at most $(n-1)k+1$. However, in our experiments for $k=3$ and $n \in \{10, \ldots, 30\}$, the length of an MSM mean is always exactly n. Thus, it is advisable to use this constraint as done in the experiments discussed above.

6.2.3 Impact of parameter *c*

For a short informal analysis of the effect of the parameter c, consider Figs. 13 and 14. Each figure shows two MSM- Mean instances for the same four time series and different values of c. The top graphs show the input time series and an optimal mean. The graphs below depict the detailed transformation structure from one of the input time series to the respective mean.

In Fig. 13 on the left c is set to 0.1; on the right c is set to 1. The four time series of the data set (samples from *OSU Leaf*) are not very similar and consequently, there is no intuitively correct mean. The three alignment plots below depict the transformation structure of the time series TS1, TS3 and TS4 to the mean. The alignment of TS2 to the mean only contains move operations and is therefore omitted. The mean is quite different from TS1, TS3, and TS4. For $c = 0.1$, this leads to many merge and split operations in the transformations. For $c = 1$, the merge and split costs are too high and many more move operations are executed, some of them being quite costly.

In the second example of four time series of the data set *Italy Power Demand*, all time series show a similar behavior. For both $c = 0.01$ (left) and $c = 0.1$ (right), the mean follows an intuitively correct curve. For both cases, the majority of transformation operations are short moves and only short consecutive intervals are merged or split in the mean or in the input time series.

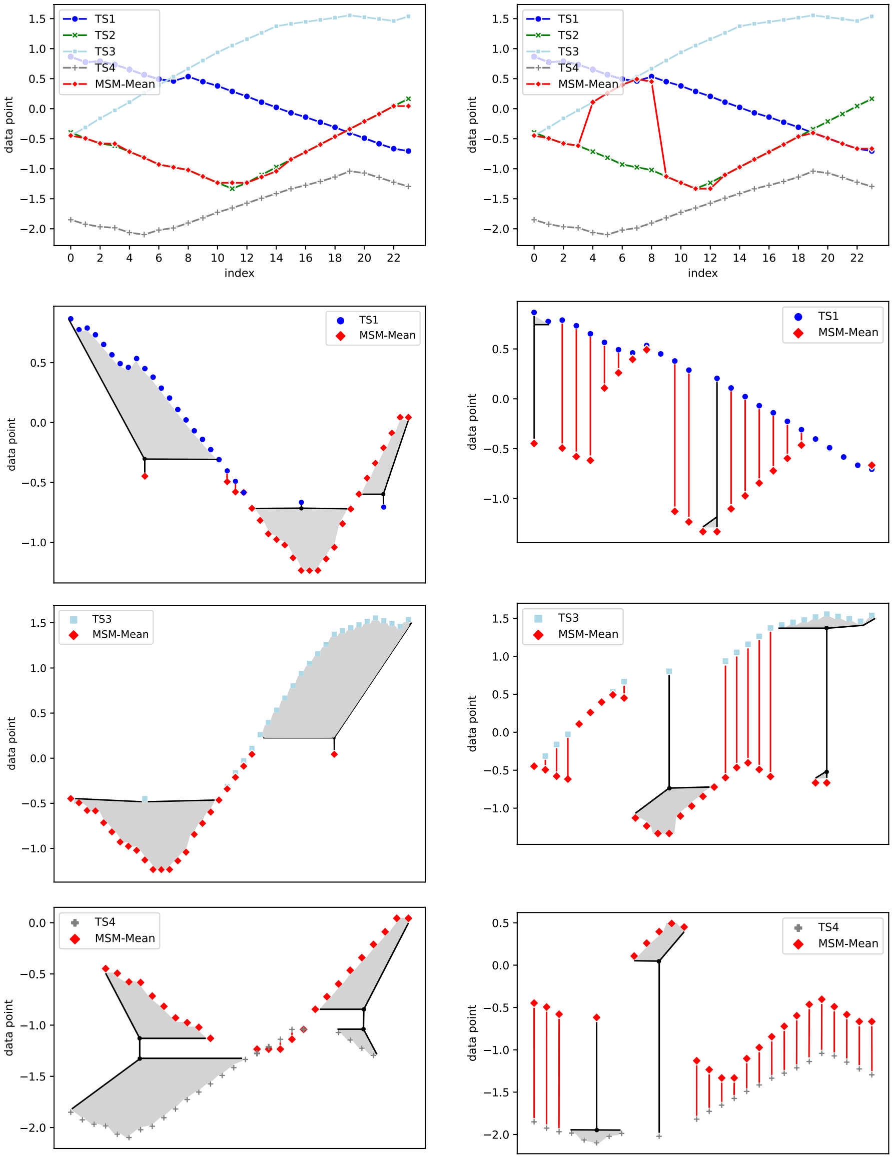


Fig. 13 OSU Leaf, left: $c = 0.1$, right: $c = 1$. The alignment plot from TS2 to the MSM mean is omitted since it only includes move operations in both cases

6.3 MSM-Mean heuristics

6.3.1 Discretization heuristic

Because the domain size of the values of a time series has a significant effect on the performance of MSM- Mean algorithm, we propose a second heuristic where the

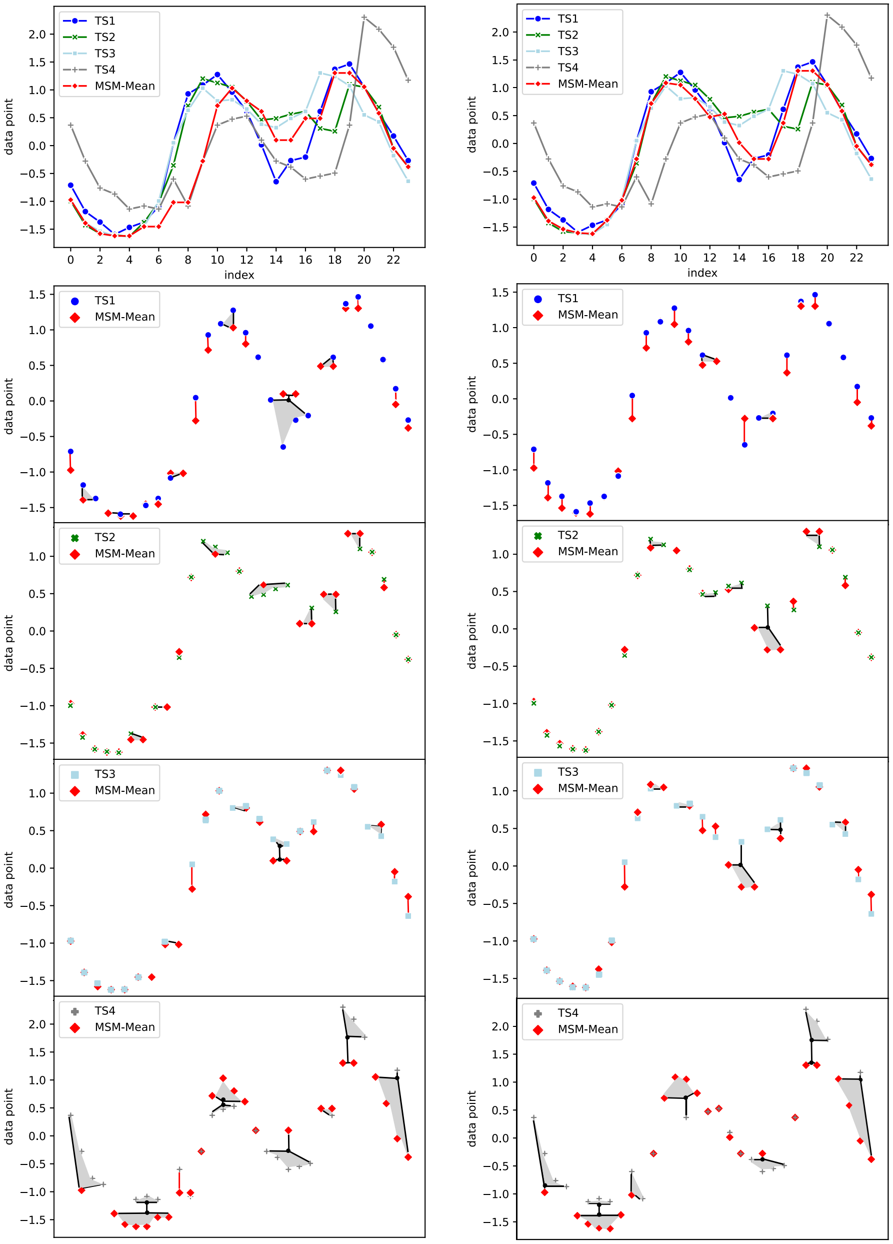


Fig. 14 Italy Power Demand, left: $c = 0.01$, right: $c = 0.1$

Fig. 15 Running time for MSM- MEAN calculation for $k = 3$ and $n = 30$ regarding the discretization heuristic

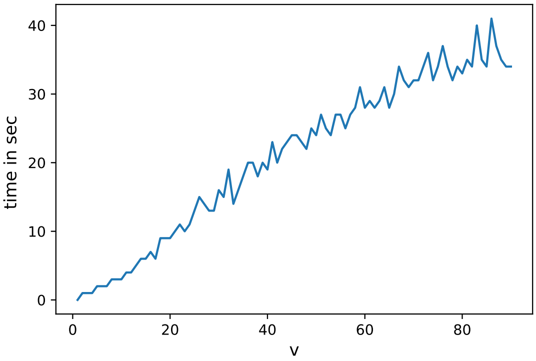
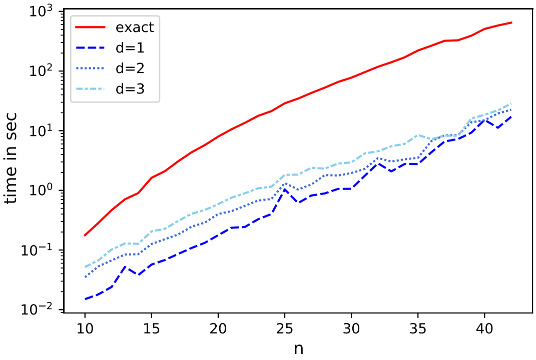


Fig. 16 Average running time of computing the exact mean and mean using the window heuristic for $d = 1, 2, 3$ for $k = 3$ and $n \in \{10, \ldots, 42\}$



domain is split into v buckets of equal length. Each value x of a time series is then replaced by the center point of the bucket to which x belongs. Thus, there are at most v different values in total.

Figure 15 shows the running time of this heuristic as a function of v for $k = 3$ and $n = 30$. There is a substantial (close to linear) decrease in the running time with a decreasing number of buckets. Moreover, the relative error is quite moderate: We observed an average and maximum error of 4.6% and 8.47%, respectively.

6.3.2 Window heuristic

In the following, we investigate the window heuristic described in Sect. 5.3 for the MSM- MEAN problem and $k = 3$, $n \in \{10, \ldots, 42\}$. We examine the window size $d = 1, 2, 3$ in our experiments. We analyze relative error of the exact mean and the means obtained from the window heuristic. As expected, the higher the window size d the smaller is the relative error. The relative error averaged over all n and all data sets was 4.8%, 3.2%, 2.4% and the maximum relative error was 9.1%, 6.4%, 5.4% for $d = 1, 2, 3$, respectively. Figure 16 shows the running time of the window heuristic in comparison to the exact computation as a function of n. Note that the y-axis plots the running time on a logarithmic scale. For all parameter settings, the running times improve substantially in comparison to the exact approach.

7 Conclusion and future work

This paper introduces the MSM- Mean problem of computing the mean of a set of time series for the Move-Split-Merge (MSM) metric. We present an exact algorithm for MSM- Mean with a better running time than a recent algorithm for computing the mean for the DTW distance. Experimental results confirm the theoretically proven superiority of our MSM- Mean algorithm in comparison to the DTW counterpart. The key observation of our method is that an MSM mean exists whose data points occur in at least one of the underlying time series. In addition, we provide an an upper bound for the length of an MSM mean. In our experiments, the maximum mean length is much shorter, rarely exceeding the length of the longest time series. The paper also provides two heuristics for speeding up the computation of the mean without sacrificing much accuracy, as shown in our experimental evaluation.

In future work we will tackle the following issues for MSM- Mean. First, we will examine how to use MSM- Mean in real clustering and classification problems. Second, we plan to develop optimization strategies such as the *A*-Algorithm* (Hart et al. 1968) for further improving the running time of our algorithm to avoid filling up the entire dynamic programming table. As a starting point, the structure of the transformation forests and the metric properties of the MSM distance could be further explored. Third, the metric properties, especially the triangle inequality, of the MSM distance enables applications of MSM means in metric indexing. Finally, we conjecture MSM- Mean to be NP-hard. Proving this conjecture could be a next research step.

More broadly, it would also be interesting to consider extensions of the MSM metric to more general types of time series data. Following the classification of Su et al. (2020), a first extension could be to consider time series with explicit time stamps for the data points. For such time series, one could consider for example merge costs that take into account the temporal distance between the merged points. Moreover, one could consider time series in higher-dimensional spaces, for example 3-dimensional trajectory data. Here, the main issue seems to be to suitably adapt the move distance to higher dimensions. One could use for example, Euclidean distances but this is only an option. After establishing appropriate generalizations of the MSM metric, it would remain to examine the performance of these generalizations in different applications, for example in trajectory clustering.

Funding Open Access funding enabled and organized by Projekt DEAL.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

References

Aach J, Church GM (2001) Aligning gene expression time series with time warping algorithms. Bioinformatics 17:6495–508

Aghabozorgi S, Shirkhorshidi AS, Wah TY (2015) Time-series clustering-a decade review. Inf Syst 53:16–38

Bader A, Kopp O, Falkenthal M (2017) Survey and comparison of open source time series databases. Datenbanksysteme für Business, Technologie und Web (BTW 2017)-Workshopband

Bagnall A, Lines J, Bostrom A, Large J, Keogh E (2017) The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. Data Min Knowl Discov 313:606–660

Berndt DJ, Clifford J (1994) Using dynamic time warping to find patterns in time series. In: Knowledge discovery in databases: papers from the 1994 AAAI workshop. Technical Report WS-94-03, vol 10, pp 359–370

Brill M, Fluschnik T, Froese V, Jain B, Niedermeier R, Schultz D (2019) Exact mean computation in dynamic time warping spaces. Data Min Knowl Discov 331:252–291

Chen Y, Keogh E, Hu B, Begum N, Bagnall A, Mueen A, Batista G (2015) The UCR time series classification archive. http://www.cs.ucr.edu/~eamonn/time_series_data/

Chen L, Gao Y, Zheng B, Jensen CS, Yang H, Yang K (2017) Pivot-based metric indexing. Proc VLDB Endow 10(10):1058–1069

Cuturi M, Blondel M (2017) Soft-DTW: a differentiable loss function for time-series. In: Proceedings of the 34th international conference on machine learning (ICML '17), vol 70, pp 894–903. PMLR

Das G, Lin K, Mannila H, Renganathan G, Smyth P (1998) Rule discovery from time series. In: Proceedings of the fourth international conference on knowledge discovery and data mining (KDD '98). AAAI Press, pp 16–22

Fréchet M (1948) Les éléments aléatoires de nature quelconque dans un espace distancié. Annales de l'Institut Henri Poincaré 10:215–310

Garcia-Arellano C, Storm AJ, Kalmuk D, Roumani H, Barber R, Tian Y, Pirahesh H (2020) Db2 event store: a purpose-built IoT database engine. Proc VLDB Endow 13(12):3299–3312

Hart PE, Nilsson NJ, Raphael B (1968) A formal basis for the heuristic determination of minimum cost paths. IEEE Trans Syst Sci Cybernet 42:100–107

Hautamäki V, Nykänen P, Fränti P, (2008) Time-series clustering by approximate prototypes. In: Proceedings of the 19th international conference on pattern recognition (ICPR '08). IEEE Computer Society, pp 1–4

Jensen SK, Pedersen TB, Thomsen C (2017) Time series management systems: a survey. IEEE Trans Knowl Data Eng 29(11):2581–2600

Jiang W (2020) Time series classification: nearest neighbor versus deep learning models. SN Appl Sci 2(4):1–17

Levenshtein VI (1966) Binary codes capable of correcting deletions, insertions, and reversals. Soviet Phys Dokl 10:707–710

Liao TW (2005) Clustering of time series data-a survey. Pattern Recognit 38(11):1857–1874

Lines J, Bagnall A (2015) Time series classification with ensembles of elastic distance measures. Data Min Knowl Discov 29(3):565–592

MacQueen J (1967) Some methods for classification and analysis of multivariate observations. In: Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, vol 1, pp 281–297

Niennattrakul V, Ratanamahatana CA, (2007) On clustering multimedia time series data using k-means and dynamic time warping. In: Proceedings of the 2007 international conference on multimedia and ubiquitous engineering (MUE '07), pp 733–738

Novak D, Batko M, Zezula P (2011) Metric index: an efficient and scalable solution for precise and approximate similarity search. Inf Syst 36(4):721–733

Paparrizos J, Gravano L (2017) Fast and accurate time-series clustering. ACM Trans Database Syst (TODS) 4(2):21–49

Paparrizos J, Liu C, Elmore AJ, Franklin MJ (2020) Debunking four long-standing misconceptions of time-series distance measures. Proceedings of the 2020 ACM SIGMOD international conference on management of data, pp 1887–1905

Petitjean F, Gançarski P (2012) Summarizing a set of time series by averaging: from steiner sequence to compact multiple alignment. Theor Comput Sci 414(1):76–91

Petitjean F, Ketterlin A, Gançarski P (2011) A global averaging method for dynamic time warping, with applications to clustering. Pattern Recognit 44(3):678–693

Petitjean F, Forestier G, Webb GI, Nicholson AE, Chen Y, Keogh E (2016) Faster and more accurate classification of time series by exploiting a novel dynamic time warping averaging algorithm. Knowl Inf Syst 4(7):11–26

Rani S, Sikka G (2012) Recent techniques of clustering of time series data: a survey. Int J Comput Appl 52(15):1–19

Sakoe H, Chiba S (1978) Dynamic programming algorithm optimization for spoken word recognition. IEEE Trans Acoust Speech Signal process 26(1):43–49

Sakurai Y, Yoshikawa M, Faloutsos C (2005) FTW: fast similarity search under the time warping distance. In: Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems. ACM, pp 326–337

Sankoff D, Kruskal JP (1983) Time warps, string edits, and macromolecules: the theory and practice of sequence comparison Time warps, string edits, and macromolecules: the theory and practice of sequence comparison, vol 10. Addison-Wesley, Boston

Schultz D, Jain B (2018) Nonsmooth analysis and subgradient methods for averaging in dynamic time warping spaces. Pattern Recognit 74:340–358

Stefan A, Athitsos V, Das G (2012) The move-split-merge metric for time series. IEEE Trans Knowl Data Eng 25(6):1425–1438

Su H, Liu S, Zheng B, Zhou X, Zheng K (2020) A survey of trajectory distance measures and performance evaluation. VLDB J 29:13–32

Ukkonen E (1985) Algorithms for approximate string matching. Inf Control 64(1–3):100–118

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.