Improving the Efficiency of Traditional DTW Accelerators

Romain Tavenard, Laurent Amsaleg

Romain Tavenard[*] , Laurent Amsaleg[**]

rtavenar@irisa.fr, lamsaleg@irisa.fr

Abstract: Dynamic Time Warping (DTW) is the most popular approach for evaluating the similarity of time series, but its computation is costly. Therefore, simple functions lower bounding DTW distances have been designed, accelerating searches by quickly pruning sequences that could not possibly be best matches. The tighter the bounds, the more they prune and the better the performance. Designing new functions that are even tighter is difficult because their computation is likely to become complex, canceling the benefits of their pruning. It is possible, however, to design simple functions with a higher pruning power by relaxing the *no false dismissal* assumption, resulting in approximate lower bound functions. This paper describes how very popular approaches accelerating DTW such as LB_Keogh and LB_PAA can be made more efficient *via* approximations. The accuracy of approximations can be tuned, ranging from no false dismissal to potential losses when aggressively set for great response time savings. At very large scale, indexing time series is mandatory. This paper also describes how approximate lower bound functions can be used with iSAX. Furthermore, it shows that a k-means-based quantization step for iSAX gives significant performance gains.

Key-words: Dynamic Time Warping; Indexing; Lower Bounds; Upper Bounds; Indexing Trees

---

Improving the Efficiency of Traditional DTW Accelerators

---

[*] TEXMEX
[**] TEXMEX

1 Introduction

Searching in very large databases of time series has received a lot of attention from researchers. Dynamic Time Warping (DTW) has proved to be an excellent similarity measure for time series. It has been successfully applied to domains as diverse as economics, life sciences and bioinformatics, pattern recognition, monitoring, speech recognition, *etc.* DTW is of great help when using any data represented as a linear series of values for discovering motifs or rules, for clustering or classifying sequences or for answering query by contents.

DTW finds the optimal alignment between two given time series. One of its strength is to cope with local distortions along the time dimension. DTW, however, is slow and costly to calculate as its time complexity is quadratic. Therefore, three families of approaches have been defined to accelerate DTW-based searches.

1.1 Lower-Bounding Functions

The first family of approaches includes cheap-to-compute functions lower bounding DTW [5, 16]. They accelerate the DTW process because they quickly prune sequences that could not possibly be a best match. Then, the full DTW is computed on the list of candidate sequences that remain. Lower bound functions have two key properties: (i) they incur no false dismissals as suggested in [3] and (ii) the tighter they are, the more they prune. At a very fine level, even LB_Keogh, known to be the tightest lower bound in the literature, can be quite far from the true value of the DTW [5]. There is therefore room for improvement, as a tighter bound would allow more pruning and faster searches. Yet, designing a tighter lower bound function is difficult as its computation is likely to become complex, thus less profitable.

The first contribution of this paper is the definition of approximate lower bounding functions built on top of the popular LB_Keogh and LB_PAA (exact) lower bounds. These approximate lower bounding functions have a user-tunable tightness: it can range from the one achieved by their exact counterpart, and in this case no false dismissals are observed, to much tighter bounds where false dismissals are possible as the DTW may get over estimated.

1.2 Indexing Time-Series

The second family of approaches includes schemes specifically designed to cope with very large databases of time series. [1, 5, 12, 15] all index time series for improved performance. Indexing shrinks the search space to some neighborhood defined around the query sequence, which, in turn, dramatically reduces the response time. Not surprisingly, lower bounding functions are at the core of such indexing schemes for even better performance. For example, [7] defines *i*SAX_MinDist that prunes sequences based on the range intervals determined for quantizing time series.

The second contribution of this paper is the definition of an approximate lower bounding function used in *i*SAX. It builds on *i*SAX_MinDist, its tightness can also be tuned by the users to offer a wide range of response time vs. accuracy trade-offs.

The performance of *i*SAX depends on the one hand on the lower bounding function used, and on the other, it depends on the way the tree used for indexing is built. *i*SAX and *i*SAX2.0 quantize time series in a tree of symbols. As discussed in [1], the quantification intervals are determined in order to be as balanced as possible assuming the data in time series is normally distributed. This is, in fact, rarely the case.

The third contribution of this paper is the use of a k-means-based quantization step for *i*SAX. Using k-means avoids to make any strong assumption on the distribution of data in time series. The resulting index better fits data. Some special care is taken to balance the tree of symbols, as this is key to performance.

1.3 Approximate Searches

The third family has approaches that have traded accuracy for response time [2, 12]. In this case, great performance gains are obtained at the cost of some potential false dismissals. What is central to these schemes is the metrics they use for assessing the similarity of time series, metrics that are no longer lower bounding the DTW. In addition to its exact *i*SAX_MinDist, *i*SAX defines such an approximate metrics. [12] shows the approximate version of *i*SAX can answer a query in less than a second, while exact *i*SAX requires about 10 minutes.

The last contribution of this paper is to show how approximate lower bounds can be used to boost the performance of the approximate version of *i*SAX.

1.4 Discussion

Experiments made with state-of-art datasets [6] show that these contributions significantly accelerate DTW. When considering small datasets, sequentially scanning the data collection is still appropriate. In this case, using approximate lower bounding functions is an order of magnitude faster than using exact lower bounds while indeed very rarely triggering false

dismissals. It is twice as fast as Iterative Deepening Dynamic Time Warping [2] while achieving comparable recall performance.

Performance gains are even more significant when considering large datasets for which indexing is needed. Approximating iSAX_MinDist allows to run iSAX two order of magnitude faster than its exact version while still returning one of the 50 actual nearest neighbours at rank 1 in 87% of the cases. Performance gains are about one order of magnitude compared to the approximate version of iSAX while achieving comparable retrieval performance, quality-wise.

This paper is structured as follows. Section 2 gives the necessary background on the DTW and on few popular techniques boosting its performance. Section 3 details how these techniques can be improved thanks to approximate lower bounding functions. Section 4 introduces iSAX$^+$, an indexing tree borrowing from iSAX that, however, relies on k-means clustering for determining quantization intervals. Note these two sections give pseudocodes. Section 5 presents the extensive experiments made using the datasets of [5] and [12] showing the algorithms we propose here outperform state-of-the art solutions. Section 6 concludes.

2   Accelerating DTW

This section reviews the background material needed for this paper. We first give a quick description of the DTW process. We then describe four very classical methods that have been designed to accelerate its computation. We then discuss IDDTW [2] which approximates DTW searches.

2.1   DTW

It has been demonstrated that similarity searches in time series is best when using DTW. DTW takes two time series, Q and C, of length m and l respectively, where:

$$Q = q_1 q_2 \ldots q_m; C = c_1 c_2 \ldots c_l \tag{1}$$

To evaluate the similarity between Q and C, a $m \times l$ matrix S is computed such that:

$$\forall (i,j) \in [1, m] \times [1, l], S_{i,j} = d(q_i, c_j) \tag{2}$$

where $d(q_i, c_j)$ is the distance between q_i and c_j. A warping path W is the set of K elements of S forming a path $W = w_1 \cdots w_K$ where $w_k = (i, j) \in [1, m] \times [1, l]$. W has to meet three conditions: it has to be continuous, monotonic and it has to meet boundary conditions (it goes from q_1, c_1 to q_m, c_l).

Dynamic programming techniques are used to efficiently find this path *via* a recurrence over a cumulative distance $\gamma_{i,j}$ defined as:

$$\forall (i,j) \in [1, m] \times [1, l], \gamma_{i,j} = \min \begin{cases} S_{i,j} + \gamma_{i-1,j-1} \\ S_{i,j} + \gamma_{i-1,j} \\ S_{i,j} + \gamma_{i,j-1} \end{cases}, \tag{3}$$

A direct implementation of DTW has a time and space complexity of $O(m \times l)$.

It has been shown that, for some applications, restricting the admissible paths around the diagonal could be beneficial in terms of both complexity and relevance of the resulting metrics. The most commonly used constraints are the Sakoe-Chiba band [11] and the Itakura parallelogram [4]. The former constrains the admissible paths to lay inside a band of fixed width around the diagonal path, as shown in Figure 1, while the latter uses a parallelogram that has the diagonal path as one of its diagonals.

2.2   Accelerator #1: LB_Keogh

LB_Keogh [5] is probably the best-known lower bound functions. LB_Keogh(Q, C) first rescales Q and C to the same length n, then builds the bounding envelope of Q by determining all its maximum and minimum values inside a Sakoe-Chiba band [11] (or an Itakura parallelogram [4]). It then sums the distances from every part of C not falling within the bounding envelope of Q to the *nearest* point from that envelope having the exact same time stamp. [5] shows, for any Q and C rescaled that LB_Keogh $\leq$ DTW. The complexity of LB_Keogh is linear in n.

2.3   Accelerator #2: LB_PAA

LB_PAA is another approach further reducing the cost of computing LB_Keogh. LB_PAA first requires to chop all sequences C stored in the database into N chunks containing a number of points that depends on the length of considered sequences. This creates sequences downsampled along their time line. Downsampled versions of C are then compared to a downsampled

(a) Sakoe-Chiba band   (b) Itakura parallelogram
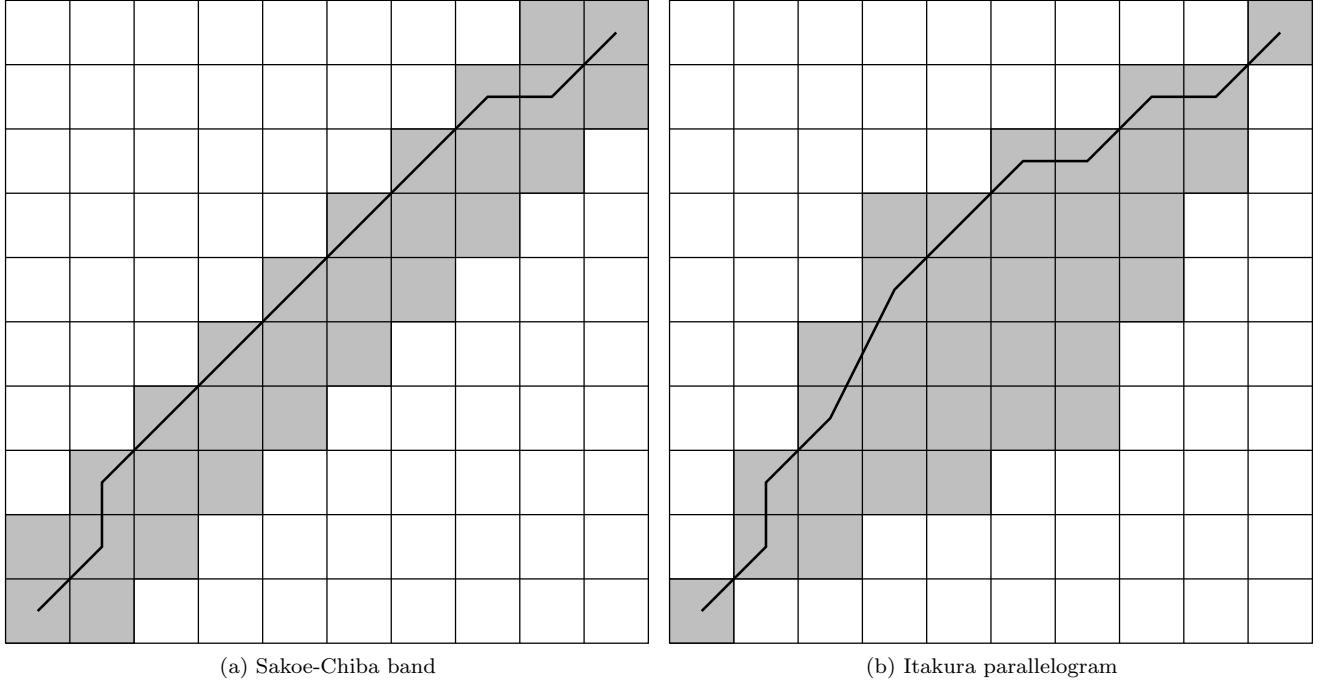
Figure 1: Example of constrained DTW paths. Admissible paths are restricted to the gray area.

version of the envelope of Q in a manner that is similar to the one for LB_Keogh. This gives the LB_PAA function, lower bounding LB_Keogh, and, therefore, DTW. Note the complexity of LB_PAA is equivalent to the complexity of computing LB_Keogh for time series of length N, which is smaller than n and thus computing LB_PAA is more efficient than computing LB_Keogh.

2.4   Accelerator #3: *i*SAX

Both LB_Keogh and LB_PAA must be computed for each sequence C from the database to find the one that is the most similar to Q. Researchers have thus worked on finding indexing techniques where only a small subset of relevant sequences from the database would have to be compared to Q, which would in turn prune the search space even more.

Since LB_PAA turns sequences into fixed-length descriptions lying in a N-dimensional space, it is possible to index them, typically in a R-Tree, as proposed in [5]. The index eventually contains minimum bounding rectangles (MBR) in which similar sequences are grouped. At query time, the algorithm finds the sequences from the database that are close to Q using the proximity of their respective MBR, quickly determined through the index.

Building on this, an even more efficient DTW indexing scheme called *i*SAX was proposed in [12]. *i*SAX also downsamples sequences, but it additionally quantizes them. *i*SAX thus gives a discrete representation to continuous sequences. It turns time sequences into a series of symbols that can easily be indexed in a tree. The quantization boundaries giving symbols for the values are determined according to a standard normal distribution. The size of the alphabet is dynamically adjusted during index construction to balance symbols. *i*SAX was later extended into *i*SAX2.0 [1], that achieves more efficient index construction thanks to bulk-loading and more balanced clusters *via* a smart symbol selection scheme to use when splitting a node from the tree.

In this case, the *i*SAX_MinDist metrics defined in [12] that is used to assess similarity between a query and a node of the tree is in the same spirit as the ones presented above, except that the range assigned to nodes are used to replace MBRs. The search process then consists in visiting each leaf of the tree in ascending order of their *i*SAX_MinDist values and the algorithm stops as soon as all remaining leaves in the tree have *i*SAX_MinDist values greater than the current best distance found.

2.5   Accelerator #4: Approximate *i*SAX

 [12] also presents a slight variant of the exact *i*SAX version that uses *i*SAX_MinDist. This variant returns approximate results as it does not scan all theoretically possible leaves but solely the one leaf with the lowest lower bound. That single leaf is entirely scanned, LB_Keogh is determined for each sequence, which possibly prunes some sequences from any further

analysis. As usual, full DTW are computed for the remaining sequences from that leaf to build the final result returned to the user. Returning data from that single leaf is of course very fast. Searching the index for time series that are similar to the query thus aborts as soon as the most probable leaf from the tree has been scanned. For the sake of clarity, we call this way of accelerating the searches *i*SAX_Approx.

2.6 IDDTW

Iterative Deepening Dynamic Time Warping (IDDTW) [2] aims, as previously introduced lower bounds for DTW, at computing DTW only for those candidate sequences in the database that are likely to be similar to the considered query. This likelihood is estimated by computing DTW between downsampled versions of the sequences. The principle is then to start comparing sequences at a very coarse resolution and keep refining the latter until the sequence can be discarded from the candidate list or, if not, until full resolution is reached. To test if a sequence can be discarded at a given resolution, a learned distribution of the estimation error at this resolution is used. Given this error distribution, if the probability that the computed estimation can lead to a lower DTW value than that of the best sequence so far is lower than a user-defined threshold, the sequence is discarded. If not, this process is repeated at a finer resolution.

This algorithm heavily relies on probabilistic estimation. Therefore, the false dismissal assumption is discarded for the sake of lower computational cost.

3 Improving Accelerators via Approximations

This section describes how the efficiency of the four DTW accelerators can be improved thanks to approximations. It first describes how the $\mathcal{A}$_Keogh and the $\mathcal{A}$_PAA approximate lower bound functions can be defined[1] from their exact counterparts LB_Keogh and LB_PAA. This improves the performance of the accelerators #1 and #2. Then, this section moves to defining $\mathcal{A}$_*i*SAX which improves the performance of the accelerator #3, indexing DTW *via* *i*SAX. It also defines $\mathcal{A}$_*i*SAX_Approx, built from *i*SAX_Approx which enhances the performance of Approximate *i*SAX. This section ends by presenting the pseudo-codes for two algorithms derived from that of [5, 12], one for a sequential-scan based search algorithm, the other for an index-based search.

3.1 $\mathcal{A}$_Keogh

The pruning power of LB_Keogh is directly linked to its tightness. The tighter it is, *i.e.*, the closer to the real value of the DTW the lower bounding values are, the better. Given a time series, computing its LB_Keogh value does not help in knowing whether it is tight, i.e., close or far from its actual DTW value. It is not the value of LB_Keogh that matters, but the relative difference between that value and the true value of DTW. It has been observed that LB_Keogh is often quite smaller than the true value of the DTW, especially with rapidly varying sequences. What is key is to have a way to distinguish the case where (i) the value of the lower bounding function is small because the DTW is also small, from the case where (ii) it is small because its calculation is poorly tight.

To reach that goal, we first describe the design of a cheap-to-compute upper bounding function that is very similar in spirit to LB_Keogh. This upper bound is used to get an indication on the tightness of the lower bounding function LB_Keogh. By repeatedly computing the lower and upper bounds on a set of training sequences, it is possible to learn and then model the overall tightness. We then introduce a mechanism to determine, given a tunable parameter that is closely linked to the tightness model, how many DTW values are likely to be over-estimated. Overall, this yields *approximate lower bounding functions*. Thanks to the model for over-estimations, it is possible to fine tune the pruning power of such functions: very aggressive functions that are likely to over-estimate DTW will have a very good pruning power but may in turn create many false dismissals, in contrast to milder functions.

3.1.1 UB_Keogh: an Envelope-based Upper Bound

[15] uses an upper bound to prune sequences that could not match a given query. In this paper, we are using an upper bound not for pruning, but to determine the interval (between the lower and upper bounds) within which the true value of the DTW is. If the interval is small, then it is likely the lower and upper bounds are tight.

One straightforward way to upper bound the DTW is to compute the Manhattan distance[2] between Q and C. A more elegant way is to rely on the envelopes around sequences that are already used for computing lower bounds. We experimentally observed that the intervals around the true values of the DTW are more stable when the bounds are defined according to the

---

[1]The $\mathcal{A}$_ prefix stand for approximate.

[2]Note that, in this paper, notations that come from the work of [11] are used, which leads to the Manhattan distance being an upper bound while, when using notations from [5], DTW is upper-bounded by Euclidean distance.
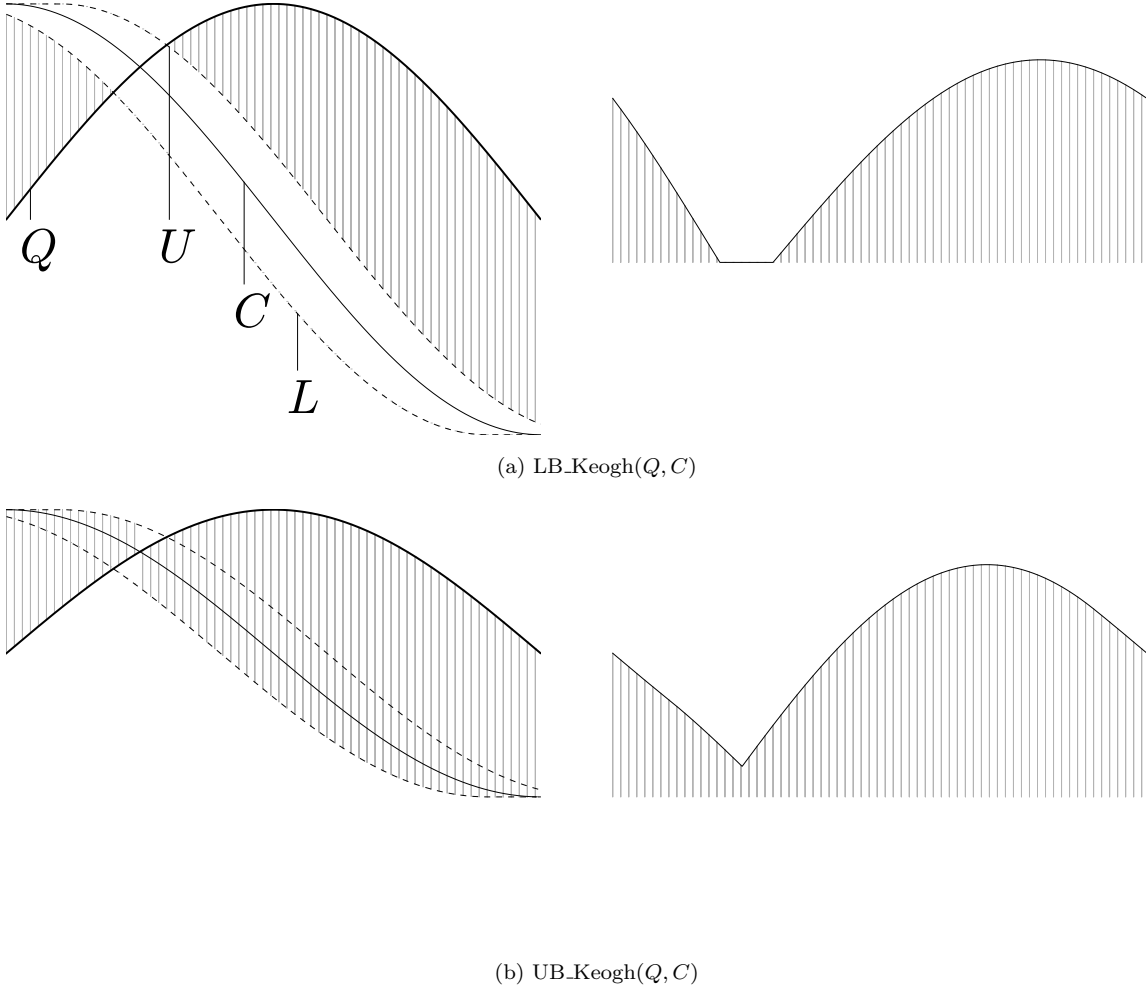
(a) LB_Keogh(Q, C)

(b) UB_Keogh(Q, C)

Figure 2: Example of LB_Keogh and UB_Keogh. U and L are defined as in [5]. The right hand side of each figure shows the overall area accounted for when computing each bound.

same underlying principle (envelopes). In addition, as it will be detailed later in this paper, relying on envelopes facilitates the computation of an upper bound on downsampled data as it is the case for LB_PAA.

Building on LB_Keogh, it is possible to define an envelope-based upper bound, called UB_Keogh. UB_Keogh is computed by summing the distances between every point in the query and the *furthest* point having the exact same time stamp on the envelope of each database sequence. The mathematical definition of UB_Keogh as well as the proof it truly upper bounds DTW are provided in the Appendix. By definition:

$$\text{LB\_Keogh} \quad \leq \text{DTW} \leq \quad \text{UB\_Keogh}. \tag{4}$$

The complexity of computing UB_Keogh is the same as the one of computing of LB_Keogh, *i.e.*, it is linear in n. Figure 2 gives a graphical illustration of LB_Keogh and UB_Keogh.

When processing Q and C, the difference between LB_Keogh and UB_Keogh gives a clear indication on the tightness of these bounds, a small difference suggesting that the bounds are tight.

3.1.2   Modeling Tightness

It is possible to learn from a training set what the values for LB_Keogh and UB_Keogh are, as well as the ones for the true DTW for each pair of sequences in the training set. A histogram of the distribution of these values, once normalized, can then be computed. Normalized values are defined as:

$$\text{s-DTW(Q,C)} = \frac{\text{DTW}(Q, C) - \text{LB\_Keogh(Q,C)}}{\text{UB\_Keogh(Q,C)} - \text{LB\_Keogh(Q,C)}}. \tag{5}$$

s-DTW is closely related to the tightness observed on the training set: a left-shifted curve for the probability density function of s-DTW suggests most LB_Keogh values are tight while most are loose when right-shifted. We validate this idea
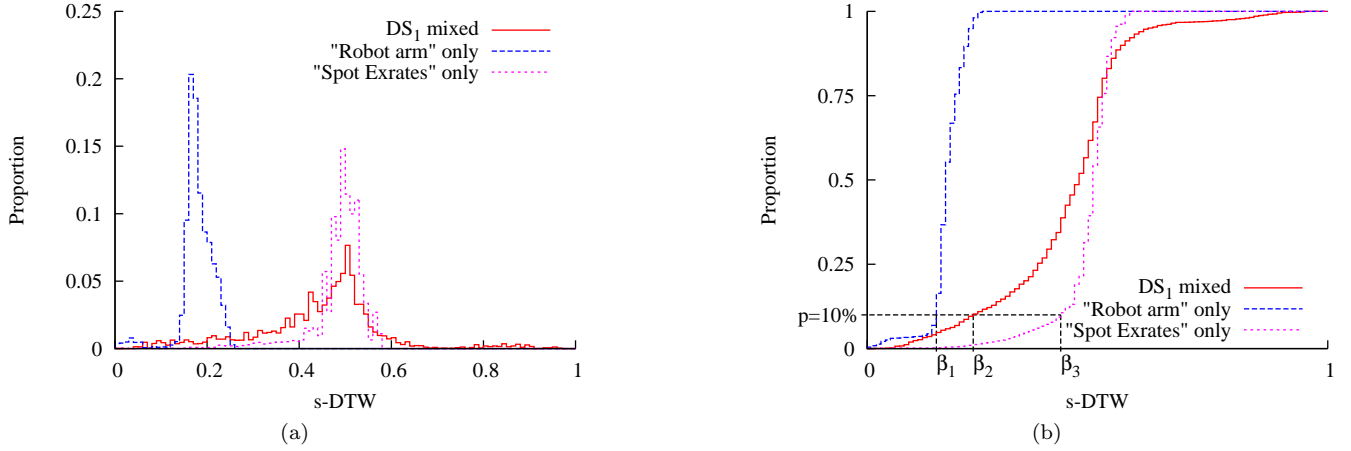
Figure 3: Experimental probability density function (a) and cumulative density function (b) for the whole merged dataset DS_1 and for its subdatasets "Robot Arm" (labeled #20 later) and "Spot Exrates" (labeled #24 later).

through experimentation. Since we believe that, at large scale, a database of sequences is highly diverse, we merged into a single database all the datasets listed in [5] (that will later be referred to as dataset DS_1).

Figure 3a shows, for DS_1, the distribution of s-DTW over all sequences. It also plots distributions observed for two sub-datasets. Few comments are in order. First, all plots are away from the x-axis zero value, meaning that, for every pair of sequence considered, LB_Keogh is strictly lower bounding DTW. This is basically the proof that there is room from improvement: it is possible to use a greater value than LB_Keogh (*i.e.* shifting decision value to the right) while still experiencing very few (or even no) DTW over-estimations (shown on the y-axis of Figure 3b). Second, there is a significant shift between the curves, which motivates the use of a learnt distribution rather than a single shift value.

The distribution of s-DTW values is an indicator of the tightness observed on the training set: we next show how it can be used as a model, to predict the tightness of sequences outside the training set.

3.1.3 Determining the Over-Estimation Rate

By computing the cumulative distribution of s-DTW, one can estimate the expected *distance over-estimation rate*. Assume one wants to speed up the search process at the price of potential false dismissals. That user decides to allow for $p = 10\%$ of over-estimated distances. Given the learned cumulative distribution, one can find the value of β on the x-axis that gives p on the y-axis (see the dashed lines on Figure 3b). β is then used to compute the approximate lower bound equal to LB_Keogh $+ \beta$(UB_Keogh $-$ LB_Keogh) that will subsequently be used to prune sequences. The resulting approximate lower bound is called $\mathcal{A}$_Keogh.

p might be meaningless for some users. Instead, they might prefer achieving a target tightness T. Finding the optimal value of p in order to achieve a target tightness T can be performed by numerically inverting the non-decreasing function that gives T as a function of p, for example by using dichotomy. If the user prefers to set a complexity constraint in terms of the expected amount of DTW computations, it is then possible to set the value for p by numerically inverting the relationship between this amount of computations and p. Finally, note that when setting $p = 0$, $\mathcal{A}$_Keogh=LB_Keogh.

3.2 $\mathcal{A}$_PAA

The previous section defined UB_Keogh and explained how to calculate $\mathcal{A}$_Keogh. It is possible to obtain UB_PAA and $\mathcal{A}$_PAA in a very similar manner. Few comments are in order, however. First, LB_PAA downsamples all sequences into N chunks. The envelope of Q is therefore defined according to its downsampled version. UB_PAA is thus also defined on this downsampled version. Mathematical definition of UB_PAA is in the Appendix. Second, LB_PAA $\leq$ DTW $\leq$ UB_PAA. Last, the formula for modeling tightness (see Eq. (5)) has to be slightly changed in a trivial manner to use LB_PAA and UB_PAA. Note also that when the rate of DTW over estimations p is set to 0, then $\mathcal{A}$_PAA=LB_PAA.

3.3 $\mathcal{A}$_Keogh and $\mathcal{A}$_PAA for Sequential Scan

LB_Keogh and LB_PAA are defined in [5]. They are both used together with a search algorithm doing a sequential scan of the entire collection. Either lower bounds are computed on every sequence C from the database to quickly get it compared to Q.

Algorithm 1 Sequential scan based searching algorithm.

Require: Q[1..n], DB[1..n_{db}][1..n], β
 (L, U) $\leftarrow$ get_envelope(Q)
 for $i = 1..n_{db}$ **do**
  lb $\leftarrow$ LB_Keogh(L, U, DB[i])
  ub $\leftarrow$ UB_Keogh(L, U, DB[i])
  $\mathcal{A}$_Keogh[i] $\leftarrow$ lb + β (ub - lb)
 end for

 argsorted_$\mathcal{A}$ $\leftarrow$ argsort($\mathcal{A}$_Keogh)

 best_so_far $\leftarrow$ ∞
 best_so_far_idx $\leftarrow$ -1
 for i in argsorted_$\mathcal{A}$ **do**
  if best_so_far $\leq$ $\mathcal{A}$_Keogh[i] **then**
   break
  end if
  if DTW(Q, DB[i]) < best_so_far **then**
   best_so_far $\leftarrow$ DTW(Q, DB[i])
   best_so_far_idx $\leftarrow$ i
  end if
 end for
 return (best_so_far, best_so_far_idx)

This section presents a slight variant of this sequential scan search process that uses either LB_Keogh, UB_Keogh and $\mathcal{A}$_Keogh or LB_PAA, UB_PAA and $\mathcal{A}$_PAA. The pseudo-code in this algorithm focuses on $\mathcal{A}$_Keogh. It is straightforward to adapt to $\mathcal{A}$_PAA.

Prerequisites n_{db} sequences are stored in a database DB. Each sequence has been rescaled to length n.

Off-Line Learning For each pair of sequences (C_i, C_j) from the learning set, DTW, LB_Keogh, UB_Keogh and s-DTW are computed. Then, the average cumulative distribution for s-DTW is calculated and a lookup table $\mathcal{C}$ keeping track of the quantiles of this distribution is built.

Query Time User Input The user inputs Q and p. Q is the query, it gets rescaled to length n. p is the average over-estimation rate. p is used to probe $\mathcal{C}$, resulting in the target quantile β.

On-Line Searching The sequential scan based searching algorithm is detailed in Algorithm 1. An important point here is that, when setting $\beta > 0$, the `break` instruction can be triggered before finding the exact nearest neighbor.

3.4 $\mathcal{A}$_iSAX and iSAX indexing

Indexing sequences is needed at large scale. The most popular time series indexing scheme is probably iSAX [12]. For efficiency, iSAX uses a lower bound function called iSAX_MinDist. Following the principles defined above, it is possible to create an approximate lower bound function $\mathcal{A}$_iSAX built on top of iSAX_MinDist.

 The definition of iSAX_MinDist is slightly more complex compared to LB_Keogh or LB_PAA as it relies on the intervals defined during the index tree construction by the quantization process. With iSAX it is possible that some intervals associated to nodes in the index tree have infinite bounds. In this case, computing an upper bound is problematic. To facilitate this computation, we had to slightly modify the tree building process by storing, for each node that has at least one infinite bound, the corresponding extremum value for the sequences that are stored in the node or one of its children nodes. Therefore, the upper bound will use the actual extremum value instead of the infinite bound in its formula. That upper bound is called iSAX_MaxDist and its mathematical definition is in the Appendix. Then, iSAX_MinDist and iSAX_MaxDist are used seamlessly in Eq. (5), eventually defining $\mathcal{A}$_iSAX.

 The resulting iSAX index based searching algorithm is detailed in Algorithm 2. Note this algorithm uses iSAX_MinDist, iSAX_MaxDist and $\mathcal{A}$_iSAX as well as the sequential scan algorithm presented above (and thus is relies on $\mathcal{A}$_Keogh).

Algorithm 2 Index based searching algorithm for iSAX.

Require: Q[1..n], $\beta_{\text{LB\_Keogh}}$, $\beta_{i\text{SAX\_MinDist}}$
  (L, U) ← get_envelope(Q)
  node_bsf ← isax_approximate_search(Q)
  (idx_bsf, dist_bsf) ← seq_scan(Q, node_bsf, $\beta_{\text{LB\_Keogh}}$)
  pq ← new_priority_queue ()
  pq.insert (0.0, root)

  while !pq.is_empty() **do**
    (dist_min, node_min) ← pq.get_min ()
    if dist_min $\geq$ dist_bsf **then**
      break
    end if
    if node_min is a leaf **then**
      (ix, dst) ← seq_scan(Q, node_min, $\beta_{\text{LB\_Keogh}}$)
      if dist_bsf $>$ dst **then**
        dist_bsf ← dst
        idx_bsf ← ix
      end if
    else
      break
      for all node in node_min.children **do**
        mind ← iSAX_MinDist(Q, node)
        maxd ← iSAX_MaxDist(Q, node)
        $\mathcal{A}\_i$SAX ← mind + $\beta_{i\text{SAX\_MinDist}}$ (maxd - mind)
        pq.insert($\mathcal{A}\_i$SAX, node)
      end for
    end if
  end while
  return (idx_bsf, dist_bsf)

Prerequisites n_{db} sequences are stored in a database DB. Each has been downsampled to length N.

Off-Line Indexing iSAX 2.0 off-line indexes all sequences as in [1].

Off-Line Learning   Same as in Section 3.3. Additionally, however, quantiles for $\mathcal{A}\_i$SAX and $\mathcal{A}$_Keogh must be learned as both lower bounding functions are used at query time.

User Input   The user inputs Q and p. The two quantiles $\beta_{\text{LB\_Keogh}}$ and $\beta_{i\text{SAX\_MinDist}}$ are obtained from p and $\mathcal{C}$.

On-Line Searching   The index based searching algorithm is detailed in Algorithm 2. It is built on iSAX (see [12]), with specifics due to the use of approximate lower bounds. Note that `isax_approximate_search`, called to initialize the process, returns the leaf node having a SAX signature that matches the one of the query[3] and seq_scan is Algorithm 1.

3.5 i**SAX_Approx indexing**

[12] describes a variant of iSAX that returns approximate results. We have described this variant in Section 2.5 and called it iSAX_Approx. iSAX_Approx determines the one best iSAX-leaf, computes LB_Keogh on all sequences from that leaf for pruning, and then computes full DTW on the remaining sequences. As it uses LB_Keogh, it is possible to patch this algorithm such that it uses $\mathcal{A}$_Keogh instead.

    It is therefore quite easy to turn iSAX_Approx into
$\mathcal{A}\_i$SAX_Approx. iSAX_Approx is already an approximate search scheme, due to its leaf picking strategy. $\mathcal{A}\_i$SAX_Approx somehow piles-up another layer of approximation as it uses $\mathcal{A}$_Keogh. Using $\mathcal{A}$_Keogh at the heart of iSAX_Approx typically

---

[3]There exists exactly one such node. `isax_approximate_search` is in fact the method defined in [12] and used in next section describing iSAX_Approx indexing.

divides by 7 the number of sequences that go through a full DTW process for comparable recall performances. The pseudo code for this $\mathcal{A}\_iSAX\_Approx$ strategy is simply made of the four first lines of Algorithm 2 as it directly returns (`idx_bsf`, `dist_bsf`) once the best iSAX node scanned by Algorithm 1.

4 iSAX$^+$: Quantizing with k-means

iSAX is a very succesful approach because it turns continuous time series into discrete sequences of symbols, for which efficient high-dimensional indexing schemes exist. The cornerstone of iSAX is therefore its quantization process which is clearly defined in [1]. It makes one strong assumption: the data in time series before quantization is assumed to be normally distributed. From that assumption, iSAX determines the quantification intervals to balance the probability with which symbols will appear in the quantized sequences. [1] shows it is key for performance to have (roughly) equiprobable symbols.

Unfortunately, the data in real-world time series is unlikely to follow a normal distribution. Note the biggest datasets used in [1] are synthetic, and have been created according to a normal law. As real-world time series rarely stick to a normal law, quantifying them as iSAX does is unlikely to give equiprobable symbols, which, in turn, hurts performance.

It is possible to relax this normal distribution assumption by using a variant of the k-means approach. k-means is a widely used unsupervised classification algorithm that takes as input a set of points $\mathbf{x}$ and a number k of clusters to build. It makes no assumption on the distribution of data. It tends to minimize intra-cluster variance defined as:

$$V = \sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \mathbf{c_i}\|^2 \tag{6}$$

where $\mathbf{c_i}$ is the centroid of cluster C_i.

Algorithms belonging to the k-means family have been used as unstructured quantizers in the context of image and video search [8, 9, 13, 14]. In this context, k-means has proved to nicely fit the real distribution of the data in space. This is the rationale for using k-means for quantizing time series. k-means, however, is also known to create Voronoi cells having a very uneven cardinality. The variant we use here does not only k-means but also balances the clusters it creates. We now detail the way k-means can be used for iSAX as well as the cluster balancing strategy.

4.1 k-means for Building the iSAX-Tree

The original iSAX tree forms a hierarchy of nodes. We therefore use k-means in a hierarchical way [8] to get a similarly shaped index tree. The root node of the original iSAX index tree has b^w children. Therefore, the root node when using the k-means approach has the same number of children nodes and results from determining $k = b^w$ centroids. Other levels below in the tree are made by running k-means with $k = 2$ as for the original iSAX which has also 2 child nodes per parent node.

As it is the case for the original iSAX approach, building a index tree with k-means requires to split a node into two childrens nodes once the parent node gets fully filled with sequences. This very traditional recursive node splitting process eventually creates leaf nodes containing at most th sequences. Not surprisingly, a node keeps track of its MBR and its associated centroid, that centroid being produced by k-means. Inserting a new sequence in the index tree is performed by going down the tree and, at each level, by determining the centroid that is the closest to the sequence to insert. If that sequence has to be inserted in a node entirely filled, then a split occurs. k-means is applied to the sequences in that node that are then moved to child nodes according to the centroids newly determined.

4.2   Balancing the k-means-based iSAX-Tree

It has been shown in [14] that a k-means computed over a large set of high dimensional features produces clusters having quite different cardinalities. This, in turn, increases the response time variance of the algorithms processing the data in clusters, and performance might be particularly bad when using highly populated clusters. Having balanced clusters reduces this variance and improves performance, overall. Note this is also why the original version of iSAX creates a balanced index tree.

[14] balances the clusters produced by a k-means as follows. It first projects all the data points as well as the centroids computed by the k-means into a higher dimensional space. Then, it iteratively enlarges the distances between the centroid of a cluster and the corresponding data points in that cluster in proportion to the cardinality of the cluster. This moves inward the frontiers of highly populated clusters, reduces their cardinality because it assigns some of the points from that cluster to other, less crowded, clusters. [14] can be ran until all clusters contain the same number of points, or it can be stopped earlier, after a given number of iterations or when the standard deviation computed over the cardinalities of clusters falls below a threshold.

The mathematical formulas used in [14] can be significantly simplified when the algorithm has to balance the cardinality of $k = 2$ clusters. In this case, they can be turned into a simple algebraic equation given in the Appendix.

Algorithm 3 Basic operations for iSAX$^+$ tree.

---

function BUILD_TREE(DB[$1..n_{\text{db}}$][$1..n$], th, b, w, α, r)
    root $\leftarrow$ new_node()
    root.insert (DB)
    while there exists a node v such that card(n) > th **do**
        if is_root (v) **then**
            split (v, α, r, b^w)
        else
            split (v, α, r, 2)
        end if
    end while
    return root
end function

---

function SPLIT(v, α, r, k)
    centroids $\leftarrow$ k-means(v.stored_data, k)
    (centroids, assign) $\leftarrow$ balance (centroids, v, k, α, r)
    for c in centroids **do**
        $v_c \leftarrow$ new_node (centroid = c)
        v_c.stored_data $\leftarrow$ assign[c]
        v_c.MBR $\leftarrow$ MBR (v_c.stored_data)
        v.children.add (v_c)
    end for
end function

---

function INSERT(v, S, α, r)
    if has_children (v) **then**
        v_0 = closest_child (v, S)
        insert (v_0, S)
    else
        if card(v) == th **then**
            split (v, α, r, 2)
            v = closest_child (v, S)
        end if
        v.stored_data.add (S)
    end if
end function

4.3 i**SAX**$^+$

This section presents the pseudocode for the complete iSAX$^+$ indexing scheme (Algorithm 3). It borrows a lot from iSAX. It uses k-means then runs [14] to create balanced clusters as described above.

    The `build_tree` function is in charge of splitting overfilled nodes upon insertion. It uses the th parameter defined for the original iSAX. At splitting time, the function `split` is invoked. It first quantizes the data using k-means. Then, it balances the children nodes using the mechanisms presented in [14] and briefly sketched above.

    Note the pseudo-code for querying the tree is not presented here as it is identical to Algorithm 2. The only (minor) difference lies in the `approximate_search` function that relies on the proximity of the centroids obtained by the k-means (and not on the set of symbols as for the original iSAX).

5   Experiments

This section presents the extensive performance evaluation showing the techniques proposed in this paper outperform state-of-the art solutions. Fhe first set of experiments compares LB_Keogh and $\mathcal{A}$_Keogh, as LB_Keogh is the best exact lower bound ever proposed. This comparison involved evaluating their respective tightness and their resulting effectiveness in pruning more sequences. We also evaluate the impact of relaxing the *no false dismissal* assumption in the case of $\mathcal{A}$_Keogh on the quality of the results returned to the user. Note this involves checking the ranking of candidate sequences that will go through a full DTW process as this ranking might differs from the one determined by LB_Keogh. Note LB_PAA and $\mathcal{A}$_PAA are also compared.
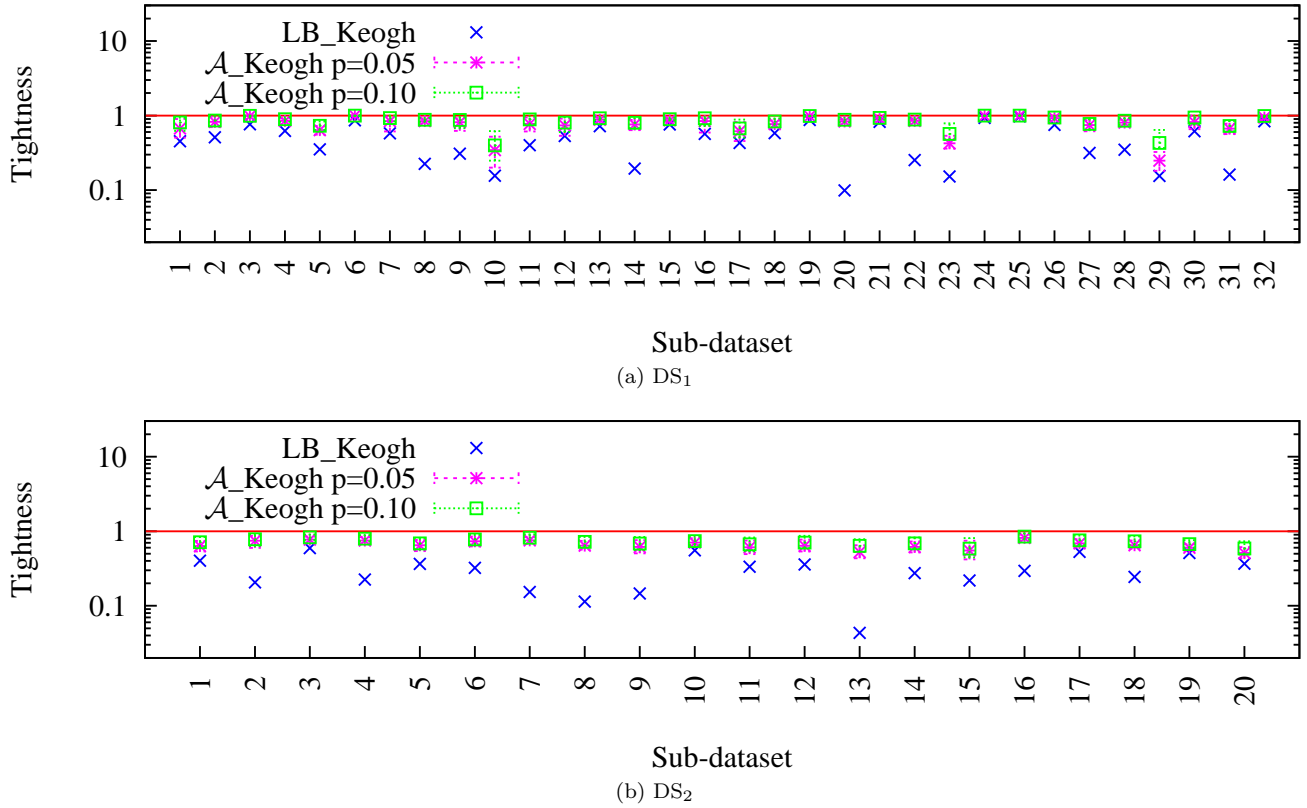
(a) DS_1

(b) DS_2

Figure 4: Median of tightness and its quartiles for $\mathcal{A}$_Keogh compared to the tightness of LB_Keogh.

The second set of experiments compares $\mathcal{A}$_Keogh with IDDTW. IDDTW also approximates DTW but it uses downsampling instead of an approximate lower bound.

The third set of experiments focuses on the impact of the above techniques on iSAX. It shows its performance is improved when using the approximate lower bounds $\mathcal{A}$_iSAX and $\mathcal{A}$_iSAX_Approx.

Finally, we give the performance of iSAX$^+$ that relies on a balanced k-means quantization step.

For the sake of reproducibility of experiments, all evaluations are conducted using publicly available [6] and widely used datasets [5, 12]. In all experiments, the baseline similarity measure is a DTW constrained by a Sakoe-Chiba band with a size set to 10% of the length of sequences, as suggested in [5].

The experiments use two datasets. The first one, referred to as DS_1, is precisely defined in [5]: it is made of 32 time series, 50 subsequences of length 256 being randomly extracted from each dataset to build the test set. In order to learn our lookup table $\mathcal{C}$, we built our own learning set per time series using 100 randomly extracted subsequences of length 256.

The second dataset, that we call DS_2, is available at [6] and was used in several publications. It is made of 20 sub-datasets, each split in advance into a training set and a test set. When running experiments involving DS_2 we naturally use these ready-made training sets to build $\mathcal{C}$. Note that these sub-datasets also provide classification labels for every sequence. When evaluating classification, we assign to the query the label of the most similar sequence returned by the search algorithm.

For datasets DS_1 and DS_2 and for both $\mathcal{A}$_Keogh and $\mathcal{A}$_PAA, we report the first-nearest-neighbor (1-NN) retrieval rate, that is, the ratio of queries for which their true nearest neighbor (according to the DTW) is ranked first.

Note that when necessary, estimator medians are reported together with their inter-quartile interval.

5.1   Compared Tightness of Lower Bounds

Figure 4 compares the tightness achieved by LB_Keogh and $\mathcal{A}$_Keogh on datasets DS_1 and DS_2. It can be seen that the observed tightness for LB_Keogh varies significantly between series: e.g., while it is tight for series #3 or #10 with DS_2, it is very loose for series #8 or #13. In contrast, $\mathcal{A}$_Keogh is overall much closer to 1 for both datasets. Note that when $p = 0.05$ or $p = 0.1$, the improved tightness is achieved with high precision, as shown by the inter-quartile intervals that do not cross the 100% tightness limit for any dataset. As predicted, setting higher values for p lead to higher achieved tightness. It is important to see that the results are more stable when using large learning datasets as it is the case of DS_2, where inter-quartile intervals are so small that they can hardly be seen on figures.

Dataset	Method	Median of ρ	Quartiles of ρ
DS$_1$	LB_Keogh	0.966	[0.874;0.996]
	$\mathcal{A}$_Keogh $p = 0.05$	0.967	[0.865;0.997]
	$\mathcal{A}$_Keogh $p = 0.1$	0.967	[0.861;0.997]
	LB_PAA	0.917	[0.501;0.991]
	$\mathcal{A}$_PAA $p = 0.05$	0.927	[0.653;0.993]
	$\mathcal{A}$_PAA $p = 0.1$	0.926	[0.653;0.993]
DS$_2$	LB_Keogh	0.914	[0.820;0.982]
	$\mathcal{A}$_Keogh $p = 0.05$	0.914	[0.819;0.982]
	$\mathcal{A}$_Keogh $p = 0.1$	0.915	[0.819;0.982]
	LB_PAA	0.614	[0.310;0.875]
	$\mathcal{A}$_PAA $p = 0.05$	0.661	[0.194;0.874]
	$\mathcal{A}$_PAA $p = 0.1$	0.663	[0.181;0.867]

Table 1: Spearman correlation coefficient. Here, the median of ρ values is computed for all subdatasets and the corresponding quartiles are reported.

5.2 Similarity of Candidate Lists

Algorithm 1 sorts all sequences from the database according to the value of $\mathcal{A}$_Keogh. That order depends on the value given to p. That order might possibly differ from the one that is determined when sequences are ordered according to their LB_Keogh value. Moreover, with $\mathcal{A}$_Keogh, when $p > 0$, the `break` instruction can be fired before identifying the best match, causing a false dismissal. It is therefore key to check if the order according to which sequences are sorted along their lower bound is somehow similar when considering DTW and LB_Keogh on the one hand or DTW and $\mathcal{A}$_Keogh on the other. If the latter order is quite similar, then it is likely that the best match will also be returned when using $\mathcal{A}$_Keogh.

The Spearman correlation coefficient ρ is a widely used metric for assessing the similarity of two ordered lists. The closer to 1 ρ is, the more similar the lists are. It is defined as:

$$\rho = 1 - \frac{6\sum_{i=1}^{n_{\mathrm{db}}}(r_i^1 - r_i^2)^2}{n_{\mathrm{db}}(n_{\mathrm{db}}^2 - 1)}, \tag{7}$$

where r_i^1 and r_i^2 are the ranks of sequence i in both lists.

Table 1 gives the median values for ρ computed on datasets DS$_1$ and DS$_2$. In this experiment the set of r_i^1 has been produced by ordering the n_{db} sequences along their DTW value; r_i^2 is either LB_Keogh, $\mathcal{A}$_Keogh, LB_PAA or $\mathcal{A}$_PAA. In all cases, the values for ρ do not vary much when p ranges from 0 to 0.1. Note also the large overlap between inter-quartile intervals suggests the observed differences for ρ between the three versions of $\mathcal{A}$_Keogh (resp. $\mathcal{A}$_PAA) are not significant.

Finally, it is interesting to notice that, in the case of $\mathcal{A}$_PAA, setting a higher value for p tends to increase ρ. In other words, taking more information from the upper bound into account tends to generate ordered candidate lists that are closer to the ones that DTW would get.

Another important point is that the observed ρ values are much smaller for LB_PAA (resp. $\mathcal{A}$_PAA) than they are for LB_Keogh (resp. $\mathcal{A}$_Keogh). This implies that smaller values should be used for p when trying to approximate LB_PAA as each DTW over-estimation is more likely to induce a false dismissal.

5.3 Retrieval performances

We used DS$_2$ to evaluate the performances of our proposed approximate lower bounds in terms of retrieval performances as well as computational cost. Retrieval is expressed by both the 1-NN retrieval rate and the correct classification rate. The computational cost is expressed as a ratio of the number of DTW computations induced by the approximate lower bound to the number of DTW computations induced by its corresponding exact lower bound.

Table 2 presents these results for $\mathcal{A}$_Keogh and $\mathcal{A}$_PAA. The first thing to notice is that, in all cases, the computational cost decreases considerably even for small values of p. Moreover, as previously suggested, small values of p should be used when considering LB_PAA to reduce the likelihood of false dismissals. The classification rate drops much slower than does the 1-NN retrieval rate, showing that when the true nearest neighbor is not ranked first, another sequence of the same class is, which is key to performance, quality-wise.

Method	p	Cost	1-NN	Classification
LB_Keogh	—	1.000	1.000	0.918
$\mathcal{A}$_Keogh	0.01	0.165	0.715	0.913
	0.05	0.062	0.410	0.879
	0.10	0.042	0.315	0.853
LB_PAA	—	1.000	1.000	0.918
$\mathcal{A}$_PAA	0.001	0.770	0.942	0.918
	0.01	0.370	0.603	0.906
	0.05	0.132	0.243	0.861

Table 2: Compared accuracy and computational cost of $\mathcal{A}$_Keogh and $\mathcal{A}$_PAA. In all cases, the cost of the exact lower bound is used as a reference.

p	Method	Cost	1-NN	10-NN	50-NN
—	iSAX_MinDist	1.0000	1.00	1.00	1.00
—	iSAX_Approx	0.0095	0.12	0.59	0.92
0.001	$\mathcal{A}$_iSAX	0.0722	0.39	0.84	0.99
0.010	$\mathcal{A}$_iSAX	0.0324	0.18	0.77	0.99
0.050	$\mathcal{A}$_iSAX	0.0034	0.03	0.26	0.77
0.001	$\mathcal{A}$_iSAX_Approx	0.0014	0.12	0.57	0.92
0.010	$\mathcal{A}$_iSAX_Approx	0.0003	0.05	0.49	0.90
0.050	$\mathcal{A}$_iSAX_Approx	0.0001	0.02	0.17	0.58

Table 3: Retrieval performance, together with the ratio of DTW computations (denoted as Cost), for iSAX.

5.4   Comparison with IDDTW

Both IDDTW and $\mathcal{A}$_Keogh approaches rely on modeling the expected error that is induced by approximation. We therefore compare both approaches in terms of accuracy and computational cost, using DS_1. Here, the computational cost is determined slightly differently from what was done previously, as IDDTW is an iterative process running DTW computations at different scales. We therefore define this cost as the total number of elementary distance computations that are needed for searching the database. For example, if a database of n_{db} sequences of length n was searched using DTW with no temporal constraint, the cost would be $n_{db} \times n^2$, since n_{db} DTW would be evaluated, each of which would use n^2 distance computations. This amount is then divided by the complexity of the reference method that is a full scan of the database using DTW constrained to a Sakoe-Chiba band.

Figure 5 shows that $\mathcal{A}$_Keogh offers great improvement over IDDTW, as comparable retrieval performances can be obtained at a cost divided by two. This can be explained by the fact that each IDDTW comparison between the query and a candidate sequence uses several estimations to infer how likely that candidate is the best match. Running such estimations multiple times (IDDTW) introduces more errors than what a single test would do ($\mathcal{A}$_Keogh). That phenomenon, a well-known problem in statistics, explains why relying on $\mathcal{A}$_Keogh better performs than using IDDTW.

5.5 iSAX

In this section, we compare the performance of $\mathcal{A}$_iSAX and $\mathcal{A}$_iSAX_Approx with the performance of the original version of iSAX2.0. We have built two indexing trees, one for learning and one for testing. The learning tree is required to determine the parameters needed by $\mathcal{A}$_iSAX and $\mathcal{A}$_iSAX_Approx. Then, the learned parameters are used to construct the other tree, subsequently used for searching. Each tree stores 100,000 sequences that are random walks of length 256. Each node of the tree can store up to 1,000 sequences. The trees are built using the iSAX2.0 algorithm to ensure better balance in the leaves cardinality. Two sets of 1,000 random walk sequences are used as queries: one for learning and the other one for testing.

A groundtruth is computed in terms of DTW, as for previous experiments. Together with the 1-NN retrieval rate, we also report 10-NN (resp. 50-NN) retrieval rate that is the ratio of returned nearest neighbors that are among the true 10 (resp. 50) nearest neighbors of the query point. iSAX_MinDist is used as a reference in terms of cost.

We measured average query times of around 5 seconds for iSAX_MinDist while approximate methods could perform several orders of magnitude faster: querying the index took on average 50 milliseconds for iSAX_Approx, 150 milliseconds for $\mathcal{A}$_iSAX with $p = 0.01$ and 5 milliseconds for $\mathcal{A}$_iSAX_Approx with $p = 0.01$. Note that these timings are coherent with the cost expressed here (that is related to the number of DTW computations), which is why we will keep presenting this cost in the following tables.
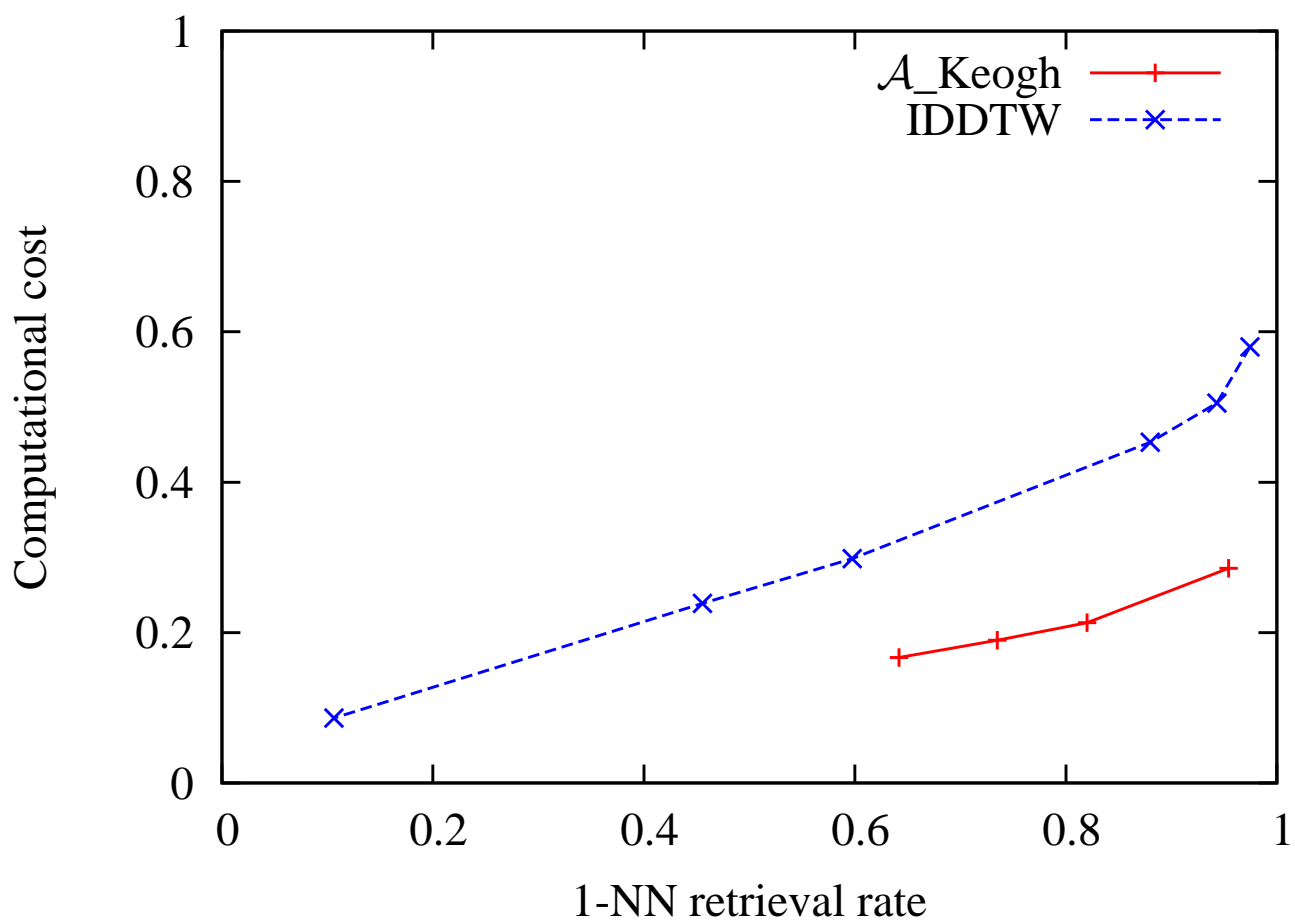
Figure 5: Comparison of IDDTW and $\mathcal{A}$_Keogh in terms of cost versus accuracy. The cost is defined as the number of elementary operations involved in the search process and accuracy is expressed in terms of 1-NN retrieval rate.

Tree	Imbalance factor γ	
	Test tree	Learning tree
iSAX 2.0	1.926	1.927
iSAX$^+$	1.11	1.10

Table 4: Compared balance of iSAX and iSAX$^+$ trees.

p	Search type	Cost	1-NN	10-NN	50-NN
—	iSAX_MinDist	1.000	1.00	1.00	1.00
—	iSAX$^+$_MinDist	1.7997	1.00	1.00	1.00
—	iSAX$^+$_Approx	0.0141	0.12	0.47	0.81
0.001	$\mathcal{A}\_i$SAX$^+$	0.6756	0.89	1.00	1.00
0.010	$\mathcal{A}\_i$SAX$^+$	0.0400	0.46	0.99	1.00
0.050	$\mathcal{A}\_i$SAX$^+$	0.0004	0.05	0.37	0.88
0.001	$\mathcal{A}\_i$SAX$^+$_Approx	0.0022	0.11	0.47	0.80
0.010	$\mathcal{A}\_i$SAX$^+$_Approx	0.0004	0.11	0.34	0.77
0.050	$\mathcal{A}\_i$SAX$^+$_Approx	0.0001	0.02	0.15	0.47

Table 5: Compared performances for approximate search using iSAX$^+$ and iSAX2.0. Sequences used here are drawn from normal distribution.

p	Search type	Cost	1-NN	10-NN	50-NN
—	iSAX_MinDist	1.000	1.00	1.00	1.00
—	iSAX_Approx	0.0129	0.20	0.81	0.95
0.001	$\mathcal{A}\_i$SAX	0.2363	0.68	0.96	1.00
0.001	$\mathcal{A}\_i$SAX$^+$	0.2914	0.95	1.00	1.00
0.010	$\mathcal{A}\_i$SAX$^+$	0.0125	0.24	1.00	1.00

Table 6: Compared performances for approximate search using iSAX$^+$ and iSAX2.0: the case of non-normal features

Results presented in Table 3 show that $\mathcal{A}\_i$SAX outperforms iSAX_MinDist and that $\mathcal{A}\_i$SAX_Approx outperforms iSAX_Approx. For example, when $p = 0.001$, iSAX_Approx runs more than 6 times more DTW computations than what $\mathcal{A}\_i$SAX_Approx does while achieving similar retrieval rates (this refers to raws #2 and #6 of the table). It is interesting to observe the retrieval 10-NN (resp. 50-NN) rates given by columns 5 and 6. The rates are excellent while the computation cost is significantly reduced.

5.6 iSAX$^+$

We compare here the performance of iSAX2.0 and iSAX$^+$ that build their index tree using two different approaches (iSAX$^+$ uses k-means). We first ran a set of experiments using the very same data sets as the ones presented above, i.e., that are made with normally distributed random walks. It is crucial to note sequences built that are particularly adapted to iSAX2.0. As we did previously, we determined the parameters for iSAX$^+$ using a first set of sequences, parameters than applied for running the experiments given below.

Both iSAX2.0 and iSAX$^+$ try as much as possible to determine their quantification intervals to get roughly equiprobable symbols. They use different mechanisms, however. To check the effectiveness of these mechanisms, we measured the imbalance of the final intervals determined on the one hand by iSAX2.0 and on the other after having ran the cluster balancing phase that follows the completion of the k-means in the case of iSAX$^+$.

Table 4 shows the ratio between the less and the more probable symbols from the trees build by iSAX2.0 and iSAX$^+$ when using the learning or the test tree. This table shows the iSAX$^+$ balancing phase achieves a much better balance of symbols probabilities than what iSAX2.0 does. This is a very nice result.

We now turn to the retrieval results when using iSAX2.0 or iSAX$^+$ for searching. These results are given in Table 5. iSAX2.0 and iSAX$^+$ show comparable retrieval performance when $p = 0.001$–slightly lower in the case of 1-NN and identical otherwise. The retrieval cost is however much lower.

One claim made in this paper is that iSAX$^+$ better performs than iSAX2.0 when non normally distributed sequences are used. We therefore ran experiments involving random walks using a Beta distribution with parameters $\alpha = \beta = 0.5$. There are 100,000 such walks in the database, each has a length of 256, and 1,000 other such walks have been created for querying the system.
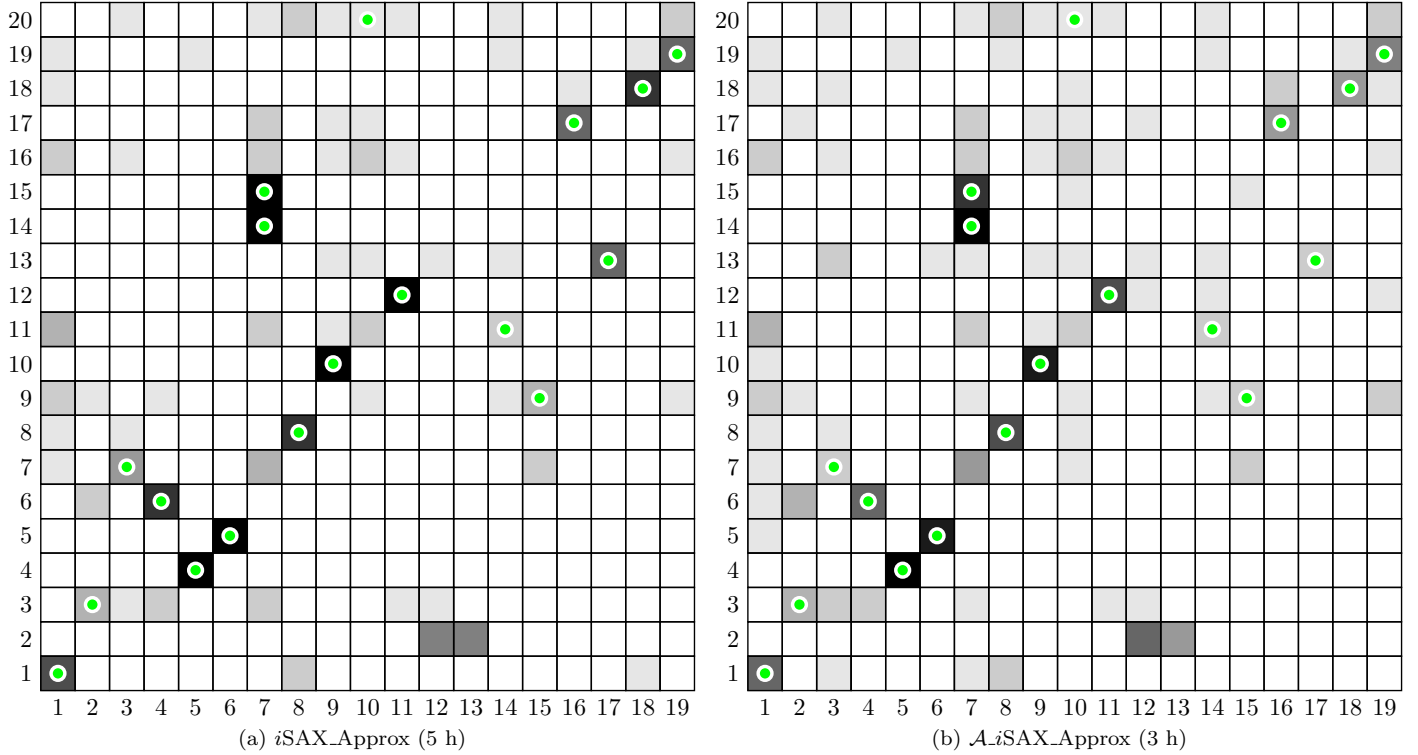
Figure 6: Correspondence matrices between human and Rhesus macaque genomes and corresponding query times. Y-axis corresponds to human chromosomes, while X-axis represents Rhesus macaque ones. Green dots correspond to matches reported in [10]. In (b), p parameter is set to 10%.

Table 6 presents the retrieval performances and the computing costs for $iSAX^+$ and $iSAX2.0$ when using these sequences. $\mathcal{A}\_iSAX^+$ is much faster than $iSAX\_MinDist$ and $iSAX\_Approx$ while achieving better recall compared to the latter.

5.7   A case study: indexing DNA datasets

As shown in [1], indexing full genomes using time series representations enable to link chromosomes between species that share common ancestors. For example, Rhesus macaques and humans have a common ancestor that lived 25 million years ago, which means their genomes share common attributes. Yet, the mapping between their chromosomes is not straightforward. One possible way to find such a mapping is to turn DNA sequences into time series and use related indexing techniques.

To do so, we use the method proposed in [1], that is each DNA symbol is changed into a numerical value equal to the one of the previous element in the sequence plus a value that depends on the symbol. Obtained sequences are then downsampled by a factor of 25 in order to reduce noise. We finally use a sliding window of length 400 and step 10 so as to extract subsequences. We use the $iSAX2.0$ algorithm with parameters $w = 10$, $b = 2$, $th = 1,000$ to index the whole Rhesus macaque genome (that is made of 10,194,500 such subsequences). The obtained index is queried using 300 sequences randomly picked from each human chromosome and their transposed versions. As in [1], we only consider longest chromosomes for the mapping and, for each of these chromosomes, we keep track of the 10 nearest neighbours among all retrieved neighbours. We then present correspondence matrices in Figure 6 by coloring each cell according to the amount of these nearest neighbours for the considered pair: the darker the cell, the more neighbours.

Query times reached by $\mathcal{A}\_iSAX\_Approx$ are significantly lower than the ones of $iSAX\_Approx$, while obtained correspondence matrices are visually similar.

6   Conclusion

Reducing the cost of computing the Dynamic Time Warping similarity measure between sequences is crucial to many applications. For some applications, indexing is not appropriate. Their performance, however, can be improved by using approximate lower bounding distance functions as described in this paper.

The accuracy of these functions can range from rough results with possible false dismissals returned extremely fast to exact results returned more slowly–the users can set this accuracy vs. speed tradeoff. For other applications, indexing is mandatory, for example to cope with scale. In this case, these approximate lower bounding distance functions can also improve the performance of retrievals, as demonstrated here.

Furthermore, this paper also shows that relying on a balanced k-means quantification step significantly improves the behavior of the very popular iSAX indexing scheme.

References

[1] A. Camerra, T. Palpanas, J. Shieh, and E. J. Keogh. iSAX 2.0: Indexing and mining one billion time series. In *Proceedings of the IEEE International Conference on Data Mining*, 2010.

[2] S. Chu, E. Keogh, D. Hart, M. Pazzani, et al. Iterative deepening dynamic time warping for time series. In *Proceedings of the SIAM International Conference on Data Mining*. Citeseer, 2002.

[3] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proceedings of the ACM SIGMOD Conference on Management Of Data*, 1994.

[4] F. Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 23(1):67–72, 1975.

[5] E. Keogh and C. Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and Information Systems*, 7(3):358–386, 2005.

[6] E. Keogh, X. Xi, L. Wei, and C. A. Ratanamahatana. The ucr time series classification/clustering homepage: www.cs.ucr.edu/~eamonn/time_series_data/, 2006.

[7] J. Lin, E. Keogh, L. Wei, and S. Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 15(2):107–144, 2007.

[8] D. Nistér and H. Stewénius. Scalable recognition with a vocabulary tree. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2161–2168, 2006.

[9] L. Paulevé, H. Jégou, and L. Amsaleg. Locality sensitive hashing: A comparison of hash function types and querying mechanisms. *Pattern Recognition Letters*, 31(11):1348 – 1358, 2010.

[10] J. Rogers, R. Garcia, W. Shelledy, J. Kaplan, A. Arya, Z. Johnson, M. Bergstrom, L. Novakowski, P. Nair, A. Vinson, et al. An initial genetic linkage map of the rhesus macaque (macaca mulatta) genome using human microsatellite loci. *Genomics*, 87(1):30–38, 2006.

[11] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26:43–49, 1978.

[12] J. Shieh and E. Keogh. iSAX: Indexing and mining terabyte sized time series. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2008.

[13] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1470–1477, 2003.

[14] R. Tavenard, H. Jégou, and L. Amsaleg. Balancing clusters to reduce response time variability in large scale image search. In *Proceedings of the IEEE Workshop on Content-Based Multimedia Indexing*, 2011.

[15] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. J. Keogh. Indexing multi-dimensional time-series with support for multiple distance measures. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 216–225, 2003.

[16] B. Yi, H. V. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *Proceedings of the IEEE International Conference on Data Engineering*, 1998.

A   Proof and mathematical definitions for upper bounds

We prove here that UB_Keogh is upper bounding DTW when the latter is restricted to a Sakoe-Chiba band. We also introduce upper bounds related to LB_PAA and *i*SAX_MinDist for which we omit the proofs as they follow the exact same principles.

Definition 1 *Let UB_Keogh be:*

$UB\_Keogh(Q,C)$

$$= \sum_{i=1}^{n} \begin{cases} (c_i - L_i) & if\ c_i > U_i \\ (U_i - c_i) & if\ c_i < L_i \\ \max(U_i - c_i, c_i - L_i) & otherwise \end{cases} \quad (8)$$

Lemma 1 *For any two sequences Q and C of length n, the following inequality stands:*

$$L_1(Q,C) \geq DTW(Q,C)$$

where the considered DTW is constrained to a Sakoe-Chiba band of width r.

Let Q and C be two sequences of length n. Manhattan distance L_1 corresponds to the alignment that follows the diagonal path. Hence, this distance is associated to one of the possible paths considered by the Dynamic Time Warping algorithm and is therefore greater than the cost of the minimal path, that is the value returned by DTW, which concludes the proof for lemma 1.

Proposition 1 *For any two sequences Q and C of length n, the following inequality stands:*

$$UB\_Keogh(Q,C) \geq DTW(Q,C)$$

where the considered DTW is constrained to a Sakoe-Chiba band of width r.

It is important to notice that each term in the sum that occurs in the definition of UB_Keogh is related to exactly one term in the computation of the L_1 distance. The only difference is that for UB_Keogh, the i-th term corresponds to the distance between the i-th point in the candidate sequence and its furthest corresponding point in the enveloppe of the query, while for L_1 the same term is equal to the distance between the i-th point in the candidate sequence and one of its possible corresponding points in the enveloppe of the query. The latter distance is then, by definition, smaller than the former and the following inequality is then straightforward, coming from lemma 1:

$$UB\_Keogh(Q,C) \geq L_1(Q,C) \geq DTW(Q,C). \quad (9)$$

Definition 2 *Let us define UB_PAA as:*

$UB\_PAA(Q,C)$

$$= \frac{n}{N} \cdot \left(\sum_{i=1}^{N} \max(\hat{U}_i - \bar{c}_i, \bar{c}_i - \hat{L}_i) \right)$$

$$+ \frac{n}{N} \cdot (\max(C) - \min(C)). \quad (10)$$

Lemma 2 *For any two sequences Q and C of length n, the following inequality stands:*

$$UB\_PAA(Q,C) \geq UB\_Keogh(Q,C).$$

Proposition 2 *For any two sequences Q and C of length n, the following inequality stands:*

$$UB\_PAA(Q,C) \geq DTW(Q,C)$$

where the considered DTW is constrained to a Sakoe-Chiba band of width r.

Definition 3 *Let us define iSAX_MaxDist as:*

$iSAX\_MaxDist(Q, R)$

$$= \sqrt{\frac{n}{N} \sum_{i=1}^{N} \begin{cases} (\bar{q}_i - B_i)^2 & si \ \bar{q}_i > H_i \\ (H_i - \bar{q}_i)^2 & si \ \bar{q}_i < B_i \\ \max(H_i - \bar{q}_i, \bar{q}_i - B_i)^2 & otherwise \end{cases}} \quad (11)$$

B   Balancing k-means

When $k = 2$, balancing k-means does not require any iterative process as proposed in [14]. It is possible to derive elevation h that the most populated clusters' centroid will get in order for both clusters to finally get equal populations without resorting to any iterative process.

Let us assume, without loss of generality, that k-means produced two centroids $\mathbf{C_1}$ and $\mathbf{C_2}$ and that cluster C_1 is more populated than C_2. Using notations introduced in Figure 7, intersection between the line $(\mathbf{C_1}, \mathbf{C_2})$ and the boundary between classes C_1 and C_2 is then $\mathbf{C_0}$, middle of the line segment $[\mathbf{C_1}, \mathbf{C_2}]$. We aim at evaluating elevation h such that this point moves to $\mathbf{C_0'}$ that is the median of projected data points. It is straightforward that if one builds a new boundary that is parallel to the original one and passes through $\mathbf{C_0'}$, both clusters will be equally populated. After solving the related system of equations, one gets:

$$h = \sqrt{2(x_2 - x_1)\left(\frac{x_1 + x_2}{2} - x_0'\right)}, \quad (12)$$

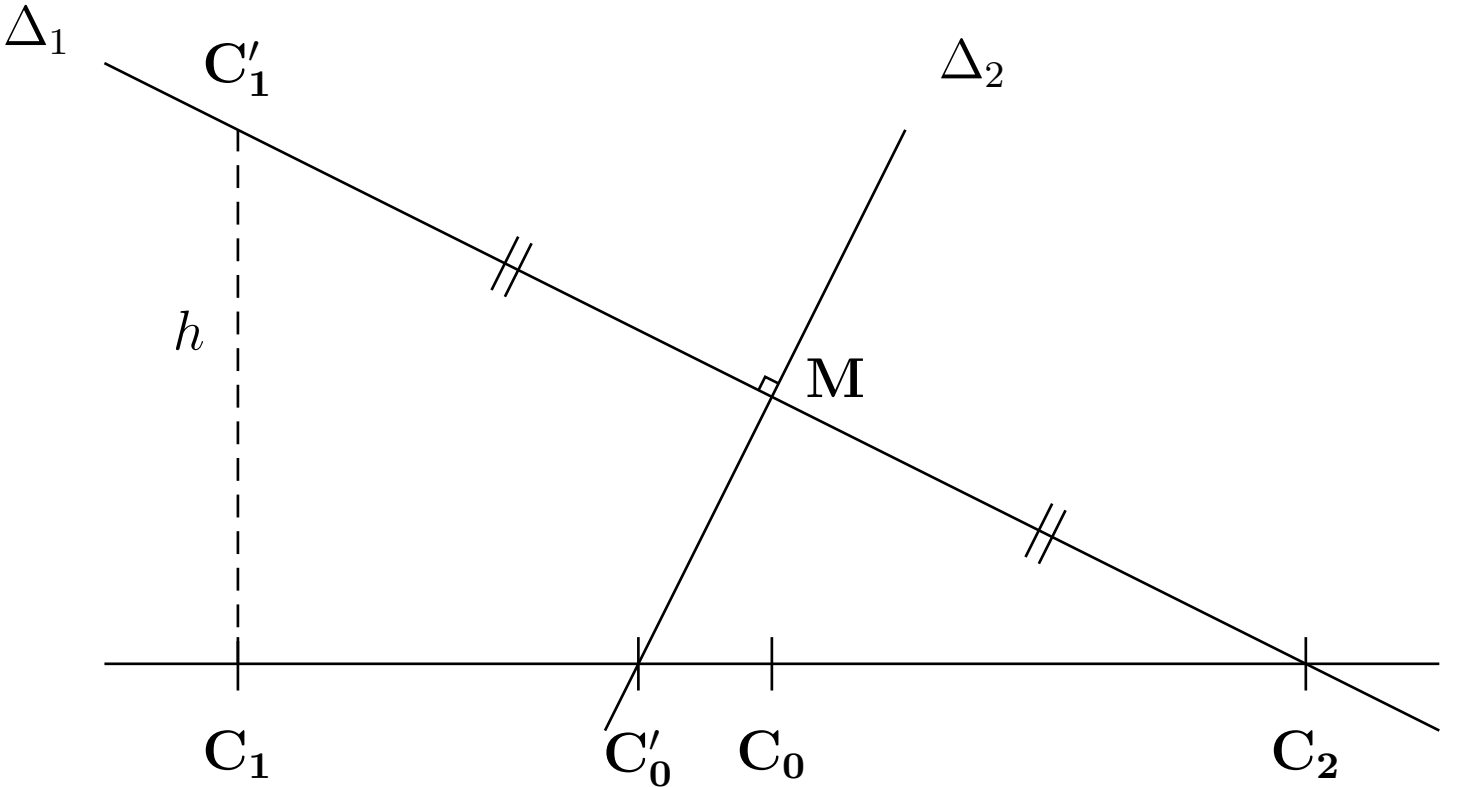where x_1 and x_2 are known from the k-means and x_0' is the median of projected data points.

Figure 7: Balancing k-means for the $k = 2$ case.