Scalable kNN Search on Vertically Stored Time Series*

Shrikant Kashyap
School of Computing
National University of Singapore
shrikant@nus.edu.sg

Panagiotis Karras
Rutgers Business School
Rutgers University
karras@rutgers.edu

ABSTRACT

Nearest-neighbor search over time series has received vast research attention as a basic data mining task. Still, none of the hitherto proposed methods scales well with increasing time-series length. This is due to the fact that all methods provide an *one-off* pruning capacity only. In particular, traditional methods utilize an index to search in a reduced-dimensionality *feature space*; however, for high time-series length, search with such an index yields many *false hits* that need to be eliminated by accessing the full records. An attempt to reduce false hits by indexing more features exacerbates the curse of dimensionality, and vice versa. A recently proposed alternative, *i*SAX, uses symbolic approximate representations accessed by a simple file-system directory as an index. Still, *i*SAX also encounters false hits, which are again eliminated by accessing records in full: once a false hit is generated by the index, there is no second chance to prune it; thus, the pruning capacity *i*SAX provides is also one-off. This paper proposes an alternative approach to time series kNN search, following a nontraditional pruning style. Instead of navigating through candidate records via an index, we access their features, obtained by a multi-resolution transform, in a *stepwise* sequential-scan manner, one level of resolution at a time, over a *vertical* representation. Most candidates are *progressively* eliminated after a few of their terms are accessed, using pre-computed information and an unprecedentedly *tight* double-bounding scheme, involving not only lower, but also *upper* distance bounds. Our experimental study with large, high-length time-series data confirms the advantage of our approach over both the current state-of-the-art method, *i*SAX, and classical index-based methods.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database applications—*Data mining*; H.2.4 [**Database Management**]: Systems—*Query processing*; H.3 [**Information Storage and Retrieval**]: Miscellaneous

General Terms

Algorithms, Experimentation, Theory, Performance

*Work supported by Singapore's MOE AcRF grant T1 251RES0807.

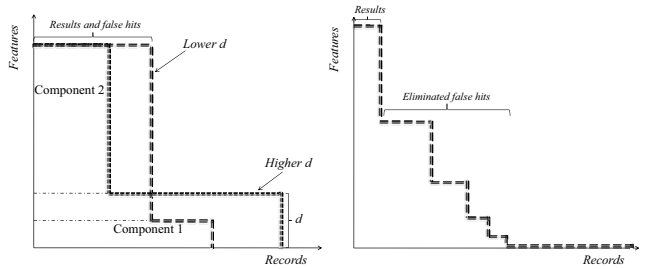
KDD'11, August 21–24, 2011, San Diego, California, USA.

1. INTRODUCTION

Similarity search among time series aims to retrieve records most similar to a query record. It attracts consistent research attention as a basic data mining task [21] with specific-domain applications ranging from finance [51] to motion tracking [15].

A time series can be represented as a point p in a D-dimensional space. A k-nearest-neighbor (kNN) query asks for the k records in a data set most similar to a query point q, according to a distance metric $DST(q, p)$. The conventional approach to this problem divides the search process in two components: First, a *filtering step* retrieves candidates by navigating a reduced-dimensionality *feature space* with an index; distance computations and object retrieval in this space, enabled by a *dimensionality reduction technique*, are less costly than in the full dimensionality, while the computed distance metric, dst, lower-bounds the actual distance, i.e. $dst(q, p) \leq DST(q, p)$; still, this step generates *false hits*, i.e. candidates not in the kNN result. A second, *refinement step* accesses the *full records* and calculates exact DST-distances to eliminate them and obtain the final results. The associated search algorithm, arrived at by a sequence of efforts in [21, 42, 35, 26, 44], alternates among these two steps, and is hence dubbed *multi-step*; in this work, we use the name *two-component framework*.

Unfortunately, it is hard to contain the number of false hits generated by the filtering step in the two-component framework; this *curse of dimensionality* is exacerbated with increasing time-series length (i.e., dimensions), as the feature-space representation becomes then less accurate, hence distance calculations with it degrade. If we attempt to alleviate this problem by increasing the feature-space dimensionality, then we render index-based search more expensive, eventually deteriorating to worse than a sequential scan [25]; if we attempt to render the search more efficient by indexing less features, the false-hit problem strikes back. Thus, a *tradeoff* emerges. The best feature-space dimensionality d strongly depends on the data set at hand and is hard to determine a priori.



(a) Classical tradeoff (b) Stepwise approach

Figure 1: Accessed data in two approaches

Figure 1(a) visualizes the said tradeoff. The area under each of the dotted lines represents the amount of data accessed, indicating

the number of records along the x axis and the number of read features of these records along the y axis. One line shows the case for a lower feature-space dimensionality d, while the other for a higher d. The two components (filtering and refinement) are denoted by the two rectangles below the dotted lines; with lower d, *fewer* records are accessed by the index (Component 1), but *more* false hits are read in full (Component 2); a higher d reduces false hits at the cost of increasing index-accessed d-dimensional records.

An award-winning work [49, 9] observed that traditional index-based methods are outperformed on average by a sequential scan if the number of dimensions exceeds around 10. In response, [49] proposed the *VA*-file, a vector approximation scheme that eschews indexing. Its *filtering step* prunes candidates via a sequential scan of bit-string representations. Still, this filtering fares poorly with skewed data [43]; the *VA*-file performs worse than a sequential scan even with uniform data in a main memory environment [18].

Recent research [46] has suggested *i*SAX, a method that consists at once a simplification and an enhancement over this classical framework; the enhancement consists of using *symbolic* approximate representations; the simplification of using a directory provided by the native file system as an index. Still, as classical approaches and the *VA*-file do, *i*SAX also uses a single approximate representation for each record and provides no intermediate level between this representation and the full record.

We argue that the main drawback of all these methods is the *one-off* filtering step they employ: once a false-hit passes their filtering process, however elaborate that process is, it has to be read in full along with an expensive distance computation. No other chance is given to prune it; filtering is an one-off affair. Even though the purpose of the index is to provide robust filtering, its pruning capacity is constrained by the accuracy of the approximate representation it employs. We contend that, to overcome this problem and enable length-scalable kNN search, one should provide ample, consecutive opportunities for candidate pruning at *multiple* levels of resolution; in other words, provide not a single filtering step, but many. Then candidates would be eliminated gradually, after passing one or more of these multiple filtering steps. To avoid confusion, we emphasize that the *multi-step* character of the algorithm in [44] involves the back-and-forth operation between false-hit elimination and further candidate retrieval; insofar as distance calculations are concerned, this classical algorithm carries out *two* steps.

In this paper, we propose a nontraditional, *genuinely multi-step* framework for time-series kNN search; our approach eschews an index and scans approximate representations instead, like the *VA*-file does; still, unlike the *VA*-file and other methods, it uses a vertical storage scheme and involves multiple filtering steps. Distance computations are carried out in a *stepwise* fashion, involving progressively more feature terms but less records; false hits are thereby gradually disqualified. Our methodology needs to employ a multi-resolution transform. The question of which transform to choose is *orthogonal* to it. For ease of exposition, we present our method using the Discrete Haar Wavelet Transform (DHWT). Figure 1(b) depicts data access with our method. The area below the dotted line again denotes accessed data. False hits are eliminated in a step-by-step manner, while gradually reading more features of remaining candidates, until we arrive at the kNN result. We analyze our chief contributions with respect to previous work as follows:

First, a vertical **storage scheme** that allows features to be read not record-by-record, but by level of resolution. A similar scheme has been proposed for image histograms in [19], storing the feature coefficients of the same dimension, for all vectors in the repository, in a separate table. This representation allows for the distance between a query point and all data vectors to be accumulated incrementally, by scanning these projections one-by-one, and computing lower and upper bounds on the employed similarity metric. This fundamental intuition is common to our work and [19]. To the best of our knowledge, such a storage scheme has not been applied in the context of disk-based kNN search among time series before.

Second, a robust **double-bounding scheme** tailored for time-series, which allows for the maximum benefits to be reaped from our storage scheme. A similar, but looser, scheme was used in [52] to reduce the communication cost among server and clients in a distributed kNN-search setting; thus, in [52] the objective is not to minimize local I/O at a client, but data transmission to the server. Our scheme goes beyond the one in [52] by leveraging sign bitmap information that renders the derived bounds much tighter.

Third, a multi-resolution **search algorithm** that leverages our storage and bounding schemes. Apart from [52], similar algorithms have been employed in [47] in the context of DNA sequences, and in [38] in the context of stream time-series. Yet these works perform pruning based solely on loose lower distance bounds, hence address *ϵ-range similarity queries* only; this methodology does not apply to the more challenging kNN-search problem. Our method applies to this problem, as it derives not only lower, but also upper distance bounds, none of which has appeared in previous works.

2. RELATED WORK

The classical approach to time series similarity search uses a *dimensionality reduction* technique along with a *multidimensional access method*. The first dimensionality reduction technique suggested for that purpose was the DFT [2]. Subsequent research proposed several indexable transforms, such as PCA [50, 34, 29], the DCT [37], the DWT [14, 41, 52], and *Chebyshev Polynomials* [11], as well as *segmentation* methods, such as the PAA [53, 33], the APCA [12], and the PLA [16]. A recent experimental comparison of diverse dimensionality reduction methods on the accuracy of approximation they provide concluded that there is "very little to choose" between them [20, 46]. Past research has also provided a host of methods for high-dimensional search [24, 45, 5, 40, 8, 32, 17, 7, 49, 13, 6, 43, 54, 28, 18, 4]. Yet such methods have not been tested on more than 64 dimensions, and are not immune to the curse of dimensionality; their performance can deteriorate to that of a sequential scan [25, 10, 18, 39, 9].

An alternative approach to time series indexing and search is given by the *indexable Symbolic Aggregate approXimation* (*i*SAX) [46], an indexable variant of SAX [39]. SAX transforms the data to their PAA representation and then symbolizes the latter to a discrete string, making better use of every available bit; a symbolic distance measure lower-bounds the PAA distance itself. The symbolic representation is then enhanced to a multi-resolution form and indexed using a standard file system for disk access. *i*SAX is reported to outperform sequential scan in exact time series NN search [46]. Thus, it provides a yardstick for other methods to be compared to.

BOND [19] proposes a physical organization of image histogram feature vectors to facilitate kNN search over the images they represent. The feature coefficients of the same dimension, for all vectors, are stored in a separate table. This representation allows for the distance between a query point and all data vectors to be accumulated incrementally, by scanning these projections one-by-one, and computing both a lower and an upper bound on the employed similarity metric. Our work shares a fundamental intuition with [19], yet the involved storage schemes, similarity metrics, queried objects, bounding methods, and application domains are different.

Our work also shares features with CoMRI [47], a grid-style index structure that facilitates similarity search among DNA sequences with a multi-resolution search algorithm, and MSM [38],

an incrementally computable representation for stream time-series data using a multi-step filtering mechanism. However, these methods only address *range* queries, as the pruning they perform is based only on *loose lower* bounds to distances, which are compared to a range similarity threshold ϵ. Unfortunately, this methodology does not apply to the more challenging *kNN search* problem.

Furthermore, several works have examined the problem of *approximate* high-dimensional similarity search, where exact answers are not necessarily required [27, 23, 3, 48]. We study the exact problem, where one needs to find the exact k most similar records.

Symbol	Meaning
$\mathcal{D}(P,Q)$	Euclidean distance between series P and Q
ℓ_i	Haar tree level of coefficient p_i
d	Haar tree depth
I_ℓ	Coefficient indexes in level ℓ
$\mathcal{D}_\ell$	$2^\ell \sum_{i\in I_\ell} \lvert p_i - q_i\rvert^2$
$\mathcal{D}_k^r$	$\sum_{\ell=r}^{d} \mathcal{D}_\ell$
$\mathcal{D}_u^r$	$\sum_{\ell=1}^{r-1} \mathcal{D}_\ell$
$\Sigma_{\ell p}^r$	$\sum_{\ell=1}^{r-1} \sum_{i\in I_\ell} 2^\ell p_i^2$
$\Sigma_{\ell q}^r$	$\sum_{\ell=1}^{r-1} \sum_{i\in I_\ell} 2^\ell q_i^2$
Σ_{pq}^r	$\sum_{\ell=1}^{r-1} \sum_{i\in I_\ell} -2^\ell p_i q_i$
$\Sigma_{\ell p}$	$\sum_\ell 2^\ell \sum_{i\in I_\ell} p_i^2$
$\Sigma_{\ell q}$	$\sum_\ell 2^\ell \sum_{i\in I_\ell} q_i^2$
O_ℓ	$\{i \in I_\ell \mid p_i q_i < 0\}$
E_ℓ	$\{i \in I_\ell \mid p_i q_i > 0\}$
$\Sigma_{pq}^{r,o}$	$\sum_{\ell=1}^{r-1} \sum_{i\in O_\ell} 2^\ell \lvert p_i q_i\rvert$
$\Sigma_{pq}^{r,e}$	$\sum_{\ell=1}^{r-1} \sum_{i\in E_\ell} 2^\ell \lvert p_i q_i\rvert$
Σ_p	$\sum_i p_i^2$
Σ_p^r	$\sum_{\ell=1}^{r-1} \sum_{i\in I_\ell} p_i^2$
Σ_q^o	$\sum_\ell \sum_{i\in O_\ell} 2^{2\ell} q_i^2$
Σ_q^e	$\sum_\ell \sum_{i\in E_\ell} 2^{2\ell} q_i^2$
$\Sigma_q^{r,o}$	$\sum_{\ell=1}^{r-1} \sum_{i\in O_\ell} 2^{2\ell} q_i^2$
$\Sigma_q^{r,e}$	$\sum_{\ell=1}^{r-1} \sum_{i\in E_\ell} 2^{2\ell} q_i^2$
$\mathcal{D}_{\mathsf{LB}}^r$	$\mathcal{D}_k^r + \Sigma_{\ell p}^r + \Sigma_{\ell q}^r - 2\sqrt{\Sigma_p^r \Sigma_q^{r,e}}$
$\mathcal{D}_{\mathsf{UB}}^r$	$\mathcal{D}_k^r + \Sigma_{\ell p}^r + \Sigma_{\ell q}^r + 2\sqrt{\Sigma_p^r \Sigma_q^{r,o}}$

Table 1: Employed notation

3. STEPWISE SIMILARITY SEARCH

Our method exploits a *multi-resolution* transform and a double-bounding scheme. This family of transforms offer a capability that other dimensionality reduction techniques do not: they allow for a distance can be computed at several *levels of resolution*. Previous research has not fully exploited the potential arising from this property. We explore this untapped potential. We present our method using the Discrete Haar Wavelet Transform (DHWT). Table 1 gathers all the notations and defining equations we introduce.

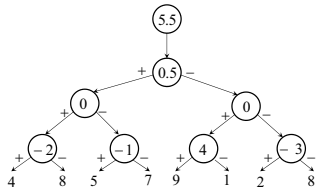


Figure 2: Discrete Haar Wavelet Transform

3.1 Distance-Bounding Computations

In its simplified non-normalized form, the DHWT can be visualized by a complete binary tree, the *Haar tree* [14, 31, 52]. The root node coefficient contains the overall average value; each other coefficient value c_i adds $+c_i$ to data cells (leaves) in its *left* subtree and $-c_i$ to those in its *right* subtree. These coefficients are computed by a recursive process that extracts *pairwise* averages and differences. The averages at one level serve as the input data for the level above; the pairwise semi-differences are the coefficients [30]. A data value is reconstructed by summing the (signed) terms along a root-to-leaf path. We opt for this intuitive formulation for the sake of clarity and conciseness; the *normalized* DWT definition includes further factors in the computation, represented by a 2^{ℓ_i} term in the following. Figure 2 shows the Haar tree representing the complete DHWT for the vector $\mathbf{D} = \{4, 8, 5, 7, 9, 1, 2, 8\}$.

Given the DHWT of a time series, $P = \{p_i\}$, and that of a query record $Q = \{q_i\}$, their squared (non-normalized) Euclidean distance can be computed on the wavelet domain [14, 52] as:

$$\mathcal{D}(P,Q) = \textstyle\sum_i 2^{\ell_i} |p_i - q_i|^2 \tag{1}$$

where $\ell_i = \lfloor \log_2 i \rfloor$ is the bottom-up DHWT level in which coefficients p_i and q_i belong [14, 52]; the top level consists of the top *two* coefficients. In a Haar tree of depth d, with $n = 2^d$ coefficients, $\ell_i = d - \lfloor \log_2 i \rfloor$, Eq. (1) can be written as:

$$\mathcal{D}(P,Q) = \textstyle\sum_{\ell=1}^{d} 2^\ell \sum_{i\in I_\ell} |p_i - q_i|^2 = \sum_{\ell=1}^{d} \mathcal{D}_\ell(P,Q) \tag{2}$$

where d is the depth of trees for the DHWTs P and Q, ℓ a tree level, $I_\ell = \{2^{d-\ell}, \ldots, 2^{d-\ell+1} - 1\}$ the range of coefficient indexes in level ℓ, and $\mathcal{D}_\ell = 2^\ell \sum_{i\in I_\ell} |p_i - q_i|^2$ the distance contribution by level ℓ (we drop the determinant (P,Q) when it is implied from context). This distance can be computed in a *stepwise* manner, reading *one level at a time*, following Eq. (2). Assume all DHWT levels from d to r have been read. Then the distance is divided in a *known* element $\mathcal{D}_k^r$, from the levels already read, and an *unknown* element $\mathcal{D}_u^r$, from the yet unread lower levels:

$$\mathcal{D} = \textstyle\sum_{\ell=r}^{d} \mathcal{D}_\ell + \sum_{\ell=1}^{r-1} \mathcal{D}_\ell = \mathcal{D}_k^r + \mathcal{D}_u^r \tag{3}$$

To progressively eliminate false hits, we need to derive *tight* double (i.e., not only lower, but also upper) bounds for the unknown element $\mathcal{D}_u^r$, and gradually refine them as the computation proceeds. At a given level r, $1 \le r \le d$, $\mathcal{D}_u^r$ is expressed as:

$$\mathcal{D}_u^r = \sum_{\ell=1}^{r-1} 2^\ell \sum_{i\in I_\ell} |p_i - q_i|^2 = \sum_{\ell=1}^{r-1} 2^\ell \sum_{i\in I_\ell} \left(p_i^2 + q_i^2 - 2p_i q_i\right)$$

Let $\Sigma_{\ell p}^r = \sum_{\ell=1}^{r-1} 2^\ell \sum_{i\in I_\ell} p_i^2$, $\Sigma_{\ell q}^r = \sum_{\ell=1}^{r-1} 2^\ell \sum_{i\in I_\ell} q_i^2$, and $\Sigma_{pq}^r = -\sum_{\ell=1}^{r-1} 2^\ell \sum_{i\in I_\ell} p_i q_i$. Then:

$$\mathcal{D}_u^r = \Sigma_{\ell p}^r + \Sigma_{\ell q}^r + 2\Sigma_{pq}^r \tag{4}$$

The first two terms in Eq. (4) can be derived using pre-computed sums $\Sigma_{\ell p} = \sum_\ell 2^\ell \sum_{i\in I_\ell} p_i^2$ for each time series record P, as well as the sum $\Sigma_{\ell q} = \sum_\ell 2^\ell \sum_{i\in I_\ell} q_i^2$ for the query record Q, and *progressively* subtracting the *partial sums* $2^\ell \sum_{i\in I_\ell} p_i^2$ and $2^\ell \sum_{i\in I_\ell} q_i^2$, respectively, for each level ℓ. The challenge we face is to bound the term Σ_{pq}^r. We treat it as follows. Let O_ℓ be the set of indexes in I_ℓ such that p_i and q_i have *opposite* signs: $O_\ell = \{i \in I_\ell \mid p_i q_i < 0\}$, and E_ℓ the index-set such that p_i and q_i have *equal* signs: $E_\ell = \{i \in I_\ell \mid p_i q_i > 0\}$. Moreover, let $\Sigma_{pq}^{r,o} = \sum_{\ell=1}^{r-1} 2^\ell \sum_{i\in O_\ell} |p_i q_i|$ and $\Sigma_{pq}^{r,e} = \sum_{\ell=1}^{r-1} 2^\ell \sum_{i\in E_\ell} |p_i q_i|$. Then the critical term in Eq. (4) can be expressed using these two non-negative terms:

$$\mathcal{D}_u^r = \Sigma_{\ell p}^r + \Sigma_{\ell q}^r + 2\Sigma_{pq}^{r,o} - 2\Sigma_{pq}^{r,e} \tag{5}$$

These two are bounded using the Cauchy-Schwartz inequality:

$$\Sigma_{pq}^{r,o} = \sum_{\ell=1}^{r-1} \sum_{i\in O_\ell} 2^\ell |p_i q_i| \le \sqrt{\sum_{\ell=1}^{r-1} \sum_{i\in O_\ell} p_i^2 \sum_{\ell=1}^{r-1} \sum_{i\in O_\ell} \left(2^\ell q_i\right)^2}$$

$$\Sigma_{pq}^{r,e} = \sum_{\ell=1}^{r-1} \sum_{i\in E_\ell} 2^\ell |p_i q_i| \le \sqrt{\sum_{\ell=1}^{r-1} \sum_{i\in E_\ell} p_i^2 \sum_{\ell=1}^{r-1} \sum_{i\in E_\ell} \left(2^\ell q_i\right)^2}$$

A computation of the quantities in the above equations that involve selected p_i^2 terms would require an explicit reading of the coefficients of the record P. Thus, these quantities can only be bounded using the pre-computed total sum $\Sigma_p = \sum_i p_i^2$ and calculating $\Sigma_p^r = \sum_{\ell=1}^{r-1} \sum_{i \in I_\ell} p_i^2$ on demand by subtracting the *partial sum* $\sum_{i \in I_\ell} p_i^2$ at each level ℓ. On the other hand, the quantities involving selected q_i^2 terms can be derived using pre-computed *coefficient sign bitmaps* of Q and P. Based on such bitmaps, the overall sums $\Sigma_q^o = \sum_\ell \sum_{i \in O_\ell} 2^{2\ell} q_i^2$ and $\Sigma_q^e = \sum_\ell \sum_{i \in E_\ell} 2^{2\ell} q_i^2$ can be pre-computed, given Q. Then, $\Sigma_q^{r,o} = \sum_{\ell=1}^{r-1} \sum_{i \in O_\ell} 2^{2\ell} q_i^2$ and $\Sigma_q^{r,e} = \sum_{\ell=1}^{r-1} \sum_{i \in E_\ell} 2^{2\ell} q_i^2$ can be calculated on demand by subtracting the *partial sums* $2^{2\ell} \sum_{i \in O_\ell} q_i^2$ and $2^{2\ell} \sum_{i \in E_\ell} q_i^2$, respectively, at each level ℓ. We group the $2^{2\ell}$ term along with the selected q_i^2 terms instead of dividing the two 2^ℓ factors among p_i^2 and q_i^2. Thus, we reassure that a 2^ℓ factor enters the bounds' calculation only when necessary and avoid an overestimation. Eventually:

$$\Sigma_{pq}^{r,o} \leq \sqrt{\Sigma_p^r \Sigma_q^{r,o}}, \qquad \Sigma_{pq}^{r,e} \leq \sqrt{\Sigma_p^r \Sigma_q^{r,e}} \tag{6}$$

By Eqs. (5) and (6), we can doubly-bound $\mathcal{D}_u^r$ as follows:

$$\Sigma_{\ell p}^r + \Sigma_{\ell q}^r - 2\Sigma_{pq}^{r,e} \leq \mathcal{D}_u^r \leq \Sigma_{\ell p}^r + \Sigma_{\ell q}^r + 2\Sigma_{pq}^{r,o} \Leftrightarrow$$

$$\Sigma_{\ell p}^r + \Sigma_{\ell q}^r - 2\sqrt{\Sigma_p^r \Sigma_q^{r,e}} \leq \mathcal{D}_u^r \leq \Sigma_{\ell p}^r + \Sigma_{\ell q}^r + 2\sqrt{\Sigma_p^r \Sigma_q^{r,o}}$$

We thus specify a *tight lower bound* $\mathcal{D}_{\mathsf{LB}}^r$, as well as a *tight upper bound* $\mathcal{D}_{\mathsf{UB}}^r$, for $\mathcal{D}$ when all levels from d to r have been read:

$$\mathcal{D}_{\mathsf{LB}}^r = \mathcal{D}_k^r + \Sigma_{\ell p}^r + \Sigma_{\ell q}^r - 2\sqrt{\Sigma_p^r \Sigma_q^{r,e}} \tag{7}$$

$$\mathcal{D}_{\mathsf{UB}}^r = \mathcal{D}_k^r + \Sigma_{\ell p}^r + \Sigma_{\ell q}^r + 2\sqrt{\Sigma_p^r \Sigma_q^{r,o}} \tag{8}$$

Using these nontraditional bounds, we can *prune* a candidate P from a kNN result, when there exist k records such that the $\mathcal{D}_{\mathsf{LB}}^r$ of P is larger than their k $\mathcal{D}_{\mathsf{UB}}^r$ values. Thus, a pre-computation of sign bitmaps and sums assists us towards more efficient search. We emphasize the fact that this double-bounding scheme is *tighter* than those used in previous works using either DHWT [14, 41, 52] or elsewhere [21, 34, 29, 37, 53, 33, 12, 11, 16]. The previously best-known lower distance bound, given in [14], consists of the first term of Eq. (7) alone, while the previously best-known upper distance bound, given in [52], uses $\Sigma_q^r = \sum_{\ell=1}^{r-1} \sum_{i \in I_\ell} 2^{2\ell} q_i^2$ in Eq. (8), which is an overestimation, as it does not take into consideration the signs of coefficients as we do. Furthermore, we re-iterate that we do not use these bounds in the same way as traditional index-based methods, but in a nontraditional manner for progressive pruning. However, in our experimental study (Section 4.2) we compare our method to a modified variant of itself that uses these looser bounds [14, 41, 52] instead of the tighter ones that we derive.

A similar *tight* analysis applies to other multi-resolution transforms. We have used the DHWT only as an arbitrary choice for ease of illustration. The tightness of the bounds we compute derives *from our analysis*, not from any property of the transform. Based on our preceding analysis and the notations we have introduced, we now outline the operation of our model in more detail.

3.2 Similarity Search

Our kNN search algorithm first computes the DHWT of the query series Q and precalculates $\Sigma_{\ell q}$ and its sign bitmap. It then reads the appropriate pre-computed sum quantities $\Sigma_{\ell p}$, Σ_p, as well as the sign bitmap, of each record P in the full candidate set $\mathcal{C}$. Having these bitmaps, it derives Σ_q^o and Σ_q^e for each $P \in \mathcal{C}$. Then it progressively reads their coefficients level-by-level, in a stepwise manner. After reading the coefficients of all *still active* candidates at level r, it adjusts the values of $\mathcal{D}_k^r$, $\Sigma_{\ell p}^r$, $\Sigma_{\ell q}^r$, Σ_p^r, $\Sigma_q^{r,o}$, $\Sigma_q^{r,e}$ to r, and updates the bounds $\mathcal{D}_{\mathsf{LB}}^r$ and $\mathcal{D}_{\mathsf{UB}}^r$ according to Eqs. (7) and (8), for each candidate record in $\mathcal{C}$. Candidates that assume a $\mathcal{D}_{\mathsf{LB}}^r$ *higher* than the running k^{th} $\mathcal{D}_{\mathsf{UB}}^r$ value are *eliminated* from $\mathcal{C}$ at each step. The process terminates when $|\mathcal{C}| = k$.

In more detail, assume a candidate record $P \in \mathcal{C}$, still active at level r. After reading the level-r coefficients of P, the values of $\mathcal{D}_k^r$, $\Sigma_{\ell p}^r$, and Σ_p^r at level r are effectively derived from their previous values at $r+1$, as follows.

$$\mathcal{D}_k^r = \mathcal{D}_k^{r+1} + 2^r \sum_{i \in I_r} |p_i - q_i|^2 \tag{9}$$

$$\Sigma_{\ell p}^r = \Sigma_{\ell p}^{r+1} - 2^r \sum_{i \in I_r} p_i^2 \tag{10}$$

$$\Sigma_p^r = \Sigma_p^{r+1} - \sum_{i \in I_r} p_i^2 \tag{11}$$

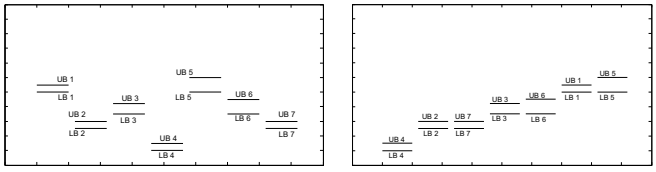
Likewise, the values of $\Sigma_{\ell q}^r$, $\Sigma_q^{r,e}$, and $\Sigma_q^{r,o}$ at level r are adjusted with respect to those at level $r+1$ as follows.

$$\Sigma_{\ell q}^r = \Sigma_{\ell q}^{r+1} - 2^r \sum_{i \in I_r} q_i^2 \tag{12}$$

$$\Sigma_q^{r,o} = \Sigma_q^{r+1,o} - 2^{2r} \sum_{i \in O_\ell} q_i^2 \tag{13}$$

$$\Sigma_q^{r,e} = \Sigma_q^{r+1,e} - 2^{2r} \sum_{i \in E_\ell} q_i^2 \tag{14}$$

While the *same* value of $\Sigma_{\ell q}^r$ is used by each candidate record P, the values of $\Sigma_q^{r,o}$ and $\Sigma_q^{r,e}$ are specific for each $P \in \mathcal{C}$, as they depend on the sign bitmap of P. Given the preceding computations, the algorithm updates the values of $\mathcal{D}_{\mathsf{LB}}^r$ and $\mathcal{D}_{\mathsf{UB}}^r$ for each still active candidate in $\mathcal{C}$, and uses them to further prune this candidate set $\mathcal{C}$, until it reaches the desired cardinality. As all pruned candidates have had a lower distance bound higher than the upper distance bounds of at least k other candidates, the final result is correct. The algorithm performs a significant amount of calculations for the derivation of bounds. Thus, a careful implementation is important to achieve its full potential. Section 3.3 provides a pseudo-code and describes some critical implementation details. Our algorithm requires that all time series to be stored in their full Haar wavelet transform, and their sign bitmaps and sum-of-squares values to be available. Our storage scheme is discussed in Section 3.6.



(a) unsorted candidate set $\mathcal{C}$ (b) $\mathcal{C}$ sorted by upper bound

Figure 3: Candidate Pruning

Figure 3 illustrates an example of our nontraditional pruning process, where $\mathcal{C}$ contains 7 candidate records (Figure 3a), which are sorted by upper bound (UB) in Figure 3b. If, for instance, $k = 4$, then the two last candidates in Figure 3b are pruned, as their LB values are higher than the 4^{th} upper bound (UB3).

3.3 Pseudo-code and Implementation

Figure 4 presents a pseudo-code for our Stepwise algorithm. Due to the large amount of calculations it performs, our algorithm requires a careful implementation to perform at its full potential. Our implementation pre-computes an array of q_i^2 values, to use when adjusting sums in Step 7 (Eqs. (13) and (14)). During the construction of this array, we also store sums of $\sum_{i \in I_r} q_i^2$ for each level r, to be used when updating $\Sigma_{\ell q}^r$, which is computed once at each level and used by all active candidates. Similar optimizations are used in all calculations, avoiding redundancies and never computing a quantity used in our equations more than once. At Step

9, we maintain two priority queues of $\mathcal{D}^r_{\mathsf{UB}}$ values: a max-heap of the k smallest ones, and a min-heap of all the rest. During updates, records can move between these two. Furthermore, we maintain a linked-list of active candidate objects in $\mathcal{C}$, with all associated sum quantities. During pruning (Step 10), we delete disqualified elements by sequentially scanning this list. We update the 2^r and 2^{2r} values (in Equations (9), (10), (12)-(14)) by division at each iteration. In terms of storage, we create multiple files per level at deeper tree levels. Terms for 1000 time series are stored in one file for each level. Each record occupies a new line in the file. We only access files that contain at least one active candidate at each level. We ensure each term occupies the same number of bytes on disk and *seek* each term using its offset in the file. Thus, both files and terms are accessed by ordered positional lookups.

Algorithm Stepwise(D, Q, k)
Input: Set of DWTs in data set D, query series Q
Required number of nearest neighbors k
Output: Set of k records in D closest to Q
1. transform Q to DWT, compute $\Sigma_{\ell q}$, sign bitmap;
2. $r = d$; candidate set $\mathcal{C}$ = all records;
3. read $\Sigma_{\ell p}$, Σ_p, sign bitmaps for records in $\mathcal{C}$;
4. calculate Σ^o_q, Σ^e_q for each record in $\mathcal{C}$;
5. **while** $|\mathcal{C}| > k$
6. at level r, read terms of records in $\mathcal{C}$;
7. adjust $\mathcal{D}^r_k$, $\Sigma^r_{\ell p}$, $\Sigma^r_{\ell q}$, Σ^r_p, $\Sigma^{r,o}_q$, $\Sigma^{r,e}_q$ to level r;
8. update $\mathcal{D}^r_{\mathsf{LB}}$, $\mathcal{D}^r_{\mathsf{UB}}$ for each candidate (Eqs. (7-8));
9. find k^{th} $\mathcal{D}^r_{\mathsf{UB}}$ value in $\mathcal{C}$;
10. prune records having $\mathcal{D}^r_{\mathsf{LB}} > k^{th}$ $\mathcal{D}^r_{\mathsf{UB}}$ from $\mathcal{C}$
11. $r = r - 1$;
12. **return** $\mathcal{C}$ as result set;

Figure 4: Stepwise Similarity Search

3.4 Proof of Soundness

The correctness of our method is based on the assumption that both calculated bounds become progressively tighter. We have hitherto assumed this to be the case. Still, this mathematical property requires proof, which we offer in the following theorem. A similar proof is given in [52], albeit for a looser bounding scheme.

THEOREM 1. *The calculated lower bound of a distance calculation does not decrease from one level $r+1$ to the successor level r in the computation.*

PROOF. We have to prove the difference between two consecutively calculated lower distance bounds, from (upper) level $r+1$ to (lower) level r, is non-negative, that is:

$$\begin{aligned}
&\mathcal{D}^r_{\mathsf{LB}} - \mathcal{D}^{r+1}_{\mathsf{LB}} \geq 0 \\
\Leftrightarrow\quad &\left(\mathcal{D}^r_k + \Sigma^r_{\ell p} + \Sigma^r_{\ell q} - 2\sqrt{\Sigma^r_p \Sigma^{r,e}_q}\right) - \\
&\left(\mathcal{D}^{r+1}_k + \Sigma^{r+1}_{\ell p} + \Sigma^{r+1}_{\ell q} - 2\sqrt{\Sigma^{r+1}_p \Sigma^{r+1,e}_q}\right) \geq 0 \\
\Leftrightarrow\quad &2^r \sum_{i\in I_r} |p_i - q_i|^2 - \sum_{i\in I_r} 2^r p_i^2 - \sum_{i\in I_r} 2^r q_i^2 \\
&+ 2\left(\sqrt{\Sigma^{r+1}_p \Sigma^{r+1,e}_q} - \sqrt{\Sigma^r_p \Sigma^{r,e}_q}\right) \geq 0 \\
\Leftrightarrow\quad &2\left(\sqrt{\Sigma^{r+1}_p \Sigma^{r+1,e}_q} - \sqrt{\Sigma^r_p \Sigma^{r,e}_q}\right) - 2\cdot 2^r \sum_{i\in I_r} p_i q_i \geq 0 \\
\Leftrightarrow\quad &\sqrt{\Sigma^{r+1}_p \Sigma^{r+1,e}_q} \geq \sqrt{\Sigma^r_p \Sigma^{r,e}_q} + 2^r \sum_{i\in I_r} p_i q_i \qquad (15)
\end{aligned}$$

By definition it is $\sum_{i\in I_r} p_i q_i = \sum_{i\in E_r} p_i q_i + \sum_{i\in O_r} p_i q_i$, where it holds that $\sum_{i\in E_r} p_i q_i = \sum_{i\in E_r} |p_i q_i| \geq 0$, while $\sum_{i\in O_r} p_i q_i = \sum_{i\in O_r} -|p_i q_i| \leq 0$. Thus, in order to prove Equation (15), it suffices to prove its stronger version:

$$\sqrt{\Sigma^{r+1}_p \Sigma^{r+1,e}_q} \geq \sqrt{\Sigma^r_p \Sigma^{r,e}_q} + 2^r \sum_{i\in E_r} p_i q_i \qquad (16)$$

Both sides of Eq. (16) are non-negative, hence it suffices to prove the inequality of their squares. The square of the left-hand term is:

$$\Sigma^{r+1}_p \Sigma^{r+1,e}_q = \left(\Sigma^r_p + \sum_{i\in I_r} p_i^2\right) \cdot \left(\Sigma^{r,e}_q + \sum_{i\in E_r} 2^{2r} q_i^2\right) =$$
$$\Sigma^r_p \Sigma^{r,e}_q + 2^{2r} \sum_{i\in I_r} p_i^2 \sum_{i\in E_r} q_i^2 + 2^{2r} \Sigma^r_p \sum_{i\in E_r} q_i^2 + \Sigma^{r,e}_q \sum_{i\in I_r} p_i^2 \qquad (17)$$

Likewise, the square of the right-hand term is:

$$\Sigma^r_p \Sigma^{r,e}_q + 2^{2r} \left(\sum_{i\in E_r} p_i q_i\right)^2 + 2\cdot 2^r \sqrt{\Sigma^r_p \Sigma^{r,e}_q} \cdot \sum_{i\in E_r} p_i q_i \qquad (18)$$

The first terms in Eqs. (17) and (18) are identical, while the second term of the former is not less than that of the latter, since:

$$\sum_{i\in I_r} p_i^2 \sum_{i\in E_r} q_i^2 \geq \sum_{i\in E_r} p_i^2 \sum_{i\in E_r} q_i^2 \geq \left(\sum_{i\in E_r} p_i q_i\right)^2$$

where the last inequality is the Cauchy-Schwartz inequality. Thus, to prove Equation (16), it suffices to prove that:

$$2^{2r} \Sigma^r_p \sum_{i\in E_r} q_i^2 + \Sigma^{r,e}_q \sum_{i\in I_r} p_i^2 \geq 2\cdot 2^r \cdot \sqrt{\Sigma^r_p \Sigma^{r,e}_q} \cdot \sum_{i\in E_r} p_i q_i \qquad (19)$$

From the inequality of arithmetic and geometric means we get:

$$2^{2r} \Sigma^r_p \sum_{i\in E_r} q_i^2 + \Sigma^{r,e}_q \sum_{i\in I_r} p_i^2 \geq 2\cdot 2^r \sqrt{\Sigma^r_p \Sigma^{r,e}_q} \sqrt{\sum_{i\in I_r} p_i^2 \sum_{i\in E_r} q_i^2} \qquad (20)$$

In addition, it holds that:

$$\sqrt{\sum_{i\in I_r} p_i^2 \sum_{i\in E_r} q_i^2} \geq \sqrt{\sum_{i\in E_r} p_i^2 \sum_{i\in E_r} q_i^2} \geq \sum_{i\in E_r} |p_i q_i| = \sum_{i\in E_r} p_i q_i \qquad (21)$$

where the latter inequality is, again, an application of the Cauchy-Schwartz inequality. Eventually, the desired result (Eq. (19)) follows by putting Eqs. (20) and (21) together. □

Along similar lines, it can be shown that the calculated *upper* bound *does not increase* from one (upper) level $r+1$ to the (lower) successor level r in the computation, i.e., the difference between two consecutively calculated upper distance bounds is *non-positive*.

Quantity	Level $r=3$	Level $r=2$
$\mathcal{D}^r_k(P_1, Q)$	4.0	24.0
$\mathcal{D}^r_k(P_2, Q)$	1.0	6.0
$\Sigma^r_{\ell p_1}$	60.0	60.0
$\Sigma^r_{\ell p_2}$	25.0	20.0
$\Sigma^r_{\ell q}$	28.0	8.0
$\Sigma^r_{p_1}$	30.0	30.0
$\Sigma^r_{p_2}$	11.25	10.0
$\Sigma^{r,o}_q(P_1)$	8.0	8.0
$\Sigma^{r,e}_q(P_1)$	8.0	8.0
$\Sigma^{r,o}_q(P_2)$	0.0	0.0
$\Sigma^{r,e}_q(P_2)$	96.0	16.0
$\mathcal{D}^r_{\mathsf{LB}}(P_1)$	61.016	61.016
$\mathcal{D}^r_{\mathsf{UB}}(P_1)$	122.984	122.984
$\mathcal{D}^r_{\mathsf{LB}}(P_2)$	0.0	8.702
$\mathcal{D}^r_{\mathsf{UB}}(P_2)$	54.0	34.0

Table 2: Calculation Example

3.5 Numerical Example

We offer a concrete numerical example of stepwise calculation of the distance between a query time series and two candidate nearest neighbors. Let $P_1 = \{4, 8, 5, 7, 9, 1, 2, 8\}$, as in Figure 2, and

$P_2 = \{2, 6, 5, 7, 4, 6, 8, 4\}$. We wish to detect the nearest neighbor to $Q = \{2, 4, 6, 8, 3, 5, 7, 5\}$. The DHWTs are:

$$\mathcal{W}(P_1) = \{5.5, 0.5, 0, 0, -2, -1, 4, -3\}$$
$$\mathcal{W}(P_2) = \{5.25, -0.25, -1, -0.5, -2, -1, -1, 2\}$$
$$\mathcal{W}(Q) = \{5, 0, -2, -1, -1, -1, -1, 1\}$$

The actual distances between the candidates and the query are $DST(P_1, Q) = 108$ and $DST(P_2, Q) = 10$. The stepwise calculation of these distances starts out by reading the level-0 coefficients (i.e., the first two coefficients) of P_1, P_2 and Q. Thus, it accumulates the known distances $\mathcal{D}^3_k(P_1, Q) = 8\times(0.5^2+0.5^2) = 4$ and $\mathcal{D}^3_k(P_2, Q) = 8 \times (0.25^2 + 0.25^2) = 1$. Furthermore, it adjusts the pre-computed sums of squares to derive the values that are needed for the calculation of bounds. Table 2 depicts the calculation of these quantities for $r = 3$ and $r = 2$.

Apart from the conventional sum-of-squares quantities, we outline the computation of the sign-dependent quantities. For example, $\Sigma^{3,o}_q(P_1)$ consists of the sum of squares of coefficients in $\mathcal{W}(Q)$ below level 3, *weighted* by $2^{2\ell}$, which are opposite-signed to the respective coefficients of P_1, i.e., coefficients in the set $O_3(Q, P_1) = \{-1, 1\}$, both in the (bottom) level 1, thus $\Sigma^{3,o}_q(P_1) = 2^2\times(1+1) = 8$. Likewise, $\Sigma^{0,e}_q(P_1)$ consists of the sum of squares of *weighted* coefficients in $\mathcal{W}(Q)$ below level 3 which are equally-signed to the respective coefficients of P_1, that is, bottom-level coefficients in the set $E_3(Q, P_1) = \{-1, -1\}$, hence $\Sigma^{3,o}_q(P_1) = 8$. In a similar fashion, $\Sigma^{3,o}_q(P_2)$ consists of the sum of $2^{2\ell}$-weighted squares of coefficients in $\mathcal{W}(Q)$ below level 3 which are opposite-signed to the respective coefficients of P_2, i.e., coefficients in the empty set, thus $\Sigma^{3,o}_q(P_1) = 0$. On the other hand, $\Sigma^{3,e}_q(P_2)$ consists of the sum of $2^{2\ell}$-weighted squares of coefficients in $\mathcal{W}(Q)$ below level 3 that are equally-signed to the respective coefficients of P_2, i.e., coefficients in the set $E_3(Q, P_2) = \{-2, -1, -1, -1, -1, 1\}$, in levels 2 and 1, hence $\Sigma^{3,e}_q(P_2) = 2^4\times(4+1)+2^2\times(1+1+1+1) = 96$.

Furthermore, $\Sigma^{2,o}_q(P_1)$ limits itself to coefficients in $\mathcal{W}(Q)$ below level 2 which are opposite-signed to the respective coefficients of P_1, that is, coefficients in the set $O_2(Q, P_1)$, which is the same as $O_3(Q, P_1)$, thus $\Sigma^{2,o}_q(P_1) = 8$. In a similar fashion, the quantities $\Sigma^r_{\ell p_1}$ and $\Sigma^r_{p_1}$ remain unchanged from level 3 to level 2, since P_1 has two zero coefficients in level 2, while $\Sigma^{2,e}_q(P_1) = 8$ also. Likewise, $\Sigma^{2,o}_q(P_2)$ remain defined by the empty set, hence $\Sigma^{2,e}_q(P_1) = 0$. Still, $\Sigma^{3,e}_q(P_2)$ is now confined to set of coefficients in $\mathcal{W}(Q)$ below level 2 that are equally-signed to the respective coefficients of P_2, i.e., coefficients in the set $E_2(Q, P_2) = \{-1, -1, -1, 1\}$, in level 1, hence $\Sigma^{3,e}_q(P_2) = 2^2\times(1+1+1+1) = 16$.

We can now compute the bounds at level $r = 3$ as:

$$\mathcal{D}^3_{\mathsf{LB}}(P_1, Q) = 4 + 60 + 28 - 2\sqrt{30 \times 8} = 61.016$$
$$\mathcal{D}^3_{\mathsf{UB}}(P_1, Q) = 4 + 60 + 28 + 2\sqrt{30 \times 8} = 122.984$$
$$\mathcal{D}^3_{\mathsf{LB}}(P_2, Q) = 1 + 25 + 28 - 2\sqrt{11.25 \times 96} = -11.727$$
$$\mathcal{D}^3_{\mathsf{UB}}(P_2, Q) = 1 + 25 + 28 + 2\sqrt{11.25 \times 0} = 54$$

The calculated value of -11.727 for $\mathcal{D}^3_{\mathsf{LB}}(P_2, Q)$ is substituted by 0 in Table 2, since a negative lower bound of distance is an anomaly due to the loosened bound for the $\Sigma^{r,e}_{pq}$ subtracted quantity. The calculation of the bounds $\mathcal{D}^r_{\mathsf{LB}}(P_1)$ and $\mathcal{D}^r_{\mathsf{UB}}(P_1)$ does not improve from level 3 to level 2, due to the zero values of P_1 coefficients at level 2 - the only difference in the calculation is that an amount of energy of the Q signal is moved from $\Sigma^r_{\ell q}$ to $\mathcal{D}^r_k(P_1, Q)$ (see Table 2); thus, both of these bounds at level 3 are already as tight as at level 2. Still, the calculations of the bounds $\mathcal{D}^r_{\mathsf{LB}}(P_2)$ and $\mathcal{D}^r_{\mathsf{UB}}(P_2)$ are refined at level $r = 2$ as below:

$$\mathcal{D}^2_{\mathsf{LB}}(P_2, Q) = 6 + 20 + 8 - 2\sqrt{10 \times 16} = 8.702$$
$$\mathcal{D}^2_{\mathsf{UB}}(P_2, Q) = 6 + 20 + 8 + 2\sqrt{10 \times 0} = 34$$

The computation can proceed in this fashion at subsequent levels. In our example, P_1 is already pruned as a candidate 1st nearest neighbor at level 3, since $\mathcal{D}^3_{\mathsf{LB}}(P_1) = 61.016 > \mathcal{D}^3_{\mathsf{UB}}(P_2) = 54$.

3.6 Storage Scheme

In our Stepwise algorithm, *features* of all records corresponding to the same level of resolution need to be read at the same time. Thus, I/O efficiency strongly depends on the availability of transform terms for all records in a *vertical* fashion. A storage scheme which allows the Stepwise method to take full advantage of its bitmap-based double-bounding scheme is one in which features (DHWT coefficients in our case) are stored grouped *by level of resolution*, instead of being grouped *by record*.

Thus, we store DWT coefficients level-by-level: level 0 for all records, followed by level 1, level 2, and so on. In disk storage terms, this scheme implies that in 'higher', lower-resolution Haar tree levels, where there are less coefficients per record, more than a single level of one record is stored per disk page. In 'lower', higher-resolution levels, as the number of terms per record increases, a single level of a single time series record can occupy more than one disk page. Sums and sign bitmaps are stored independently.

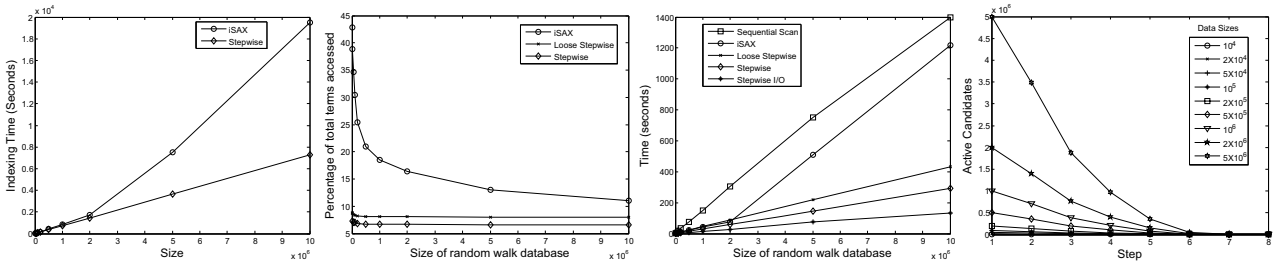
In more detail, level ℓ of the Haar tree (Figure 2) includes 2^ℓ DWT terms (level 0 has 2 terms). Assuming a page size of B bytes and b bytes per term, a page stores $\frac{B}{b}$ terms. Thus, level $\log \frac{B}{b}$ of a single record fits exactly in one disk page. Thereafter, level ℓ of each record occupies $2^{\ell-\log \frac{B}{b}}$ disk pages. For $\ell < \log \frac{B}{b}$, we store the 2^ℓ level-ℓ coefficients of $2^{\log \frac{B}{b} - \ell}$ time series on the same disk page. For example, if $B = 1$KB and $b = 4$bytes, then the 128 level-7 coefficients of 2 records are stored on the same page.

4. EXPERIMENTAL EVALUATION

In this section we experimentally evaluate the performance of our Stepwise method compared to previous time-series kNN search techniques. In Section 4.2, we compare to *i*SAX [46]. Then, in Section 4.3, we compare to classical two-component methods; we examine these techniques due to their long history, even though their utility has been called into question [46]. Algorithms were implemented in MS Visual C# 2005. Experiments ran on an Intel Xeon 3Ghz server with 64GB of main memory and 150GB of hard disk space running Windows Server 2003.

4.1 Description of Data

We used two synthetic data collections. The first, RandomWalk, is created using the Random Walk time series generator available in the *i*SAX package [1]. The seed of the generator was set at 1416. We produced data sets of 256-length time series, ranging from $10,000$ to $10M$. Each of these time series is z-normalized (i.e., has mean 0 and standard deviation 1). The second collection, Patterns, is a synthetic data set created using the data generation algorithms in [22]; these generators were also used in [20] and provided to us by those authors. The data consists of patterns belonging to 4 classes. The positions and durations of the patterns are randomized. Around the patterns, the signal is characterized by independent Gaussian noise. We generated $1M$ records of length 1024, and used prefixes of them ranging from $10,000$ to $1M$ in our experiments. The pattern range used by the process of [22] was set at $[80, 110]$. To follow the conventions used in [46], we z-normalized each time series of length 1024 (i.e., rendered the mean of each record 0 and its standard deviation 1). We also generated



(a) Indexing Time (b) Fraction of Data accessed (c) Search Time (d) Active Candidates per Step

Figure 5: Comparison to *i*SAX: Random Walk Data

records of length 256 and 512, 1024, with pattern range $[16, 32]$ as queries for our experiments with classical methods.

We also used two real world data sets characterized by diverse features, originating from the UCR archive[1], and created by sampling a collection of constant-length subsequences from very large time series. ECG, described in [36], results from concatenating 5 electrocardiograms to create a time series of length 144002; 10000 subsequences of length 256 were extracted as before. EEG derives from the concatenation of 2 electroencephalogram signals of albin rats, creating a series of length $10,957,312$; 5000 subsequences of length 256 were extracted at instances separated by 256 time intervals $(1-256, 257-512, \ldots)$. For the purposes of evaluating scalability versus length, we extracted 1000 time series of length 1024 from the ECG electrocardiogram data at instances separated by 100 time intervals. Prefixes of the samples series were used to evaluate performance for varying time series length.

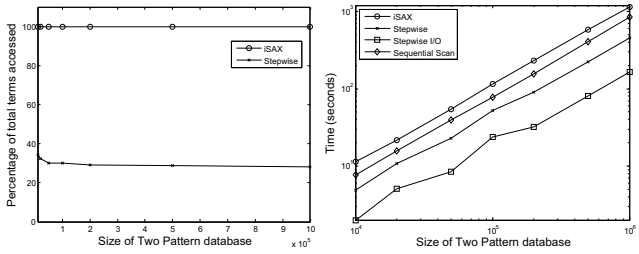
4.2 Comparison to *i*SAX

We first compare our method to *i*SAX [46] on large data sets that do not fit in main memory. We focus on examining how the two methods scale with data set size. The *word length* for *i*SAX was set at 10, while an *i*SAX file contained 100 time series, using the settings of [46]. The storage scheme of Stepwise was configured to store 1000 time series per file at each level. Separate files stored the employed sums and sign bitmaps. We emphasize that the first two experiments of this section are on z-normalized data, following the standard set in [46], and the third on non-normalized real data.

Our first set of experiments is on the RandomWalk data. First, we report the time required for *indexing* or creating all needed files, with each method. Our results in Figure 5(a) show that this time is more for *i*SAX. Next, we measure the *ratio* of accessed data to the full amount of data on disk. As Stepwise employs no index, and *i*SAX uses a minimal directory-based index with small overhead, this benchmark allows for a fair comparison. The *i*SAX code is configured to run for 1NN queries, hence we compare our methods on such queries. Figure 5(b) presents our results averaged over 30 random query runs for both methods. Stepwise maintains a fairly constant ratio of accessed data. This result indicates its reliability. *i*SAX starts out with a clear disadvantage for small sizes; reduces its percentage as size grows, but remains at a disadvantage.

We have emphasized that Stepwise owes its effectiveness to its *tight* bitmap-based double-bounding scheme, much tighter than those in previous works. To demonstrate its impact, we run a *loose* variant of Stepwise that employs the DHWT bounds calculated in [14, 41, 52] (see Section 3.1); this loose variant of Stepwise is *not* suggested in these works; [14, 41] apply traditional index-based pruning, while [52] conducts *distributed* time-series similarity search, where the aim to reduce not the I/O cost, but the communication cost among server and clients. Figure 5(b) shows that the loose variant of Stepwise reads more terms than its tight counterpart, but still less than *i*SAX. Thus, remarkably, not only the tight-bound version of our Stepwise approach, but also its loose-bound variant, outperforms the state-of-the-art *i*SAX method in this experiment. Figure 5(c) shows our wall-clock time results, including the time for search by sequential scan. The data access advantage of Stepwise translates to a wall-clock time advantage. We observe that the time of *i*SAX grows linearly at the *same* rate as the time of sequential scan, while that of *stepwise* grows more modestly.

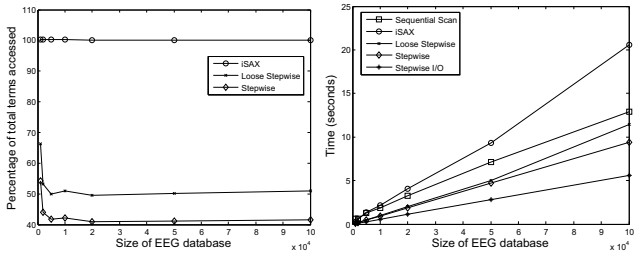
Stepwise raises high computational demands. In order to gauge their effect, we run a version of Stepwise that *eschews* all calculations, using precalculated quantities instead; it performs disk I/O only. Figure 5(c) shows the time required by this I/O-only method.



(a) Fraction of Data accessed (b) Search Time

Figure 6: Comparison to *i*SAX: Two-Pattern Data

Our second set of experiments is on the Patterns data of length 1024. We issue (z-normalized) random-walk queries of the appropriate length and average the results over 30 queries. Figure 6 shows the results on the fraction of data accessed and the wall-clock time. The results reconfirm our previous findings, revealing a poorer performance for *i*SAX. Now *i*SAX needs to access most of the data, while Stepwise accesses only a small fraction thereof. This fraction is predictably higher with these length-1024 z-normalized data, but still safely low. Most significantly, we find that the runtime of *i*SAX can be higher than that of a sequential scan. Stepwise performs two orders of magnitude better than *i*SAX in this experiment, as seen in Figure 6(b), which is in *log-log* axes.



(a) Fraction of Data accessed (b) Search Time

Figure 7: Comparison to *i*SAX: EEG (real data)

[1] See http://www.cs.ucr.edu./~eamonn/time_series_data

We also compared to *i*SAX on non-uniform and non-normalized real data. Figure 7 shows our results with different sizes of the EEG data set, queried by time series extracted from the same electroencephalogram series. The time of a sequential scan is also shown in Figure 7(b). These results corroborate our previous findings. Stepwise performs better than *i*SAX, which does not manage to access significantly less than 100% of the data. This finding is even more noteworthy, as it is observed with real data. Thus, we expect that a full sequential scan will outperform *i*SAX in this experiment. Figure 7(b) verifies our expectation. We also include measurements for the *loose* variant of Stepwise, which reads much more data than its *tight* counterpart. Still, their time difference is not so pronounced; this result indicates that Stepwise poses a tradeoff between disk I/O and bounding computations: it takes *more* CPU time to perform the distance-bounding computations that *save* I/O time. The results with the I/O-only variant confirm this observation. We expect that Stepwise will then have a still *greater* advantage over less CPU-intensive methods on a *faster* CPU.

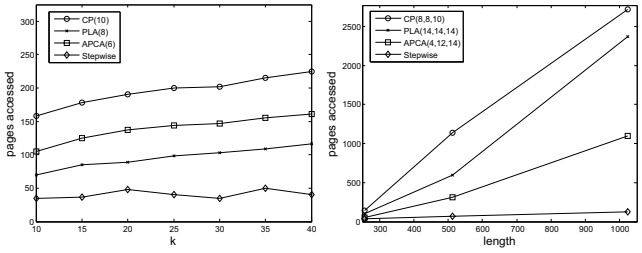
We have argued that Stepwise has an advantage over one-off pruning methods thanks to its nontraditional *progressive* candidate elimination. Still, this approach may not be worthwhile if candidates are already eliminated in the first pruning step. To confirm its worthiness, Figure 5(d) presents the number of *active candidates* left after each pruning step in the experiments in Figure 5; each line shows the average number of candidates per step for queries on one data size. While the largest elimination is done in the first step (which reads the four terms in the first two DHWT levels), a substantial number of candidates remains. Even if each new step eliminates less candidates than its predecessor, this elimination may still amount to a larger chunk of terms, as each subsequent DHWT level involves *twice* the size of its predecessor. Thus, even when only a few candidates are left, avoiding to read the next levels for some of them is worth the pruning effort.

4.3 Comparison to Classical Techniques

We now compare our approach to the most advanced classical kNN-search techniques, namely the methods based on APCA [12], CP [11], and PLA [16], with an R-tree index. Existing implementations of these methods operate in *main memory*, simulating disk page accesses, hence use data sets that fit in main memory. We conduct our investigation using the *same* conventions as [12, 11, 16] in this respect. We do not compare to predecessor methods, such as, e.g., the one using the DWT; the pruning power of the two-component method with DWT is the same as with PAA [33], which is in its turn superseded by the APCA method that we compare to. Thus, any benefit that we observe for Stepwise cannot be attributed to any assumed benefit of using the DWT. Our search algorithm and pruning scheme are not the same as the ones employed by these index-based methods, hence we cannot conduct our comparison using a pruning power metric; index-based method have been traditionally compared in terms of such metrics to *each other*, yet the rationale for this comparison does not apply in our case.

We measure the I/O cost in terms of page accesses necessitated by each method for kNN query processing with respect to length. Such I/O cost is measured in [12, 11, 16]. To create a level playing field, we assume zero cache and measure the number of pages faults, that is, page accesses. For Stepwise, we measure page accesses according to the disk storage scheme discussed in Section 3.6; by this scheme, pages are accessed in sequential order, while some pages are skipped; no page is accessed more than once. With all methods, we report data for 1KB disk page size and 4bytes per coefficient. Our results for larger page size (as is common in modern machines) were *less* favorable to classical methods.

Previous research has usually set the dimensionality of the feature space for methods based on a spatial access index at a fixed value, such as 12, with all the data it experimented with [12, 11, 16]. Yet a fixed dimensionality may not always represent the best point in the tradeoff between filtering and refinement (Figure 1(a)). We try dimensionality ranging from 3 to 20 with each index-based method and data set, and select the one that achieves the best performance; the optimal choice is strongly data-dependent.



(a) I/O vs. k (b) I/O vs. length

Figure 8: Page accesses vs. k & length, ECG

In extensive experiments, Stepwise outperformed these classical techniques. We also measured the preprocessing time required by Stepwise to be negligible compared to the indexing time required by these methods. We present two representative results. We first show the I/O cost of kNN search operations on the ECG data as a function of k. We create 100 query records of appropriate length using the data generation algorithms in [22] (see Section 4.1), and average the results over them on each data set. Figure 8(a) shows our results. The numbers in parentheses in the legend show the chosen dimensionality that exhibited best search performance with the given method. We observe a clear advantage for Stepwise. Interestingly, the I/O cost of Stepwise does not grow monotonically with k. A higher value of k may allow for lower cost, since the search terminates as soon as a candidate set is reduced to k members; on the other hand, the bound-based pruning potential is larger for lower values of k; thus, the effect of k on the efficiency of search is not straightforward. We also examine how our findings scale with respect to time series length. We use different-length versions of the ECG data on 100 queries of appropriate length created and with results averaged as before. Figure 8(b) presents these results, with $k = 10$. Again, the best-choice R-tree dimensionality is shown in parentheses per method *and length*. The advantage of Stepwise becomes more discernable with increasing length.

Overall, our results confirm our expectations, the comparison to *i*SAX being most significant. They also indicate that Stepwise provides a viable solution for data sets of longer time series records, with which the problem is more challenging.

5. CONCLUSIONS

This paper revisited the time-series kNN search problem and provided a solution that can effectively and scalably handle high-length time series. Instead of utilizing an index, we search in a *stepwise* manner, whereby records are *progressively* read and pruned at multiple levels of resolution; thus false hits are eliminated without being read in their full length. Our method could be used with any multi-resolution transform that allows for the computation of distance bounds at multiple levels of resolution; for ease of presentation, we opted for the Discrete Haar Wavelet Transform. Using pre-computed information, we provide *double* (i.e., not only lower, but also upper) bounds, *tighter* than those calculated in previous works. We utilize these bounds in a nontraditional, efficient gradual pruning of candidates facilitated by a *vertical* storage scheme where features are stored grouped by *level of resolution*. Thus,

we build a novel solution to the problem. Our experimental study shows that our method has an advantage over the state-of-the-art *i*SAX method and classical index-based methods.

Acknowledgments

We are grateful to Eamonn Keogh, who provided us with the *i*SAX code, data sets, and data generators, and to Lei Chen, who shared with us codes for the classical methods. We also thank Stavros Papadopoulos and Yin Yang for occasional musings on this topic.

6. REFERENCES

[1] *i*SAX page. http://www.cs.ucr.edu/ eamonn/iSAX/iSAX.html.
[2] R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient similarity search in sequence databases. In *FODO*, 1993.
[3] S. Arya, T. Malamatos, and D. M. Mount. Space-time tradeoffs for approximate nearest neighbor searching. *J. ACM*, 57(1), 2009.
[4] I. Assent, R. Krieger, F. Afschari, and T. Seidl. The TS-tree: efficient time series search and retrieval. In *EDBT*, 2008.
[5] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *SIGMOD*, 1990.
[6] S. Berchtold, C. Böhm, H. V. Jagadish, H.-P. Kriegel, and J. Sander. Independent quantization: An index compression technique for high-dimensional data spaces. In *ICDE*, 2000.
[7] S. Berchtold, C. Böhm, and H.-P. Kriegal. The pyramid-technique: towards breaking the curse of dimensionality. In *SIGMOD*, 1998.
[8] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. In *VLDB*, 1996.
[9] S. Blott and R. Weber. What's wrong with high-dimensional similarity search? *PVLDB*, 1(1):3–3, 2008.
[10] C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373, 2001.
[11] Y. Cai and R. Ng. Indexing spatio-temporal trajectories with Chebyshev polynomials. In *SIGMOD*, 2004.
[12] K. Chakrabarti, E. Keogh, S. Mehrotra, and M. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM TODS*, 27(2):188–228, 2002.
[13] K. Chakrabarti and S. Mehrotra. The hybrid tree: An index structure for high dimensional feature spaces. In *ICDE*, 1999.
[14] K.-P. Chan and A. W.-C. Fu. Efficient time series matching by wavelets. In *ICDE*, 1999.
[15] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *SIGMOD*, 2005.
[16] Q. Chen, L. Chen, X. Lian, Y. Liu, and J. X. Yu. Indexable PLA for efficient similarity search. In *VLDB*, 2007.
[17] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB*, 1997.
[18] B. Cui, B. C. Ooi, J. Su, and K.-L. Tan. Indexing high-dimensional data for efficient in-memory similarity search. *IEEE TKDE*, 17(3):339–353, 2005.
[19] A. P. de Vries, N. Mamoulis, N. Nes, and M. Kersten. Efficient kNN search on vertically decomposed data. In *SIGMOD*, 2002.
[20] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. Querying and mining of time series data: experimental comparison of representations and distance measures. *PVLDB*, 1(2), 2008.
[21] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD*, 1994.
[22] P. Geurts. Pattern extraction for time series classification. In *PKDD*, 2001.
[23] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, 1999.
[24] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD*, 1984.
[25] J. M. Hellerstein, E. Koutsoupias, and C. H. Papadimitriou. On the analysis of indexing schemes. In *PODS*, 1997.
[26] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM TODS*, 24(2):265–318, 1999.
[27] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, 1998.
[28] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang. iDistance: An adaptive B^+-tree-based indexing method for nearest neighbor search. *ACM TODS*, 30(2):364–397, 2005.
[29] K. V. R. Kanth, D. Agrawal, and A. Singh. Dimensionality reduction for similarity searching in dynamic databases. In *SIGMOD*, 1998.
[30] P. Karras and N. Mamoulis. One-pass wavelet synopses for maximum-error metrics. In *VLDB*, 2005.
[31] P. Karras and N. Mamoulis. The Haar$^+$ tree: a refined synopsis data structure. In *ICDE*, 2007.
[32] N. Katayama and S. Satoh. The SR-tree: an index structure for high-dimensional nearest neighbor queries. In *SIGMOD*, 1997.
[33] E. J. Keogh, K. Chakrabarti, M. J. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowl. Inf. Syst.*, 3(3):263–286, 2001.
[34] F. Korn, H. V. Jagadish, and C. Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. In *SIGMOD*, 1997.
[35] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. Fast nearest neighbor search in medical image databases. In *VLDB*, 1996.
[36] A. Koski, M. Juhola, and M. Meriste. Syntactic recognition of ECG signals by attributed finite automata. *Pattern Recognition*, 28(12):1927–1940, 1995.
[37] J.-H. Lee, D.-H. Kim, and C.-W. Chung. Multi-dimensional selectivity estimation using compressed histogram information. In *SIGMOD*, 1999.
[38] X. Lian, L. Chen, J. X. Yu, J. Han, and J. Ma. Multiscale representations for fast pattern matching in stream time series. *IEEE TKDE*, 21(4):568–581, 2009.
[39] J. Lin, E. Keogh, L. Wei, and S. Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Min. Knowl. Discov.*, 15(2):107–144, 2007.
[40] K. I. Lin, H. V. Jagadish, and C. Faloutsos. The TV-tree: an index structure for high-dimensional data. *The VLDB Journal*, 3(4):517–542, 1994.
[41] I. Popivanov and R. J. Miller. Similarity search over time-series data using wavelets. In *ICDE*, 2002.
[42] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *SIGMOD*, 1995.
[43] Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima. The A-tree: An index structure for high-dimensional spaces using relative approximation. In *VLDB*, 2000.
[44] T. Seidl and H.-P. Kriegel. Optimal multi-step k-nearest neighbor search. In *SIGMOD*, 1998.
[45] T. K. Sellis, N. Roussopoulos, and C. Faloutsos. The R^+-tree: A dynamic index for multi-dimensional objects. In *VLDB*, 1987.
[46] J. Shieh and E. Keogh. *i*SAX: indexing and mining terabyte sized time series. In *KDD*, 2008.
[47] H. Sun, Ö. Öztürk, and H. Ferhatosmanoğlu. CoMRI: A compressed multi-resolution index structure for sequence similarity queries. In *CSB*, 2003.
[48] Y. Tao, K. Yi, C. Sheng, and P. Kalnis. Quality and efficiency in high dimensional nearest neighbor search. In *SIGMOD*, 2009.
[49] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, 1998.
[50] D. Wu, A. Singh, D. Agrawal, A. El Abbadi, and T. R. Smith. Efficient retrieval for browsing large image databases. In *CIKM*, 1996.
[51] H. Wu, B. Salzberg, and D. Zhang. Online event-driven subsequence matching over financial data streams. In *SIGMOD*, 2004.
[52] M.-Y. Yeh, K.-L. Wu, P. S. Yu, and M.-S. Chen. LEEWAVE: Level-wise distribution of wavelet coefficients for processing kNN queries over distributed streams. In *VLDB*, 2008.
[53] B.-K. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary L_p norms. In *VLDB*, 2000.
[54] C. Yu, B. C. Ooi, K.-L. Tan, and H. V. Jagadish. Indexing the distance: An efficient method to kNN processing. In *VLDB*, 2001.