LEEWAVE: Level-Wise Distribution of Wavelet Coefficients for Processing kNN Queries over Distributed Streams

Mi-Yen Yeh[†,‡] Kun-Lung Wu[‡] Philip S. Yu[§] Ming-Syan Chen[†]

[†]Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan
[‡]IBM T.J. Watson Research Center, Hawthorne, NY
[§]Department of Computer Science, University of Illinois at Chicago, Chicago, IL

miyen@arbor.ee.ntu.edu.tw, klwu@us.ibm.com, psyu@cs.uic.edu, mschen@cc.ee.ntu.edu.tw

ABSTRACT

We present LEEWAVE − a bandwidth-efficient approach to searching range-specified k-nearest neighbors among distributed streams by LEvEl-wise distribution of WAVElet coefficients. To find the k most similar streams to a range-specified reference one, the relevant wavelet coefficients of the reference stream can be sent to the peer sites to compute the similarities. However, bandwidth can be unnecessarily wasted if the entire relevant coefficients are sent simultaneously. Instead, we present a level-wise approach by leveraging the multi-resolution property of the wavelet coefficients. Starting from the top and moving down one level at a time, the query initiator sends only the single-level coefficients to a progressively shrinking set of candidates. However, there is one difficult challenge in LEEWAVE: how does the query initiator prune the candidates without knowing all the relevant coefficients? To overcome this challenge, we derive and maintain a *similarity range* for each candidate and gradually tighten the bounds of this range as we move from one level to the next. The increasingly tightened similarity ranges enable the query initiator to effectively prune the candidates without causing any false dismissal. Extensive experiments with real and synthetic data show that, when compared with prior approaches, LEEWAVE uses significantly less bandwidth under a wide range of conditions.

1. INTRODUCTION

Processing data streams has become increasingly important as more and more emerging applications are required to handle a large amount of data in the form of rapidly arriving streams. Examples include data analysis in sensor networks, program trading in financial markets, video surveillance and weather forecasting. In response, many organizations [1, 3, 5, 9, 30, 32] have started developing data stream processing systems (DSPS).

Finding k-nearest neighbors (kNN) is one of the most common applications in computing. Processing kNN queries has been one of the most studied problems in traditional non-streaming database research. It is also believed to be the case in data stream processing. For a kNN query, the DSPS will find the k streams that have more similar patterns than others to a given pattern contained in a reference stream. Compared to kNN query processing in traditional databases, stream-based kNN query processing is much more challenging. It must handle an endlessly growing amount of data with limited resources. Nevertheless, many researchers have started working on various aspects of stream-based kNN query processing [13, 17, 19, 21]. But, these works mainly focus on the case where data streams are collected and processed at a central site.

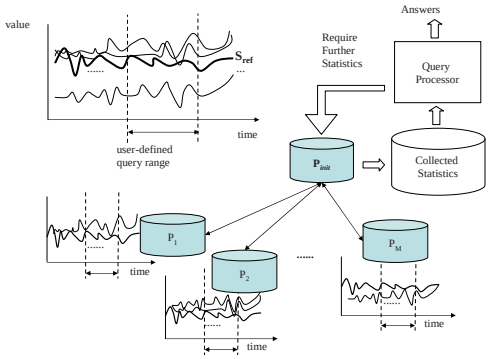

Figure 1: System model.

In many real-world applications, however, data streams are usually collected in a decentralized manner. For example, to forecast the weather and track global climate changes, meteorologists collect streams of measurements, like temperatures, from observation stations located over a wide area. In surveillance, video cameras are set up in many places and continuously capture images from various angles. Finally, readings from a sensor network are collected in a distributed fashion. In these cases, it is inefficient to gather all of the distributed streams to a central site before doing any query processing. It is even impossible to do so when the available network bandwidth is limited. Hence, there is a need to develop a bandwidth-efficient approach to processing kNN queries among distributed streams.

In this paper, we study the problem of processing distributed kNN (k-similarity) queries. The system model is shown in Fig 1, where there are M distributed sites, each monitoring one or more streams. These sites can communicate with each other via a communication network. Given a reference stream S_{ref} maintained by an initiator site, P_{init}, the goal is to find the k streams among all M sites with the highest similarities to S_{ref} than other streams in

PVLDB '08, August 23-28, 2008, Auckland, New Zealand

the user-defined time range T.

An obvious solution to distributed kNN query processing is to transmit to all other sites all the relevant data of S_{ref} in the specified time range, referred to as the query object. After receiving the query object from P_{init}, each peer site computes the similarities between locally maintained streams and S_{ref}, and then reports its local kNN streams to P_{init}. Finally, P_{init} determines the true kNN after it receives results from all the peer sites. This kind of scheme was called Concurrent Processing (CP) in [26]. Unfortunately, concurrent processing is not a good one because it requires large bandwidth (size of the query stream multiplied by $(M-1)$ plus k candidate objects from each of the $M-1$ sites) and a significant part of it can be unnecessarily wasted, especially if $M \gg k$ or if T is large. To save the bandwidth, the authors in [26] further suggested a probabilistic processing (PRP) method, which reduces the amount of data transmitted back from all other peer sites. Similar to CP, PRP first sends the query object to all other sites as well. However, it requests only $k/M+1$ similar objects from each local site. Formulating the current set of best matches, the query initiator further asks objects from sites with possible candidates. PRP indeed saves bandwidth in returning candidate objects; however, sending the entire query object initially to a large number of sites still consumes huge bandwidth.

Searching for a better solution, we notice that summary sketches, instead of complete details, of the streams are usually maintained in a data streaming environment. Among various sketches, the wavelet-based one, especially the Haar wavelet summarization, has been widely adopted in many stream-oriented applications due to its efficiency and simplicity [6, 31, 34]. More importantly, the Haar wavelet decomposition provides multiple resolutions in time and frequency domains like all other wavelet decompositions do. In a coarser resolution, there are fewer wavelet coefficients, each representing a longer subsection of the original data; while in a finer resolution, there are more wavelet coefficients, but each represents a smaller subsection of the data. More specifically, coefficients in a lower resolution give a rough outline of the original data, while those in a higher resolution disclose more details. This gives us inspiration that we can use fewer coefficients in a coarser resolution to filter out as many candidates as possible and then use more coefficients in a finer resolution to refine the answers. The multi-resolution property of wavelets has also been exploited by many other applications. For example, using haar wavelet coefficients to index images, the authors in [2] showed that it saved computation time while still providing good results when answering queries using approximated images in coarser resolutions. Similarly, when considering clustering of time series, the authors in [20] also leverages the multi-resolution property of wavelets to avoid local minima problem and save computation time. The coarser representations are used to decide initial cluster centers while the finer ones are used to get final results.

Armed with this insight, we present LEEWAVE $-$ a bandwidth-efficient approach to processing kNN queries in a distributed streaming environment by LEvEl-wise distribution of WAVElet coefficients. In essence, instead of simultaneously distributing all the relevant coefficients of the reference stream to other peer sites, the query initiator, P_{init}, sends the coefficients one level at a time, starting from the top (the coarsest) level. At each step, with returned level distances from the peer sites, P_{init} progressively prunes the candidates. As we progress to a lower level, which usually contains more coefficients, the number of candidates becomes much smaller. As a result, significant bandwidth savings can be realized because the wavelet coefficients at a lower level are sent to a much smaller set of candidate sites. More importantly, during the process of candidate pruning, we guarantee that there is no false dismissal[1] which is not provided in previous work on multi-resolution indexing [2].

However, there is one difficult challenge in LEEWAVE: At each step, how does P_{init} prune the candidate streams without knowing all the relevant wavelet coefficients? In fact, it is impossible at an intermediate level to compute the true similarities. Without them, it is difficult, if not impossible, to prune the candidates. To address this challenge, we derive and maintain a *similarity range* for each candidate stream. The upper and lower bounds of a similarity range can be incrementally updated at P_{init} with level-wise distances returned from the peer sites. More importantly, a similarity range gradually becomes tighter as we move from one level to the next. These increasingly tightened similarity ranges enable P_{init} to effectively prune the candidate streams without causing any false dismissal.

To evaluate the effectiveness of LEEWAVE, we conduct extensive experiments using both real and synthetic data. For comparisons, we also implement the CP and PRP approaches. We measure the total bandwidth consumed in finding the k most similar streams to a reference one. The results show that, under a wide range of conditions, LEEWAVE consumes significantly less bandwidth, especially when $M \gg k$ or T is large.

Our contributions can be summarized as follows:

- We introduce LEEWAVE as a bandwidth-efficient approach to processing kNN queries in a distributed streaming environment by level-wise distribution of wavelet coefficients. The relevant coefficients of the reference stream are sent to a progressively shrinking set of candidate streams/sites one level at a time.
- To enable the query initiator to prune the candidate streams without any false dismissal, we derive and maintain a similarity range based on wavelet coefficients for each candidate stream. The similarity range is increasingly tightened by incorporating level-wise distances computed and returned by a peer site.
- We conduct extensive experimental studies to evaluate LEEWAVE. The results show that, when compared with prior approaches, LEEWAVE uses significantly less bandwidth under a wide range of conditions.

The remainder of this paper is organized as follows. Related work is discussed in Section 2. Preliminaries are given in Section 3, including wavelet decomposition and coefficient maintenance. The application of LEEWAVE to the processing of distributed kNN queries is described in Section 4. Section 5 shows the experimental results. Finally, the paper is concluded in Section 6.

2. RELATED WORK

Wavelet transform plays an important role in the field of time series analysis [27]. Although there are many other data decomposition methods such as discrete Fourier transform, singular value decomposition, piecewise linear approximation and so on, none of them provides the multiple resolutions in both time and frequency domain. The property of multi-resolution and the feasibility of online updating provides various options for maintaining synopses in

[1]Note that not all the wavelet coefficients are retained in a steaming environment. As a result, the accuracy of our scheme is based only on the retained wavelet coefficients. Namely, if there is an error in the kNN w.r.t. the raw data, the error is solely due to the discarded coefficients. Our scheme does not introduce any extra error.

the streaming environment. Bulut et al. provided a wavelet-based index structure which incrementally summarizes data in multiple resolutions [6]. Recent data are sketched using finer resolutions while obsolete ones are sketched using coarser resolutions. These indexes can then be used to answer point queries, range queries and inner product queries. Zhu et al. considered burst detection using a summary structure called SWT, which is a shifted-wavelet tree based on the Haar wavelet transform [34]. Teng et al. applied the Haar wavelet transform concept to discover frequent temporal patterns of data streams [31].

Searching k-nearest neighbors is an important research topic in a streaming environment [13, 17, 19, 21]. In [13], the authors proposed to continuously retrieve the latest L points of a stream as a query pattern and then find its nearest neighbors from a time series database. Base on traditional indexing methods, the proposed scheme achieves efficient query response via prefetching. In [19], given an error bound, approximate k-nearest neighbors are searched among stream snapshots. In [21], the authors proposed a new indexing technique based on scalar quantization to provide efficient nearest-neighbor search among multiple streams. In [17], based on the Haar wavelet synopses, the authors provided an efficient approach to finding the k-nearest neighbors under an arbitrary range constraint. All these works assume that streams are collected and processed at a central site.

However, in practice, most streams are generated in geographically distributed places. Therefore, more and more research works have started to focus on distributed streams. These works include finding recently frequent itemsets [23], top-k monitoring [4], tracking approximate quantiles [11], processing aggregation and thresholding queries [22, 25, 28], content-based indexing for inner product queries and similarity queries [7], and so forth. To the best of our knowledge, there is no prior work on processing kNN queries among multiple distributed streams.

In a distributed environment, among various kind of kNN queries and without considering the streaming environment, the work in [26] analyzed four schemes to tackle the k nearest neighbor queries, which are most similar to our problem. Among them, the Probabilistic Processing (PRP) claimed to be an optimistic search by the authors. In PRP, the query initiator (primary server) first submits the query object to all other sites. Then, only $k/M + 1$, where M is the total number of sites, nearest candidates of each peer site need to be sent back. From the current set of best matches, the initiator further requests objects from possible sites. The PRP method reduces the bandwidth consumption by retrieving as fewer candidates from peer sites as possible. However, sending the entire query object to all other sites initially still needs too much bandwidth, especially when the size of the query object or the total number of sites is large.

In [10], the concept of multi-resolution indexing is employed to reduce the computation for shape image retrieval. Instead of considering all vantage points of images, the authors computed an approximate distance range for each candidate image using a smaller number of vantage points. From the approximate distance ranges, a set of candidate images were chosen to compute the true distances with all the vantage points of the chosen candidate images. However, their scheme is different from LEEWAVE in many respects. First, they did not use wavelet coefficients as LEEWAVE does. Second, they did not progressively tighten their distance ranges one level at a time as LEEWAVE does.

3. PRELIMINARIES

3.1 Wavelet decomposition

Table 1: Haar wavelet decomposition.

	averages	wavelet coefficients
raw data	$\{4, 6, 7, 4, 8, 6, 5, 7\}$	-
high resolution	$\{5, 5.5, 7, 6\}$	$\{-1, 1.5, 1, -1\}$
mid resolution	$\{5.25, 6.5\}$	$\{-0.25, 0.5\}$
low resolution	$\{5.875\}$	$\{-0.625\}$

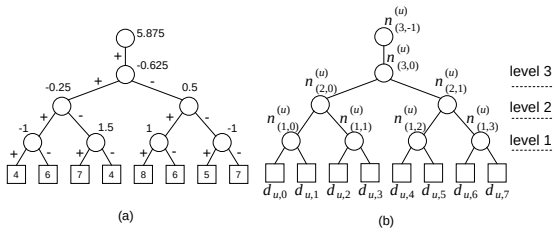

Figure 2: (a) The error tree for Example 1; (b) The notation of an error tree proposed in [17].

Among different wavelet transformations, the Haar wavelet [16] decomposition is the first and also the most popular one. It is achieved by averaging two adjacent data values of a sequence of data at different time resolutions. Then, only the overall average and differences are kept. We show a simple example here in Table 1 to illustrate the idea of the Haar wavelet decomposition. Assume the original data are $\{4, 6, 7, 4, 8, 6, 5, 7\}$. The first pairwise averages are $\{(4 + 6)/2 = 5, (7 + 4)/2 = 5.5, (8 + 6)/2 = 7, (5 + 7)/2 = 6\}$. Then the corresponding difference can be obtained: $\{4-5 = -1, 7-5.5 = 1.5, 8-7 = 1, 5-6 = -1\}$. Based on the data of this resolution, a lower resolution can be obtained in a similar way as described above. Finally, the wavelet coefficients containing the overall average and the difference values of each resolution are $\{5.875, -0.625, -0.25, 0.5, -1, 1.5, 1, -1\}$.

To better illustrate the Haar wavelet decomposition, a widely used data structure called *error tree* is proposed in [24]. The error tree for Example 1 is shown in Fig. 2(a). This tree is composed of wavelet coefficients as nodes and signs as edges. The root of this tree is the overall average and all the other non-leaf nodes are differences at various resolutions. The leaf nodes represent the raw data, but these raw data are not maintained. Instead, one can always reconstruct the raw data by tracing the error tree. Along each path from the root, each data value in a leaf node is equal to the sum of the value of a node multiplied by the sign below it. For example, the fifth data value, 8, can be reconstructed by $+5.875 - (-0.625) + 0.5 + 1 = 8$.

We also use an error tree to illustrate our idea, the same as the one used in [17], which is shown in Fig. 2(b). Here, each non-leaf node is labeled with an identifier with two attributes as subscripts: *level* and *placement*. A node with a label of $n^{(u)}_{(l,p)}$ means that it is in the p-th placement of level l in the error tree corresponding to stream S_u. This notation can be efficiently maintained when data keep steaming in. Moreover, when not all wavelet coefficients are retained, we can easily find the relative positions of the retained coefficients in the error tree via the node labels.

3.2 Coefficient maintenance

Generally, not all the wavelet coefficients in an error tree are retained because the data volume tends to be huge and the memory space is limited in a streaming environment. To meet different error requirements between the raw data and the retained coefficients, many on-line approaches to selecting wavelet synopses have been

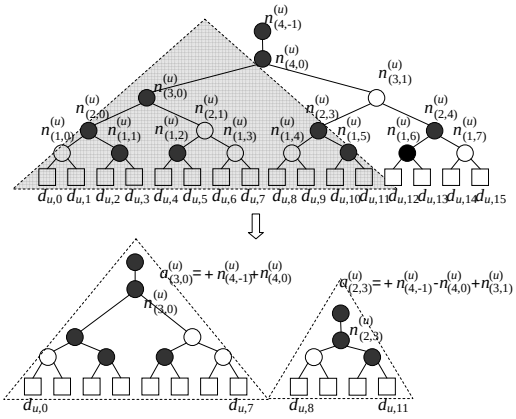

Figure 3: Extracting complete error subtrees and relevant coefficients from the whole error tree based on a desired time range.

proposed. These requirements include minimizing the L^2-norm average error [14], minimizing the maximum absolute/relative error [18], minimizing the weighted L^p-norm error [15], and providing a guaranteed accuracy [12], to name a few.

Given the retained coefficients of a stream, we can extract the relevant coefficients within any time range $[t_s,\ t_e]$. As suggested in [17], the extraction procedure costs a time complexity of $O(\log^2 N)$, where N is the total number of data values in a stream. The extration method is outlined as follows. When a time range $[t_s, t_e]$ is given, it first decomposes the range into several subranges where each of them corresponds to a complete error subtree. Using Fig. 3 as an example, suppose the given range is $[t_0, t_{11}]$, which contains the shaded triangular area in Fig. 3. We can decompose it into two complete error subtrees where one covers $[t_0, t_7]$ and the other $[t_8, t_{11}]$. For each complete error subtree, the new average node will be computed by traversing from the original root node $n^{(u)}_{(4,-1)}$ to the root of the subtree. In Fig. 3, the black nodes represent retained coefficients, while the white ones are those being discarded. When traversing the path to get the new average, the missing nodes are just treated as zero. For example, the new average node $a^{(u)}_{(2,3)}$ equals to $n^{(u)}_{(4,-1)} - n^{(u)}_{(4,0)} + n^{(u)}_{(3,1)}$, where in fact $n^{(u)}_{(3,1)}$ is a missing node and just be treated as zero.

4. THE LEEWAVE APPROACH TO PROCESSING DISTRIBUTED KNN QUERIES

Based on the maintained wavelet synopses, the goal of our kNN query is to find the k most similar streams to S_{ref}. We denote a kNN query as $Q(S_{ref}, k, t_s, t_e)$, where k is the desired number of most similar streams, and $[t_s,\ t_e]$ defines the time range of interest. For the similarity measure between two streams, we adopt the commonly used Euclidean distance in this paper.

4.1 Computing similarities using wavelet coefficients

Given wavelet coefficients of two streams, we can compute the Euclidean distance directly from the coefficients themselves without doing inverse wavelet transform back to the original data [8]. By reformulating the distance computation proposed in [17], for a given time range T=[t_s, t_e], the distance between two streams S_u and S_v can be computed as follows:

$$dst(S_u, S_v)|_{t_s}^{t_e} = [\sum_{T_i} \sum_{n^{(u)}_{(l,p)} \text{ or } n^{(v)}_{(l,p)} \in C_{T_i}} D^{(u,v)}_{(l,p)} \times 2^l]^{1/2}, \quad (1)$$

where

$$D^{(u,v)}_{(l,p)} = \begin{cases} [n^{(u)}_{(l,p)} - n^{(v)}_{(l,p)}]^2 & \text{if } n^{(u)}_{(l,p)} \in C_{T_i} \ \&\ n^{(v)}_{(l,p)} \in C_{T_i}, \\ [n^{(u)}_{(l,p)}]^2 & \text{if } n^{(u)}_{(l,p)} \in C_{T_i} \ \&\ n^{(v)}_{(l,p)} \notin C_{T_i}, \\ [n^{(v)}_{(l,p)}]^2 & \text{if } n^{(u)}_{(l,p)} \notin C_{T_i} \ \&\ n^{(v)}_{(l,p)} \in C_{T_i}, \end{cases}$$

T_i is one of the subrange in T with a complete error subtree, and C_{T_i} is the set of retained coefficients which are in the subrange T_i.

Eq. (1) computes the distance between two streams based on the complete error subtrees. Unfortunately, this error-subtree-based distance computation is not useful in LEEWAVE because peer sites only receive the retained coefficients of S_{ref} one level at a time. We need a level-wise approach to computing the distance for LEEWAVE.

To better illustrate our idea, we introduce three definitions.

DEFINITION 1. *We define the square sum of Euclidean distance between stream S_u and S_v as $Dst(S_u, S_v)$. Namely,*

$$Dst(S_u, S_v) = dst(S_u, S_v)^2.$$

Also, for ease of exposition, when we use the word "distance" from now on, we mean this square sum of Euclidean distance, $Dst(u, v)$.

DEFINITION 2. *We define the* level-l distance *between streams S_u and S_v as the distance of retained coefficients at level l.*

$$Dst^l(S_u, S_v)|_{t_s}^{t_e} = \sum_p D^{(u,v)}_{(l,p)} \times 2^l,$$

where $D^{(u,v)}_{(l,p)}$ satisfies the same condition as in Eq.(1)*.*

DEFINITION 3. *We define the accumulated distance between streams S_u and S_v from the highest level L down to level ρ (the top-down accumulated distance) as:*

$$accDst^\rho(S_u, S_v)|_{t_s}^{t_e} = \sum_{l=\rho}^{L} Dst^l(u, v)|_{t_s}^{t_e}.$$

If we combine Definitions 2 and 3, and assume that the height of the whole retained error tree is L, then Eq.(1) can be reformulated as:

$$\begin{aligned} Dst(S_u, S_v)|_{t_s}^{t_e} &= dst(S_u, S_v)|_{t_s}^{t_e\,2} \\ &= \sum_p D^{(u,v)}_{(1,p)} \times 2^1 + ... + \sum_p D^{(u,v)}_{(L,p)} \times 2^L \\ &= \sum_{l=1}^{L} \sum_p D^{(u,v)}_{(1,p)} \times 2^l \\ &= \sum_{l=1}^{L} Dst^l(u, v)|_{t_s}^{t_e} \\ &= accDst^1(u, v)|_{t_s}^{t_e} \quad (2) \end{aligned}$$

Eq. (2) is important to LEEWAVE as it suggests a level-wise approach to computing the distance between two streams. LEEWAVE will use Eq. (2) to derive a similarity range for each candidate stream and gradually tighten this range in order to prune the

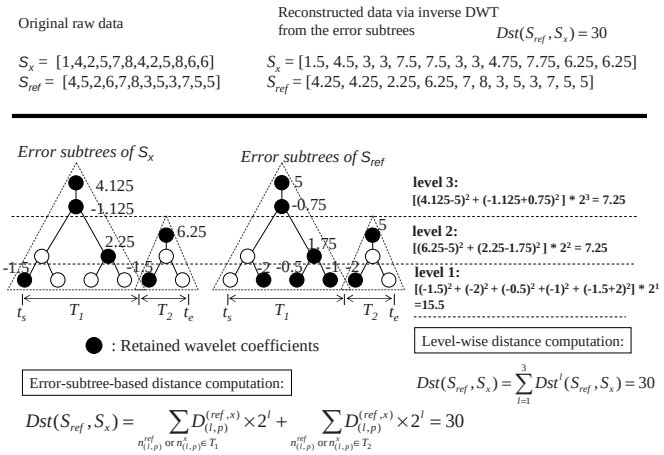

Figure 4: Example of level-wise vs. error-subtree-based distance computation.

candidates. Before we delve into the details, let us look at an example to understand the subtle, yet crucial, difference between Eq. (1) and (2). Note that, for ease of exposition, we will omit the notation of $[t_s, t_e]$ from now on in our description of the distance between two streams.

Fig. 4 shows two different approaches to computing the distance between two streams, S_{ref} and S_x. Assume that during the time range $[t_s,t_e]$, S_x has 12 raw data values $[1, 4, 2, 5, 7, 8, 4, 2, 5, 8, 6, 6]$ and S_{ref} also has 12 raw data values $[4, 5, 2, 6, 7, 8, 3, 5, 3, 7, 5, 5]$. These raw data are transformed into separate error trees. Then a series of complete error subtrees and retained coefficients are extracted. To verify the correctness, we also show the reconstructed data and use them to compute the distance. However, in practice, we never need to do such reconstruction. The distance computation between two streams is done completely with retained wavelet coefficients.

EXAMPLE 1. Error-subtree-based computation: *Based on Eq.*(1), *the distance between two streams is the summation of the distance between each pair of their corresponding complete error subtrees. For the first pair of error subtrees in range* T_1, *the distance is:* $((4.125-5)^2 + (-1.125+0.75)^2) \times 2^3 + ((2.25-1.75)^2+) \times 2^2 + ((-1.5)^2+(-2)^2+(0.5)^2+(-1)^2) \times 2^1 = 23.25$. *For the second pair of error subtrees in range* T_2, *the distance is:* $(6.25-5)^2 \times 2^2 + (-1.5+2)^2 \times 2^1 = 6.75$. *The final distance* $Dst(S_{ref}, S_x)$ *is* $23.25 + 6.75 = 30$. *If we check the distance from the reconstruction data by inverse DWT, we get exactly the same distance value.* ■

EXAMPLE 2. Level-wise computation: *From Fig. 4, we can also compute the total distance in a level-wise manner. The total distance computed this way is also 30, which is the same as the computed distance in Example 1.* ■

4.2 LEEWAVE for a kNN query

The central idea of LEEWAVE is as follows. The query initiator sends the coefficients of S_{ref} one level at a time. At each step, each candidate peer site reports the level-l distance and other necessary information to the initiator site. The initiator then gradually prunes the candidates with the returned level distances. However, the key challenge is: how does the initiator prune the candidates?

Note that, even with Eq. (2), the query initiator still cannot prune the candidate streams without risking some false dismissal. This is because, at a given level, it does not know how much those not-yet-seen coefficients of the candidate streams at lower levels will contribute to the final distances.

To overcome this problem, we maintain a similarity range into which the exact distance may fall for each candidate stream. To estimate the similarity range, we first decompose Eq.(2) into two parts: one is the accumulated distance so far and the other is the distance from those not-yet-seen coefficients.

$$\begin{aligned} & Dst(S_{ref}, S_x) \\ = & \sum_{l=1}^{L} Dst^l(S_{ref}, S_x) \\ = & accDst^{\rho}(S_{ref}, S_x) + \sum_{l=1}^{\rho-1} Dst^l(S_{ref}, S_x). \end{aligned} \quad (3)$$

At level ρ, the $accDst^{\rho}(S_{ref}, S_x)$ can be easily maintained by P_{init}. However, the second term of Eq. (3) is still unknown. Let us further decompose the second terms:

$$\begin{aligned} & \sum_{l=1}^{\rho-1} Dst^l(S_{ref}, S_x) \\ = & \sum_{l=1}^{\rho-1} \sum_{p} [n^{(ref)}_{(l,p)} - n^{(x)}_{(l,p)}]^2 \times 2^l \\ = & \sum_{l=1}^{\rho-1} \sum_{p} ([n^{(ref)}_{(l,p)}]^2 + [n^{(x)}_{(l,p)}]^2 - 2n^{(ref)}_{(l,p)} n^{(x)}_{(l,p)}) \times 2^l. \end{aligned} \quad (4)$$

In Eq. (4), P_{init} can easily compute the first term using its own coefficients. It can also compute the second term, $\sum_{l=1}^{\rho-1} \sum_p ([n^{(x)}_{(l,p)}]^2 \times 2^l)$, by first receiving $\sum_{l=1}^{L-1} \sum_p ([n^{(x)}_{(l,p)}]^2 \times 2^l)$ at the initial step and then gradually subtracting $\sum_p [n^{(x)}_{(\rho,p)}]^2 \times 2^{\rho}$ from it at each subsequent level ρ. The only problem left is the last term in Eq. (4), which involves the product of coefficients at levels below ρ. Neither P_{init} nor any candidate site can compute this term at the current level ρ. To guarantee no false dismissal, we would like to find a substitute for this term which is an overestimate but can be computed with level-wise coefficients.

Fortunately, we do find such a substitute. According to Cauchy-Schwarz inequality [29], we can find an upper bound of the inner product of two vectors in real space, where this upper bound is the product of the linear square sum of each vector:

$$(\sum_{i=1}^{h} \alpha_i \beta_i)^2 \leq \sum_{i=1}^{h} \alpha_i^2 \times \sum_{i=1}^{h} \beta_i^2, \text{ where } \alpha_i, \beta_i \in \mathbb{R}.$$

Now we if let $\alpha_i = -[n^{(ref)}_{(l,p)}] \times 2^l$, and $\beta_i = [n^{(x)}_{(l,p)}]$, then the last term of Eq. (4) has an upper bound as follows:

$$\begin{aligned} & 2 \times \sqrt{\sum_{l=1}^{\rho-1} \sum_{p} (-[n^{(ref)}_{(l,p)}] \times 2^l)^2 \times \sum_{l=1}^{\rho-1} \sum_{p} [n^{(x)}_{(l,p)}]^2} \\ = & 2 \times \sqrt{\sum_{l=1}^{\rho-1} \sum_{p} ([n^{(ref)}_{(l,p)}] \times 2^l)^2 \times \sum_{l=1}^{\rho-1} \sum_{p} [n^{(x)}_{(l,p)}]^2}. \end{aligned}$$

Procedure: LEEWAVE for a kNN query
Input: P_{init}, S_{ref}, k, $T = [t_s, t_e]$
Output: The k most similar streams to S_{ref}

P_{init}:	*A candidate peer site P_x:*
1. Extract relevant coefficients of S_{ref} in $[t_s,t_e]$.	
2. Send t_s, t_e, and coefficients of S_{ref} at level L to all other $M-1$ sites.	3. For each local candidate stream, S_x, compute and return a 3-tuple ($Dst^L(S_{ref}, S_x)$, $\sum_{l=1}^{L-1}\sum_p[n_{(l,p)}^{(x)}]^2$, $\sum_{l=1}^{L-1}\sum_p([n_{(l,p)}^{(x)}]^2 \times 2^l)$) to P_{init}.
4. Compute the upper and lower bound of the similarity range based on Eq. (5) for each candidate stream. Do the first pruning. Then, sort out a list of candidate sites and streams.	
5. **for** ($\rho = L-1$; (ρ! $= 0$ && !**done**); $\rho = \rho - 1$) {	
6. Send level-ρ coefficients of S_{ref} and the list of candidate streams to each candidate peer site.	7. Compute and return a 2-tuple ($Dst^\rho(S_{ref}, S_x)$, $\sum_p[n_{(\rho,p)}^{(x)}]^2$) for each local candidate stream, S_x.
8. Update the upper and lower bound of the similarity range based on Eq. (5) for each candidate stream. Do pruning. Set **done** to **true** if there are no more than k candidate streams left.	
9. Ask the contents of the final k streams within the query range.	10. Send back corresponding contents.
11. }	

Figure 5: Algorithm for distributed kNN query processing using LEEWAVE.

Therefore, at level ρ, the true distance between S_{ref} and S_x, which is described in Eq.(3), is bounded in the following range:

$$accDst^\rho(S_{ref}, S_x) \le Dst(S_{ref}, S_x) \le$$
$$accDst^\rho(S_{ref}, S_x) + \sum_{l=1}^{\rho-1}\sum_p([n_{(l,p)}^{(ref)}]^2 + [n_{(l,p)}^{(x)}]^2) \times 2^l$$
$$+2 \times \sqrt{\sum_{l=1}^{\rho-1}\sum_p([n_{(l,p)}^{(ref)}] \times 2^l)^2 \times \sum_{l=1}^{\rho-1}\sum_p[n_{(l,p)}^{(x)}]^2}. \quad (5)$$

With Eq. (5), it is now possible to maintain both the lower bound and the upper bound of this similarity range in a level-wise manner with $Dst^\rho(S_{ref}, S_x)$ and $\sum_p[n_{(\rho,p)}^{(x)}]^2$ returned by a peer site at each level ρ. This is because the term $\sum_{l=1}^{\rho-1}\sum_p[n_{(l,p)}^{(x)}]^2$ can now be incrementally computed at P_{init} by subtracting $\sum_p[n_{(l,p)}^{(x)}]^2$ from $\sum_{l=1}^{\rho}\sum_p[n_{(l,p)}^{(x)}]^2$, which was computed at the previous level $\rho+1$. More importantly, as we move from one level to the next, the similarity range becomes tighter. This is because the lower bound of a similarity range is non-decreasing, based on Definition 3, and the upper bound is non-increasing.

THEOREM 1. *The upper bound of a similarity range is non-increasing when we move from level ρ to level $\rho - 1$.*

Proof: See the Appendix. ■

We are now ready to describe the details of how LEEWAVE processes a distributed kNN query in a level-wise manner and how P_{init} gradually prunes the candidates. Fig. 5 shows the algorithm of distributed kNN query processing using LEEWAVE. At the first step, P_{init} extracts the relevant wavelet coefficients of S_{ref} in the range of $[t_s, t_e]$. Then, it sends t_s, t_e and the level-L coefficients of S_{ref} to all other $M-1$ peer sites. Each peer site then extracts relevant coefficients for each stream it monitors. And it returns 3 numbers for each local candidate stream, S_x. They are the level-L distance, $Dst^L(S_{ref}, S_x)$, and two other numbers that will be used by P_{init} to progressively tighten the similarity range:

$$\sum_{l=1}^{L-1}\sum_p[n_{(l,p)}^{(x)}]^2 \quad \text{and} \quad \sum_{l=1}^{L-1}\sum_p([n_{(l,p)}^{(x)}]^2 \times 2^l).$$

After receiving the 3 numbers from each candidate stream, P_{init} updates the lower and upper bounds of a similarity range based on Eq. (5) for each candidate stream. It will then do some initial pruning, if possible, and sort out a list of candidate streams that might be the final k most similar streams. Then, it moves to the next level. For a given level ρ, P_{init} sends the level-ρ coefficients of S_{ref} and the list of candidate streams to a candidate site. A candidate site will compute and return two level-specific numbers: $Dst^\rho(S_{ref}, S_x)$ and $\sum_p[n_{(\rho,p)}^{(x)}]^2$. P_{init} will update the two bounds of a similarity range, making it tighter, with these two level-specific numbers. With increasingly tighter ranges, P_{init} can better prune the candidate list. The algorithm ends when there are no more than k candidate streams left.

To prune, P_{init} first sorts the candidate streams in an ascending order based on the upper bounds of their similarity ranges. Any candidate stream whose similarity lower bound is higher than the upper bound of the k-th streams in the sorted list cannot be the final answer, and thus is pruned. From Theorem 1, we can guarantee that there is no false dismissal under this pruning strategy.

EXAMPLE 3. *In this example, we use a concrete example to demonstrate the increasingly tightened similarity range for a candidate stream. Consider the case in Fig. 4. Table 2 shows the flow of data exchanges between P_{init} and P_x, the values of various terms in Eq.* (5) *maintained by P_{init}, and the similarity range of S_x. Each column shows the level currently in progress.*

When $\rho = 3$, $Dst^3(S_{ref}, S_x) = 7.25$ is obtained as shown in Fig. 4, $\sum_{l=1}^{2}\sum_p[n_{(l,p)}^{(x)}]^2 = ((-1.5)^2 + (-1.5)^2 + 2.25^2 + 6.25^2) = 48.625$, and $\sum_{l=1}^{2}\sum_p[n_{(l,p)}^{(x)}]^2 \times 2^l = (((-1.5)^2 + (-1.5)^2) \times 2^1 + (2.25^2 + 6.25^2) \times 2^2) = 185.5$. The terms $\sum_{l=1}^{\rho-1}\sum_p[n_{(l,p)}^{(ref)} \times 2^l]^2$ and $\sum_{l=1}^{\rho-1}\sum_p[n_{(l,p)}^{(ref)}]^2 \times 2^l$ can be computed by P_{init} similarly. With above values, the first similarity range for stream S_x is $[7.25, 7.25 + 130.75 + 185.5 + 2 * (486 \times$

Table 2: Example of an increasingly tightened similarity range for a candidate stream.

	$\rho = 3$	$\rho = 2$	$\rho = 1$
P_{init} to P_x	$((5, -0.75), t_s, t_e)$	$(1.75, 5)$	$(-2, -0.5, -1, -2)$
P_x to P_{init}	$(7.25, 48.625, 185.5)$	$(7.25, 44.125)$	$(15.5, 4.5)$
$\sum_{l=1}^{\rho-1}\sum_p[n_{(l,p)}^{(ref)} \times 2^l]^2$	486.0	37.0	0
$\sum_{l=1}^{\rho-1}\sum_p[n_{(l,p)}^{(ref)}]^2 \times 2^l$	130.75	18.5	0
$\sum_{l=1}^{\rho-1}\sum_p[n_{(l,p)}^{(x)}]^2$	48.625	4.5	0
$\sum_{l=1}^{\rho-1}\sum_p[n_{(l,p)}^{(x)}]^2 \times 2^l$	185.5	9	0
similarity range for S_x	$[7.25, 630.95]$	$[14.5, 67.81]$	$[30, 30]$

$48.625)^{1/2} = 630.95]$. *This completes the values in the first column.*

When ρ=2, P_x sends back $Dst^2(S_{ref}, S_x) = 7.25$, and $\sum_p[n_{(2,p)}^{(x)}]^2 = (2.25^2 + 6.25^2) = 44.125$. After receiving this 2-tuple, P_{init} then subtracts these two numbers from the maintained terms in the middle rows. Take ρ=2 as an example, $\sum_{l=1}^{2-1}\sum_p[n_{(l,p)}^{(x)}]^2 = 48.625 - 44.125 = 4.5$, and $\sum_{l=1}^{2-1}\sum_p[n_{(l,p)}^{(x)}]^2 \times 2^l = 185.5 - 44.125 \times 2^2 = 9$. For the sum related to coefficients of S_{ref}, P_{init} updates them similarly. Then, the second similarity range for S_x is $[7.25 + 7.25 = 14.5, 14.5 + 18.5 + 9 + 2 * (37 \times 4.5)^{1/2} = 67.81]$.

The last row in Table 2 shows the similarity range of S_x at each level. It clearly shows that the similarity range does indeed become tighter, $[7.25, 630.95]$ to $[14.5, 67.81]$ to $[30, 30]$, as we move from one level to the next. ■

5. PERFORMANCE STUDY

We conducted a series of experiments with both real and synthetic data to evaluate LEEWAVE. We compared LEEWAVE with the CP and PRP approaches. All approaches were implemented in Visual C++ and the experiments were run on a PC with 2.8GHz CPU and 2GB RAM.

We compared the total bandwidth requirements for LEEWAVE with those for the Concurrent Processing (CP) and Probabilistic Processing (PRP) approaches proposed in [26]. We focused on the impacts of query range T, k and the total number of sites M on the bandwidth consumption. The total bandwidth consumption was calculated by adding up the data transmitted from P_{init} to all other candidate peer sites and those transmitted back from each peer candidate site. Each data value sent was counted as one unit of bandwidth. For coefficients in the same level, the total bandwidth was 2 (value and placement index) multiplied by the total number of coefficients at that level plus 1 (level number).

For the CP approach, the total retained coefficients of the reference stream within the query range plus 3 additional values (k, t_s, t_e) were sent to $M-1$ peer sites. The number of data transmitted from the $M-1$ peer sites to P_{init} is decided by the number of candidate streams (at most k) returned. It is the summation of the total returned coefficient size within the query range plus the stream index and the corresponding distances.

For the PRP approach, the bandwidth is computed as the same way as CP does. Only the number of returned candidates is different. Besides, PRP needs to send the current best k-th value to sites which are possible to have candidates and request them back.

For LEEWAVE, we summed up the data transmitted at each level. For the highest level, only the level-L relevant coefficients of the reference stream plus (t_s, t_e) were counted from P_{init} to the $M-1$ peer sites. The number of data values sent back from each of the $M-1$ peer sites for each locally maintained stream was 4, including the stream index, the level-L distance and two other values for pruning. For a subsequent level ρ, P_{init} sends the level-ρ coefficients and the candidate stream list for a candidate site. Then, for each local candidate stream, a candidate site only sends back the level-ρ distance and another data value needed for P_{init} to do the pruning. When the final k neighbors are decided, the size of those requested stream patterns are added in.

It it noted that LEEWAVE is independent of the way wavelet coefficients are chosen. Without loss of generality, here the wavelet coefficients were retained using the method proposed in [14], which retains the B largest coefficients in terms of absolute normalized coefficient values. We randomly picked one stream from our dataset as the reference stream and performed kNN queries using three approaches. Since the total bandwidth used for processing kNN queries depends on the reference stream, we averaged the bandwidth consumption over a few different reference streams for each bandwidth value we reported.

5.1 Experiments with real data

The real data we used here were the daily average temperature data of 300 cities around the world, which were obtained from the temperature data archive of the University of Dayton[2]. The data from each city was regarded as a stream, each of which has $3,416$ data points. In addition, these streams were evenly distributed among the M sites for all the experiments.

The first experiment examined the impacts of k and the number of sites M on bandwidth consumption for a given T=2048. The results are shown in Fig. 6. The value of k was varied from 5 to 30 and M was varied from 10, 30, 60, 100, 150 and 300. Compared with CP and PRP, LEEWAVE saves a significant bandwidth. When M is small, for example 10, the number of streams in a site is always larger than k. This means that CP will always retrieve exactly k candidates from each peer sites. Fig. 6(a) shows that the required bandwidth of CP grows linearly as k increases. On the other hand, as shown in Fig. 6(b), the bandwidth required for PRP also increases but at a slower rate since (k/M+1) grows slowly. However for a bigger M, the bandwidth is insensitive to k for both CP and PRP. For CP, since the number of streams in each site is smaller than k, it always retrieves all of them back. For PRP, when $k \ll M$, $(k/M + 1)$ remains the same value, i.e., 1, for different Ms. In this regard, the bandwidth used for transmitting the query stream to other sites dominates the total bandwidth consumption. The larger the number of sites, the more the bandwidth is consumed. Also, this is the reason why both CP and PRP almost have the same bandwidth consumption when M is 300. On the contrary, instead of sending the entire relevant coefficients of the reference stream to all other sites, LEEWAVE sends as fewer

[2]http://www.engr.udayton.edu/weather/

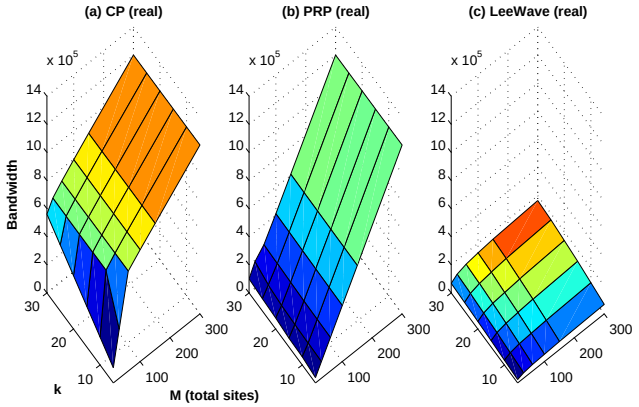

Figure 6: Impacts of k and M on bandwidth consumption with real data.

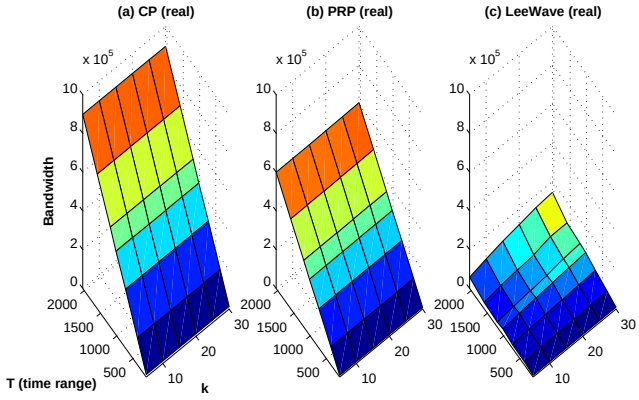

Figure 7: Impacts of T and k on bandwidth consumption with real data.

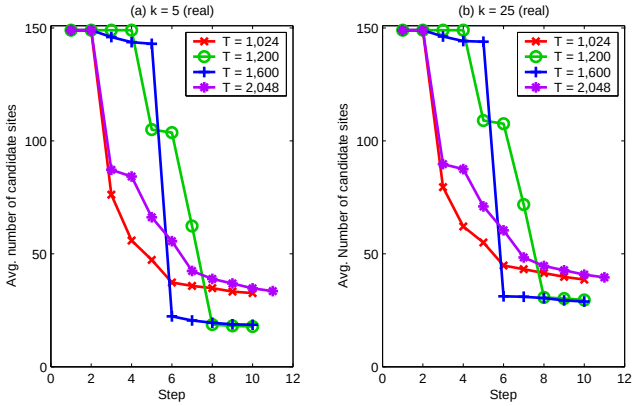

Figure 8: Size of candidate sites at each step of LEEWAVE in Fig. 7.

coefficient as possible to only those candidate sites by leveraging the multi-resolution property of wavelet coefficients. This saves a lot of bandwidth as shown in Fig. 6(c).

The second experiment examined the impacts of query range T and k on bandwidth consumption for a given $M = 150$. The results are shown in Fig. 7. In this experiment, k was varied from 5 to 30 and the query range was varied from the following set of values: 90, 365, 730, 1,024, 1,200, 1,600 and 2,048. From Fig. 7(a) and (b), the bandwidth consumption of the CP and PRP approaches increases significantly as the query range increases, because more relevant coefficients need to be sent to the peer sites. Fig. 7(b) shows that PRP has less bandwidth consumption than CP does. When M equals 150, each site contains only 2 streams on average. Therefore, compare with the CP approach which retrieves all 2 streams for each site, the number, $(k/M + 1) = 1$, allows the PRP approach to save almost half in the bandwidth consumption in returning data. Note that the PRP approach may need to send few more candidates in the second round to make sure all real k stream patterns are obtained. In contrast, LEEWAVE continues to maintain a substantially smaller bandwidth requirement, even as the query range increases. Specifically, for $k = 5$ and $T = 2,048$, the bandwidth requirement of LEEWAVE is only about 6.5% that of CP and 9.7% that of PRP. Considering the impacts of k, from 7(a) and (b), the bandwidth consumption of CP and PRP is not sensitive to k, because it always sends the entire relevant coefficients of the reference stream to all other sites. On the other hand, the bandwidth requirement of LEEWAVE increases slightly as k increases, as shown in Fig. 7(c). This is because it uses the k^{th} lowest upper bound to do pruning. When k is larger, the upper bound is higher, which means the pruning ability becomes less effective.

In addition, from Fig. 7(c) we observe that the 3D surface is not smooth for LEEWAVE, especially along the query-range axis. The reason is as follows. For a different query range, we extracted different series of complete error subtrees, with different heights and subranges. Hence, the relevant retained coefficients might be rather different for different query ranges. Since LEEWAVE computes the distance in a top-down, level-wise fashion, the retained coefficients at different levels under different query ranges have different influences on the pruning effectiveness. To see the details of such impacts, we collected the average number of candidate sites at each step during the query processing in LEEWAVE. In Fig. 8, we plotted the number of candidate sites at each step (level) when k=5 in Fig. 8(a) and k=25 in Fig. 8(b). First we look at the case when $k = 5$. From Fig. 7(c), the bandwidth consumption is higher for $T = 1,024$ (the 4^{th} line along the k-axis) than for $T = 1,200$ (the 5^{th} line along the k-axis.) Then we examine the charts shown in Fig. 8(a). Although the size of candidate sites drops faster for $T = 1,024$ than for $T = 1,200$ at the initial few steps, the reduction is faster at the final few steps (step 8 to 10) for $T = 1,200$ than for $T = 1,024$. Note that there are usually more coefficients retained at the lower levels. Hence, the sizes of candidate sites at the final few steps dominate the total bandwidth consumption. As a result, the total bandwidth is smaller when $T = 1,200$ than when $T = 1,024$. For the case of $k = 25$, the final sizes of candidate sites are closer for both $T = 1,024$ and $1,200$. As a result, the bandwidth drop between these two ranges is less obvious (see Fig. 7(c)). For $T = 2,048$, for both k=5 and k=25 cases, although it generally has a smaller size of candidate sites than others, however, a lot more retained coefficients are involved for a larger range, consuming more bandwidth.

The third experiment, as shown in Fig. 9, examined the impacts of query range T and total number of sites M on bandwidth consumption when k is fixed at 10. The time range settings were the same as those used in the previous experiment. The number of sites was also increased from 10 to 300 as the first experiment did. From Fig. 9(a) and (b), the bandwidth consumption of CP and PRP increases significantly not only as query range T increases, but also as M increases. In contrast, Fig. 9(c) shows that LEEWAVE is much less sensitive to M. This is because unnecessary coefficients are

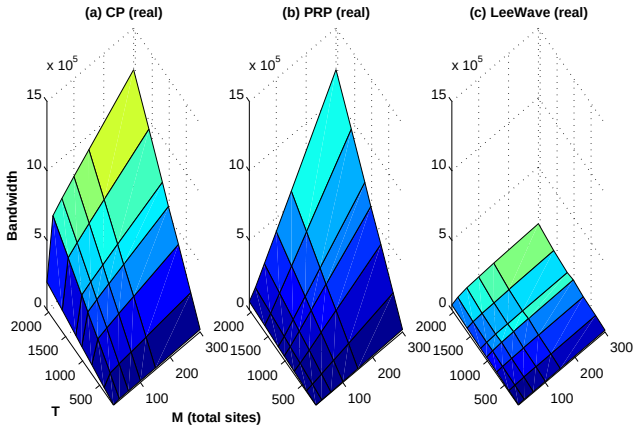

Figure 9: Impacts of T and M on bandwidth consumption with real data.

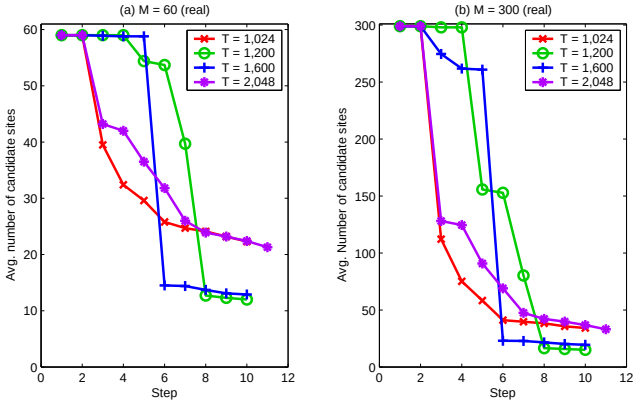

Figure 10: Size of candidate sites at each step of LEEWAVE in Fig. 9.

not distributed by the query initiator. It is noted that there is an obvious bend for CP in Fig. 9(a) for every fixed T when M jumps from 10 to 30. Similar situation also happens at Fig. 6(a). When M is not bigger than 30, each site has more than k=10 streams, therefore both the bandwidth for sending entire relevant coefficients to and retrieving k candidates from all other sites increase. That is the reason why the slope of the curve is steeper. On the other hand, when M is bigger than 30, the number of streams in each site is always less than k, then all streams are always requested by the query initiator. Then only the bandwidth for sending relevant coefficients of the reference stream is increasing, which results in a lower slope for the curve.

For LEEWAVE, the results under different query ranges behave similarly to those from the previous set of experiments. We also show the average number of candidate sites at each step for $M = 60$ and $M = 300$ cases in Fig. 10.

Before the end of this section, we zoom in to smaller values of T and M, on the bandwidth consumption for the three approaches, using the 2-D representation. The results are shown in Fig. 11. The default T is 365, M is 30, and k is 10. From Fig. 11(a), we discover that when the query time range is small, which means a smaller size of relevant coefficients, LEEWAVE may not outperform either CP or PRP. The same is true in Fig. 11(b) when M is not substantially large. We fixed T at 365 and M at 30, and let the value of k vary,

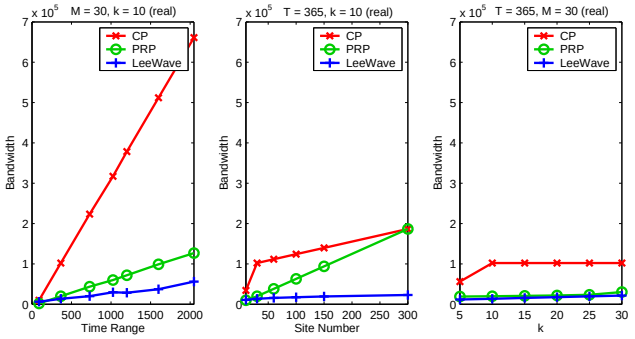

Figure 11: Bandwidth consumption for smaller values of T and M.

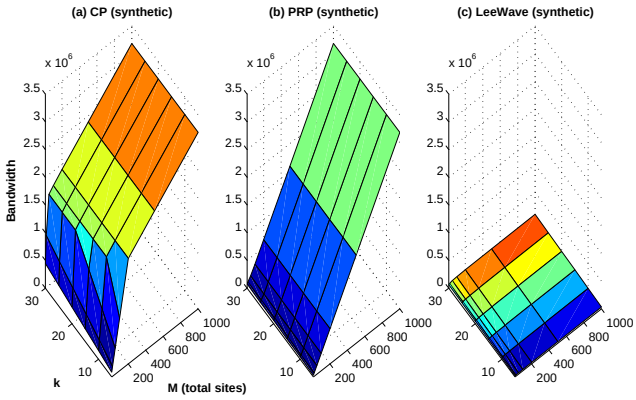

Figure 12: Impacts of k and M on bandwidth consumption with synthetic data.

as shown in Fig. 11(c), to give a better illustration of the former statement. However, when the query range is large, or when the number of distributed sites is large, LEEWAVE indeed is a much better solution in bandwidth savings.

5.2 Evaluation with synthetic data

The synthetic data were generated by a random walk data model proposed in [33]. For a stream S_i, it was generated as follows:

$$S_i = 100 + \sum_{j=1}^{i} (u_j - 0.5),$$

where u_j was randomly picked from [0,1].

We generated 1,000 streams in total, where each stream has 20,000 data points. For a given number of sites, the 1,000 streams were evenly distributed among them.

The first experiment for synthetic dataset studied the impacts of k and the number of sites M on bandwidth consumption for a given T=2048. The results are shown in Fig. 12. The k was varied from 5 to 30 and M was varied from 10, 20, 50, 100, 200, 500 and 1000. We see that LEEWAVE significantly saves bandwidth, when compared with the CP and PRP approaches. Similar to the curves in Fig. 6, when the number of sites is small, the bandwidth usage for both the CP and PRP approaches increases linearly as k increases. CP increases faster than PRP. For a bigger M, the bandwidth is insensitive to k for both CP and PRP.

Note that the query range used here was the same as the one used in Fig. 6. Also, the ratio of retained coefficient number to

Table 3: The bandwidth consumption when k=30, M=100, and T=2048.

	CP	PRP	LEEWAVE
real data	795759.6	411008.2	132315.5
16.6% of CP, 32.2% of PRP			
synthetic data	1738185	321973.8	91588.4
5.3% of CP, 28.4% of PRP			

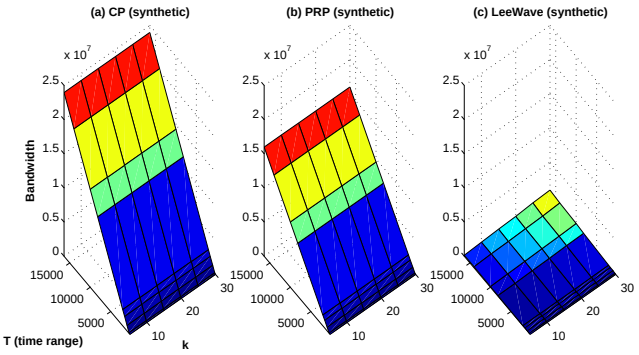

Figure 13: Impacts of T and k on bandwidth consumption with synthetic data.

original data size was also the same. When M=100, T=2048, and k=30, the bandwidth requirements of three approaches for the two data sets are shown in Table. 3. The bandwidth consumption of CP for synthetic data is higher than that for real data. It is because CP retrieves back 10 streams from each site on average for synthetic data compared with 3 streams for real data. For PRP, it requires almost the same bandwidth for both data sets. For LEEWAVE, it requires 16.6% of the bandwidth that is consumed by CP for real data. On the other hand, the ratio is 5.3% for synthetic data. It shows that LEEWAVE saves even more bandwidth for the synthetic dataset compared with CP. This is also true when LEEWAVE is compared with PRP. One reason is due to the nature of the data sets. The synthetic data were generated randomly. Hence, the deviations between streams were much larger than those between temperature streams of different cities. It is easier to separate apart those dissimilar streams in synthetic data set by using only the first few levels of coefficients in distance computation. The other reason is that by pruning candidate streams efficiently, LEEWAVE can quickly decide the final $k = 10$ out of either 300 streams for real data or 1000 streams for synthetic data without sending the entire relevant coefficients of the reference stream to all other sites. This again proves the superiority of LEEWAVE approach.

The second experiment in this section, shown in Fig. 13, studied the impacts of query range T and k on bandwidth consumption for a given $M = 500$. The query range was varied from the following: $300, 512, 3,000, 6,000, 8,192, 10,000, 14,000$ and $16,384$. k was varied from 5 to 30. From Fig. 13, LEEWAVE consumes dramatically less bandwidth, when compared with the CP and PRP approaches. Similar to the previous experiment sets, LEEWAVE saves a lot of bandwidth by quickly pruning candidates. This can be clearly seen in Fig. 14, where the size of candidate sites shrinks quickly after the first few steps. Sometimes the final answers can be obtained at an intermediate level. This is why the size of candidate sites approaches to zero in Fig. 14. When k is larger, the size of candidate sites is higher. This also shows that a higher upper bound has less pruning ability.

Finally, the last experiment, shown in Fig. 15, studied the impacts of the query range T and the number of sites M on bandwidth consumption when k=10. The settings of T were the same as those used in the previous experiment. The value of M was varied from $10, 20, 50, 100, 200, 500$ to $1,000$. From Fig. 15(a) and (b), we observe the same phenomena as in real data. The bandwidth consumption of CP and PRP increases significantly as both query range and the number of sites increase. In contrast, as shown in Fig. 15(c), LEEWAVE saves a huge amount of bandwidth. From Fig. 16, the speed at which the size of candidate sites reduces is faster when $M = 1,000$ than when $M = 200$. This shows that LEEWAVE outperforms the CP and PRP approaches, especially when the number of sites is large.

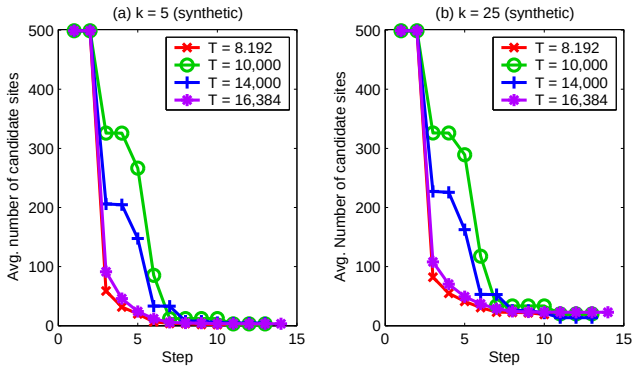

Figure 14: Size of candidate sites at each step of LEEWAVE in Fig. 13.

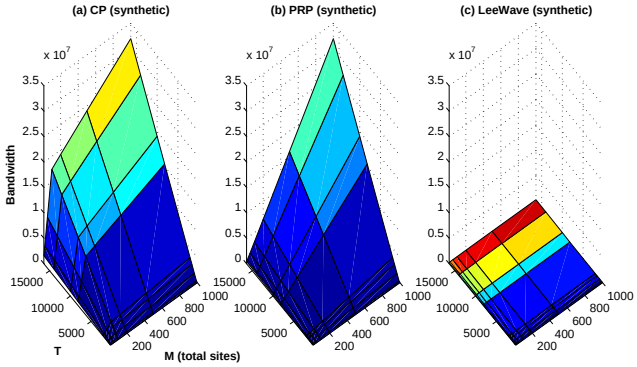

Figure 15: Impacts of T and M on bandwidth consumption with synthetic data.

At the end of this section, we also plot a 2-D plot which focuses on smaller values of T and M. The default T is 1000, M is 50, and k is 10. Fig. 17(a) shows the bandwidth consumption under different M when T is fixed at 1000. Fig 17(b) shows the bandwidth consumption under different T when M is fixed at 50. Fig. 17(c) further shows the results under different k for smaller T and M. As shown in Fig. 17, LEEWAVE may not be a better solution when the time range or when the number of sites is relatively small. However, when the query range is large or when the number of distributed sites is large, LEEWAVE indeed is a significantly better approach in bandwidth savings.

6. CONCLUSION

In this paper, we presented LEEWAVE - a bandwidth-efficient approach to processing range-specified kNN queries in a distributed

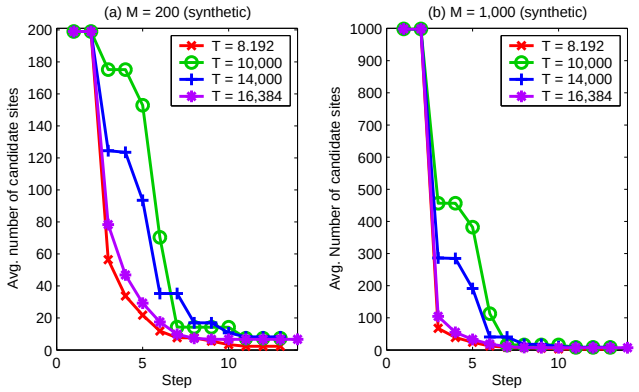

Figure 16: Size of candidate sites at each step of LEEWAVE in Fig. 15.

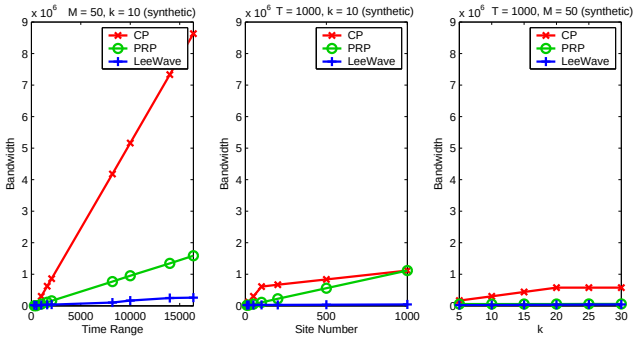

Figure 17: Bandwidth consumption for smaller values of T and M.

streaming environment. Leveraging the multi-resolution property of wavelet coefficients, LEEWAVE distributes the relevant wavelet coefficients to the peer sites in a level-wise fashion. Starting from the top level and moving down one level at a time, the query initiator only sends single-level coefficients to a gradually reduced set of candidate sites. In order to overcome the challenge of pruning the candidates without knowing all the relevant coefficients, we devised and maintained a similarity range for each candidate stream. This similarity range is tightened with each returned level distance. This increasingly tightened similarity range enables the query initiator to effectively prune the candidates. Significant bandwidth savings are achieved by avoiding sending unnecessary coefficients. We conducted extensive experiments with both real and synthetic data. The results show that (1) When compared with the CP and PRP approaches under a wide range of conditions, LEEWAVE uses significantly less bandwidth, especially when query range or the number of sites is large. (2) When the deviations among the streams are large, the performance advantage of LEEWAVE is more significant.

7. REFERENCES

[1] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. The design of the Borealis stream processing engine. In *CIDR*, 2005.

[2] M. G. Albanesi, M. Ferretti, and A. Giancane. A compact wavelet index for retrieval in image database. In *Proc. of IEEE Int. Conf. on Image Analysis and Processing*, 1999.

[3] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, R. Motwani, I. Nishizawa, U. Srivastava, D. Thomas, R. Varma, and J. Widom. STREAM: The Stanford stream data manager. *IEEE Data Engineering Bulletin*, 26, 2003.

[4] B. Babcock and C. Olston. Distributed top-k monitoring. In *ACM SIGMOD*, 2003.

[5] H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, E. Galvez, J. Salz, M. Stonebraker, N. Tatbul, R. Tibbetts, and S. Zdonik. Retrospective on Aurora. *VLDB Journal*, 2004.

[6] A. Bulut and A. K. Singh. SWAT: Hierarchical stream summarization in large networks. In *IEEE ICDE*, 2003.

[7] A. Bulut, R. Vitenberg, and A. K. Singh. Distributed data streams indexing using content-based routing paradigm. In *IPDPS*, 2005.

[8] F. K.-P. Chan, A. W.-C. Fu, and C. Yu. Haar wavelets for efficient similarity search of time-series: With and without time warping. *IEEE TKDE*, 15(3):686–705, 2003.

[9] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, V. Raman, F. Reiss, and M. A. Shah. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.

[10] T.-C. Chiueh, A. Ballman, and K. Kreeger. Multi-resolution indexing for shape images. In *ACM CIKM*, 1998.

[11] G. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi. Holistic aggregates in a networked world: distributed tracking of approximate quantiles. In *ACM SIGMOD*, 2005.

[12] G. Cormode, M. Garofalakis, and D. Sacharidis. Fast approximate wavelet tracking on streams. In *EDBT*, 2006.

[13] L. Gao, Z. Yao, and X. S. Wang. Evaluating continuous nearest neighbor queries for streaming time series via pre-fetching. In *ACM CIKM*, 2002.

[14] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. One-pass wavelet decompositions of data streams. *IEEE TKDE*, 15(3):541–554, 2003.

[15] S. Guha and B. Harb. Wavelet synopsis for data streams: minimizing non-euclidean error. In *ACM SIGKDD*, 2005.

[16] A. Haar. Zur theorie der orthogonalen funktionensysteme. *Mathematische Annalen*, 69, 1910.

[17] H.-P. Hung and M.-S. Chen. Efficient range-constrained similarity search on wavelet synopses over multiple streams. In *ACM CIKM*, 2006.

[18] P. Karras and N. Mamoulis. One-pass wavelet synopses for maximum-error metrics. In *VLDB*, 2005.

[19] N. Koudas, B. C. Ooi, K.-L. Tan, and R. Zhang. Approximate NN queries on streams with guaranteed error/performance bounds. In *VLDB*, 2004.

[20] J. Lin, M. Vlachos, E. J. Keogh, and D. Gunopulos. Iterative incremental clustering of time series. In *EDBT*, 2004.

[21] X. Liu and H. Ferhatosmanoglu. Efficient k-NN search on streaming data series. In *SSTD*, 2003.

[22] A. Manjhi, S. Nath, and P. B. Gibbons. Tributaries and deltas: efficient and robust aggregation in sensor network streams. In *ACM SIGMOD*, 2005.

[23] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. Finding (recently) frequent items in distributed data streams. In *IEEE ICDE*, 2005.

[24] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-based

histograms for selectivity estimation. In *ACM SIGMOD*, 1998.

[25] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *ACM SIGMOD*, 2003.

[26] A. N. Papadopoulos and Y. Manolopoulos. Distributed processing of similarity queries. *Distributed and Parallel Databases*, 9:67–92, 2001.

[27] I. Popivanov and R. J. Miller. Similarity search over time-series data using wavelets. In *IEEE ICDE*, 2002.

[28] I. Sharfman, A. Schuster, and D. Keren. A geometric approach to monitoring threshold functions over distributed data streams. In *ACM SIGMOD*, 2006.

[29] J. M. Steele. *The Cauchy-Schwarz Master Class: An Introduction to the Art of Mathematical Inequalities*. Cambridge University Press, New York, NY, USA, 2004.

[30] StreamBase Systems. http://www.streambase.com/, May 2007.

[31] W.-G. Teng, M.-S. Chen, and P. S. Yu. Resource-aware mining with variable granularities in data streams. In *SIAM Data Mining*, 2004.

[32] K.-L. Wu, P. S. Yu, B. Gedik, K. W. Hildrum, C. C. Aggarwal, E. Bouillet, W. Fan, D. A. George, X. Gu, G. Luo, and H. Wang. Challenges and experience in prototyping a multi-modal stream analytic and monitoring application on system S. In *VLDB*, Sept. 2007.

[33] Y. Zhu and D. Shasha. Statstream: statistical monitoring of thousands of data streams in real time. In *VLDB*, 2002.

[34] Y. Zhu and D. Shasha. Efficient elastic burst detection in data streams. In *ACM SIGKDD*, 2003.

Appendix

Proof of Theorem 1: At level ρ, the upper bound is:

$$accDst^{\rho}(S_{ref}, S_x) + \sum_{l=1}^{\rho-1}\sum_{p}([n_{(l,p)}^{(ref)}]^2 + [n_{(l,p)}^{(x)}]^2) \times 2^l + 2 \times \sqrt{\sum_{l=1}^{\rho-1}\sum_{p}([n_{(l,p)}^{(ref)}] \times 2^l)^2 \times \sum_{l=1}^{\rho-1}\sum_{p}[n_{(l,p)}^{(x)}]^2}. \quad (6)$$

From level ρ to $\rho - 1$, the upper bound is reduced by the amount of:

$$-Dst^{\rho-1}(S_{ref}, S_x) + \sum_{p}([n_{(\rho-1,p)}^{(ref)}]^2 + [n_{(\rho-1,p)}^{(x)}]^2) \times 2^l + 2 \times (\sqrt{\sum_{l=1}^{\rho-1}\sum_{p}([n_{(l,p)}^{(ref)}] \times 2^l)^2 \times \sum_{l=1}^{\rho-1}\sum_{p}[n_{(l,p)}^{(x)}]^2} - \sqrt{\sum_{l=1}^{\rho-2}\sum_{p}([n_{(l,p)}^{(ref)}] \times 2^l)^2 \times \sum_{l=1}^{\rho-2}\sum_{p}[n_{(l,p)}^{(x)}]^2}). \quad (7)$$

To prove the upper bound is non-increasing, we need to prove that Eq. (7) is ≥ 0. For ease of exposition, we let $\alpha_{(l,p)} = [n_{(l,p)}^{(ref)}] \times 2^l$, and $\beta_{(l,p)} = [n_{(l,p)}^{(x)}]$. By expanding the term $Dst^{\rho-1}(S_{ref}, S_x) = \sum_p([n_{(\rho-1,p)}^{(ref)}]^2 + 2n_{(\rho-1,p)}^{(ref)} n_{(\rho-1,p)}^{(x)} + [n_{(\rho-1,p)}^{(x)}]^2) \times 2^l$ and using substitutes, Eq. (7) becomes:

$$2\alpha_{(\rho-1,p)}\beta_{(\rho-1,p)} + 2 \times (\sqrt{\sum_{l=1}^{\rho-1}\sum_{p}\alpha_{(l,p)}^2 \times \sum_{l=1}^{\rho-1}\sum_{p}\beta_{(l,p)}^2} - \sqrt{\sum_{l=1}^{\rho-2}\sum_{p}\alpha_{(l,p)}^2 \times \sum_{l=1}^{\rho-2}\sum_{p}\beta_{(l,p)}^2}). \quad (8)$$

Now the task becomes to prove Eq. (8) ≥ 0. We divide Eq. (8) into two cases, which is when $\alpha_{(\rho-1,p)}\beta_{(\rho-1,p)} \geq 0$ and $\alpha_{(\rho-1,p)}\beta_{(\rho-1,p)} < 0$. If it is the former case, Eq. (8) ≥ 0 must be true. Therefore, we only need to prove the later case. By reformulating Eq. (8) and omitting the factor 2, to prove Eq. (8), we need to prove the following:

$$\sqrt{\sum_{l=1}^{\rho-1}\sum_{p}\alpha_{(l,p)}^2 \times \sum_{l=1}^{\rho-1}\sum_{p}\beta_{(l,p)}^2} \geq -\alpha_{(\rho-1,p)}\beta_{(\rho-1,p)} + \sqrt{\sum_{l=1}^{\rho-2}\sum_{p}\alpha_{(l,p)}^2 \times \sum_{l=1}^{\rho-2}\sum_{p}\beta_{(l,p)}^2} \quad (9)$$

Since both the left-hand side and the right-hand side of Eq. (9) are positive, we can square the terms of both side while the inequality still holds. The square value of left-hand side of Eq. (9) is:

$$\begin{aligned} & \sum_{l=1}^{\rho-1}\sum_{p}\alpha_{(l,p)}^2 \times \sum_{l=1}^{\rho-1}\sum_{p}\beta_{(l,p)}^2 \\ = \; & (\alpha_{(\rho-1,p)}^2 + \sum_{l=1}^{\rho-2}\sum_{p}\alpha_{(l,p)}^2) \times (\beta_{(\rho-1,p)}^2 + \sum_{l=1}^{\rho-2}\sum_{p}\beta_{(l,p)}^2) \\ = \; & \alpha_{(\rho-1,p)}^2\beta_{(\rho-1,p)}^2 + \alpha_{(\rho-1,p)}^2 \sum_{l=1}^{\rho-2}\sum_{p}\beta_{(l,p)}^2 \\ & + \beta_{(\rho-1,p)}^2 \sum_{l=1}^{\rho-2}\sum_{p}\alpha_{(l,p)}^2 + \sum_{l=1}^{\rho-2}\sum_{p}\alpha_{(l,p)}^2 \sum_{l=1}^{\rho-2}\sum_{p}\beta_{(l,p)}^2 \end{aligned} \quad (10)$$

The square value of the right-hand side of Eq. (9) is:

$$\begin{aligned} & \alpha_{(\rho-1,p)}^2\beta_{(\rho-1,p)}^2 \\ & - 2 \times \alpha_{(\rho-1,p)}\beta_{(\rho-1,p)} \sqrt{\sum_{l=1}^{\rho-2}\sum_{p}\alpha_{(l,p)}^2 \times \sum_{l=1}^{\rho-2}\sum_{p}\beta_{(l,p)}^2} \\ & + \sum_{l=1}^{\rho-2}\sum_{p}\alpha_{(l,p)}^2 \sum_{l=1}^{\rho-2}\sum_{p}\beta_{(l,p)}^2. \end{aligned} \quad (11)$$

Compare Eq. (10) with Eq. (11), by eliminating the same terms, we only need to prove

$$\alpha_{(\rho-1,p)}^2 \sum_{l=1}^{\rho-2}\sum_{p}\beta_{(l,p)}^2 + \beta_{(\rho-1,p)}^2 \sum_{l=1}^{\rho-2}\sum_{p}\alpha_{(l,p)}^2 \geq -2 \times \alpha_{(\rho-1,p)}\beta_{(\rho-1,p)} \sqrt{\sum_{l=1}^{\rho-2}\sum_{p}\alpha_{(l,p)}^2 \times \sum_{l=1}^{\rho-2}\sum_{p}\beta_{(l,p)}^2}. \quad (12)$$

By using the inequality of arithmetic and geometric means, Eq. (12) holds, and so does Eq. (9). Q.E.D.