SSH (Sketch, Shingle, & Hash) for Indexing Massive-Scale Time Series

Chen Luo
Department of Computer Science
Rice University
Houston, Texas
cl67@rice.edu

Anshumali Shrivastava
Department of Computer Science
Rice University
Houston, Texas
anshumali@rice.edu

ABSTRACT

Similarity search on time series is a frequent operation in large-scale data-driven applications. Sophisticated similarity measures are standard for time series matching, as they are usually misaligned. Dynamic Time Warping or DTW is the most widely used similarity measure for time series because it combines alignment and matching at the same time. However, the alignment makes DTW slow. To speed up the expensive similarity search with DTW, branch and bound based pruning strategies are adopted. However, branch and bound based pruning are only useful for very short queries (low dimensional time series), and the bounds are quite weak for longer queries. Due to the loose bounds branch and bound pruning strategy boils down to a brute-force search.

To circumvent this issue, we design SSH (Sketch, Shingle, & Hashing), an efficient and approximate hashing scheme which is much faster than the state-of-the-art branch and bound searching technique: the UCR suite. SSH uses a novel combination of sketching, shingling and hashing techniques to produce (probabilistic) indexes which align (near perfectly) with DTW similarity measure. The generated indexes are then used to create hash buckets for sub-linear search. Our results show that SSH is very effective for longer time sequence and prunes around 95% candidates, leading to the massive speedup in search with DTW. Empirical results on two large-scale benchmark time series data show that our proposed method can be around 20 times faster than the state-of-the-art package (UCR suite) without any significant loss in accuracy.

1. INTRODUCTION

Mining for similar or co-related time series is ubiquitous, and one of the most frequent operations, in data driven applications including robotics,medicine [35, 8], speech [38], object detection in vision [49, 46], High Performance Computing (HPC) and system failure diagnosis [32, 47], earth science [34], finance [15], and information retrieval [40] etc.

The focus of this paper is on the problem of similarity search with time series data. A time series X is defined as a sequence of values $X = \{x_1, x_2, ..., x_m\}$ associated with timestamps:$\{t(x_1), t(x_2), ..., t(x_m)\}$ that typically satisfy the relationship $t(x_i) = t(x_{i-1}) + \tau$, where τ is the sampling interval and m is the number of points in the time series. Formally, given a dataset $D = \{X_i | 1 \leq i \leq N\}$ and a query time series Q, we are interested in efficiently computing

$$X^* = \arg\max_{X \in D} S(Q, X), \tag{1}$$

where $S(X, Y)$ is some similarity of interest between time series X and Y. This problem is generally prohibitively expensive for large-scale datasets, especially for latency critical application. We shall concentrate on the computational requirement of this problem.

Finding the right similarity measure for time series is a well-studied problem [39], and the choice of this measure is dependent on the application. It is further well known that while matching time series, it is imperative, for most applications, to first align them before computing the similarity score. Dynamic time warping or DTW is widely accepted as the best similarity measure (or the default measure) over time series, as pointed out in [39]. DTW, unlike L_1 or L_2 distances, takes into account the relative alignment of the time series (see Section 2.1 for details). However, since alignment is computationally expensive, DTW is known to be slow [39].

Owing to the significance of the problem there are flurry of works which try to make similarity search with DTW efficient. The popular line of work use the branch-and-bound technique [13, 25, 24]. Branch and bound methods use bounding strategies to prune less promising candidates early, leading to savings in computations. A notable among them is the recently proposed UCR suite [39]. The UCR suite showed that carefully combining different branch-and-bound ideas leads to a significantly faster algorithm for searching. They showed some very impressive speedups, especially when the query time series is small. UCR suite is currently the fastest package for searching time series with DTW measure, and it will serve as our primary baseline.

Branch and bounds techniques prune down candidates significantly while dealing with small queries (small subsequence search). For short queries, a cheap lower bound is sufficient to prune the search space significantly leading to impressive speedups. However, when the query length grows, which is usually the case, the bounds are very loose, and they do not result in any effective pruning. Our empirical finding suggests that existing branch-and-bound leads to almost no pruning (less than 1%, see Section 3) when querying with longer time series, making the UCR suite expensive. Branch-and-bound techniques, in general, do not scale well when dealing with long time series. Nevertheless, it should be noted that branch and bound techniques give exact answers. It appears that if we want to solve the search problem exactly, just like classical near neighbor search, there is less hope to improve the UCR suite. We will discuss this in details in Section 3.

Indexing algorithms based on hashing are well studied for reducing the query complexity of high-dimensional similarity search [36, 45, 42]. Hashing techniques are broadly divided into two categories: 1) Data Independent Hashing [42, 45] and 2) Learning-based (Data Dependent) Hashing [50, 51].

To the best of our knowledge, there is only one recent hashing algorithm tailored for the DTW measure: [22]. This algorithm falls into the category of learning-based hashing. Here, the authors demonstrated the benefit of kernel-based hashing (Learning-based) scheme [27] for DTW measure on medium scale datasets (60k time series or less). However, the computation of that algorithm scales poorly $O(n^2)$ where n is the number of time series. This poor scaling is due to the kernel matrix $(n \times n)$ and its decomposition which is not suitable for large-scale datasets like the ones used in this paper with around 20 million time series.

In addition, the method in [22], as a learning based hashing, requires an expensive optimization to learn the hash functions on data samples followed by hash table construction. Any change in data distribution needs to re-optimize the hash function and repopulate the hash tables from scratch. This static nature of learning-based hashing is prohibitive in current big-data processing systems where drift and volatility are frequent. Furthermore, the optimization itself requires quadratic $O(n^2)$ memory and computations, making them infeasible to train on large datasets (such as the one used in this paper where n runs into millions).

In contrast, data independent hashing enjoys some of the unique advantages over learning-based hashing techniques. Data independent hashing techniques derive from the rich theory of Locality Sensitive Hashing (LSH) [14] and are free from all the computational burden. Furthermore, they are ideal for high-speed data mining in a volatile environment because drift in distribution does not require any change in the algorithms and the data structures can be updated dynamically. Owing to these unique advantages data independent hashing is some of the heavily adopted routines in commercial search engines [18]

However, data independent methodologies for time series are limited to vector based distance measures such as L_p [12, 1] or cosine similarity. As argued before, vector based distances are not suitable for time series similarity. Unfortunately, there is no known data independent hashing scheme tailored for the DTW measure. Lack of any such scheme makes hashing methods less attractive for time series mining, particularly when alignments are critical. A major hurdle is to design an indexing mechanism which is immune to misalignments. In particular, the hashes should be invariant to spurious transformations on time series such as shifting. In this work, we provide a data independent hashing scheme which respects alignments, and our empirical results show that it correlates near perfectly with the desired DTW measure. The focus of this paper will be on data-independent hashing schemes which scale favorably and cater the needs of frequently changing modern data distributions.

Our Contributions: We take the route of randomized hashing based indexing to prune the candidates more efficiently compared to branch-and-bound methods. We propose the first data-independent Hashing Algorithm for Time Series: SSH (Sketch, Shingle, & Hash). Indexing using SSH can be around 20x faster than the current fastest package for searching time series with DTW, UCR suite [39]. Our proposal is a novel hashing scheme which, unlike existing schemes, does both the alignment and matching at the same time. Our proposal keeps a sliding window of random filters to extract noisy local bit-profiles (sketches) from the time series. Higher order shingles (or n-grams with large n like 15 or more) from these bit-profiles are used to construct a weighted set which is finally indexed using standard weighted minwise hashing which is a standard locality sensitive hashing (LSH) scheme for weighted sets.

Our experiments show that the ranking under SSH aligns near perfectly with the DTW ranking. With SSH based indexing we can obtain more than 90% pruning even with long queries where branch-and-bound fails to prune more than 7%. Our proposed method is simple to implement and generates indexes (or hashes) in one pass over the time series vector. Experimental results on two large datasets, with more than 20 million time series, demonstrate that our method is significantly (around 20 times) faster than the state-of-the-art package without any noticeable loss in the accuracy.

The rest of this paper is organized as follows. Section 2 introduces background of our work. In Section 3 we discuss why pruning strategies can not work well when dealing with long queries. We then describe our approach in Section 4. Section 5 presents our experimental results.

2. BACKGROUND

Let us now review several backgrounds of our work. We introduce DTW (Dynamic Time Warping) for time series in Section 2.1. We then introduce Locality Sensitive Hashing and Weighted Minwise Hashing in Section 2.2.

2.1 Dynamic Time Warping and Expensive Computation

One of the peculiarities of time series similarity which is different from general vector similarity is its invariance with warping or shift in time. For example, a series $X = \{x_1, x_2, ..., x_m\}$, associated with timestamps:

$$\{t(x_1), t(x_2), ..., t(x_m)\}$$

should be very similar to a slightly shifted time series $X' = \{x_3, x_4, ..., x_m, y, z\}$ over the same time stamps. This high similarity is because there is a significantly long subsequence of X and X', which are identical (or very similar). Traditional measures such as L_2 distance are not suitable for such notions of similarity as they are sensitive to shifts. Dynamic Time Warping (DTW) was designed to align various systematic inconsistencies in the time series, which is the main reason behind its wide adoption.

To compute the DTW distance we construct an m-by-m matrix W, where the (i-th,j-th) element of the matrix W denotes the difference between i-th component of X and j-th component of Y. The DTW distance finds the path through the matrix that minimizes the total cumulative distance between X and Y (Fig. 1). The optimal path is the one that minimizes the warping cost:

$$DTW(X, Y) = \min \sqrt{\sum_{k=1}^{K} w_k}$$

where, w_k is the $k-th$ element of a warping path P, which is a contiguous set of elements that represent a mapping between X and Y. The overall computation of DTW is given by a dynamic program, please see [39] for more details.

DTW is costly as it requires $O(m^2)$ computations using a dynamic programming solution, where m is the time series

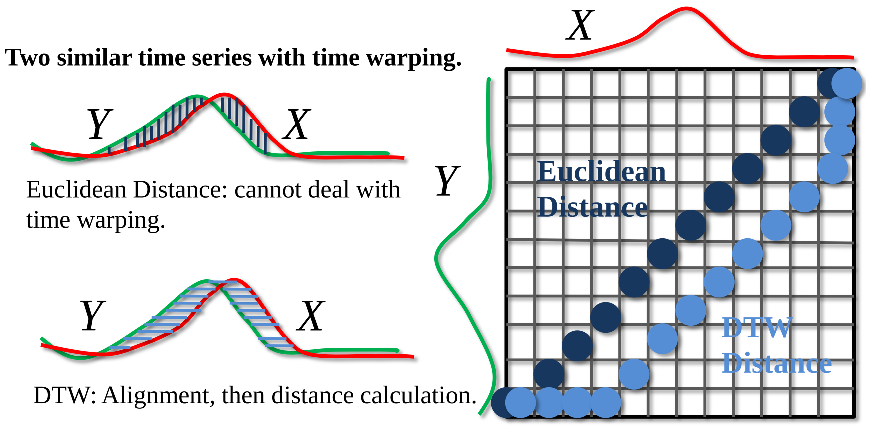

Two similar time series with time warping.
Y
X
Euclidean Distance: cannot deal with time warping.
Y
X
DTW: Alignment, then distance calculation.
X
Y
Euclidean Distance
DTW Distance

Figure 1: The difference between Euclidean and DTW distances of two time series X and Y. The DTW distance computes the similarity of the best alignment and hence can deal with time warping of X and Y.

length. DTW computes the optimal alignment of the two given time series followed by calculating the optimal similarity after the alignment. As expected, alignment is a slow operation. To make searching, with DTW, efficient a common strategy is to resort of branch and bound based early pruning [39].

2.2 Locality Sensitive Hashing and Weighted Minwise Hashing

2.3 Locality Sensitive Hashing (LSH)

Locality-sensitive hashing (LSH) [2, 31] is common for sub-linear time near neighbor search. The basic idea of LSH is to hash input items to different buckets so that similar items map to the same "buckets" with high probability.

LSH generates a random hash map h which takes the input (usually the data vector) and outputs a discrete (random) number. For two data vectors x and y, the event $h(x) = h(y)$ is called the collision (or agreement) of hash values between x and y. The hash map has the property that similar data vectors, in some desired notion, have a higher probability of collisions than non-similar data vectors. Informally, if x and y are similar, then $h(x) = h(y)$ is a more likely event, while if they are not similar then $h(x) \neq h(y)$ is more likely. The output of the hash functions is a noisy random fingerprint of the data vector [9, 37, 23], which being discrete is used for indexing training data vectors into hash tables. These hash tables represent an efficient data structure for similarity search [20].

For the details of Locality-sensitive hashing, please refer [2, 31].

2.4 Weighted Minwise Hashing

Weighted Minwise Hashing is a known LSH for the Weighted Jaccard similarity [28]. Given two positive vectors $x,\ y \in \mathbb{R}^{\mathbb{D}}$, $x,\ y > 0$, the (generalized) Weighted Jaccard similarity is defined as

$$\mathbb{J}(x,y) = \frac{\sum_{i=1}^{D} \min\{x_i, y_i\}}{\sum_{i=1}^{D} \max\{x_i, y_i\}}. \qquad (2)$$

$\mathbb{J}(x,y)$ is a frequently used measure for comparing web-documents [5], histograms (specially images), gene sequences, etc. Recently, it was shown to be a very effective kernel for large-scale non-linear learning [29]. WMH leads to the best-known LSH for L_1 distance, commonly used in computer vision, improving over [12].

Weighted Minwise Hashing (WMH) (or Minwise Sampling) generates randomized hash (or fingerprint) $h(x)$, of the given data vector $x \geq 0$, such that for any pair of vectors x and y, the probability of hash collision (or agreement of hash values) is given by,

$$Pr(h(x) = h(y)) = \frac{\sum \min\{x_i, y_i\}}{\sum \max\{x_i, y_i\}} = \mathbb{J}(x,y). \qquad (3)$$

A notable special case is when x and y are binary (or sets), i.e. $x_i, y_i \in \{0,\ 1\}^D$. For this case, the similarity measure boils down to $\mathbb{J}(x,y) = \frac{\sum \min\{x_i,y_i\}}{\sum \max\{x_i,y_i\}} = \frac{|x \cap y|}{|x \cup y|}$.

Weighted Minwise Hashing (or Sampling), [5, 7, 33] is the most popular and fruitful hashing technique for indexing weighted sets, commonly deployed in commercial big-data systems for reducing the computational requirements of many large-scale search [6, 3, 17, 18, 26, 11]. Recently there has been many efficient methodologies to compute weighted minwise hashing [33, 21, 43, 44, 16, 41].

3. LONGER SUBSEQUENCES AND ISSUES WITH BRANCH AND BOUND

Branch and bound strategies are used for reducing the searching cost by pruning off bad candidates early. The core idea behind branch and bound is to keep a cheap-to-compute lower bound on the DTW distance. For a given query, if the lower bound of the current candidate exceeds the best seen DTW then we ignore this candidate safely, simply using cheap lower bounds. This strategy eliminates the need for computing the costly DTW.

UCR Suite [39] combines several branch and bound strategies and makes time series searching process very fast. Three main branch and bound strategies are used in UCR Suite [39]: LB_{Kim} [25] lower bound, LB_{Keogh} lower bound, and LB_{Keogh2} lower bound [24]. LB_{Kim} uses the distance between the First (Last) pair of points from Candidate time series and the Query time series as the lower bound. The complexity of calculating LB_{Kim} is $O(1)$. LB_{Keogh} and LB_{Keogh2} [24] uses the Euclidean distance between the candidate series and Query series.

Table 1: Percentage of candidates that pruned by UCR Suite on ECG Data set and Random Walk Data set. With the increasing of the time series length, the ability of lower bounds used by UCR Suite to prune candidates deteriorate as the bounds suffer from the curse of dimensionality.

Time Series Length	128	512	1024	2048
UCR Suite Pruned (ECG)	**99.7%**	**94.96%**	**18.70%**	**7.76%**
UCR Suite Pruned (Random Walk)	**98.6%**	**14.11%**	**30.2%**	**3.5%**

The complexity of this lower bound is $O(n)$, where n is the time series length. These three branching and bounds strategies can prune bad candidates in $O(1)$ (or O(n)) time which are significantly smaller than the time needed to compute DTW distance ($O(n^2)$ time).

However, the lower bound gets weaker with the increase in the length of the time series, due to the curse of dimensionality. This weakening of bounds with dimensionality makes branch-and-bound ideas ineffective. We demonstrate this phenomenon empirically on two large-scale datasets (also used in our experiment see Section 5). Table.1 shows the percentage of candidates that are pruned by the three lower bounding strategies as well as the UCR Suite which combines all the three.

The code of this experiment are taken from the UCR Suite package [1], which is publicly available. This implementation of UCR Suite uses three pruning lower bound: LB_{Kim} [25] lower bound, LB_{Keogh} lower bound, and LB_{Keogh2} lower bound [24]. For each time series, this UCR Suite package compares all the three lower bound for each time series and uses the lowest one to do pruning.

From Table.1 we can see that when the time series is short (e.g. 128), the pruning strategies performs quite well (98% to 99% of the time series pruned). However, when the time series length is around 1000 or more, then all the three criteria are completely ineffective (only 3% to 7% of the time series pruned), and the search boils down to nearly brute force. We observe the same trend on both the datasets as evident from Table. 1.

Intuitively, as the length of the query time series increases, the number of possible good alignments (or warping) also increases. A myopic $O(n)$ lower bound is, therefore, not effective to eliminate all the possibilities.

4. OUR PROPOSAL: SSH (SKETCH, SHINGLE & HASH)

SSH (Sketch, Shingle & Hash): We propose a new hashing scheme, for time series, such that hash collisions are "active" indicator of high similarity while ignoring misalignments if any. Our hashing scheme consists of the following stages:

1. **Sliding Window Bit-profile (Sketch) Extraction:** We use sliding window of random filter to generate a binary string B_X (sketch) of the given time series X.
2. **Shingle(n-grams) Generation:** We generate higher order shingles from the bit string B_X. This process generates a weighted set S_X of shingles.
3. **Weighted MinHash Computation:** Our final hash value is simply the weighted minwise hashes of S_X, which we use as our indexes to create hash tables for time series.

[1]http://www.cs.ucr.edu/ eamonn/UCRsuite.html

Next, we go over each of the three steps in detail.

4.1 Sliding Window Bit-profile Extraction

We have a successful set of methodologies based on shingling [28] to deal with massive-scale discrete sequential data such as text or strings. However, time series data contains continuous values making shingling based approaches inapplicable. Furthermore, variations in sampling intervals, frequency, and alignments make the problem worse.

Our first step solves all this problem by converting time series with continuous values to discrete sequential objects which can be handled using shingling later. We use the idea of sketching time series with a sliding window of random filters [19], which was shown to capture trends effectively. In particular, we produce a bit string (sketch) from the time series. Each bit in this string captures crude information of some small subsequence in the time series.

To generate local bit-profile, we use a randomly generated filter which is a small vector r, of appropriately chosen size W, as shown in Figure 2. This filter slides over the time series with an appropriately selected step size δ. During each slide, the filter r is multiplied to the current W length subsequence of the time series, and a bit indicating the sign of the output is stored. In technical terms, this is a signed projection of the selected window [30, 10]. These crude sketches are robust to various perturbations in the values of time series.

More formally, Given a time series $X = (x_1, x_2, ..., x_m)$, the length W of vector r, step size δ. The extracted information is a (bit) sign stream, given by:

$$B_X = (B_X^{(1)}, B_X^{(2)}, ..., B_X^{(N_B)}). \tag{4}$$

Where $N_B = (m - W)/\delta$ is the size of the sign stream B_X. And each $B_X^{(i)}$s is calculated as follow:

$$B_X^{(i)} = \begin{cases} +1 & : r.X_s^{(i)} \geq 0 \\ -1 & : r.X_s^{(i)} < 0 \end{cases} \tag{5}$$

In above, $X_s^{(i)} = \{x_{i*\delta}, x_{i*\delta+1}, ..., x_{i*\delta+W-1}\}$ is the subseries of length W.

For example, given a time series $X = (1, 2, 4, 1)$, a small filter $r = (0.1, -0.1)$, and a step size $\delta = 2$. Then the extracted sign stream is:

$$\begin{aligned} B_X &= (sign((1,2)*(0.1,-0.1)), sign((4,1)*(0.1,-0.1))) \\ &= (sign(-0.1), sign(0.3)) \\ &= (-1, +1) \end{aligned} \tag{6}$$

In this step, we choose r as a spherically symmetric random vector with length W, i.e. the entries of r are $i.i.d$

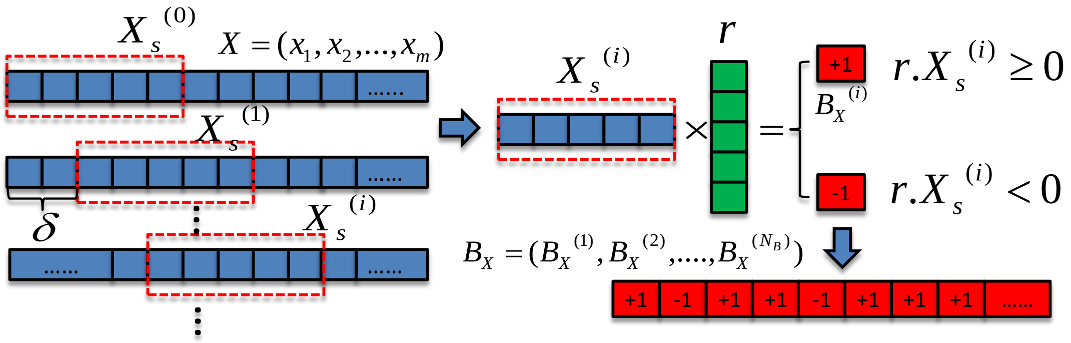


Figure 2: For each time series X, we convolve it with a sliding window (red dash box), with shift δ, of random gaussian filter r and generate a bit depending on the sign of the inner product. After the complete slide, the process generate a binary string (sketch) B_X which captures the pattern in the time series.

normal, i.e., $r \sim N(0,1)$. This choice is a known locality sensitive hashing for cosine similarity [31]. From the theory of signed random projections [31] these bits are 1-bit dimensionality reduction of the associated small subsequence which was multiplied by r. It ensures that bit matches are a crude probabilistic indicator of the closeness of local profiles.

4.2 Shingle(n-grams) Generation

After the sketching step we have a bit-string profile B_X from the time series X, where the i^{th} bit value $B_X(i)$ is a 1-bit summary (representation) of a small subsequence of X, i.e. $X_s^{(i)} = \{x_{i*\delta}, x_{i*\delta+1}, ..., x_{i*\delta+W-1}\}$. Therefore, for two vectors X and Y, $B_X(i) = B_X(j)$ indicates that the small subsequences $X_s^{(i)} = \{x_{i*\delta}, x_{i*\delta+1}, ..., x_{i*\delta+W-1}\}$ and $Y_s^{(j)} = \{y_{j*\delta}, y_{j*\delta+1}, ..., y_{i*\delta+W-1}\}$ are likely to be similar due to the LSH property of the bits.

4.2.1 *Intuition of why this captures alignments as well as similarity?*

If for two time series X and Y there is a common (or very similar) long subsequence, then we can expect that a relatively large substring of B_X will match with some other significant substring of B_Y (with possibly some shift). However, the match will not be exact due to the probabilistic nature of bits. This situation is very similar to the problem of string matching based on edit distance, where token based (or n-gram based) approach has shown significant success in practice. The underlying idea is that if two bit strings B_X and B_Y has a long common (with some corruption due to probabilistic nature) subsequence, then we can expect a significant common n-grams (and their frequency) between these bit strings. It should be noted that n-gram based approach automatically takes care of the alignment as shown in Fig. 4.

For example, given a bit string $B_X = (+1, +1, -1 - 1 + 1 + 1)$, and shingle length $n = 2$, then we can get a weighted set:

$$S_X = \{(+1,+1):2, (+1,-1):1, (-1,+1):1, (-1,-1):1\}.$$

Formally, given the bit (or sign) stream

$$B_X = (B_X^{(1)}, B_X^{(2)}, ..., B_X^{(N_B)})$$

for a time series X, we construct weighted set S_X by including all substrings of length n (n-gram) occurring in B_X

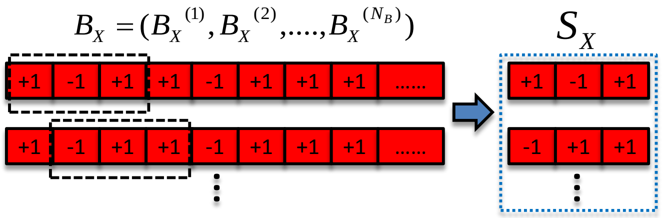


Figure 3: Shingle(n-grams) Generation: Give the bit string sketch generated from step-1, we treat it as string and generate n-grams shingles. The shingling process outputs a weighted set.

with their frequencies as their corresponding weight (see Figure.3).

$$S_X = \{S_i, w_i \mid S_i = \{B_X^{(i)}, B_X^{(i+1)}, ..., B_X^{(i+n-1)}\}, \ 0 < i < n\} \quad (7)$$

Notice that, the set is a weighted set, w_i denotes the number of tokens (or patterns) $S_i = \{B_X^{(i)}, B_X^{(i+1)}, ..., B_X^{(i+n-1)}\}$ present in the time series. The intuition here is that the Weighted Jaccard similarity between the sets S_X and S_Y, generated by two different time series X and Y, captures the closeness of the original time series. This closeness is not affected by spurious shifting of the time series.

4.3 Weighted MinHash Computation

The Shingle(n-grams) generation step generates a weighted set S_X for the given time series X. Since we want to capture set similarity, our final hash value is simply the weighted minwise hashing of this set. We use these weighted minwise hashes as the final indexes of the time series X, which can be utilized for creating hash tables for sub-linear search.

Weighted minwise hashing (or Consistent Weighted Sampling) is a standard technique for indexing weighted sets [7]. There are many efficient methodologies to compte them [33, 21, 43, 44, 16, 41]. Please refer to [41] for details.

4.4 Overall Framework

Given a time series search data set $D = \{X_i | 1 \leq i \leq N\}$, the query time series Q, and the corresponding parameters W, r, δ. Our goal is to output the top-k most similar time series of Q. The proposed framework contains two steps (1) Preprocessing Step: preprocess all the time series, and

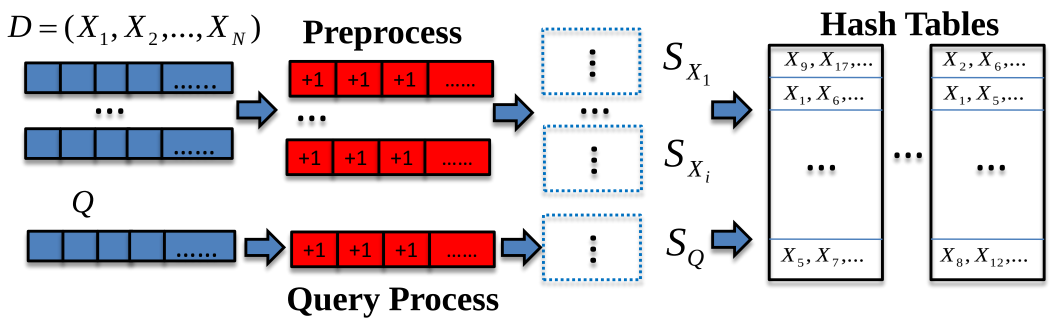
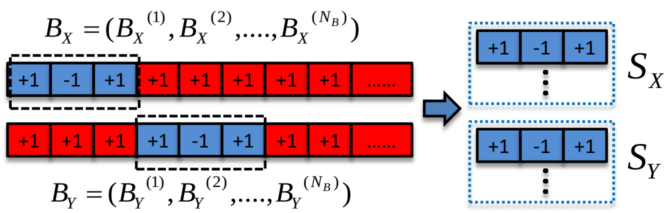


Figure 5: Overall framework contains two steps: (1) Preprocess and (2) Query process. In Preprocess stage, all the time series in data set D hased in to hash tables following three processing steps: (1) Sliding Window Bit Profile Extraction, (2) Shingle (n-gram) Generation, (3) Weighted MinHash Computation. In Query Process, given a query time series, find the associated buckets in the hash tables using the same s-step hashing schema.



$B_X = (B_X^{(1)}, B_X^{(2)}, ..., B_X^{(N_B)})$
+1 -1 +1 +1 +1 +1 +1 +1 ......
+1 +1 +1 +1 -1 +1 +1 +1 ......
$B_Y = (B_Y^{(1)}, B_Y^{(2)}, ..., B_Y^{(N_B)})$
S_X
S_Y

Figure 4: **SSH Illustration on two different time series:** Two different time series X and Y has same pattern (Blue window). We use n-gram to extract patterns and use the pattern set S_X and S_Y to represent the time series, then the time warping of time series is solved by set similarity.

hash them into hash tables using our 3-step SSH scheme (2) Query step, given a time series, find the associated buckets in the hash tables using the same 3-step SSH scheme. Select top-k among the candidates retrieved from the buckets. The detailed steps of our proposed framework are illustrated in Figure.5 and summarized in Algorithm 1, and Algorithm 2.

Algorithm 1 Pre-Processing

```
1: Input: Given $D = \{$~~$X_i|0 < i < N - 1$~~$\}$, the sub-series of length $W$. a spherically symmetric random vector $r$ with length $W$, step size $\delta$, n-gram Shingle Length $n$, number of hash tables $d$.
2: Output: Constructed $d$ hash tables.
3: Initialization $i = 0$
4: for Each time series $S_i$ in $D$ do
5:     Extract the information of time series $S_i$ using the method introduced in Section. 4.1.
6:     Using $n$-gram method introduced in Section 4.1.
7:     Using weighted minhash algorithm introduced in Section 4.3 to hash each time series into $d$ different hash tables.
8: end for
9: return Constructed $d$ hash tables.
```

Algorithm 1 shows the Preprocessing stage using the SSH scheme. This stage takes the time series data sets $D = \{X_i|1 \leq i \leq N\}$ as input, and construct d hash tables. In Algorithm 1, for each time series S_i in D, we perform Sliding Window Bit Profile Extraction (line 5), Shingle (n-gram) Generation (line 6), and Weighted MinHash Computation (line 7). These three SSH steps (line 5-7) hashes the time series into appropriate hash tables for future queries.

Algorithm 2 Query Process

```
1: Input: Given $D = \{X_i|0 < i < N - 1\}$, the sub-series of length $W$. a spherically symmetric random vector $r$ with length $W$, step size $\delta$, n-gram Shingle Length $n$, and the number of return series $k$.
2: Output: Top $k$ time series in $D$.
3: Extract the information of time series $Q$ using the method introduced in Section. 4.1.
4: Using $n$-gram method introduced in Section 4.1 to get the weighted set of $Q$.
5: Using weighted minhash algorithm introduced in Section 4.3 to ge the hash value of $Q$.
6: Initialize the retrieved set $R$ to null
7: for Each Hash table $T_i$ do
8:     Add all the time series in the probed bucket to $R$
9: end for
10: return Search $R$ for top-$k$ time series using UCR Suite algorithm
```

Algorithm 2 shows the querying process with the SSH scheme. This stage takes the query time series Q as input and returns top-k time series. We use the same SSH steps, Sliding Window Bit Profile Extraction (line 3), Shingle (n-gram) Generation (line 4), and Weighted MinHash Computation (line 5) on the query time series Q to generate the indexes. Using these indexes, we then probe the buckets in respective hash tables for potential candidates. We then report the top-k similarity time series, based on DTW (line 7-10). The reporting step requires full computation of DTW between the query and the potential candidates. To obtain more speedups, during the last step, we use the UCR suite branch-and-bound algorithm to prune the potential candidate further.

4.5 Discussions and Practical Issues

The SSH procedure leads to a weighted set which combines noisy sketching with cheap shingle based representation. Shingling (or Bag-of-Words or n-grams statistics) is a very powerful idea and has led to state-of-the-art representations for a variety of structured data which includes text, images, genomes, etc. It is further known that shingling is a lossy description because it does not capture complete information of the sequence data, and therefore do no have provable guarantees. Nevertheless, reasonably higher order shingles are still the best performing methods in the information retrieval task with both text and image datasets. For example, the state-of-the-art method for image retrieval, as implemented in popular openCV [4] package, compute various noisy features such as SIFT and then represent the image as bag-of-words of those SIFT features. The main argument that goes in favor of noisy representations is that real world high-dimensional datasets come from a distribution which makes the problem much simpler than the combinatorial hardness associated with their raw representations. A noisy representation many times is good enough to capture the essence of that distribution and thus can save significantly over methods which try to solve the problem exactly in the combinatorial formulation.

In SSH procedure there are three main parameters: Length W of the spherically symmetric random vector r, step size δ, and n-gram Shingle Length n. Different choice of these parameters will impact the performance of SSH.

As with other shingling methods, the right selection of the parameters is usually dependent on the data and the similarity distribution. We can easily choose these parameters using a holdout dataset that suits our task. Since we are interested in retrieving with the DTW measure, we determine values of these parameters such that the rankings under hash collisions nearly agree with the ranking of DTW over a small sample of the dataset. We introduce details, and thorough analysis of these parameters study in section 5.

It should be further noted that the overall hashing scheme can be computed in just one pass over the time series. As we scan, can keep a sliding window over the time to calculate the inner product with filter r. Each bit generated goes into a buffer of size n. For every n-gram generated, we can hash the tokens and update the minimum on the fly.

5. EXPERIMENT

In this section, we describe our experimental evaluation of SSH procedure on two benchmark data set: ECG time series data and Random Walk time series data. Since our proposal is a new indexing measure for DTW similarity, just like [39], our gold standard accuracy will be based on the DTW similarity.

The experiments run on a PC (Xeon(R) E3-1240 v3 @ 3.40GHz $\times$ 8 with 16GB RAM). All code are implemented in C++. We use g++ 4.8.4 compiler. To avoid complications, we do not use any c++ compiler optimization tools to speed up the program.

5.1 Datasets

To evaluate the effectiveness of our method for searching over time series, we choose two publicly available large time series data which were also used by the UCR suite paper [39]: Random Walk, and ECG [2]. Random Walk is a benchmark dataset, which is often used for testing the similarity search methods [39]. The ECG data consists of 22 hours and 23 minutes of ECG data (20,140,000 data points).

We also processed both the datasets in the same manner as suggested in [39]. The process is as follow: Given the very long time series $S = (s_1, s_2, ..., s_m)$, and a time series length t ($t = 128,\ 512,\ 1024,\ 2048$). We extract a time series data set $D = \{S^i | 1 < i < m - t + 1\}$, where each S^i in D is denoted as:

$$S_i = (s_i, s_i + 1, ..., s_{i+t-1})$$

After we get the time series data set $D = \{S^i | 1 < i < m - t + 1\}$, we can then do the similarity searching tasks.

For our proposed SSH method, the choices of W, δ and n were determined using a holdout sample.

For the 22 Hour ECG time series data. We choose window size $W = 80$, $\delta = 3$, and $n = 15$ for n-gram based set construction, and we use 20 hash tables, for each hash table using our hash as index. For the Random Walk Benchmark time series data. We choose window size $W = 30$, $\delta = 5$, and $n = 15$ for n-gram based set construction, and we use 20 hash tables. The effect and choice of these values are explained in Section 5.5.

5.2 Baselines

Since UCR suite is the state-of-the-art algorithm for searching over time series data, we use it as our best branch-and-bound baseline. Note that branch-and-bound baselines are exact.

We point out here that there is no known data independent hashing scheme for time series data that can handle misalignment of time series. So, as another sanity check, we also compare the performance of vanilla hashing scheme the signed random projections (**SRP**) to confirm if the alignment is a critical aspect. For SRP, we simply regard the time series as long vectors. If alignment is not critical then treating time series as vectors is a good idea and we can expect SRP to perform well.

5.3 Accuracy and Ranking Evaluation

Task: We consider the standard benchmark task of near-neighbor search over time series data with DTW as the gold standard measure.

To understand the variance of performance we run our hashing algorithm SSH, as described in Section 4, for searching top-k near neighbors. The gold standard top-k neighbors were based on the actual DTW similarity. For the SRP baseline, we replace SSH indexing procedure with SRP hash function. For a rigorous evaluation we run these algorithms with a different values of $k = \{5,\ 10,\ 20,\ 50\}$.

Evaluation Metric We use precision and NDCG (Normalized Discounted Cumulative Gain) [48] to evaluate the accuracy and rankings of our method.

Precision for a search result is defined as

$$Precision = \frac{relevantseen}{k},$$

here relevant seen denotes the number of top-k gold standard time series returned by the algorithm.

Just observing the precision is not always a good indicator of rankings. We also evaluate the rankings of the top-k candidates reported by our algorithm with the rankings generated by DTW measure. We use NDCG (Normalized Discounted Cumulative Gain), which is a widely used ranking evaluation metric. NDCG [48] is defined as:

$$nDCG = \frac{DCG}{IDCG},$$

[2]http://www.cs.ucr.edu/ eamonn/UCRsuite.html

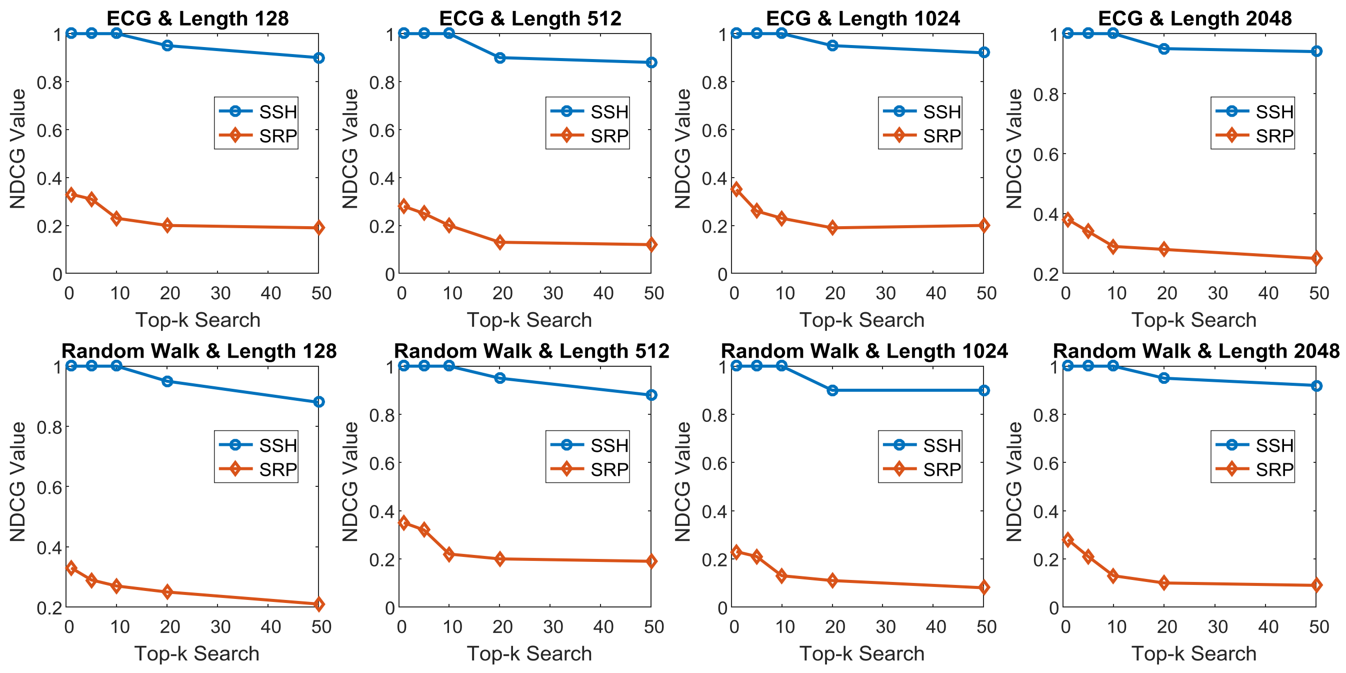


Figure 6: The NDCG (Normalized Discounted Cumulative Gain) of SSH and SRP (Sign Random Projection) on ECG and Random Walk Datasets. The Gold Standard Ranking was based on DTW Distance.

Table 2: Accuracy of SSH Framework and SRP (Sign Random Projection) on ECG and Random Walk Dataset for retrieving top-k ($k = 5, 10, 20, 50$) time series. We can see that SRP performs very poorly due to lack of alignment. The proposed SSH Framework on the other hand is significantly accurate. We can conclude that alignment is critical for these datasets.

Dataset	**Time Series Length**	**Method**	**Top-5**	**Top-10**	**Top-20**	**Top-50**
ECG	128	SSH	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{0.95 \pm 0.05}$	$\mathbf{0.90 \pm 0.02}$
		SRP	0.20 ± 0.00	0.10 ± 0.10	0.10 ± 0.05	0.04 ± 0.02
	512	SSH	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{0.90 \pm 0.05}$	$\mathbf{0.88 \pm 0.02}$
		SRP	0.00 ± 0.00	0.10 ± 0.10	0.05 ± 0.05	0.04 ± 0.02
	1024	SSH	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{0.95 \pm 0.05}$	$\mathbf{0.92 \pm 0.02}$
		SRP	0.00 ± 0.00	0.00 ± 0.00	0.05 ± 0.05	0.02 ± 0.02
	2048	SSH	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{0.95 \pm 0.05}$	$\mathbf{0.94 \pm 0.02}$
		SRP	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
Random Walk	128	SSH	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{0.95 \pm 0.00}$	$\mathbf{0.88 \pm 0.02}$
		SRP	0.00 ± 0.00	0.20 ± 0.10	0.10 ± 0.05	0.04 ± 0.02
	512	SSH	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{0.95 \pm 0.00}$	$\mathbf{0.86 \pm 0.02}$
		SRP	0.00 ± 0.00	0.00 ± 0.00	0.05 ± 0.00	0.04 ± 0.02
	1024	SSH	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{0.90 \pm 0.10}$	$\mathbf{0.90 \pm 0.04}$
		SRP	0.00 ± 0.00	0.10 ± 0.00	0.05 ± 0.05	0.04 ± 0.00
	2048	SSH	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{0.95 \pm 0.05}$	$\mathbf{0.92 \pm 0.02}$
		SRP	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.02 ± 0.02

Table 3: CPU Execution time (in seconds) of UCR Suite and our proposed hashing method on ECG and Random Walk Dataset, with increasing query length. Hashing algorithm is consistently faster than UCR suite and gets even better with increase in query length. For longer query time series, hashing can be 20x faster.

Dataset	**Method**	**128**	**512**	**1024**	**2048**
ECG	SSH	**2.30024**	**5.50103**	**39.578**	**339.57**
	Branch-and-Bounds (UCR Suite)	7.90036	20.2823	309.578	7934.615
Random Walk	SSH	**1.21002**	**3.20156**	**15.2061**	**216.48035**
	Branch-and-Bounds (UCR Suite)	3.32005	42.12	297.652	1934.615

Table 4: Percentage of time series filtered by the SSH for different query length. Hashing, unlike branch and bound, becomes more effective for longer sequences.

Dataset	**Method**	**128**	**512**	**1024**	**2048**
ECG	SSH Algorithm (Full)	**99.9%**	**98.8%**	**90.8%**	**95.7%**
	Pruned by Hashing alone (SSH)	72.4%	76.4%	88.7%	95.4%
	Branch-and-Bounds (UCR Suite)	99.7%	94.96%	18.70%	7.76%
Random Walk	SSH Algorithm (Full)	**99.6%**	**97.6%**	**94.2%**	**92.6%**
	Pruned by Hashing alone(SSH)	75.4%	86.4%	91.7%	92.4%
	Branch-and-Bounds (UCR Suite)	98.6%	82.7%	30.2%	3.5%

where DCG can be calculated as $DCG = \sum_{i=1}^{k} \frac{R_i}{\log_2(i)}$, and IDCG denotes the DCG for the ground truth ranking result. R_i denotes the graded relevance of the result at position i. In this experiment, R_i is calculated as $R_i = k - i$.

Result: The ranking result on ECG and Random walk dataset is shown in Figure. 6. We summarize the precision of different methodologies in Table. 2. Note, UCR suite is an exact method so it will always have an accuracy of 100% and NDCG is always 1.

We can see from Figure. 6 that the proposed SSH based ranking achieves near perfect NDCG value for most values of k and gracefully decreases for large k. This deterioration with increasing k is expected as hashing techniques are meant for high similarity region. On the contrary, the performance of SRP is quite poor irrespective of the values of k, indicating the importance of alignment.

In Table. 2, we can see that for the top-5 and top-10 similarity search tasks, our proposed method can get 100% accuracy for both the benchmark datasets. For large $k \geq 20$ we see some loss in the accuracy. As expected, hashing based methods are very accurate at high similarity levels which are generally of interest for near-neighbor search. The performance of SRP, as expected, is very poor indicating the need for alignment in time series data. The success of our method clearly shows that our proposal can get the right alignment. We can clearly see that despite SSH being the approximate method the impact of approximation is negligible on the final accuracy. The accuracy trends are consistent for both the datasets.

5.4 Speed Comparison

We now demonstrate the speedup obtained using the SSH procedure. We compute the average query time which is the time required to retrieve top-k candidates using Algorithm 4. The query time includes the time needed to compute the SSH indexes of the query.

The CPU execution time of our method and exact search method using UCR Suite is shown in Table 3. We can clearly see that hashing based method is significantly faster in all the experiments consistently over both the data sets irrespective of the length of the query.

It can be clearly seen from the table that when the query length increases, the UCR suite scales very poorly. For searching with long time series (e.g. 2048 or higher) hashing based method is drastically efficient compared to the UCR Suite. It can be around 20 times faster than the UCR suite.

To understand the effectiveness of hashing in pruning the search space, we also show the number of time series that were filtered by our proposed algorithm. We also highlight the candidates pruned by hashing alone and separate it from the total candidates pruned which include additional pruning in step 10 of Algorithm 2. We summarize these percentages in Table. 4. Hashing itself prunes down the candidates drastically. For shorter queries, it is advantageous to use branch-and-bound to filter further the candidates returned by hashing. As expected for longer queries hashing is sufficient, and the additional branch and bound pruning by our algorithm leads to negligible advantages. Hashing based pruning works even better with an increase in the time series length. This is expected because hashing based method is independent of dimensions and only pick time series with high similarity. These numbers also demonstrate the power of our method when dealing with long time series.

It should be further noted that hashing based filtering, unlike branch and bound, does not even require to evaluate any cheap lower bound and thus are truly sub-linear. In par-

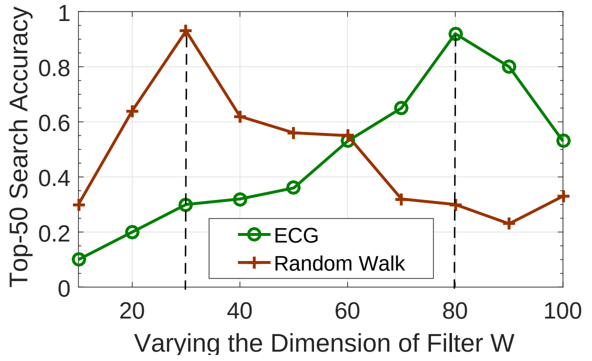


Figure 7: Accuracy with respect to the filter dimension W for the two data sets.

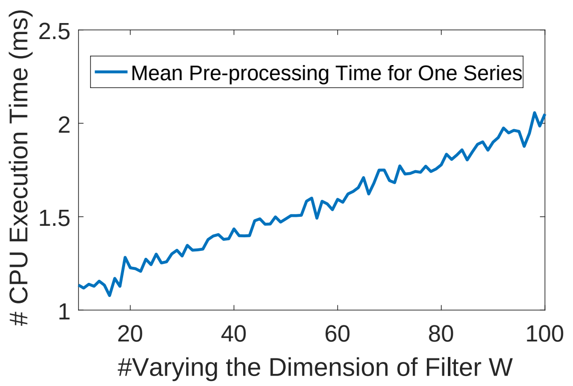


Figure 8: Preprocessing time with respect to the filter dimension W.

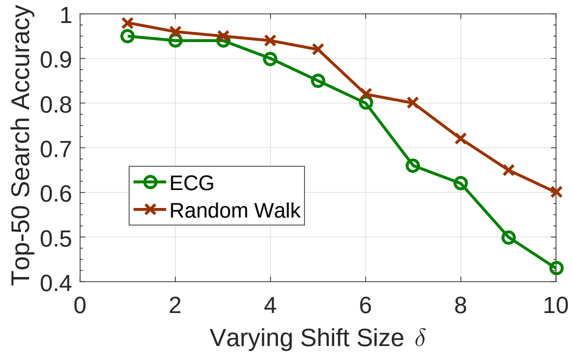


Figure 9: Accuracy with respect to the shift size δ.

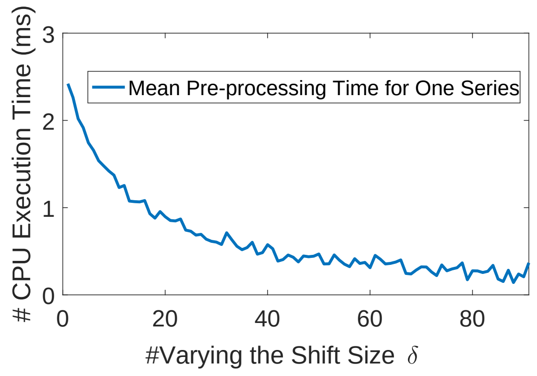


Figure 10: Preprocessing time with respect to the shift size δ

ticular, branch and bound prune a candidate by computing a cheap lower bound which still requires enumerating all the time series. Hashing eliminates by bucketing without even touching the pruned candidates.

5.5 Parameter Study

As we introduced in Section 4.5. The proposed hashing scheme takes W (the dimension of the filter r), δ (the shift size) and n (shingle length) as parameters. The choice of these parameters is critical for the performance. In this section, we shall show the impact of these three parameters on the retrieving accuracy and execution time. This study will also explain the selection procedure of the parameters we used in our experiments.

5.5.1 *W (the dimension of filter)*

The dimension W of the filter r in our framework is a critical parameter. If W is too large, then the 1-bit sketch is likely to be non-informative and won't capture temporal trends. Also, it may merge significant patterns of the time series. On the other hand, if the choice of W is too small, then the sub-series may only contain component information which can be very noisy. Thus there is a trade-off. From the execution time perspective view, if we choose large W, the preprocessing execution time will increase due to the inner product operation. As a result, proper choice of W is an imperative step of our framework.

Fig. 7 shows the precision of SSH indexes, for top-50 neighbors, with varying filter dimension W for the two data sets. We can see from the figure that when the W is small, the accuracy is reduced. With the increase of the filter dimension W, the accuracy starts increasing and after reaching a sweet spot drops again. We achieve the sweet spot at W=80 for ECG time series data and W=30 for random walk data respectively.

Fig. 8 shows the preprocessing time with varying filter dimension W for the two data sets. From the result, we can see that the average running time for preprocessing on a single time series is linear to the dimension of W. This is because the preprocessing time will increase due to the increase in the number of inner product operation.

5.5.2 *δ (the shift size)*

The shift size δ of the SSH scheme shows a consistent trend of decreasing the accuracy with an increase in δ. The best δ is $\delta = 1$. However, a smaller δ increase the execution complexity of SSH because of the number of inner products. Large δ leads to information loss. Fig. 9 shows the accuracy with the shift size δ for the two data sets, whereas Fig. 10 shows the preprocessing time by varying the shift size δ for the two data sets. From the result, we can see that the average running time for preprocessing increases with the decreasing of the shift size. To balance this accuracy-time trade-off we chose $\delta = 3$ for ECG and $\delta = 5$ for random walk data respectively.

5.5.3 *n (shingle length)*

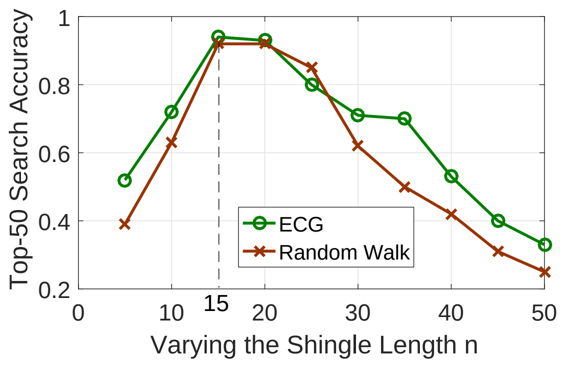


Figure 11: Accuracy by varying the shingle length n for the two data sets.

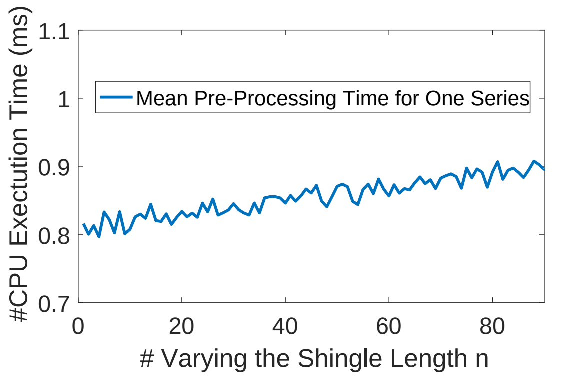


Figure 12: Preprocessing execution time by varying the filter dimension n for the two data sets.

The shingle length n in SSH turns out to be a sensitive and critical parameter. Just like the behavior of n-grams in the text, too large to too little n hurts the performance.

Fig. 11 shows the accuracy by varying the Shingle length n for the two data sets. We can see from the figure that when the n is too small, the accuracy is poor. With the increasing of the shingle length n, the accuracy also increase. For both ECG and Random Walk datasets, n=15 seems to be the right sweet spot. With further increasing in n, the accuracy start to decrease.

Fig. 12 shows the preprocessing execution time by varying the filter dimension n for the two data sets. As expected, we can see that the average running time for preprocessing is linear to the dimension of n. When the shingle length n increases, the constructed weighted set S will become larger, thus the execution time will also increase.

6. CONCLUSIONS

DTW is a widely popular but expensive similarity measure. Speeding up DTW is an important research direction. Branch and bound based candidate pruning was the most popular method for improving search efficiency with DTW. However, branch-and-bound techniques suffer from the curse of dimensionality. Our proposed framework provides an alternative route of randomized indexing to prune the candidates more efficiently.

We have proposed SSH (Sketch, Shingle & Hash) the first indexing scheme which does both the alignment and matching on time series data. Unlike branch and bound based approaches our scheme does not deteriorate with an increase in the length of query time series are is free from the curse of dimensionality.

SSH combines carefully chosen three step procedure for indexing time series data which as we demonstrate are ideal for searching with DTW similarity measure. For similarity search with time series data, we show around 20x speedup over the fastest package UCR suite on two benchmark datasets.

Acknowledgments

We are thankful for UCR time series Team for providing the UCR Suite code and the corresponding time series data. This work was supported by Rice Faculty Initiative Award 2016.

7. REFERENCES

[1] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *International Conference on Foundations of Data Organization and Algorithms*, pages 69–84. Springer, 1993.

[2] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 459–468. IEEE, 2006.

[3] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *WWW*, pages 131–140, 2007.

[4] G. Bradski and A. Kaehler. *Learning OpenCV: Computer vision with the OpenCV library.* " O'Reilly Media, Inc.", 2008.

[5] A. Z. Broder. On the resemblance and containment of documents. In *the Compression and Complexity of Sequences*, pages 21–29, Positano, Italy, 1997.

[6] A. Z. Broder. Filtering near-duplicate documents. In *FUN*, Isola d'Elba, Italy, 1998.

[7] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. In *WWW*, pages 1157 – 1166, Santa Clara, CA, 1997.

[8] J. P. Caraça-Valente and I. López-Chavarrías. Discovering similar patterns in time series. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 497–505. ACM, 2000.

[9] J. L. Carter and M. N. Wegman. Universal classes of hash functions. In *STOC*, pages 106–112, 1977.

[10] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, Montreal, Quebec, Canada, 2002.

[11] S. Chien and N. Immorlica. Semantic similarity between search engine queries using temporal correlation. In *WWW*, pages 2–11, 2005.

[12] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokn. Locality-sensitive hashing scheme based on p-stable distributions. In *SCG*, pages 253 – 262, Brooklyn, NY, 2004.

[13] 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. *Proceedings of the VLDB Endowment*, 1(2):1542–1552, 2008.

[14] A. Gionis, P. Indyk, R. Motwani, et al. Similarity search in high dimensions via hashing. In *VLDB*, volume 99, pages 518–529, 1999.

[15] C. W. J. Granger and P. Newbold. *Forecasting economic time series.* Academic Press, 2014.

[16] B. Haeupler, M. Manasse, and K. Talwar. Consistent weighted sampling made fast, small, and easy. Technical report, arXiv:1410.4266, 2014.

[17] M. R. Henzinge. Algorithmic challenges in web search engines. *Internet Mathematics*, 1(1):115–123, 2004.

[18] M. Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 284–291. ACM, 2006.

[19] P. Indyk, N. Koudas, and S. Muthukrishnan. Identifying representative trends in massive time series data sets using sketches. In *Proceedings of the 26th International Conference on Very Large Data Bases*, VLDB '00, pages 363–372, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[20] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, Dallas, TX, 1998.

[21] S. Ioffe. Improved consistent sampling, weighted minhash and l1 sketching. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 246–255. IEEE, 2010.

[22] D. C. Kale, D. Gong, Z. Che, Y. Liu, G. Medioni, R. Wetzel, and P. Ross. An examination of multivariate time series hashing with applications to health care. In *Data Mining (ICDM), 2014 IEEE International Conference on*, pages 260–269. IEEE, 2014.

[23] R. M. Karp and M. O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987.

[24] E. Keogh, L. Wei, X. Xi, M. Vlachos, S.-H. Lee, and P. Protopapas. Supporting exact indexing of arbitrarily rotated shapes and periodic time series under euclidean and warping distance measures. *The VLDB JournalâĂŤThe International Journal on Very Large Data Bases*, 18(3):611–630, 2009.

[25] S.-W. Kim, S. Park, and W. W. Chu. An index-based approach for similarity search supporting time warping in large sequence databases. In *Data Engineering, 2001. Proceedings. 17th International Conference on*, pages 607–614. IEEE, 2001.

[26] N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: similarity measures and algorithms. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 802–803. ACM, 2006.

[27] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *2009 IEEE 12th international conference on computer vision*, pages 2130–2137. IEEE, 2009.

[28] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of massive datasets.* Cambridge University Press, 2014.

[29] P. Li. 0-bit consistent weighted sampling. In *KDD*, 2015.

[30] P. Li and A. C. König. Theory and applications b-bit minwise hashing. *Commun. ACM*, 2011.

[31] P. Li, A. Shrivastava, J. L. Moore, and A. C. König. Hashing algorithms for large-scale learning. In *Advances in neural information processing systems*, pages 2672–2680, 2011.

[32] C. Luo, J.-G. Lou, Q. Lin, Q. Fu, R. Ding, D. Zhang, and Z. Wang. Correlating events with time series for incident diagnosis. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1583–1592. ACM, 2014.

[33] M. Manasse, F. McSherry, and K. Talwar. Consistent weighted sampling. Technical Report MSR-TR-2010-73, Microsoft Research, 2010.

[34] M. Mudelsee. *Climate time series analysis.* Springer, 2013.

[35] T. Oates, M. D. Schmill, and P. R. Cohen. A method for clustering the experiences of a mobile robot that accords with human judgments. In *AAAI/IAAI*, pages 846–851, 2000.

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

[37] M. O. Rabin. Fingerprinting by random polynomials. Technical Report TR-15-81, Center for Research in Computing Technology, Cambridge, MA, 1981.

[38] L. Rabiner and B.-H. Juang. Fundamentals of speech recognition. 1993.

[39] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 262–270. ACM, 2012.

[40] J. Rao, X. Niu, and J. Lin. Compressing and decoding term statistics time series. In *European Conference on Information Retrieval*, pages 675–681. Springer, 2016.

[41] A. Shrivastava. Simple and efficient weighted minwise hashing. In *NIPS*, 2016.

[42] A. Shrivastava and P. Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *NIPS*, Montreal, CA, 2014.

[43] A. Shrivastava and P. Li. Densifying one permutation hashing via rotation for fast near neighbor search. In *ICML*, Beijing, China, 2014.

[44] A. Shrivastava and P. Li. Improved densification of one permutation hashing. In *UAI*, Quebec, CA, 2014.

[45] A. Shrivastava and P. Li. Asymmetric minwise hashing for indexing binary inner products and set containment. In *Proceedings of the 24th International Conference on World Wide Web*, pages 981–991. ACM, 2015.

[46] M. Sonka, V. Hlavac, and R. Boyle. *Image processing, analysis, and machine vision.* Cengage Learning, 2014.

[47] C. Sun, H. Zhang, J.-G. Lou, H. Zhang, Q. Wang, D. Zhang, and S.-C. Khoo. Querying sequential

software engineering data. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 700–710. ACM, 2014.

[48] Y. Wang, L. Wang, Y. Li, D. He, W. Chen, and T.-Y. Liu. A theoretical analysis of ndcg ranking measures. In *Proceedings of the 26th Annual Conference on Learning Theory (COLT 2013)*, 2013.

[49] M.-H. Yang, D. J. Kriegman, and N. Ahuja. Detecting faces in images: A survey. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(1):34–58, 2002.

[50] D. Zhang, F. Wang, and L. Si. Composite hashing with multiple information sources. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 225–234. ACM, 2011.

[51] D. Zhang, J. Wang, D. Cai, and J. Lu. Self-taught hashing for fast similarity search. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 18–25. ACM, 2010.