Efficient Shapelet Discovery for Time Series Classification

Guozhong Li, Byron Choi, Jianliang Xu, Sourav S Bhowmick, Kwok-Pan Chun, and Grace Lai-Hung Wong

Abstract—Time-series shapelets are discriminative subsequences, recently found effective for time series classification (TSC). It is evident that the quality of shapelets is crucial to the accuracy of TSC. However, major research has focused on building accurate models from some shapelet candidates. To determine such candidates, existing studies are surprisingly simple, e.g., enumerating subsequences of some fixed lengths, or randomly selecting some subsequences as shapelet candidates. The major bulk of computation is then on building the model from the candidates. In this paper, we propose a novel *efficient shapelet discovery* method, called BSPCOVER, to discover a set of high-quality shapelet candidates for model building. Specifically, BSPCOVER generates abundant candidates via Symbolic Aggregate approXimation with sliding window, then prunes identical and highly similar candidates via *Bloom filters*, and *similarity matching*, respectively. We next propose a p*-Cover algorithm* to efficiently determine discriminative shapelet candidates that maximally represent each time-series class. Finally, any existing shapelet learning method can be adopted to build a classification model. We have conducted extensive experiments with well-known time-series datasets and representative state-of-the-art methods. Results show that BSPCOVER speeds up the state-of-the-art methods by more than 70 times, and the accuracy is often comparable to or higher than existing works.

Index Terms—Time series classification, shapelet discovery, efficiency, accuracy

1 Introduction

Time series classification (TSC) has attracted considerable attention from both academia and industry. The classical approach to solving the TSC problem is the whole series-based approach [1]. This method combines classifiers, such as 1-Nearest Neighbor (1NN), and similarity metrics, such as euclidean Distance or Dynamic Time Warping distance. A recent trend in TSC is to find small patterns that represent classes of time series. Among these patterns, shapelet-based methods (e.g., [7], [9], [20]) have repeatedly demonstrated superior accuracy. Intuitively, shapelets can be understood as discriminative subsequences that maximally represent classes of time series. Recent research has combined shapelets with learning approaches (e.g., [7], [9]) to learn a few shapelets that can distinguish time series of different classes.

The two key steps of shapelet-based methods are *shapelet discovery* and *model building*. Recent research on shapelets has given much attention on accurate model building. In contrast, this paper shows that *shapelet discovery not only significantly improves the efficiency of model building but also often improves the accuracy of the model built*.

The fundamental challenge of shapelet discovery is to efficiently discover high-quality shapelets for model building. First, since shapelet can be any subsequence, whose length is smaller than or equal to the length of raw time series, the number of candidates is large. Given an instance of a time series, the number of shapelet candidates is, however, quadratic to its length. Second, the shapelets should be discriminative enough to classify time series [24]. Shapelet discovery is worthwhile because non-discriminative subsequences harm both the efficiency and accuracy of model building. Further, subsequences that discriminate similar time series instances of one class into different classes do not improve the accuracy but also adversely affect the efficiency of model building. Third, due to efficiency issues in model building for some large datasets, only a relatively small set of candidates are selected. For example, only shapelets of three different lengths are used in [9]. It is technically intriguing to systematically select candidates for efficient model building.

Previous shapelet-based methods can be divided into three categories. **I.** Brute force algorithms (e.g., [25]) exhaustively search shapelets from raw time series sequences. Information gain is adopted to measure the quality of shapelet candidates. The time complexity is $O(m^2n^4)$, where m is the number of time series instances and n is the length of time series. **II.** Some previous studies (e.g., learning time-series shapelets (LTS) [9] and shapelet transformation (ST) [18]) simply set a few fixed ratios of the raw time series lengths as the final shapelet lengths and hence the number

- *G. Li, B. Choi, and J. Xu are with the Department of Computer Science, Hong Kong Baptist University, Hong Kong. E-mail: {csgzli, bchoi, xujl}@comp.hkbu.edu.hk.*
- *S. S. Bhowmick is with the School of Computing Engineering, Nanyang Technological University, Singapore639798. E-mail: assourav@ntu.edu.sg.*
- *K.-P. Chun is with the Department of Geography, Hong Kong Baptist University, Hong Kong. E-mail: kpchun@hkbu.edu.hk.*
- *G. L.-H. Wong is with the Faculty of Medicine, The Chinese University of Hong Kong, Hong Kong. E-mail: wonglaihung@cuhk.edu.hk.*

Manuscript received 7 Mar. 2019; revised 19 Apr. 2020; accepted 11 May 2020. Date of publication 19 May 2020; date of current version 3 Feb. 2022. (Corresponding author: Guozhong Li.)
Recommended for acceptance by T. Palpanas.
Digital Object Identifier no. 10.1109/TKDE.2020.2995870

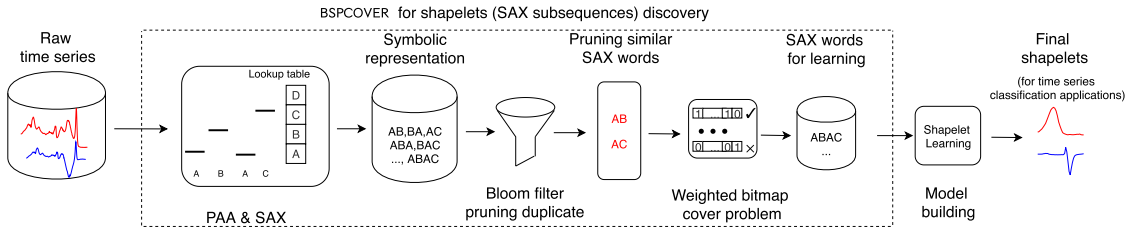


Fig. 1. The overview of shapelet discovery.

of shapelets for model building. These ratios are sensitive to both datasets and algorithms. For example, the number of shapelets for the dataset **Beef** [4] with a length of 470 is 71 $(0.15 \times n)$ for LTS and 235 $(0.5 \times n)$ for ST. The complexity is $O(Imn^2)$, where the learning approach LTS [9] is adopted and I is the iteration number. From the experiments on [7] on **Beef**, LTS takes *more than one week* to learn a model on training data and do the classification on test data. **III.** The state-of-the-art approaches SAX-VSEQL, SAX-VFSEQL [14], and ELIS [7] exploit symbolic representations of time series. The symbolic representation methods, such as Piecewise Aggregate Approximation (PAA) [12], and Symbolic Aggregate approXimation (SAX) [15] can reduce the dimensionality of the original time series. A technique is derived from TF-IDF to measure the quality of the candidates for ELIS, which can still be costly. ELIS takes *over six hours* to learn a model and do classification for **Beef**.

In this paper, we propose *an efficient shapelet discovery method, called* BSPCOVER, *to discover high-quality shapelets* for model building. An overview of BSPCOVER is presented in Fig. 1.

The first step of BSPCOVER follows existing work on shapelets that simply applies symbolic representation methods, specifically PAA [12] and SAX [15], to reduce the dimensionality of raw time series. In particular, they transform raw data into discrete representations (SAX subsequences, a.k.a SAX words). A sliding window method is adopted to generate numerous SAX words. The core of BSPCOVER then consists of three main steps. **I.** BSPCOVER derives a bloom filter for each class of time series to efficiently prune the same SAX words that also exist in other classes. Such SAX words have less discriminative power and only deteriorate overall efficiency and accuracy. Since bloom filters do not produce false negatives, the SAX words of a class that do not exist in the bloom filters of other classes are definitely not in those other classes. Such SAX words can be candidates for model building. **II.** The second step is a non-metric distance-based pruning of similar SAX words, which eliminates the effect of similar SAX words that exist in all classes. First, where the distance between the raw time series violates triangle inequality, an intrinsic property of time series, the discrete representation of time series inherits this property. Second, there are two situations in which similar SAX words are pruned. First, where similar SAX words exist in different classes, they are all pruned. In the second case, for words in the same class, only one representative word may be kept for further processing. **III.** For each set of similar SAX words of a class, we count their term frequency, as weight. Then, we design weighted bitmap structures of SAX words to quantify the quality of shapelet candidates, which is utilized to discover a set of SAX words that have the maximal weight and represent all the instances in each class. The shapelet selection problem is then formalized as a weighted bitmap cover problem, which can be reduced from the classical weighted set cover problem. We propose a heuristic algorithm to solve this problem. The complexity of the algorithm is $O(mn^2)$ only.

Finally, to complete a TSC solution, we compute the discriminative shapelets from the candidates as follows. We transform the SAX words back to the raw representation of time series. Existing learning methods can be applied for modifying the selected shapelets to improve the accuracy further. In the paper, we revise LTS [9] to build one-vs-rest classifiers.

We have conducted extensive experiments by running current state-of-the-art methods and our implementation of BSPCOVER on well-known time-series datasets. We have obtained consistent results as they are reported from the paper, unless stated otherwise. We note that BSPCOVER is faster than most shapelet-based methods, and its accuracy remains competitive with collective of transformation-based ensembles (COTE) [2], which is by far the best classifier today, according to a recent comprehensive survey of TSC [1].

We summarize the shapelet discovery and model building time complexity of some shapelet-based methods in Table 1. (While the complexities of FS are low, its accuracy is lower than other methods.) Model building of other methods is more costly than shapelet discovery. Due to shapelet discovery, I of BSPCOVER can often be 10 times smaller that of LTS and ELIS. Shapelet discovery makes model building converge much faster in practice.

The contributions of this paper are summarized as follows:

- We propose an approach, BSPCOVER, to discover discriminative shapelets efficiently for classifier model building for TSC. Bloom filters are adopted and a non-metric distance measure is proposed to prune non-discriminative shapelets.
- We design a data structure of SAX words, namely weighted bitmap, to measure the quality of SAX words. Then, we formalize a weighted bit cover problem based on this structure.
- The SAX words selection problem is reduced to the classical weighted set cover problem by the weighted

TABLE 1
Shapelet Discovery and Model Training Time Complexity of Some Shapelet-Based Methods [1]

Method	FS	ST & COTE	LTS	ELIS	BSPCOVER
Shapelet discovery	$O(mn^2)$	negligible	negligible	$O(mn^2 + mn\log(mn))$	$O(mn^2)$
Model building	-	$O(m^2n^4)$	$O(Imn^2)$	$O(Imn^2)$	$O(Imn^2)$

bitmap structure, and a heuristic algorithm is introduced for discovering a set of SAX words to build the classifier model.
- Extensive experiments on UCR datasets (UCRARCHIVE) for TSC studies [4] verify that our proposed approach is significantly more efficient and yet competitive in terms of accuracy when compared to existing shapelet-based methods.

Organization. The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 provides some preliminaries. The details of our proposed method are given in Section 4. Section 5 reports the experimental results. Section 6 concludes the paper and presents avenues for future work. For self-containedness, we summarize the adopted learning algorithm in Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2020.2995870.

2 Related Work

There are a large number of existing methods for TSC problem. Interested readers may refer to an excellent review paper [1]. However, the efficiency of TSC has not been the primary concern of research. In this section, we mainly review two representative methods, namely the *symbolic representation*-based methods, and the *shapelet*-based methods.

2.1 Symbolic Representation-Based Methods

There are a few symbolic representation-based methods such as PAA [7], SAX [14], [16], SFA [22], DFT and other discretization techniques. Another notable example is the dictionary-based method. These methods approximate and reduce the dimensionality of series through transforming raw data into representative words. Bag-of-Patterns (BOP) [17] builds a classifier from SAX, calculating the euclidean distance with the BOP representation of a new instance to determine its class. Another approach, SAX-VSM [23], adopts SAX techniques, proposes a vector space model (VSM), and uses TF-IDF to rank time series patterns. Le *et al.* [14] combine various variable-length bag-of-symbolic-words representations and an efficient linear sequence learning approach (SAX-VSEQL [11]) for efficient TSC.

2.2 Shapelet-Based Methods

Ye *et al.* propose the seminal work of time series shapelets [24]. Shapelets offer interpretable classification but require high computation time. Lines *et al.* propose ST [18] for selecting shapelets to represent the original time series. Classification methods such as Neural Network and SVM can be applied to determine shapelets. LTS [9] utilizes logistic regression with stochastic gradient descent to learn shapelets. It increases the classification accuracy of the UCR Time Series Classification Archive. However, the training phase is time-consuming because of the large number of shapelet candidates. A few parameters, such as the numbers and the lengths of shapelets, are manually tuned to select candidates for model building. Grabocka *et al.* provide fast shapelet discovery method [10] with an online clustering/pruning technique and a supervised shapelet selection technique. Rakthanmanon *et al.* propose fast shapelets [20] transforming the raw time series data into SAX words to reduce shapelet discovery time. The execution time is significantly reduced when compared to [24]. However, the accuracy is often clearly lower than more recent work [1].

There have been recent works on exploiting deep learning to solve the TSC problem (e.g., [8]). As motivated, this paper undertakes the shapelet-based approach due to its interpretability and competitive accuracy [1].

TABLE 2
Summary of Frequently Used Notations

Notation	Meaning
T	time series $T = (t_1, t_2, \ldots, t_n)$ of the length n
$\bar{T}$	the PAA result of time series T
Σ	the alphabet set of SAX words, e.g., {A,B,C,D,E}
$\hat{T}$	the SAX result of time series T
D	time series T_j with the class label C_j, $1 \le j \le m$
$\mathcal{C}$	the label set of dataset D
D_C	the time series instances of class C in D
$\mathcal{S}$	the shapelets
ω	compression ratio of time series data
θ	the threshold for determining ω
ϕ	the minimum sliding window ratio of $\hat{T}$
τ	a small real value

3 Preliminaries

In this section, we present some preliminaries and summarize the notations and their meanings in Table 2.

Definition 1 (Time series T). *A time series T is an ordered-value sequence $T = (t_1, t_2, \ldots, t_i, \ldots, t_n)$, where n is the length of T, t_i is the value observed at timestamp i.*

Definition 2 (Time series dataset D). *A time series dataset D is a set of time series T_j with $C_j = \mathsf{label}(T_j), j \in [1, m]$, where $C_j \in \mathcal{C}$ is the class label of the dataset, $\mathcal{C} = \{0, 1, 2, \ldots, |\mathcal{C}| - 1\}$, and $|\mathcal{C}|$ denotes the number of classes.*

Definition 3 (Subsequence). *Given a time series T, a subsequence of T is, $T(a, b) = (t_a \ldots, t_b)$, where $1 \le a \le b \le n$, a and b are the beginning and ending positions to the subsequence, respectively.*

Definition 4 (Distance Between Two Sequences [7]). *The distance of sequence T_p of the length n_p and T_q of the length n_q is denoted as (w.l.o.g. assuming $n_q \ge n_p$),*

$$\mathsf{dist}(T_p, T_q) = \min_{j=1,\ldots,n_q-n_p+1} \frac{1}{n_p} \sum_{l=1}^{n_p} (t_{q,j+l-1} - t_{p,l})^2. \quad (1)$$

Definition 5 (PAA Sequence $\bar{T}$ [12]). *A time series T can be represented by a PAA sequence $\bar{T} = (\bar{t_1}, \bar{t_2}, \ldots, \bar{t_{\lceil \frac{n}{\omega} \rceil}})$, where the i-th element of $\bar{T}$, $\bar{t_i}$, is calculated by the following equation:*

$$\bar{t_i} = \frac{1}{\omega} \sum_{j=\lceil (i-1)\omega+1 \rceil}^{\lceil i\omega \rceil} t_j, \quad (2)$$

where ω is the number of consecutive values for averaging.

Hence, ω is sometimes referred as the compression ratio of the PAA sequence over the raw time series data.

Definition 6 (SAX Sequence $\hat{T}$ [15]). *A SAX sequence $\hat{T} = \hat{t_1}, \hat{t_2}, \ldots, \hat{t_{\lceil \frac{n}{\omega} \rceil}}$ can be mapped from $\bar{T}$ with an alphabet by*

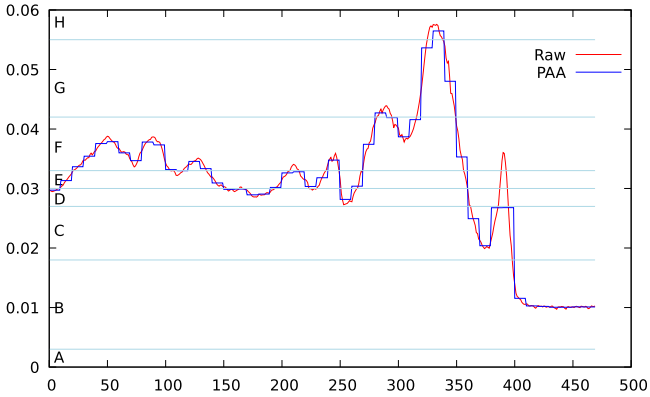


Fig. 2. An example of PAA with $\omega = 10$ and SAX with $|\Sigma| = 8$ from the **Beef** dataset.

the breakpoints of values following the Gaussian distribution [13]: The alphabet $\Sigma = \{A_1, A_2, \ldots, A_d\}$, the size of Σ is the number of characters (a.k.a symbols), denoted as $|\Sigma|$.

Example 1. Fig. 2 shows a time series data instance (red solid line) from the **Beef** dataset with the length $n = 470$. The PAA sequence (blue solid line) is with $\omega = 10$, and the SAX sequence is **DEFFF**$\cdots$**BBBBB** (breakpoints are the light blue dash lines) with $|\Sigma| = 8$ characters.

Definition 7 (Shapelet S_i [24]). *A shapelet S_i of the length L in class C_j, where $C_j \in \mathcal{C}$, is a time series subsequence, which can represent class C_j and discriminate C_j from other classes, i.e., $\mathcal{C} \setminus \{C_j\}$. That is, $\mathsf{dist}(T_j, S_i)$ is smaller for all $\mathsf{label}(T_j) = C_j$, whereas $\mathsf{dist}(T_{j'}, S_i)$ is larger for all $\mathsf{label}(T_{j'}) = \mathcal{C} \setminus \{C_j\}$.*

According to Eqn. (1), the distance between the j-th time series T_j and a shapelet candidate S_i of length L_{S_i} is defined as follows:

$$d_i^j = \mathsf{dist}(T_j, S_i), \tag{3}$$

Definition 8 (Shapelet Transformation [18]). *Shapelet transformation is a method to transform a time series T_j, w.r.t. the shapelets $\mathcal{S} : \{S_1, \ldots, S_k\}$, into a new data space $(d_1^j, \cdots, d_k^j)$, where $d_i^j = \mathsf{dist}(T_j, S_i)$, and $\mathsf{dist}(T_j, S_i)$ is the distance between T_j and a shapelet S_i in $\mathcal{S}$.*

For simplicity, we also use $\mathsf{dist}(T_j, \mathcal{S})$ to denote the distance between T_j and $\mathcal{S}$, *i.e.*, $\mathsf{dist}(T_j, \mathcal{S}) = (d_1^j, \cdots, d_k^j)$. Shapelets can be applied to reduce the dataset by using the transformation in Definition 8. The number of data values of the dataset D is reduced from $m \times n$ to $m \times k$, where k is often much smaller than n.

Putting these together, we are ready to present the problem statement below.

Problem Statement. Given an integer k and a time series dataset D, where each time series has a class label, we aim to efficiently discover the shapelets $\mathcal{S} : \{S_1, \ldots, S_{k'}\}$, $k' \leq k$, for each class, such that

$$\arg\max_{\mathcal{S}} \sum_{T,T' \in D \mathsf{label}(T) \neq \mathsf{label}(T')} ||\mathsf{dist}(T, \mathcal{S}) - \mathsf{dist}(T', \mathcal{S})||. \tag{4}$$

The shapelets discovered are used to learn a classification model. The details are presented in Appendix, available in the online supplemental material.

4 Efficient SAX Subsequence Computation

In this section, we present the details of the BSPCOVER approach for discovering a set of SAX subsequences (a.k.a *SAX words*) of high-quality for model building. For presentation clarity, we present BSPCOVER using Algorithm 1-5.

Algorithm 1. SAX Transformation for Class C

Input: Time series dataset $D = T^{m \times n}$
Output: SAX words set Ω_C for each class C
Initialize $\Omega_C = \emptyset$;
$\omega = \pi(n)$ // Step function shown in Fig. 3
$\phi = \frac{L_{\min}}{\lceil \frac{n}{\omega} \rceil}$;
foreach j $\in \{1, 2, \ldots, m\}$ **do**
  {PAA and SAX sequence generation}
  foreach i $\in \{1, 2, \ldots, \lceil \frac{n}{\omega} \rceil\}$ **do**
    foreach r $\in \{\lceil (i-1)\omega + 1 \rceil, \ldots, \lceil i\omega \rceil\}$ **do**
      $\bar{T}^{j \times i} = \bar{T}^{j \times i} + T^{j,r}$;
    $\bar{T}^{j \times i} = \frac{\bar{T}^{j \times i}}{\omega}$;
    $\hat{T}^{j \times i} =$ SAX lookup table$(D, \bar{T}^{j \times i})$ [16];
  {SAX words generation}
  foreach $l \in \{\phi \cdot \lceil \frac{n}{\omega} \rceil, 2 \cdot \phi \cdot \lceil \frac{n}{\omega} \rceil, \ldots, \lceil \frac{n}{\omega} \rceil\}$ **do**
    foreach x $\in \{1, 2, \ldots, \lceil \frac{n}{\omega} \rceil - l + 1\}$ **do**
      $\hat{e} = \hat{T}(x, x + l - 1)$;
      $\Omega_C = \Omega_C + \hat{e}$;
return Ω_C

Algorithm 2. Construction of Bloom Filters and the SAX Words' bitmaps of Class C

Input: SAX words set Ω_C
Output: Candidate set with their bitmaps stored in bloom filter $\Omega_C^{\mathcal{BF}}$
{Constructing bloom filters}
Initialize $\mathcal{BF}_C$ for each C in $\mathcal{C}$;
foreach C $\in \mathcal{C}$ **do**
  foreach $\hat{e} \in \Omega_C$ **do**
  $\mathcal{BF}_C$.add$(\hat{e})$ // using MD5 hash functions

{Pruning duplicate SAX words}
foreach C $\in \mathcal{C}$ **do**
  Initialize $\Omega_C^{\mathcal{BF}} = \emptyset$;
  foreach $\hat{e} \in \Omega_C$ **do**
    if $\bigvee_{\bar{C} = \mathcal{C} \setminus \{C\}} \mathcal{BF}_{\bar{C}}$.lookup$(\hat{e})$ == false **then**
      $\Omega_C^{\mathcal{BF}}$.add$(\hat{e})$;

{Constructing bitmaps}
foreach C $\in \mathcal{C}$ **do**
  foreach $\hat{e} \in \Omega_C^{\mathcal{BF}}$ **do**
    $\hat{e}$.bitmap $= 0^{|D_C|}$;
    for $j \in \{1, 2, \ldots, |D_C|\}$ **do**
      if $\hat{e} \in \hat{T}_j$ **then**
        $\hat{e}$.bitmap$(j) = 1$;
return $\Omega_C^{\mathcal{BF}}$

Overview. The raw time series data are reduced to SAX sequences, then a sliding window is used to generate SAX words of variable lengths (Algorithm 1). Mining some high-quality SAX words from voluminous SAX words is inefficient. Our main techniques compute a small set of SAX word candidates as follows. **I.**We construct a bloom filter for each class to efficiently prune the same SAX words that exist in all classes,

and build a bitmap structure for each remaining SAX word (Algorithm 2); **II.**We prune similar SAX words that appear in all classes, and then define a weighted bitmap of each SAX word (Algorithm 3); and **III.**We formulate the discovery problem as a weighted set cover problem, and propose a heuristic method to solve it (Algorithm 4 and Algorithm 5).

Algorithm 3. Computing the Weights for Non-Similar SAX Words

Input: Candidates set with bloom filter $\Omega^{\mathcal{BF}}$
Output: Weighted candidates after pruning, Ω^{sim}
{Pruning similar words}
foreach $C \in \mathcal{C}$ **do**
  foreach $\hat{e}_1 \in \Omega_C^{\mathcal{BF}}$ **do**
    foreach $\hat{e}_2 \in \Omega_{\mathcal{C}\setminus\{C\}}^{\mathcal{BF}}$ **do**
      if isSimilar $(\hat{e}_1, \hat{e}_2)$ **then**
        $\Omega_{\mathcal{C}\setminus\{C\}}^{\mathcal{BF}}$.remove$(\hat{e}_2)$;
        flag = true;
    if flag then
      $\Omega_C^{\mathcal{BF}}$.remove$(\hat{e}_1)$;

{Counting the weights of remaining SAX words}
foreach $C \in \mathcal{C}$ **do**
  Initialize $\Omega_C^{sim} = \emptyset$;
  while $\Omega_C^{\mathcal{BF}} \neq \emptyset$ **do**
    $\hat{e}_1 = \Omega_C^{\mathcal{BF}}$.pop() ;
    Ω_C^{sim}.add$(\hat{e}_1)$;
    foreach $\hat{e}_2 \in \Omega_C^{\mathcal{BF}} \setminus \{\hat{e}_1\}$ **do**
      if isSimilar$(\hat{e}_1, \hat{e}_2)$ **then**
        $\hat{e}_1$.weight++ ;
        $\hat{e}_2$.weight++ ;
return Ω^{sim}

Algorithm 4. OneSaxCover: Heuristic Algorithm for the Weighted Bitmap Cover Problem

Input: Candidate set after pruning Ω^{sim}
Output: Final candidate set Ω_C^{fin}, for all $C \in \mathcal{C}$
foreach $C \in \mathcal{C}$ **do**
Initialize $\Omega_C^{fin} = \emptyset$;
Bitmap$_C = 0^{|D_C|}$ // all pos. are uncovered
Ω_C^{sim}.sort() // by the number of 1 values and then the weight of bitmap of SAX words in Ω_C^{sim}
{Select k SAX words}
do
  $\hat{e}_1 = \Omega_C^{sim}$.pop();
  Ω_C^{fin}.add$(\hat{e}_1)$;
  Bitmap$_C$ = Bitmap$_C$ + $\hat{e}_1$.bitmap // mark covered pos. by $\hat{e}_1$
  if Bitmap$_C == 1^{|D_C|}$ **then**
    break // fully covered
  else
    {Pruning similar SAX words}
    foreach $\hat{e}_2 \in \Omega_C^{sim} \setminus \{\hat{e}_1\}$ **do**
      if isSimilar$(\hat{e}_1, \hat{e}_2)$ **then**
        Ω_C^{sim}.remove$(\hat{e}_2)$;
    Pos = Bitmap$_C$.pos() // uncovered pos.
    Ω_C^{sim}.sort(Pos) // by the number of 1 values from uncovered pos., then the weight of bitmap of SAX words in Ω_C^{sim}
while Bitmap$_C \neq 1$ // pos. are not fully covered
return Ω_C^{fin} (for all $C \in \mathcal{C}$)

Algorithm 5. PSaxCover: Determining p-Cover SAX Words

Input: Ω^{sim}, p, k
Output: Final candidates Ω^{fin}
Initialize $\Omega^{fin} = \emptyset$ and $i = 0$;
while $i < p$ **do**
  $\Omega^{fin'}$ = OneSaxCover(Ω^{sim}) ;
  Ω^{fin} = $\Omega^{fin} + \Omega^{fin'}$;
  Ω^{sim} = $\Omega^{sim} - \Omega^{fin'}$;
  i++ ;
{Pruning Ω_C^{fin} to avoid overly-covered time series}
if $|\Omega_C^{fin}| > k$ **then**
  foreach $C \in \mathcal{C}$ **do**
    Initialize Bitmap$_C = 0^{|D_C|}$;
    foreach $\hat{e} \in \Omega_C^{fin}$ **do**
      Bitmap$_C$ = Bitmap$_C$ + $\hat{e}$.bitmap ;
    while $\exists\, j$ s.t. Bitmap$_C[j] > p$ and $|\Omega_C^{fin}| > k$ **do**
      $\hat{e} = \Omega_C^{fin}$.mincost() ;
      if $\hat{e}$ is null **then**
        break ;
      else
        Ω_C^{fin}.remove$(\hat{e})$;
        Bitmap$_C$ = Bitmap$_C$ − $\hat{e}$.bitmap ;
return Ω^{fin}

4.1 SAX Transformation

For completeness, this subsection recalls the relevant details of the transformation this paper adopts. BSPCOVER transforms raw time series data into PAA sequences, SAX sequences, and then SAX words to reduce their information but enhance processing efficiency. The parameters of the transformation can be easily set, since excessive transformed SAX words are further pruned by our techniques.

We present the procedure for compressing each time series T of a dataset D in Algorithm 1. It applies PAA to the raw time series data through averaging ω consecutive data values (Lines 5-9). After generating the PAA sequence $\bar{T}$, Algorithm 1 transforms $\bar{T}$ to SAX sequence $\hat{T}$ by replacing the PAA sequence value from a lookup table [16] (Line 10). It then generates SAX words through sliding windows (Lines 12-15), whose lengths are determined by the minimum sliding window ratio ϕ (Line 12). We then elaborate upon the two parameters ω and ϕ.

Determining ω. According to the finding in [10], when ω increases from 1 to 5, the shapelet discovery time is significantly reduced and the accuracy remains fairly stable. With ω increasing over 5, the discovery time of shapelets shows no obvious decline but the accuracy deteriorates slightly. Next, according to the experiment results from related studies [19], [20], few shapelets are of the full length of the raw time series. We obtain the same findings from our experiments.

Based on these two findings, to preserve more information of time series, and meanwhile to improve efficiency, we set ω to range from $\omega_{\min} = 1$ to $\omega_{\max} = 5$ for the different datasets, and specify a step function of ω for n. For instance, the step function π used in our experiments for determining ω is shown in Fig. 3.

Determining ϕ. The minimum sliding window ratio ϕ can be determined for the different datasets as follows. If the lengths of subsequences are smaller than $L_{\min}$, normally 10,

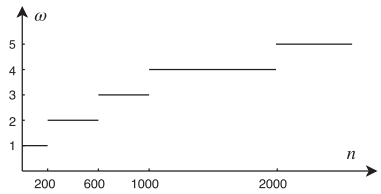


Fig. 3. The step function π of ω.

their discriminative power is small. Thus, given the minimum length of the subsequence, the minimum sliding window ratio of each dataset can be computed in Line 3 of Algorithm 1 and sliding windows of various lengths are generated in Line 12. By default, $L_{\min}$ is set to $n \times 10\%$, denoted as $\phi = 10\%$. If the default value of $L_{\min}$ is smaller than 10, we then set $L_{\min}$ to 10 to deduce the ϕ.

Furthermore, it is not necessary to consider subsequences of all possible lengths but those with a difference of ϕ. The rationale is that the discriminative power of subsequence $T(p, q)$ is similar to the subsequence $T(p, q + \Delta)$, if Δ is tiny,

$$\text{dist}(T(p, q), T(p, q + \Delta)) < \tau, \tag{5}$$

where τ is a small value.

If the distance between two subsequences is smaller than τ, their discriminative powers are considered similar. Thus, the minimum sliding window ratio is used to generate SAX words (Lines 3 and 12).

Example 2. The SAX transformation process, from raw time series to PAA sequences and SAX words, is shown in Fig. 4. Suppose the instance belongs to a class, with a raw time series length $n = 8$, a compression ratio $\omega = 2$ and an alphabet size $|\Sigma| = 4$, the raw time series is transformed into **CDBD**. The minimum sliding window ratio $\phi = 0.5$; thus, the sliding window sizes are 2 and 4. The SAX words are {**CD, DB, BD, CDBD**} and are appended to the SAX words set Ω_1.

Complexity. The complexity of Algorithm 1 is $O(mn)$, which is composed of two parts. The first PAA sequence part (Lines 6-10) is $O(mn)$; and the time complexity of SAX words generation part (Lines 12-15) is $O(\frac{mn}{\omega\phi})$.

4.2 Bloom Filter and Bitmap of SAX Words

Because of the SAX transformation of Section 4.1, the number of SAX words generated by Algorithm 1 is much larger than those of LTS [9] and ELIS [7]. Only three lengths (at most) of shapelets exist in LTS, which are smaller than one-fifth of ours on average. The maximum length of the compressed times series in ELIS is 50 for the PAA sequences in all datasets, which indicates that the maximum ω of ELIS is over 10. Identical SAX words often exist in different classes, which do not have discriminative power. If these SAX words are selected as candidates for model building, the accuracy does not improve.

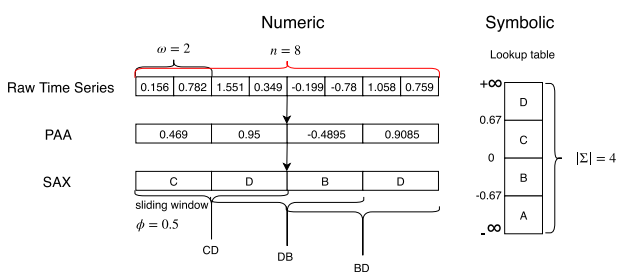


Fig. 4. An example of SAX transformation.

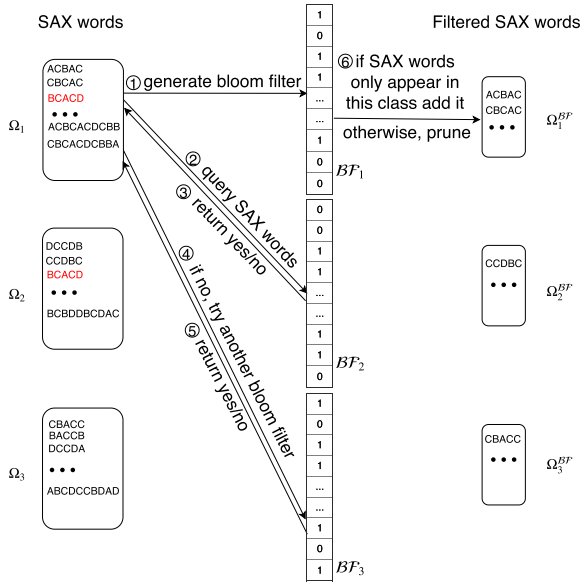


Fig. 5. Pruning redundant and non-discriminative SAX words (of 3 classes) using bloom filters.

We propose to use bloom filters in BSPCOVER to efficiently prune the SAX words that do not have discrimination, i.e., those present in all classes.[1] The bloom filters using 13 independent MD5 hash functions are constructed for each class, to achieve low false positive probability (Lines 3-5, Algorithm 2).

4.2.1 Filtering Non-Discriminative SAX Words

In the filtering step, each SAX word is treated as the query. For class C, all the SAX words are taken as the query for bloom filters of other classes. There are two types of query results. The first is "*possibly in set*", which means the SAX word exists in other classes with high probability and is removed in all classes' candidates. The other is "*definitely not in set*", which indicates that this SAX word is specifically found in its own class, and therefore, can only represent class C. Thus, it is kept in $\Omega_C^{\mathcal{BF}}$ for further processing in Section 4.3. The process of pruning is shown in Lines 8-12 in Algorithm 2.

Example 3. The function of bloom filters is illustrated with Fig. 5. We generate a bloom filter for each class (shown in ①) and take one SAX word $\hat{e}$ from Ω_1 as the query for the other two bloom filters generated by Ω_2 and Ω_3 in ② and ④. Two kinds of operations are shown in ⑥. If the SAX word can pass the bloom filters, this SAX word $\hat{e}$ may have high discriminative power and can be inserted into the candidates set $\Omega_1^{\mathcal{BF}}$. Otherwise, the SAX word is

1. To the best of our knowledge, this paper is the first work to use bloom filters to prune SAX words in shapelet discovery. A previous paper [3] utilized bloom filters to store the SAX words for motif discovery.

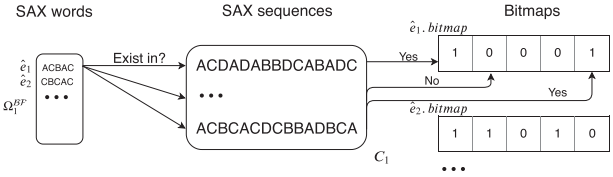


Fig. 6. An example of (partial) constructing bitmaps of SAX words in Class 1 (C_1).

pruned, such as **BCACD**, which exists in Ω_1 and Ω_2. Then, we continue the next SAX word until all words in Ω_1 are processed and carry out the same operation for SAX words of Ω_2 and Ω_3.

4.2.2 Building Compact Representation of SAX Words

To quantify the quality of candidate SAX words $\Omega_C^{\mathcal{BF}}$ of a class C, we propose to generate a bitmap structure for each SAX word $\hat{e}$ (Lines 15-20 in Algorithm 2). The length of bitmap is the number of instances in the class $|D_C|$ and the initial value of each bit is set to 0. If $\hat{e}$ exists in the sequence $\hat{T}_j$, then $\hat{e}.\mathsf{bitmap}(j) = 1$ and we say that $\hat{e}$ *covers* $\hat{T}_j$. That is, the value of jth position of bitmap indicates whether the SAX word candidate covers $\hat{T}_j$. The larger the number of 1 values of the bitmap of $\hat{e}$, the more instances $\hat{e}$ covers and the better the quality of $\hat{e}$.

Algorithm 2 computes the bitmaps of all SAX words in Lines 15-20. In the end, we obtain the candidates set and their bitmaps. Fig. 6 shows an example of (partial) bitmaps for $\Omega_1^{\mathcal{BF}}$.

Complexity. The complexity of Algorithm 2 is $O(\frac{mn}{\omega\phi}|D_C|)$, which is composed of three parts: Lines 3-5 of $O(\frac{mn}{\omega\phi})$, Lines 8-12 of $O(\frac{mn}{\omega\phi})$ and Lines 15-20 of $O(\frac{mn}{\omega\phi}|D_C|)$.

4.3 Similar SAX Word Pruning

In the previous subsection, the bloom filter is proposed to prune duplicated SAX words that have low discriminative power. There are still often numerous similar SAX words in different classes. For example, after the pruning, over 50% of all SAX words of the dataset **Beef** are similar. Such similar words in different classes' shapelet candidates evidently reduce both efficiency and accuracy.

A natural technique for reducing excessive SAX words for model building is to prune similar SAX words that exist in all classes. There are two technical details. First, similarity measures of time series violate triangle inequality. That is, if T_1 is similar to T_2 and T_2 is similar to T_3, T_1 is not necessarily similar to T_3. Second, it is required to differentiate the similar SAX words that exist in self class and other classes and to select one SAX word to represent similar ones in self class. In this subsection, we define a non-metric distance measure for the similarity measure of SAX words. As for the second technical issue, we propose to use term frequency as the weight of a SAX word to quantify the quality of SAX words. The greater the weight, the better the quality.

In Definition 10, we propose a non-metric distance for SAX words. When we use it to determine similar SAX words (Definition 11), we propose a threshold, which is dependent on the words' lengths (specifically, $\frac{l}{\phi\cdot\lceil\frac{n}{\omega}\rceil}$).

Definition 9 (SAX L1 Distance (L1D)). *Given two SAX words with the same length, $\hat{e}_1 = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_l)$ and $\hat{e}_2 = (\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_l)$, the* SAX L1 distance between $\hat{e}_1$ and $\hat{e}_2$ *is defined as follows.*

$$\mathrm{L1D}(\hat{e}_1, \hat{e}_2) = \Sigma_{i=1}^{l}|\hat{x}_i - \hat{y}_i|, \tag{6}$$

Definition 10 (Matching Distance of SAX Words (MATCHD)). *Given two SAX words of the same length, $\hat{e}_1 = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_l)$ and $\hat{e}_2 = (\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_l)$, four kinds of subsequences are defined as $\hat{e}_1^{1+i} = (\hat{x}_{1+i}, \hat{x}_{2+i}, \ldots, \hat{x}_l)$, $\hat{e}_2^{l-i} = (\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_{l-i})$, $\hat{e}_1^{l-i} = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_{l-i})$, and $\hat{e}_2^{1+i} = (\hat{y}_{1+i}, \hat{y}_{2+i}, \ldots, \hat{y}_l)$, where $i \in [1, \frac{l}{\phi\cdot\lceil\frac{n}{\omega}\rceil}]$.*

The matching distance of SAX words $\hat{e}_1$ and $\hat{e}_2$, *denoted as* MATCHD$(\hat{e}_1, \hat{e}_2)$, *is defined below.*

$$\textsc{matchd}(\hat{e}_1, \hat{e}_2) = \min\{\mathrm{L1D}(\hat{e}_1^{1+i}, \hat{e}_2^{l-i}), \mathrm{L1D}(\hat{e}_2^{1+i}, \hat{e}_1^{l-i})\}, \tag{7}$$

Some SAX words do not match well may be just because they are generated by sliding window and they do not align well. Hence, we generate $\hat{e}_1^{1+i}$ and $\hat{e}_1^{l-i}$ by removing i characters ($i \in [1, \frac{l}{\phi\cdot\lceil\frac{n}{\omega}\rceil}]$) from the beginning and the end of $\hat{e}_1$. We generate $\hat{e}_2^{1+i}$ and $\hat{e}_2^{l-i}$ similarly. MATCHD uses them to compute the smallest distance between the matching of the middle parts of $\hat{e}_1$ and $\hat{e}_2$. Such definition leads to effective pruning of similar SAX words.

Example 4. Consider the dataset **Beef** and two of its SAX words $\hat{e}_1$ = **BCCDA** and $\hat{e}_2$ = **CCDAA**. **I.** For the L1 distance, L1D($\hat{e}_1, \hat{e}_2$) = 1+0+1+3+0 = 5; **II.** For MATCHD, the maximum removing number of these two SAX words is $\frac{5}{\lceil 0.05\times\frac{470}{5}\rceil} = 1$, MATCHD between them is the minimum of two following situations, one is removing the first character of $\hat{e}_1$ and the last character of $\hat{e}_2$ L1D(**BCCD**, **CCDA**) = 5; another is L1D(**CCDA**, **CCDA**) = 0. Hence, MATCHD$(\hat{e}_1, \hat{e}_2)$ = 0.

Definition 11 (Similar SAX Words). *Given two SAX words of the same length, $\hat{e}_1 = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_l)$ and $\hat{e}_2 = (\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_l)$, ISSIMILAR is defined as follows:* ISSIMILAR$(\hat{e}_1, \hat{e}_2) =$

$$\begin{cases} \mathsf{true}, & \textsc{matchd}(\hat{e}_1, \hat{e}_2) = 0, \text{or} \\ & \mathrm{L1D}(\hat{e}_1, \hat{e}_2) \le \frac{l}{\phi\cdot\lceil\frac{n}{\omega}\rceil} \text{ and } \bigwedge_{i\in[1,l]} |\hat{x}_i - \hat{y}_i| \le 1, \\ \mathsf{false}, & \text{otherwise.} \end{cases} \tag{8}$$

Formula 8 states that $\hat{e}_1$ and $\hat{e}_2$ are similar if there is a match using MATCHD. Alternatively, two SAX words are similar if they are of the same length l and the distance (Definition 9) is no larger than a ratio between l and the minimum sliding window size $\phi\cdot\lceil\frac{n}{\omega}\rceil$. Since the SAX words are generated by $\phi\cdot\lceil\frac{n}{\omega}\rceil$, l can be divisible by the minimum sliding window size (Line 3, Algorithm 1).

The pseudocode of pruning similar SAX words and counting the weights of the remaining SAX words are shown in Algorithm 3. If a SAX word $\hat{e}_1$ has similar SAX words in another class (Line 5), Algorithm 3 prunes those similar words, according to Eqn. (8) (Lines 6-7). $\hat{e}_1$ is also pruned (Line 10) because it is not discriminative.

Next, in Lines 12-21, we compute the weight of each SAX word, which records the number of similar SAX words in its class. Algorithm 3 records each weighted SAX word in the set Ω_C^{sim}. Ω_C^{sim} is initialized to an empty set (Line 14). Algorithm 3 calculates the similarity with Eqn. (8) between

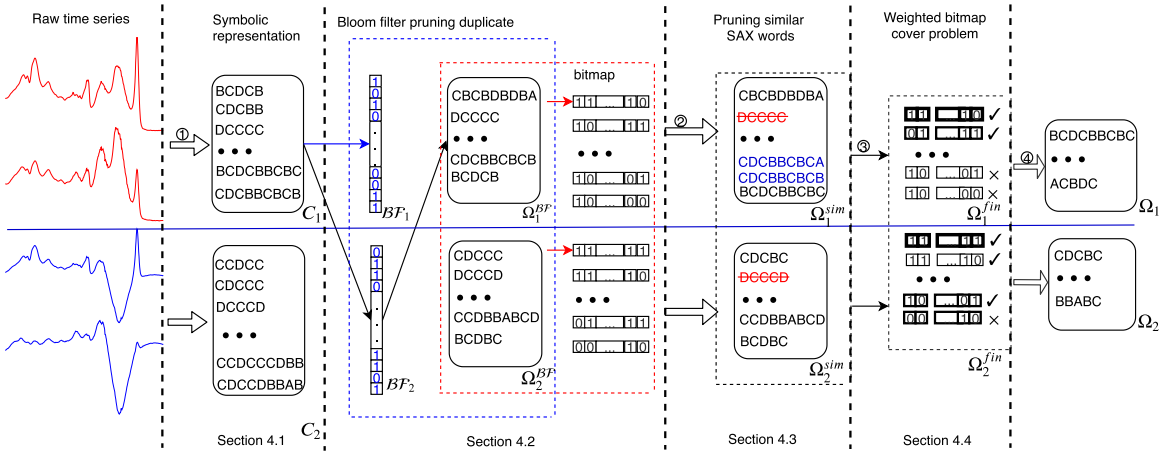


Fig. 7. An example of finding SAX words for model building (The figure is best viewed in color).

each SAX word and others in $\Omega_C^{\mathcal{BF}}$ (Lines 18-19) and updates the weight (Lines 20-21).

Complexity. The complexities of Lines 1-10 and Lines 12-21 are both $O(m(\frac{n}{\omega\phi})^2)$. Hence, the time complexity of Algorithm 3 is $O(m(\frac{n}{\omega\phi})^2)$.

4.4 SAX Word Cover for Model Building

In this subsection, the details of the BSPCOVER method to compute SAX words for model building are presented. In a nutshell, the problem is formulated as a weighted bitmap cover problem, through finding the minimum subset to cover the time series instances rather than discovering the SAX words directly. We provide some definitions for explaining the approach below.

Definition 12 (p-Cover). *A SAX word $\hat{e}$* covers *a SAX sequence $\hat{T}_j$* if and only if *there is a subsequence of $\hat{T}_j$ that is identical to $\hat{e}$. SAX sequence $\hat{T}_j$ is p-Covered* if and only if*$\hat{T}_j$ is covered by p distinct SAX words.*

To implement the covering relationships between SAX words and SAX sequences of class C, we utilize the bitmap structure. The number of bits of bitmap is $|D_C|$, and $\hat{e}$.bitmap $(j) = 1$ if and only if $\hat{e}$ covers $\hat{T}_j \in D_C$.

Definition 13 (Weighted Bitmap Cover (WBC) Problem). *Given a dataset D_C and a set of SAX words $\Omega_C^{sim} = \{\hat{e}_1, \ldots, \hat{e}_q\}$, the* weighted bitmap cover *problem is to determine a subset Ω^{fin} of Ω_C^{sim} such that:*

- *all $\hat{T}_j$ are p-covered by Ω^{fin}, and*
- *the total* weight *of Ω^{fin} is maximized.*

When $p = 1$, the weighted bitmap cover problem is equivalent to the weighted set cover problem, which is an NP-complete problem. The p value of the WBC problem is not smaller than 1 and hence, WBC is also an NP-complete problem.

It is known that the weighted set cover problem can be practically solved by Chvatal [5], which follows one rule: choosing the set that contains the largest number of uncovered elements at each stage. The approximation ratio is $\log q$, where q is the number of SAX subsequences in Ω_C^{sim}. Therefore, we extend the greedy algorithm to Algorithm 4 to solve the WBC.

Algorithm 4 finds candidate SAX words for each class C. It initializes the final SAX word set Ω_C^{fin} and a special bitmap, called Bitmap_C. Its length is $|D_C|$ and all of its bits have a value of 0 (Lines 3-4), where Bitmap$[i] = 1$ when the instance has been covered by some of the SAX words in Ω_C^{fin}, or 0 otherwise. Next, Algorithm 4 sorts the SAX words based on the time series they cover and then the number of similar SAX words (i.e., the number of 1 values and weight of their bitmaps) in descending order (Line 5). A larger number of 1 values and a greater weight indicate that the SAX word is of a higher quality.

Algorithm 4 selects the next SAX words from Ω_C^{sim}, adds it into Ω_C^{fin} (Lines 8-9), then adds its bitmap to Bitmap (Line 10). Then, if Bitmap = 1, all instances of the class are covered by the current SAX words, and Algorithm 4 breaks the loop and finds SAX words for the next class (Lines 11-12). Otherwise, Bitmap is not equal to 1 and the similar SAX words of $\hat{e}_1$ in Ω_C^{sim} are removed so that they are not passed to the potentially costly model building phase (Lines 15-17). The remaining candidates are sorted, similar to Line 5, at the positions that are not covered by Ω_C^{sim} (Lines 18-19). Then, Algorithm 4 processes the remaining SAX words iteratively, from Line 20.

Analysis. The complexity of Algorithm 4 is $O(m(\frac{n}{\omega\phi})^2)$. The approximation ratio of Algorithm 4 is $\log q$.

Example 5. Fig. 7 shows the *whole* process of finding SAX words from raw time series. All the time series instances are transformed into SAX words, as shown in ①. Then, the SAX words are utilized to build the bloom filter $\mathcal{BF}_C$ for each class C, which prunes identical SAX words in other classes (blue dotted rectangle). Next, the bitmap structures for the remaining SAX words are built to quantify their qualities (red dotted rectangle). ② The procedure prunes similar SAX words in all classes. The red SAX words **DCCCC** and **DCCCD** of different classes are similar, whereas the blue SAX words of the same class are similar. The red ones are pruned and the blue ones are passed to compute their weights. ③ Algorithm 4 utilizes the weight bitmap structures (the last rectangle) to select the minimum subset of bitmaps to cover all the instances.

Algorithm 5 (Lines 2-7) applies Algorithm 4 p times to determine the p-cover SAX words. The approximation ratio is $(\log q)^p$.

It should be noted that the SAX words Ω_C^{fin} after the loop (Lines 3-7) may cover some time series instances more than

p times. A large Ω_C^{fin} reduces the efficiency of model building. Hence, if Ω_C^{fin} has more than k shapelets, we greedily prune some SAX words (Lines 13-19 of Algorithm 5). This is again implemented by a bitmap Bitmap_C. $\mathsf{Bitmap}_C[j]$ records the number of times T_j is covered by Ω_C^{fin} (Lines 11-12). If there is a T_j covered more than p times and the number of shapelets is still larger than k (Line 13), we implement a function mincost (Line 14) to compute an $\hat{e}$ has the smallest cost and after the removal of $\hat{e}$, Ω^{fin} is still a p-cover. The cost of $\hat{e}$ is the number of time series $\hat{e}$ covers. If there is such an $\hat{e}$, we remove it from Ω_C^{fin} (Lines 18-19).

4.5 Parameter Settings

Determining p. The pseudocode of Algorithm 6 shows how to determine p. In Lines 3 and 6, BSPCOVER denotes the composition of Algorithm 1-5. First, we call BSPCOVER using $p = 1$ and then learn a model to compute its accuracy. In the do-while loop, Algorithm 6 computes the model at $p + 1$ (Lines 6-7), and the ratio between accuracy and time (Line 8). If the ratio is smaller than the user-specified threshold (Δ), p is returned (Lines 9-10, 15). Otherwise, the current accuracy of p is updated by that of $p + 1$, the value of p is updated and the loop continues (Lines 11-14).

Algorithm 6. Determining p for BSPCOVER

Input: Accuracy threshold Δ, and time series dataset D, number of shapelets k
Output: p
1 Initialize $p = 1$;
2 $\Omega^{fin} = \text{BSPCOVER}(D, p, k)$;
3 $acc_p = \text{SHAPELETSLEARNING}(\Omega^{fin})$ // LTS [9]
4 **do**
5 $\Omega^{fin'} = \text{BSPCOVER}(D, p + 1, k)$;
6 $acc_{p+1} = \text{SHAPELETSLEARNING}(\Omega^{fin'})$;
7 $\Delta_{cur} = \frac{acc_{p+1} - acc_p}{T_{p+1} - T_p}$;
8 **if** $\Delta_{cur} > \Delta$ **then**
9 **break** ;
10 $acc_p = acc_{p+1}$;
11 $p = p + 1$;
12 $\Omega^{fin} = \Omega^{fin'}$;
13 **while** true
14 **return** p

Cost Model for Setting Parameters. We then present a cost model for BSPCOVER to analyze its efficiency for a dataset D. The efficiency of BSPCOVER is mainly contributed by three components: **I.** the number of iterations I; **II.** the number of cover times p; and **III.** other factors ϵ, such as program initialization and operating system. From the experiments, we observe that the relationships between time and the factors are linear. Thus, the total time cost can be estimated as follows.

$$T_{\text{BSPCOVER}} = T_I + T_p + \epsilon = c_1 I + c_2 p + \epsilon, \quad (9)$$

where both I and $p \in \mathbb{N}$. T_I represents the time of iterations; T_p denotes the time caused by the covering time p; and $c_{i,i\in 1,2}$ are the constants scaling factors for I and p.

Remarks. The SAX words discovered by BSPCOVER (Ω^{fin}) are passed to a model learning algorithm. This paper has revised [9]. For completeness, we summarize the algorithm in Appendix, available in the online supplemental material. We remark that BSPCOVER can be incorporated with other model learning algorithms.

5 Experimental Results

This section presents an experimental evaluation of BSPCOVER applied to the well-known benchmark datasets used for TSC research. The major findings are that BSPCOVER is both efficient and accurate, more so than two methods used as benchmark classifiers, Rotation Forest [1] and 1NN-DTW [21], and many other shapelet-based methods including ST [18], LTS [9], ELIS [7], Fast shapelets [20], Scalable Shapelet Discovery [10] and COTE [2]. In addition, we present some performance characteristics of BSPCOVER by varying some important parameters. The source code used for this experiment is publicly available at https://goo.gl/sN9Kz7.

5.1 Environment

We implemented the proposed algorithms in JAVA. The model building (shapelet learning) part was implemented using multithreading, whereas other parts of our algorithms were single-threaded. All the experiments were conducted on a machine with a Xeon E5-2630 v4 @ 2.2 GHz (2S/10C)/ 256 GB RAM/128 GB SWAP, running on CentOS 7.6 (64-bit). The parameters of the bloom filters of each dataset can be easily set because they are memory efficient.

5.2 Datasets and Parameters

A well-known benchmark of TSC datasets, namely UCRARCHIVE [4], was tested. Due to space limitations, we selected 45 representative datasets from the data type *Device (4), ECG (5), Image (11), Motion (7), Sensor (8), Stimulated (5), and Spectro (5)*; the number in parentheses is the number of datasets of the type (about half of each data type). Table 3 shows the settings of the experiments with the datasets, where *Train*, *Test*, *Class*, and *Length* are the numbers of time series in the training set and the testing set, the number of classes, and the length of time series, respectively.

By running experiments and referencing previous work [7] [9], we tuned several important parameters of our method, including the learning rate η, the regularization λ, the number of iterations I in shapelet learning and the optimal p of the WBC algorithm. The default value of η was set to a small value 0.1 in consideration of the performance and the chances of reaching local minima. λ was chosen from $\{0.01, 0.1, 1\}$, while I was selected from $\{1000, 2000, 3000, 5000\}$. The range of p was from [1,6]. The values of the parameters of each dataset are shown in Table 3.

5.3 Baselines

We compared BSPCOVER with ten different methods. Due to space restrictions, we provide only brief details of each method. Interested readers may refer to the original paper for details.

- *Two benchmarks [1].* Two benchmark classifiers (**RotF** and **DTW-Rn-1NN**) are more competitive than many methods. Hence, we include them in our experiments.
- *Shapelet Transformation (ST) [18].* This method combines a weighted ensemble of standard classifiers,

TABLE 3
Datasets and Parameters

Dataset	Train	Test	Class	Length	λ	I	p
ArrowHead	36	175	3	251	0.01	1000	5
Beef	30	30	5	470	0.01	1000	5
Beetle/Fly	20	20	2	512	0.01	1000	6
CBF	30	900	3	128	0.01	1000	4
ChlorineConcentration	467	3840	3	166	0.01	3000	3
Coffee	28	28	2	286	0.01	1000	2
Computers	250	250	2	720	0.1	1000	3
CricketZ	390	390	12	300	0.1	1000	5
DiatomSizeReduction	16	306	4	345	0.01	3000	5
DistalPhalanxOutlineCorrect	276	600	2	80	0.01	1000	1
Earthquakes	139	322	2	512	0.1	1000	1
ECG200	100	100	2	96	0.1	1000	2
ECG5000	500	4500	5	140	0.1	1000	1
ECGFiveDays	23	861	2	136	0.01	1000	1
ElectricDevices	8926	7711	7	96	0.01	1000	1
FaceAll	560	1690	14	131	0.01	1000	3
FaceFour	24	88	4	350	1	1000	4
FacesUCR	200	205	14	131	1	3000	6
FordA	1320	3601	2	500	0.1	1000	1
GunPoint	50	150	2	150	0.1	1000	3
Ham	109	105	2	431	0.1	1000	1
HandOutlines	370	1000	2	2709	0.01	1000	1
Haptics	155	308	5	1092	0.1	1000	2
InlineSkate	100	550	7	1882	0.01	1000	6
InsectWingbeatSound	220	1980	11	256	0.01	1000	2
LargeKitchenAppliances	375	375	3	720	0.1	1000	2
Mallat	55	2345	8	1024	0.01	1000	3
Meat	60	60	3	448	0.1	1000	2
NonInvasiveFetalECGThorax1	1800	1965	42	750	0.1	1000	1
OSULeaf	200	242	6	427	0.1	3000	4
Phoneme	214	1896	39	1024	0.01	1000	1
RefrigerationDevices	375	375	3	720	0.01	1000	1
ShapeletSim	20	180	2	500	0.1	5000	6
SonyAIBORobotSurface1	20	601	2	70	0.01	5000	5
SonyAIBORobotSurface2	27	953	2	65	0.01	1000	3
Strawberry	370	613	2	235	0.1	1000	4
Symbols	25	995	6	398	0.1	1000	4
SyntheticControl	300	300	6	60	0.1	1000	2
ToeSegmentation1	40	228	2	277	0.01	1000	1
TwoLeadECG	23	1139	2	82	0.1	5000	5
TwoPatterns	1000	4000	4	128	0.01	2000	3
UWaveGestureLibraryY	896	3582	8	315	0.01	1000	2
Wafer	1000	6164	2	152	0.1	1000	4
WormsTwoClass	77	181	2	900	0.1	1000	4
Yoga	300	3000	2	426	0.01	5000	5

and new time series are classified with a weighted vote.

- *Learning Time-Series Shapelets (LTS) [9].* Logistic regression with gradient descent adjusts the selected shapelets for TSC. It has significantly improved accuracy.
- *Fast Shapelets (FS) [20].* FS combines SAX and random masking techniques to enhance efficiency.
- *Scalable Shapelet Discovery (SD) [10].* SD filters out candidates improving classification with an online clustering/pruning technique. The dimensionality reduction ratio r is set to $1/2$ and pruning distance percentile p is 25 for simplicity if there is no support from [10].
- *ELIS [7].* ELIS is the current state-of-the-art of efficient shapelet-based methods. It employs PAA and TF-IDF to improve the efficiency of discovering shapelet candidates. Then, the logistic regression classifier is applied to adjust the shapelets. We follow ELIS [7] to set parameters whenever possible. Otherwise, the parameters are fixed as follows: the iteration number is 1000; the regularization λ is 0.01; and the learning rate η is set to 0.1.
- *COTE [2].* COTE is the meta ensemble method, which is a combination of classifiers.
- *ResNet [8].* The main characteristic of ResNet is the shortcut residual connection between consecutive convolutional layers.
- BSPCOVER**-Random**. (*abbrev.* **Random**) The only difference between this algorithm and BSPCOVER is that it randomly selects the same number of final SAX word candidates as BSPCOVER does. That is, **Random** replaces our heuristic algorithm (in Sec. 4.4) with random selection.

5.4 Experiments on Efficiency

We observe from our results that the efficiency of ELIS is about two orders of magnitude faster than that of LTS [7]. On the other hand, we find that COTE and ST are slower than LTS [1]. Thus, we compare FS and ELIS with BSPCOVER to investigate efficiency.

5.4.1 Comparison With ELIS and FS,SD

We present the runtime of the total time of ELIS, FS, SD and our method BSPCOVER, and the improvement of FS and SD over BSPCOVER and that of BSPCOVER over ELIS, in Table 4. (The unit of the numbers is seconds by default, and we use d and h to denote for days and hours, respectively.)

On these 45 datasets, BSPCOVER is consistently faster than ELIS: 70 times (on average) faster in terms of total time. In particular, it is the fastest method on **MoteStrain**, whose efficiency is 315 times superior to ELIS. One reason is that our method finds the discriminative shapelets more efficiently prior to model building. This can be verified with the shapelet discovery time, which is 26 times faster than ELIS on average. The iteration number of our method is smaller, enhancing efficiency 80 times over in model building. It confirms the shapelet candidates discovered by our method of high quality. Compared to FS and SD, BSPCOVER is slower in most datasets. While FS and SD are 13 and 488 times faster than BSPCOVER on average, it is known to result in a *significant* drop in accuracy. Our results also reveal that BSPCOVER is clearly more accurate than FS and SD, which exhibit the worst accuracies among all compared methods.

5.4.2 Efficiency by Varying p

Fig. 8 presents the total time with the increase of p. For the first three datasets **Beetle**/**Fly**, **Diatom**., **FaceFour**, the times maintain an approximate linear relationship with p. Regarding **TwoLeadECG**, there is a similar relationship when p is small. However, the total time stabilizes when p exceeds 5. The reason is that all the candidates for representing the classes have been discovered when $p = 5$. The model building time is significantly more computationally costly than BSPCOVER's, which is observed in the "discover" and "train" columns of "Ours" of Table 5.

TABLE 4
Efficiency of Our Method and Related Methods on UCRArchive (Units is Second; d and h Denotes Days and Hours)

Dataset	SD	FS	ELIS	BSPCOVER	SD vs BSPCOVER	FS vs BSPCOVER	BSPCOVER vs ELIS
ArrowHead	2.83	7.68	3.4h	55.57	19.63	7.23	220.83
Beef	0.87	52.13	6.9h	131.17	150.77	2.52	188.97
Beetle/Fly	1.62	13.96	122.87	42.92	26.49	3.08	2.86
CBF	1.19	2.04	518.49	16.43	13.81	8.07	31.56
ChlorineConcentration	1.82	128.56	31.8h	173.86	95.53	1.35	659
Coffee	0.98	4.1	53.3	10.96	11.18	2.67	4.86
Computers	55.6	398.47	34.4h	1049.52	18.88	2.63	117.9
CricketZ	7.84	741.89	40.6h	5.8h	2663.27	28.3	6.96
DiatomSizeReduction	1.02	4.33	749.03	30.04	29.45	6.94	24.93
DistalPhalanxOutlineCorrect	4.05	15.2	1.1h	52.39	12.93	3.45	73.61
Earthquakes	39.75	1152.06	2.7h	2957.36	73.4	2.57	3.27
ECG200	1.12	2.96	325.94	48.34	43.16	16.35	6.74
ECG5000	6.95	36.89	7.5h	600.37	86.38	16.27	44.76
ECGFiveDays	1.31	1.01	24.16	1.38	1.05	1.37	17.51
ElectricDevices	1.7h	782.76	27.4d	5.8h	3.5	26.64	113.73
FaceAll	12.34	155.43	11.3h	1541.8	124.93	9.92	26.54
FaceFour	1.51	17.15	181.63	32.67	21.64	1.91	5.56
FacesUCR	4.83	40.91	10.7h	1265.7	262.05	30.94	30.33
FordA	301.9	4.5h	548.78	10.4h	124.01	2.32	0.01
GunPoint	1.09	1.69	486.48	8.97	8.23	5.31	54.23
Ham	2.07	172.19	2.02h	126.1	60.92	0.73	57.67
HandOutlines	72.88	10.9h	/	1.2h	59.28	0.11	/
Haptics	6.45	2086.17	3.6d	3.2h	1786.05	5.52	27.23
InlineSkate	5.42	2.1h	8.5d	4.2h	2856.09	1.97	48.66
InsectWingbeatSound	7.94	147.67	1.1d	646.49	81.42	4.38	144.77
LargeKitchenAppliances	650.37	1018.59	4.2d	3.9h	21.59	13.72	25.68
Mallat	4.34	419.98	3.9h	2896.15	667.32	6.9	4.87
Meat	1.94	34.62	2975.9	44.02	22.69	1.27	67.6
NonInvasiveFatalECGThorax1	30.34	5.6h	/	11.1h	1317.07	1.99	/
OSULeaf	7.41	381.4	18.02h	1.9h	923.08	17.71	9.61
Phoneme	462.23	2.6h	53.7d	12.7h	98.91	4.82	101.4
RefrigerationDevices	80.87	1.3h	3.5d	2.5h	111.29	1.9	34.08
ShapeletSim	3.24	26.23	28.33	455.23	140.5	17.35	0.06
SonyAIBORobotSurface1	1.15	0.85	271.38	4.19	3.64	4.91	64.77
SonyAIBORobotSurface2	0.93	0.83	292.42	3.78	4.06	4.54	77.36
Strawberry	5.02	93.33	8.4h	235.17	46.84	2.52	128.15
Symbols	1.04	16.17	2315.93	90.43	86.95	5.59	25.61
SyntheticControl	1.91	8.87	1.5h	249.29	130.52	28.11	20.99
ToeSegmentation1	2.04	7.75	1.9h	19.91	9.76	2.57	343.61
TwoLeadECG	1.86	0.91	397.09	20.32	10.92	22.22	19.54
TwoPatterns	7.71	195.12	20.99h	4.96h	2315.95	91.69	4.22
UWaveGestureLibraryY	35.89	1231.53	5.9d	2.2d	5296.18	157.26	2.65
Wafer	6.39	64.15	2.7h	825.96	129.26	12.88	11.58
WormsTwoClass	5.75	2085.67	6.3h	1124.08	195.49	0.54	20.24
Yoga	5.81	333.24	1.2d	2.9h	1796.9	31.79	9.46
Average					488.09	13.84	67.07

5.4.3 Cost Model

We conducted an experiment to compute the constants $c_{i,i\in 1,2}$ for each factor in a cost model and validate whether it can fit the runtimes of BSPCOVER. As discussed in Section 4.4, the efficiency of BSPCOVER is mainly contributed to by three components. The total time cost is estimated as follows. For the illustration purpose, we present only the dataset **Beef** to derive the constants c_i and the same procedure can be utilized to explore c_i for other datasets. The main idea is to compute the constant through modifying one parameter and fixing others for each factor as control variables. For instance, below is the cost model for **Beef**;

$$T_{\text{BSPCOVER}} = c_1 I + c_2 p + \epsilon = 90 \cdot I + 4,667 \cdot p + 27,389, \quad (10)$$

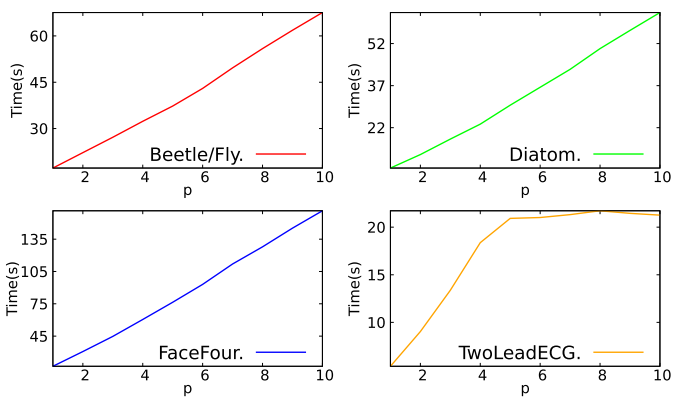


Fig. 8. Efficiency vs p (of p-Cover).

TABLE 5
Efficiency of Our Method and ELIS on UCRARCHIVE

Dataset	ELIS (seconds)				Ours (seconds)				Improvement	
	discover	train	test	total	discover	train	test	total	discover	total
Beef	109.61	24675.86	1.70	24847.17	15.72	114.32	1.13	131.17	6.97	188.42
Beetle/Fly	51.80	70.68	0.39	122.87	14.35	28.12	0.45	42.92	3.61	2.86
Bird/Chicken	60.14	42.46	0.26	102.86	11.14	21.97	0.07	33.18	5.40	3.10
Coffee	50.56	2.73	0.01	53.3	1.96	8.86	0.14	10.96	25.80	4.86
Diatom.	60.42	688.05	0.56	749.03	2.62	26.06	1.36	30.04	23.06	24.93
ECGFiveDays	13.79	9.7	0.67	24.16	0.36	0.48	0.55	1.38	38.31	17.51
FaceFour	117.33	63.62	0.68	181.63	3.06	28.87	0.74	32.67	38.34	5.56
GunPoint	40.99	444.54	0.95	486.48	0.67	8.09	0.21	8.97	61.18	54.23
ItalyPower	0.72	636.7	0.55	637.97	0.06	1.85	0.23	2.14	12.00	298.12
Lighting7	554.88	15838.13	3.68	16396.69	21.4	573.67	1.72	596.79	25.93	27.47
MoteStrain	3.84	1470.2	1.2	1475.24	0.19	3.74	0.74	4.67	20.21	315.90
SonyAIBO1	2.31	268.76	0.31	271.38	0.24	3.34	0.61	4.19	9.63	64.77
Symbols	227.58	2082.63	5.72	2315.93	6.04	91.83	5.17	103.04	37.68	22.48
Trace	374.78	73.11	0.37	448.26	5.92	44.02	0.3	50.24	63.31	8.92
TwoLeadECG	4.45	391.49	1.15	397.09	0.19	19.35	0.78	20.32	23.42	19.54
Average									26.32	70.58

TABLE 6
Accuracy of Our Method and Related Methods on UCRARCHIVE

Dataset	RotF	DTW_Rn_1NN	ST	LS	FS	SD	COTE	ELIS	ResNet	Random	BSPCOVER
ArrowHead	73.71	80	73.71	84.57	59.43	65.7	81.14	81.43	84.5	75.33	80.57
Beef	86.67	66.67	90	86.67	56.67	50.7	86.67	63.33	75.3	54.21	73.33
BeetleFly	90	65	90	80	70	75	80	85	85	80	90
CBF	92.89	99.44	97.44	99.11	94	97.5	99.56	90.44	99.5	92.58	99.67
ChlorineConcentration	84.74	65	69.97	59.24	54.64	55.3	72.71	27.39	84.4	45.63	61.22
Coffee	100	100	96.43	100	92.86	96.1	100	96.43	100	93.47	100
Computers	70	62.4	73.6	58.4	50	58.8	74	50	81.5	57.74	67.2
CricketZ	65.64	73.59	78.72	74.1	46.41	67.3	81.54	78.95	81.2	68.91	74.1
DiatomSizeReduction	87.25	93.46	92.48	98.04	86.6	89.6	92.81	89.86	30.1	81.69	87.25
DistalPhalanxOutlineCorrect	75.72	72.46	77.54	77.9	75	71.7	76.09	57.83	77.1	77.52	83.17
Earthquakes	74.82	72.66	74.1	74.1	70.5	63.6	74.82	77.64	71.2	75.49	81.68
ECG200	85	88	83	88	81	81.8	88	80	87.4	85	92
ECG5000	94.58	92.51	94.38	93.22	92.27	92.4	94.6	72.69	93.4	92.64	94.44
ECGFiveDays	90.82	79.67	98.37	100	99.77	95.3	99.88	95.45	97.5	89.95	100
ElectricDevices	78.58	63.08	74.7	58.75	57.9	59.3	71.33	8.65	72.9	18.19	24.24
FaceAll	91.12	80.77	77.87	74.85	62.6	71.4	91.78	75.56	83.9	72.44	76.33
FaceFour	81.82	89.77	85.23	96.59	90.91	82	89.77	95.46	95.5	93.32	96.59
FacesUCR	80.29	90.78	90.59	93.9	70.59	84.7	94.24	63.63	95.5	72.14	78.29
FordA	84.47	66.52	97.12	95.68	78.71	77.6	95.68	67.6	92.0	90.12	96.31
GunPoint	92	91.33	100	100	94.67	93.1	100	97.57	99.1	87.38	100
Ham	71.43	60	68.57	66.67	64.76	61.9	64.76	63.81	75.7	71.87	76.19
HandOutlines	91.08	87.84	93.24	48.11	81.08	79.9	91.89	/	91.1	75.28	86.7
Haptics	43.83	41.56	52.27	46.75	39.29	35.6	52.27	41.56	51.9	41.31	45.13
InlineSkate	37.09	38.73	37.27	43.82	18.91	38.5	49.45	35.46	37.3	32.67	38.73
InsectWingbeatSound	63.64	57.37	62.68	60.61	48.94	44.1	65.25	59.55	50.7	50.82	57.42
LargeKitchenAppliances	60.8	79.47	85.87	70.13	56	57.1	84.53	33.33	90	82.37	86.13
Mallat	94.93	91.43	96.42	95.01	97.61	92.6	95.39	81.58	97.2	72.51	76.8
Meat	96.67	93.33	85	73.33	83.33	93.3	91.67	55	96.8	70.14	75
NonInvasiveFatalECGThorax1	90.53	82.9	94.96	25.9	71.04	81.4	93.13	/	94.5	82.36	91.47
OSULeaf	57.02	59.92	96.69	77.69	67.77	56.6	96.69	76.45	97.9	75.39	83.88
Phoneme	12.97	22.68	32.07	21.84	17.35	15.8	34.92	15.19	33.4	17.27	20.73
RefrigerationDevices	56.53	44	58.13	51.47	33.33	46.1	54.67	40	52.5	59.91	54.67
ShapeletSim	41.11	69.44	95.56	95	100	67.2	96.11	100	77.9	79.25	84.44
SonyAIBORobotSurface1	80.87	69.55	84.36	81.03	68.55	85	84.53	87.85	95.8	80.06	88.35
SonyAIBORobotSurface2	80.8	85.94	93.39	87.51	79.01	78	95.17	93.17	97.8	79.93	93.49
Strawberry	97.3	94.59	96.22	91.08	90.27	88.4	95.14	83.85	98.1	89.57	94.29
Symbols	79.3	93.77	88.24	93.17	93.37	90.1	96.38	78.29	90.6	86.23	93.37
SyntheticControl	97.33	98.33	98.33	99.67	91	98.3	100	99.33	99.8	95.57	99.67
ToeSegmentation1	53.07	75	96.49	93.42	95.61	88.2	97.37	98.24	96.3	81.71	96.49
TwoLeadECG	97.01	86.83	99.74	99.65	92.45	86.7	99.3	99.82	100	92.19	99.65
TwoPatterns	92.8	99.85	95.5	99.33	90.83	98.1	100	99.75	100	92.64	99.8
UWaveGestureLibraryY	71.44	70.18	73.03	70.3	59.58	67.1	75.85	69.32	67.0	57.15	64.01
Wafer	99.45	99.59	100	99.61	99.68	99.3	99.98	99.43	99.9	96.49	99.81
WormsTwoClass	68.83	58.44	83.12	72.73	72.73	64.1	80.52	71.82	74.7	71.24	74.59
Yoga	82.43	84.3	81.77	83.43	69.5	62.5	87.67	83.9	87.0	81.27	88.2
Total best acc	4	1	10	6	2	0	14	2	11	0	11
Ours 1-to-1 Wins	28	29	18	25	40	36	13	34	19	45	-
Ours 1-to-1 Draws	3	2	3	7	1	0	3	1	1	0	-
Ours 1-to-1 Losses	14	14	24	13	4	9	29	8	25	0	-
Rank Mean	6.34	6.68	3.78	5.16	8.43	8.58	3.02	7.46	3.65	8.21	3.78
Wilcoxon Test p-vlaue	0.003	0.001	0.079	0.048	0.00	0.00	0.002	0.00	0.122	0.00	-

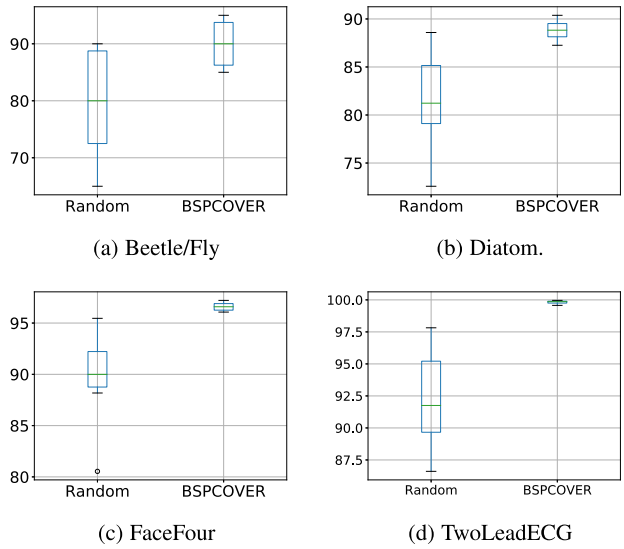


Fig. 9. Accuracy of Random vs BSPCOVER in box plots.

where both I and $p \in \mathbb{N}$ and I is often a multiple of 1000.

The standard deviation of c_1, c_2 and ϵ are $2.57, 8.94$, and 14.58; and the root-mean-square error of T_{BSPCOVER} is 979.26, less than 1 second. Hence, users can employ the above method to reasonably estimate the runtimes and determine the values of I and p for the datasets.

5.5 Experiments on Accuracy

The experiment accuracy results for baselines (with the except of **Random**) are all taken from the paper [1] with a single train/test. The accuracy results for 45 datasets are presented in Table 6.

5.5.1 Comparison With Other Methods

While the current most accurate classifier remains COTE, our method is very competitive. In particular, it is close to ST and ResNet, and superior to the others on average. BSPCOVER exhibits the highest accuracy in 11 datasets. Its accuracy is ranked the $3rd$, with a rank mean of 3.78 and the rank of most datasets is top-three among all nine methods. The 1-to-1 wins numbers for RotF, DTW_Rn_1NN, LS, FS, SD, ELIS and Random are all more than half of all the datasets. It is not surprising that BSPCOVER is more efficient than COTE (and all other shapelet-based methods except FS and SD). In other words, BSPCOVER shows that it is possible to make a small sacrifice in accuracy to achieve a significant improvement in efficiency.

We conducted the Friedman test [6] among all the methods and Wilcoxon signed-rank test between other ten methods and our BSPCOVER. The Friedman test is a well-known non-parametric statistical test, to detect the differences in 45 datasets across 11 methods. Our result is that $p = 0.00$, which is smaller than 0.05. Thus, we can reject the null hypothesis, and there is a difference among these 11 methods. However, we cannot figure out the difference from exactly which methods only according to the Friedman test. We further conducted the Wilcoxon test against all baselines and found out that all results are statistically significant at $p < 0.05$, except **ST**, **ResNet** (larger than 0.05) and **COTE** (based on opposite ranks) from the last row in Table 6.

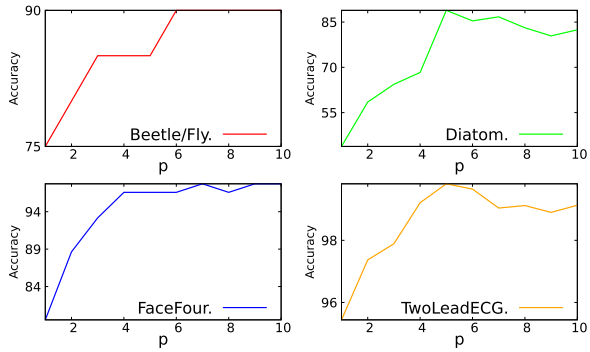


Fig. 10. Accuracy vs p (of p-Cover).

5.5.2 Comparison With BSPCOVER-Random

From the last two columns of Table 6, we can observe that the results of BSPCOVER are better than those of BSPCOVER-Random in all the datasets, which proves the superiority of our heuristic algorithm. We compared the accuracy of two methods with box plots, generated from ten runs for each dataset. Due to limitations of space, some representative datasets are shown in Fig. 9. The box plots show the minimum, first quartile, median, third quartile, and maximum values of four datasets, namely **Beetle**/**Fly**, **Diatom.**, **FaceFour**, and **TwoLeadECG**. The metrics of the box plot in our BSPCOVER method are better than those of the BSPCOVER-Random method.

5.5.3 Accuracy by Varying p

To investigate the performance of p-Cover value, a set of experiments is designed to verify the trend between p-Cover value and accuracy. Four datasets, **Beetle**/**Fly**, **Diatom.**, **FaceFour**, and **TwoLeadECG** are selected. We can observe in Fig. 10 that as the p-Cover value increases, the accuracy also rises at the beginning. The accuracy typically stabilizes after p-Cover reaches the optimal value.

5.5.4 Accuracy by Varying I

The quality of the shapelets discovered can be enhanced by increasing the number of iterations I in model building. Hence, we conducted another experiment that varies I and studied the accuracies of BSPCOVER and the most recent approach, ELIS. We chose four datasets due to their difference

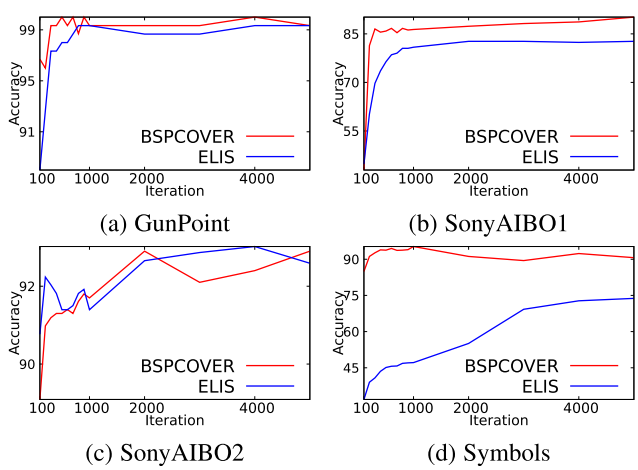


Fig. 11. Accuracy vs iterations of BSPCOVER and ELIS.

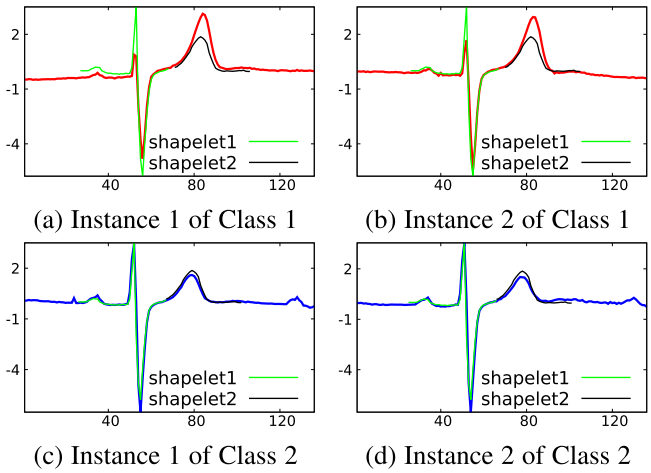


Fig. 12. *ECGFiveDays* shapelets encodes hyperacute T wave.

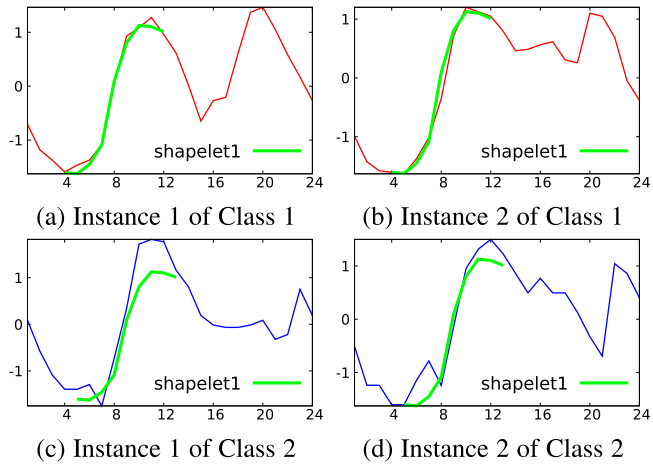


Fig. 13. A shapelet of *ItalyPowerDemand* highlighting the morning heating demand difference of summer and winter months.

in iteration numbers for convergence is not significant. We observe in Fig. 11 that the accuracy in all four subfigures increases with the iteration number. Our BSPCOVER discovers high-quality shapelets in three datasets, **GunPoint**, **Sony AIBO1**, and **Symbols** and the accuracy stabilizes early.

5.6 Experiments on Interpretability

One of the strengths of shapelets is their interpretability, which can help practitioners to understand their data. This experiment investigates whether the shapelets discovered by our method are interpretable. We present results from two datasets because they do not require much domain background. We calculate the best matching score (best matching location) of each shapelet in the raw time series and users can exploit them to greedily select a small set of shapelets.

5.6.1 Interpretability of ECGFiveDays Shapelets

Fig. 12 shows how the shapelets discriminate the different classes of the **ECGFiveDays** dataset. ECG stands for Electrocardiography. The time series in Figs. 12a and 12b belong to Class 1 (abnormal ECG) and those in Figs. 12c and 12d belong to Class 2 (normal ECG). Our selected shapelets are colored in green and black for Class 2. More importantly, the shapelets highlight the major different fragments in two classes. (Shapelet 1 is QRS complex and Shapelet 2 is T wave of ECG.) Intuitively, the peak of T wave is larger than that of QRS complex in Class 1. It is indeed called hyperacute T wave in medicine, due to ischemia or hyperkalemia or other diseases.

5.6.2 Interpretability of ItalyPowerDemand Shapelets

Another interpretability result comes from the **ItalyPower Demand** dataset, shown in Fig. 13. The time series instances in Figs. 13a and 13b belong to Class 1 (summer months, from April to September), and those in Figs. 13c and 13d exist in Class 2 (winter months, from October to March). The most representative shapelet selected by BSPCOVER for Class 1 illustrates the difference between the two classes. The difference is that the power demand in Class 1 is lower than that in Class 2 from 4am to 12pm, which turns out to be the heating in the morning during the winter time. (Air conditioning was still fairly rare in Italy when the data was collected.)

6 Conclusion

This paper has proposed an efficient method of shapelet discovery, called BSPCOVER. BSPCOVER has four steps: it transforms raw time series by adopting PAA and SAX techniques; it adopts bloom filters to prune duplicate SAX words and proposes an algorithm to prune similar non-discriminative SAX words; and it computes a subset of shapelet candidates with maximal weights to cover each class p times. Hence, potentially costly learning algorithms are applied to a discriminative subset only. Our experiments verify that BSPCOVER is more efficient than almost all shapelet-based methods (with the sole exception of FS) and its accuracy is comparable to or slightly lower than the most accurate classifier, COTE.

In future work, we plan to study the TSC problem with multivariate time series and missing values. A shapelet-based approach for multivariate TSC is, however, in its infancy. Since multivariate time series contain multiple variables, instead of only one variable, shapelet candidates of different variables can be of different lengths, and the qualities of shapelets are hence hard to measure. We will apply the techniques to time series of tidal water level and time series in electronic patient records, respectively.

Acknowledgements

This work was partly supported by HKRGC GRF 12201119, 12232716, 12201518, 12200817, and 12201018, and NSFC 61602395.

References

[1] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, "The great time series classification bake off: A review and experimental evaluation of recent algorithmic advances," *Data Mining Knowl. Discovery*, vol. 31, no. 2, pp. 606–660, 2017.

[2] A. Bagnall, J. Lines, J. Hills, and A. Bostrom, "Time-series classification with COTE: The collective of transformation-based ensembles," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 2, pp. 2522–2535, Sep. 2015.

[3] N. Begum and E. Keogh, "Rare time series motif discovery from unbounded streams," *Proc. VLDB Endowment*, vol. 8, no. 2, pp. 149–160, 2014.

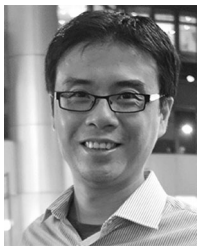
[4] Y. Chen *et al.*, "The UCR time series classification archive, Jul. 2015. [Online]. Available: www.cs.ucr.edu/eamonn/time_series_data/
[5] V. Chvatal, "A greedy heuristic for the set-covering problem," *Math. Operations Res.*, vol. 4, no. 2, pp. 233–235, 1979.
[6] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, 2006.
[7] Z. Fang, P. Wang, and W. Wang, "Efficient learning interpretable shapelets for accurate time series classification," in *Proc. IEEE 34th Int. Conf. Data Eng.*, 2018, pp. 497–508.
[8] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: A review," *Data Mining Knowl. Discovery*, vol. 33, pp. 917–963, 2019.
[9] J. Grabocka, N. Schilling, M. Wistuba, and L. Schmidt-Thieme, "Learning time-series shapelets," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2014, pp. 392–401.
[10] J. Grabocka, M. Wistuba, and L. Schmidt-Thieme, "Fast classification of univariate and multivariate time series through shapelet discovery," *Knowl. Inf. Syst.*, vol. 49, no. 2, pp. 429–454, 2016.
[11] G. Ifrim and C. Wiuf, "Bounded coordinate-descent for biological sequence classification in high dimensional predictor space," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2011, pp. 708–716.
[12] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra, "Dimensionality reduction for fast similarity search in large time series databases," *Knowl. Inf. Syst.*, vol. 3, no. 2, pp. 263–286, 2001.
[13] R. J. Larsen and M. L. Marx, *An Introduction to Mathematical Statistics and its Applications*, vol. 5. London, U.K.: Pearson, 2017.
[14] T. Le Nguyen, S. Gsponer, and G. Ifrim, "Time series classification by sequence learning in all-subsequence space," in *Proc. IEEE 33rd Int. Conf. Data Eng.*, 2017, pp. 947–958.
[15] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A symbolic representation of time series, with implications for streaming algorithms," in *Proc. 8th ACM SIGMOD Workshop Res. Issues Data Mining Knowl. Discovery*, 2003, pp. 2–11.
[16] J. Lin, E. Keogh, L. Wei, and S. Lonardi, "Experiencing sax: A novel symbolic representation of time series," *Data Mining Knowl. Discovery*, vol. 15, no. 2, pp. 107–144, 2007.
[17] J. Lin, R. Khade, and Y. Li, "Rotation-invariant similarity in time series using bag-of-patterns representation," *J. Intell. Inf. Syst.*, vol. 39, no. 2, pp. 287–315, 2012.
[18] J. Lines, L. M. Davis, J. Hills, and A. Bagnall, "A shapelet transform for time series classification," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2012, pp. 289–297.
[19] A. Mueen, E. Keogh, and N. Young, "Logical-shapelets: An expressive primitive for time series classification," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2011, pp. 1154–1162.
[20] T. Rakthanmanon and E. Keogh, "Fast shapelets: A scalable algorithm for discovering time series shapelets," in *Proc. SIAM Int. Conf. Data Mining*, 2013, pp. 668–676.
[21] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Trans. Acoustics Speech Signal Process.*, vol. 26, no. 1, pp. 43–49, Feb. 1978.
[22] P. Schäfer and U. Leser, "Fast and accurate time series classification with weasel," in *Proc. ACM Conf. Info. Knowl. Manage.*, 2017, pp. 637–646.
[23] P. Senin and S. Malinchik, "Sax-VSM: Interpretable time series classification using sax and vector space model," in *Proc. IEEE 13th Int. Conf. Data Mining*, 2013, pp. 1175–1180.
[24] L. Ye and E. Keogh, "Time series shapelets: A new primitive for data mining," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2009, pp. 947–956.
[25] L. Ye and E. Keogh, "Time series shapelets: A novel technique that allows accurate, interpretable and fast classification," *Data Mining Knowl. Discovery*, vol. 22, no. 1–2, pp. 149–182, 2011.

Guozhong Li is working toward the PhD degree in the Department of Computer Science, Hong Kong Baptist University. His research interests include time series data mining. He is a member of the Database Group at Hong Kong Baptist University (http://www.comp.hkbu.edu.hk/∼db/).

Byron Choi received the B.Eng. degree in computer engineering from the Hong Kong University of Science and Technology (HKUST), in 1999, and the MSE and PhD degrees in computer and information science from the University of Pennsylvania, in 2002 and 2006, respectively. He is an associate professor with the Department of Computer Science, Hong Kong Baptist University.

Jianliang Xu is a professor with the Department of Computer Science, Hong Kong Baptist University (HKBU). He held visiting positions at Pennsylvania State University and Fudan University. He has published more than 150 technical papers in these areas, most of which appeared in leading journals and conferences including SIGMOD, VLDB, ICDE, the *ACM Transactions on Database Systems*, the *IEEE Transactions on Knowledge and Data Engineering*, and the *VLDB Journal*.

Sourav S Bhowmick is an associate professor with the School of Computer Science and Engineering, Nanyang Technological University. His current research interests include data management, data analytics, computational social science, and computational systems biology. He has published many papers in major venues in these areas such as SIGMOD, VLDB, ICDE, SIGKDD, MM, the *IEEE Transactions on Knowledge and Data Engineering*, the *VLDB Journal*, and the *Bioinformatics*.



Kwok Pan Chun received the PhD degree from Imperial College London, U.K.. He is an assistant professor with the Department of Geography, Hong Kong Baptist University. His research interests include nonstationary environmental processes related to the atmosphere, hydrosphere, and biosphere. He develops tools and numerical models for analysing dynamic systems and projecting possible changes.

Grace Lai-Hung Wong is a professor with the Department of Medicine and Therapeutics, Consultant Hepatologist, Center for Liver Health, The Chinese University of Hong Kong. She has published more than 160 articles in peer-reviewed journals including Gastroenterology, Hepatology and Gut. She is currently a reviewer of 44 biomedical journals, the editorial board members of five journals and associate editor of *Journal of Gastroenterology and Hepatology*.