Ensemble Grammar Induction For Detecting Anomalies in Time Series

Yifeng Gao
George Mason University
Fairfax, VA
ygao12@gmu.edu

Jessica Lin
George Mason University
Fairfax, VA
jessica@gmu.edu

Constantin Brif
Sandia National Laboratories
Livermore, CA
cnbrif@sandia.gov

ABSTRACT

Time series anomaly detection is an important task, with applications in a broad variety of domains. Many approaches have been proposed in recent years, but often they require that the length of the anomalies be known in advance and provided as an input parameter. This limits the practicality of the algorithms, as such information is often unknown in advance, or anomalies with different lengths might co-exist in the data. To address this limitation, previously, a linear time anomaly detection algorithm based on grammar induction has been proposed. While the algorithm can find variable-length patterns, it still requires preselecting values for at least two parameters at the discretization step. How to choose these parameter values properly is still an open problem. In this paper, we introduce a grammar-induction-based anomaly detection method utilizing ensemble learning. Instead of using a particular choice of parameter values for anomaly detection, the method generates the final result based on a set of results obtained using different parameter values. We demonstrate that the proposed ensemble approach can outperform existing grammar-induction-based approaches with different criteria for selection of parameter values. We also show that the proposed approach can achieve performance similar to that of the state-of-the-art distance-based anomaly detection.

1 INTRODUCTION

Time series anomaly detection is an important task, with applications in a broad variety of domains. Many approaches have been proposed in recent years, but often they require that the length of the anomalies be known in advance and provided as an input parameter. Recently, this limitation has been addressed by introducing a time series anomaly detection approach that is based on grammar induction and has a linear time complexity with respect to the data size (the time series length).

Typically, the grammar-induction-based anomaly detection follows a four-step process. In the first step, the input time series is converted into a discrete sequence of symbols via a sliding window; this discretization depends on two parameters. In the second step, grammar induction (e.g., via the Sequitur algorithm [15]) is applied to the discrete sequence to quickly identify grammar rules that are repeating strings of symbols. The third step maps the repeating strings back to the time series subsequences that the strings represent. Finally, a meta time series named *rule density curve* is computed, which records the frequency of grammar rules at each point and is used to detect and rank the anomalies. Specifically, it is assumed that anomalies correspond to rarely occurring strings and hence are indicated by minima of the rule density curve.

While the grammar-induction-based algorithm can find variable-length patterns, it still requires preselecting values for at least two parameters at the discretization step. These two discretization parameters are the number of segments (i.e., the length of a word, also called PAA size which will be explained later), and the alphabet size. How to choose the parameter values properly is still an open problem, especially in an unsupervised setting where no training data are available.

Figure 1 presents an example that illustrates the challenge of choosing proper parameter values, even with the knowledge of ground truth information that allows us to evaluate the quality of anomalies detected. Figure 1.top shows a snippet of dishwasher electricity usage time series. An anomalous cycle that has an unusual short power usage period is highlighted in red. We ran the algorithm with different parameter value combinations, and the results are shown in Figure 1.bottom. According to the figure, the best parameter value combination (indicated by arrow) is significantly different from the second best combination. Moreover, we see that parameter values that are close to the optimal values actually perform badly. As a result, guessing the "best" parameter values can be very tricky.
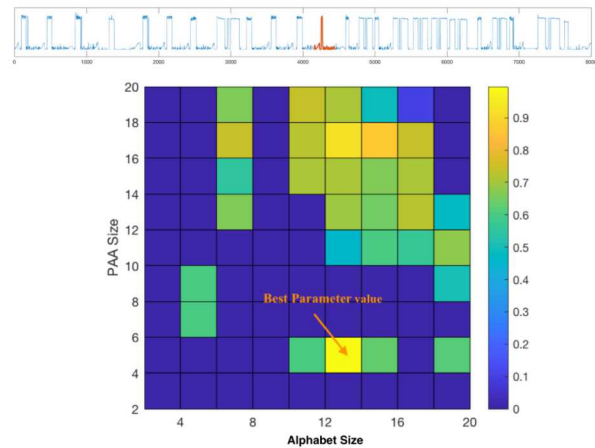

Figure 1: (top) A snippet of dishwasher electricity usage time series. An anomalous cycle is highlighted in red. (bottom) Performance of the grammar-induction-based algorithm with different discretization parameter values in the task of detecting the anomalous subsequence.

To overcome the challenge mentioned above, in this paper, we introduce a robust version of the grammar-induction-based anomaly detection method, which utilizes ensemble learning. Intuitively, instead of using a single combination of parameter values, the method generates the final result based on a set of results obtained using different parameter values. We design an approach that combines the results returned by each ensemble

member corresponding to a different combination of parameter values. Furthermore, in the discretization step, we use an approach to compute multi-resolution words to efficiently represent time series subsequences. This approach can dramatically reduce the cost of discretization and increase the scalability of the proposed work.

The contribution of this paper is summarized as follows:

- The proposed ensemble grammar induction can achieve better performance than the grammar induction with a single combination of parameter values.
- The proposed ensemble grammar induction is particularly suitable for unsupervised anomaly detection, where no training set is available to perform a grid search for the best parameter values.
- The proposed approach has a linear time complexity with respect to the data size and can achieve performance comparable to that of the state-of-the-art distance-based approach that has a quadratic time complexity.
- We adapt a fast algorithm to discretize subsequences, which reduces the cost of repeated computation.
- We demonstrate that the ensemble approach can be applied to find meaningful anomalies in real-world application.

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 presents definitions and notations used in the paper. The process used for discretization and numerosity reduction is described in Section 4. The grammar-induction-based anomaly detection is presented in Section 5. The ensemble approach is introduced in Section 6. The experimental results are shown in Section 7, and conclusions are summarized in Section 8.

2 RELATED WORK

First, we describe the state-of-the-art time series anomaly detection approach. Keogh et al. [9] introduced a concept named *time series discord*, which is the subsequence that has the largest one–nearest-neighbor (1-NN) distance, hence it represents the most unusual subsequence in the time series. The authors introduced an algorithm named HOTSAX to effectively detect the time series discord. Recently, a series of matrix-profile-based approaches, STOMP [23] and STAMP [21], have been introduced for fast computation of 1-NN distances for every subsequence. It has been shown that compared with the original method, STOMP and STAMP can achieve more stable and generally better performance. However, all these methods have a quadratic time complexity with respect to the data size, and the accuracy is sensitive to the length of the discord that has to be specified in advance as an input parameter [20].

In previous work, we proposed a series of approximate time series pattern discovery algorithms called GrammarViz based on grammar induction [10, 19, 20]. The idea is that by learning a context-free grammar from a discrete sequence of symbols that approximates the original time series, one can identify the repeating strings. Recently, in [18], we have extended the idea of grammar-induction-based anomaly detection by introducing the concept of rule density curve, which is a meta time series that records the frequency of grammar rules (repeating strings) at each point of the original time series. Since anomalies correspond to rarely occurring strings, they are indicated by minima of the rule density curve. It has been shown that this approach can

achieve competitive performance compared to the state-of-the-art while having linear time complexity. However, the algorithm requires the user to preselect values of two important parameters, and the performance can be greatly affected by these values.

Several ensemble algorithms have been proposed for unsupervised anomaly detection [17, 22]. Most of them use the average performance as the ground truth and select a very small number of detectors to approximate the average performance. All these approaches are introduced for point-based anomaly detection. How to use them for detecting anomalous subsequences is still unclear. Besides, since subsequences extracted from a time series via a sliding window are highly overlapped, the detector used in these approaches [1] cannot resolve this problem.

Other works [2, 4, 8] also focus on anomaly detection; however, these techniques often focus on detecting anomalies in event logs (discrete or mixed data type time series) and the lengths of the anomalies are often very short.

3 NOTATIONS AND PROBLEM DEFINITION

We first describe the fundamental definitions related to time series and grammar induction. We then formulate the problem of time series anomaly detection.

3.1 Notations and Definitions

We start with the definitions related to time series:

Time series $T = t_1, \ldots, t_N$ is a set of observations ordered by time.

Subsequence $T_{p,q}$ of a time series T is a subsequence of elements in T starting from position p and ending at position q, of length $n = q - p + 1$. Typically, $n \ll N$, and $1 \le p \le N - n + 1$.

Subsequences can be extracted from time series via a sliding window. In many applications, we are interested in finding unusual "shapes." Therefore, anomaly discovery result is more meaningful when the method can maintain offset- and amplitude-invariance during the anomaly detection process. This can be achieved by normalizing all subsequences prior to applying an anomaly detection algorithm. **z-normalization** is a procedure that normalizes the mean and standard deviation of a subsequence to zero and one, respectively.

Since the proposed anomaly detection approach is based on grammar induction, we next introduce the definitions related to grammar induction using a toy example shown in Figure 2. Intuitively, grammar induction is a process that induces a hierarchical grammar structure from a **token sequence** S, which is a sequence of discrete tokens (words). In the example, S consists of nine two-letter tokens. The hierarchical grammar structure is represented by a set of **grammar rules**. Each grammar rule represents a repeating string of token segments in S. In the example, two rules R_1 and R_2 represent the repeating token segments ab, bc and cc, cc, respectively. Following the terminology used in previous work [15, 20], each grammar rule is also called a **non-terminal** and each token stored in the token sequence is called a **terminal**.

Using the set of grammar rules, the original token sequence S is represented by the compressed sequence R_0. Since compressibility is a measure of regularity (and hence incompressibility is a measure of anomalousness), the compressed sequence plays an important role in motif (repeated patterns) discovery and anomaly detection.
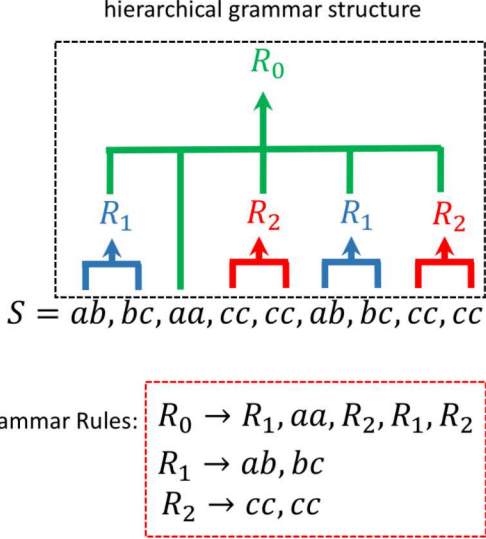
Figure 2: Example of grammar induction applied to a token sequence.

Clearly, the grammar induction approach cannot be directly used for real-valued time series. To use it, we need first to approximate the original time series by a discretized token sequence. We describe the time series discretization process in detail in Section 4.

3.2 Problem of Anomaly Detection

Based on the definitions above, we can formulate the problem of anomaly detection. In this work, we follow the previous anomaly detection framework GrammarViz [20], and determine a hierarchical grammar structure for a time series, through the processes of time series discretization and grammar induction. The anomaly candidates are subsequences that cannot be compressed by induced grammar rules.

To illustrate the basic idea behind this process of anomaly detection, we use another toy example of a simple token sequence:

$$S = aa, bb, cc, xx, aa, bb, cc. \qquad (1)$$

The grammar structure induced from S is shown in Table 1. We see that S contains a repeating pattern, aa, bb, cc, represented by the grammar rule R1. The token xx, however, does not appear in any grammar rules (i.e., it is incompressible). It is not hard to see that token xx is *structurally* dissimilar from the rest of the sequence. In the grammar-induction-based time series anomaly detection framework, the subsequence that xx represents is considered an anomaly candidate.

Table 1: Example of grammar rules induced from token sequence S of Eq. (1)

Grammar rules	Expanded sequence
$R_0 \rightarrow R_1, xx, R_1$	$aa, bb, cc, xx, aa, bb, cc$
$R_1 \rightarrow aa, bb, cc$	aa, bb, cc

4 DISCRETIZATION AND NUMEROSITY REDUCTION

Time series discretization [12] is a common step in many time series data mining tasks, including anomaly detection [5, 9]. Since grammar induction requires discrete input, it is necessary to approximate the time series by a token sequence first. In general, discretization offers several advantages including noise removal, dimensionality reduction, and improved efficiency.

In this section, we first describe the algorithm called *Symbolic Aggregate approXimation* (SAX), a widely used time series discretization technique. We then describe *numerosity reduction*, a procedure that removes repeating consecutive tokens to form a more compact token representation of a time series [11, 20].

4.1 Symbolic Aggregate approXimation

In this section, we describe SAX, a popular technique used to discretize univariate time series.

SAX consists of two steps. At the first step, the *Piecewise Aggregate Approximation* (PAA) is used to convert a normalized subsequence of length n from a time series T into a representation of a lower dimension $w < n$ (w is called PAA size). Specifically, the subsequence is divided into w equal-sized windows, and the average value of the elements within each window is computed. In other words, the PAA coefficients vector [12] is a w-dimensional vector that consists of the average values from w equal-sized segments of the input subsequence. PAA coefficients are an approximate representation of the original subsequence.

At the second step, the PAA coefficients vector is mapped to w symbols from an alphabet of size a, according to a breakpoint table [12], defined such that the regions are approximately equal-probable under the Gaussian distribution. This maximizes the chances that the symbols occur with an approximately equal probability. These w symbols form a SAX word.

Figure 3 illustrates the SAX process for an example subsequence (shown as the blue curve). The bold flat lines represent the values of PAA coefficients computed from their respective segments in the subsequence. The breakpoint table with a from 2 to 4 is also shown in the figure. Since we set $a = 3$ in this example, two breakpoints in the second column of the table are used to generate three regions: $(-\infty, -0.43), [-0.43, 0.43), [0.43, \infty)$. The PAA coefficients falling into these three regions are mapped to symbols a, b, and c, respectively. In this example, the SAX word $abca$ is formed to approximate the original subsequence.

This description shows how SAX is applied to a given subsequence. In order to discretize the entire time series, SAX is usually applied via a sliding window of length n.
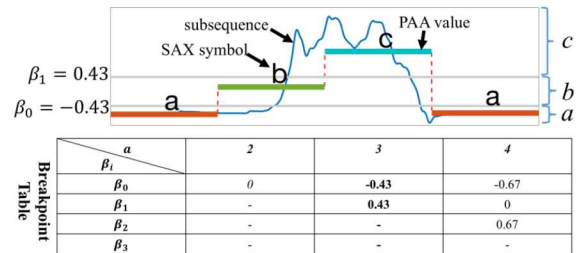

Figure 3: Example of the SAX process, with $a = 3$ and $w = 4$, which approximates the original subsequence by the word $abca$.

4.2 Numerosity Reduction

In practice, since neighboring subsequences are only off by one point, the neighboring tokens generated by SAX are often identical to each other. This phenomenon, however, will result in an overwhelming number of grammar rules that represent trivial matches, and this redundancy will significantly affect both the scalability and the quality of results. To avoid this problem, a *numerosity reduction* step is added to further compress the token sequence. Specifically, whenever there is a sequence of one token repeating consecutively multiple times, numerosity reduction will output only the first occurrence of the token along with its offset.

For example, a token sequence

$$S = ba, ba, ba, dc, dc, aa, ac, ac \qquad (2)$$

can be compressed to

$$S_{NR} = ba_1, dc_4, aa_6, ac_7, \qquad (3)$$

where the subscripts indicate the positions in the original uncompressed token sequence. S_{NR} contains all information needed to retrieve the original token sequence.

5 GRAMMAR INDUCTION

In this section, we first describe Sequitur [15], a grammar induction algorithm with a linear time complexity. We then describe the process of computing the rule density curve and ranking anomalies.

5.1 Sequitur

Sequitur is a linear-time greedy algorithm to induce a context-free grammar structure from a discrete token sequence.

Sequitur maintains a list of grammar rules and a table of digrams based on the input sequence. A digram is a pair of consecutive tokens (terminals or non-terminals) in the sequence. Two principles, digram uniqueness and rule utility, are applied to constrain the rules during grammar induction. Digram uniqueness requires that digrams stored in the digram table should be unique. Rule utility requires that rules that only appear once should be removed to minimize the size of grammar.

To illustrate how Sequitur works, consider an example token sequence generated by SAX with parameters $w = 2, a = 3, n = 16$, after the numerosity reduction step:

$$S_{NR} = ab_1, bc_8, aa_{15}, cc_{21}, ca_{25}, ab_{29}, bc_{34}, aa_{40}. \qquad (4)$$

A step-by-step grammar induction process is shown in Table 2. From Step 1 to Step 6, since neither digram uniqueness nor rule utility is violated, the algorithm simply reads the first six tokens and adds digrams into the digram table, respectively. In Step 7, the algorithm finds that the digram $\{ab, bc\}$ occurs in the digram table twice. Therefore, in Step 8, the algorithm forms a new rule, $R_1 \rightarrow ab, bc$. The new non-terminal symbol R_1 is generated to replace all occurrences of $\{ab, bc\}$, and the digram table is updated to maintain the uniqueness of digrams by removing $\{ab, bc\}$ and adding $\{R_1, aa\}$ and $\{ca, R_1\}$. Similarly, in Step 9, the digram $\{R_1, aa\}$ appears twice and therefore is replaced, in Step 10, by a new rule R_2. After the digram table is updated in Step 10, the algorithm finds that the rule R_1 only appears once. Therefore, in Step 11, R_1 is expanded to satisfy rule utility.

After processing the entire token sequence, the original token sequence is compressed into $R_0 \rightarrow R_2, cc, ca, R_2$ and the string cc, ca is identified as an anomaly candidate since it cannot be compressed.

5.2 Rule Density Curve

We next describe the construction of the rule density curve [20]. Simply stated, a rule density curve is a meta time series, in which each value is equal to the number of grammar rules that "cover" the respective time point. For anomaly detection, we are interested in the intervals for which the rule densities are the lowest. These intervals correspond to subsequences (more precisely, the SAX strings that represent these subsequences) that rarely appear in grammar rules, and hence are potentially anomalous. For the example sequence shown in Table 2, the time series points corresponding to the token subsequence cc, ca will have a count of zero since cc, ca does not appear in any grammar rules.

To construct the rule density curve, we map each instance of a rule back to the subsequence index based on the index recorded in the numerosity reduction step, and then we keep track of the number of rules that cover each time point.
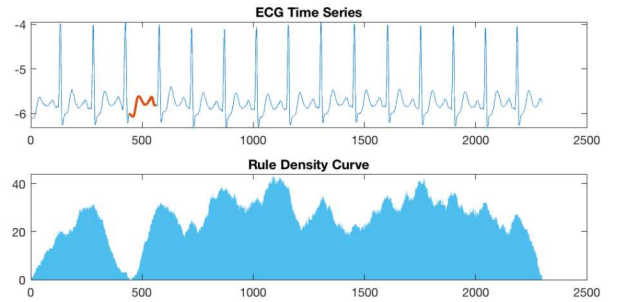

Figure 4: An example of the rule density curve generated for an ECG time series.

Once the rule density curve is constructed, we can locate the potentially anomalous subsequences by finding the local minima of the curve and ranking them based on their respective rule density values.

An example is shown in Figure 4. Figure 4.top shows an electrocardiogram (ECG) time series and Figure 4.bottom shows the rule density curve computed from this time series. An anomaly candidate, highlighted in red in Figure 4.top, corresponds to the minimum of the rule density curve. According to [20], this location corresponds to an anomalous premature heart beat.

5.3 Challenges of Parameter Selection

Although the grammar-induction-based approach described above has been successfully used in time series data mining problems [7, 20], it has two drawbacks that can strongly affect the quality of patterns found. First, approximation errors are inevitably introduced due to the information loss at the discretization step. For example, two subsequences represented by the same SAX word may actually be dissimilar and have a large distance. Second, Sequitur is a greedy algorithm which cannot guarantee the globally optimal result. Therefore, the grammar rules learned may not perfectly reflect the repeating and anomalous patterns in the time series. In summary, a single run of grammar induction with fixed parameter values may simply not be enough to detect high quality anomalies.

In this paper, in order to mitigate the limitations of the existing grammar-induction-based anomaly detection framework while still maintaining high efficiency, we introduce an ensemble approach to generate a rule density curve based on multiple runs with different parameter values.

Table 2: Example of Sequitur inducing grammar from token sequence S_{NR} of Eq. (4)

Step	Grammar rules	Digrams
1.	$S \to ab_1$	
2.	$S \to ab_1, bc_8$	$\{ab, bc\}$
3.	$S \to ab_1, bc_8, aa_{15}$	$\{ab, bc\}, \{bc, aa\}$
4.	$S \to ab_1, bc_8, aa_{15}, cc_{21}$	$\{ab, bc\}, \{bc, aa\}, \{aa, cc\}$
5.	$S \to ab_1, bc_8, aa_{15}, cc_{21}, ca_{25}$	$\{ab, bc\}, \{bc, aa\}, \{aa, cc\}, \{cc, ca\}$
6.	$S \to ab_1, bc_8, aa_{15}, cc_{21}, ca_{25}, ab_{29}$	$\{ab, bc\}, \{bc, aa\}, \{aa, cc\}, \{cc, ca\}, \{ca, ab\}$
7.	$S \to \boldsymbol{ab_1, bc_8}, aa_{15}, cc_{21}, ca_{25}, \boldsymbol{ab_{29}, bc_{34}}$	$\boldsymbol{\{ab, bc\}}, \{bc, aa\}, \{aa, cc\}, \{cc, ca\}, \{ca, ab\}, \boldsymbol{\{ab, bc\}}$
8.	$S \to R_1, aa_{15}, cc_{21}, ca_{25}, R_1$ $R_1 \to ab, bc$	$\{R_1, aa\}, \{bc, aa\}, \{aa, cc\}, \{cc, ca\}, \{ca, ab\}, \{ca, R_1\}$ $\{ab, bc\}$
9.	$S \to \boldsymbol{R_1, aa_{15}}, cc_{21}, ca_{25}, \boldsymbol{R_1, aa_{40}}$ $R_1 \to ab, bc$	$\boldsymbol{\{R_1, aa\}}, \{bc, aa\}, \{aa, cc\}, \{cc, ca\}, \{ca, ab\}, \{ca, R_1\}, \boldsymbol{\{R_1, aa\}}$ $\{ab, bc\}$
10.	$S \to R_2, cc_{21}, ca_{25}, R_2$ $\boldsymbol{R_1 \to ab, bc}$ $R_2 \to R_1, aa$	$\{R_2, cc\}, \{cc, ca\}, \{ca, R_2\}$ $\{ab, bc\}$ $\boldsymbol{\{R1, aa\}}$
11.	$S \to R_2, cc_{21}, ca_{25}, R_2$ $R_2 \to ab, bc, aa$	$\{R_2, cc_{21}\}, \{cc, ca\}, \{cc_{21}, R_2\}$ $\{ab, bc\}, \{bc, aa\}$

6 ENSEMBLE GRAMMAR INDUCTION

In this section, we introduce the proposed ensemble-based grammar induction approach.

6.1 Ensemble Rule Density Curve

The algorithm that generates the ensemble rule density curve from a set of multiple grammar induction runs is shown in Algorithm 1. First, we generate N rule density curves using values for PAA size w and alphabet size a randomly chosen from intervals $[2, w_{\max}]$ and $[2, a_{\max}]$, respectively (Lines 4–6). Second, we remove low-quality curves based on their standard deviations (Lines 7–10). Third, we normalize each remaining curve to the same scale (Line 11). Finally, we combine the results from all normalized curves by computing the median (Line 14).

Algorithm 1: Ensemble Rule Density Curve

1: **Input**: time series T, sliding window length n, ensemble size N, maximum PAA size $w_{\max}$, maximum alphabet size $a_{\max}$, ensemble selectivity τ
2: **Output**: ensemble rule density curve d_e
3: $\mathcal{D} = []$ {Compute N rule density curves with random parameter values and compute the standard deviation for each of them}
4: **for** $i = 1$ to N **do**
　　{Randomly generate parameter values; any w, a combination is used only once}
5:　$w, a = \text{GenerateRandomParam}(w_{\max}, a_{\max})$
6:　$d_i = \text{GrammarInduction}(T, n, w, a)$
7:　$s_i = \text{ComputeStd}(d_i);$
8: **end for**
9: index = ArgSort(s) {Sort the standard deviations in descending order}
10: **for** $i = 1$ to τN **do** {Keep $\tau\%$ of the density curves and normalize each of them}
11:　$d_{\text{norm}} = d_{\text{index}[i]} / \max(d_{\text{index}[i]})$
12:　$\mathcal{D}.\text{add}(d_{\text{norm}})$
13: **end for**
14: $d_e = \text{ComputeMedian}(\mathcal{D})$ {Compute the median for all normalized rule density curves}
15: **return** d_e

6.1.1 Removing Low-Quality Rule Density Curves. Not all rule density curves provide reliable information about anomalies. Intuitively, low-quality curves correspond to situations where a grammar rule set has a similar frequency everywhere, and they should be removed from the ensemble to improve the effectiveness of anomaly detection. Towards this end, the algorithm computes the standard deviation for each of the generated rule density curves (Line 8). We then rank the curves based on the standard deviation in descending order and only keep the top $\tau\%$ of the curves to form the ensemble set $\mathcal{D}$.

An example that illustrates the quality variation among different rule density curves is shown in Figure 5. All four curves in Figure 5 are generated from the ECG time series shown in Figure 4.top. The first two curves colored in blue are the top-2 curves based on the ranking by the standard deviation. The last two curves colored in red are the bottom-2 curves based on this ranking. As seen from the figure, it is very hard to determine the anomaly location from the bottom-2 curves. In contrast, the top-2 curves reveal the anomaly location clearly. This example demonstrates that using the rule density curves with higher standard deviations can help identify anomalies and potentially reduce the number of false positives.
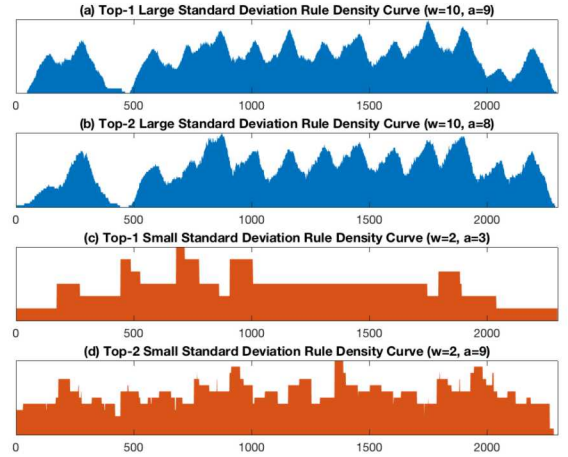

Figure 5: Examples of rule density curves generated from the ECG time series with different parameter values.

6.1.2 Normalizing Rule Density Curves. Intuitively, each rule density curve may have a different scale. For example, using a coarse discretization resolution (i.e., small PAA size and alphabet size) tends to result in a large average frequency of grammar rules since it is more likely to find a match due to a smaller dictionary size. Conversely, using a fine discretization resolution (i.e., large PAA size and alphabet size) tends to result in a small average frequency of grammar rules. Consequently, without a normalization process, some anomaly detectors may undeservedly dominate the decision.

To avoid this problem, we normalize each of the rule density curves in the ensemble set, so that each point of the curve falls in the range [0, 1]. It is worth noting that we do not use min-max normalization because we want to preserve the significance of the locations where the rule density is zero.

6.1.3 Combining Rule Density Curves. At the final step, we compute an ensemble rule density curve d_e based on all normalized curves kept in the set. In this paper, the specific way in which we combine all rule density curves in the ensemble is by computing the median value at each time point. Once the ensemble rule density curve is generated, the algorithm ranks the anomaly candidates through the same process as described in Sec. 5.2.

6.2 Multi-resolution SAX Word Computation

Since the ensemble approach requires computing different SAX words (corresponding to different parameter values) for the same subsequence, it is important to increase the efficiency of the discretization process. Therefore, in this subsection, we describe a fast way to compute multi-resolution SAX words [6].

6.2.1 Fast Computation of SAX Words with Different Values of w. We first describe a fast way to compute the PAA coefficients [16]. First, two vectors of statistical features for a time series T are pre-computed: $\text{ESum}_x(x) = \sum_{i=1}^{x} t_i$ and $\text{ESum}_{xx}(x) = \sum_{i=1}^{x} t_i^2$. Given a subsequence $T_{p,q}$ of length n, the SAX representation can be computed by Algorithm 2. In this algorithm, the mean and variance of $T_{p,q}$ are computed in constant time (Lines 3–5). The cost of computing the PAA coefficients (Lines 6–8) is $O(w)$ for a single resolution, which is faster than the trivial approach whose computation cost is $O(n)$ ($w < n$).

Algorithm 2: Fast Compute PAA (FastPAA)

1: **Input**: subsequence $T_{p,q}$, Esum_x,Esum_{xx}, PAA size w
2: **Output**: PAA representation A
3: $E_x = \text{Esum}_x(q) - \text{Esum}_x(p)$
4: $E_{xx} = \text{Esum}_{xx}(q) - \text{Esum}_{xx}(p)$
5: $n = q - p + 1$, $\mu = E_x/n$, $\sigma = \sqrt{(E_{xx} - E_x^2/n)/(n-1)}$
6: **for** every PAA segment **do**
7: $\quad A_i = \left(\frac{\text{Esum}_x(A_{i,e}) - \text{Esum}_x(A_{i,s})}{n/w} - \mu \right) / \sigma$ {$A_{i,s}$ and $A_{i,e}$ are the start and end points of the ith PAA segment}
8: **end for**
9: **return** A

6.2.2 Fast Computation of SAX Words with Different Values of a. To efficiently compute SAX words with multiple resolutions, we adapt an algorithm we introduced in a previous work [6]. Specifically, given a maximum alphabet size $a_{\max}$, to fast compute SAX words with different alphabet sizes, we first gather
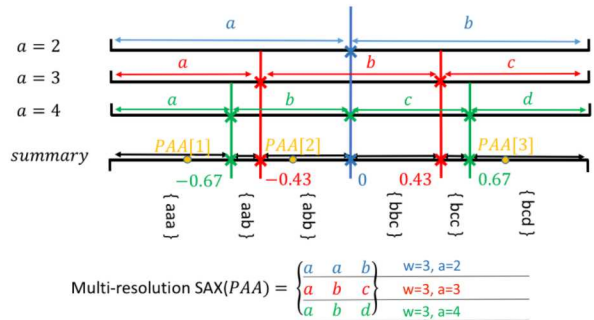

Figure 6: Fast computation of multi-resolution SAX.

breakpoint tables of all alphabet size values used in the ensemble. For each interval between any two breakpoints, a symbol sequence containing corresponding symbols up to $a_{\max}$ resolution is recorded. We represent any PAA coefficient belonging to the interval by the pre-computed symbol sequence.

An example with alphabet sizes from 2 to 4 is shown in Figure 6. In the figure, the set of breakpoints for all alphabet sizes from 2 to 4 (labeled by '×' in each line) are projected to the line denoted as "summary." All distinct breakpoints with a from 2 to 4 in the breakpoint table (Fig. 3.bottom) create 6 intervals, and each interval stores a sequence of symbols. The ith position in such a sequence stores the corresponding symbol for alphabet size $a = i + 1$. For example, the 3 PAA coefficients that fall in intervals $(-\infty, -0.63]$, $(-0.43, 0]$ and $(0.63, \infty)$ (denoted by yellow dots) are mapped to symbol sequences *aaa*, *abb*, and *bcd*, respectively. A symbol matrix is then created by concatenating such symbol sequences (bottom of Figure 6). The ith row of the symbol matrix represents a SAX word generated with alphabet size $i + 1$. For example, the first row *aab* is the SAX word generated for $a = 2$. To construct the matrix of SAX words with all alphabet sizes, for each PAA coefficient, we only need to perform at most 3 comparisons to determine the interval via binary search, which is the same as generating fixed-resolution SAX. By using binary search to determine which interval the PAA coefficient belongs to, we can find its SAX representations in all resolutions from $a = 2$ to $a = a_{\max}$ with a time complexity of $O(2\log(a_{\max}))$. When $a_{\max} = 20$, the cost of computing all resolutions is similar to computing a fixed resolution.

6.2.3 Overall Improvement. The time complexity of computing SAX words with multiple resolutions in a straightforward manner (without acceleration) is $O(nw_{\max}a_{\max} + w_{\max}^2 a_{\max}^2)$. In contrast, with the proposed acceleration, the computation cost is $O(w_{\max}^2 \log(a_{\max}))$, which is a significant improvement.

7 EXPERIMENTAL EVALUATION

We perform a series of numerical experiments to evaluate the accuracy and speed of ensemble grammar induction applied to time series anomaly detection. In all experiments, unless noted otherwise, the parameter values are: $a_{\max} = 10$, $w_{\max} = 10$, $N = 50$, and $\tau = 40\%$. All the experiments are conducted on a 16 GB RAM laptop with quad core processor of 2.5 GHz. We first show the performance on real-world time series. We then evaluate the impact of parameter sets used in the ensemble. Finally, we analyze the scalability of the algorithm and conduct a case study on an electric load time series.

Table 3: Properties of datasets used for experimental evaluation

Dataset	Time Series Length	Segment Length	Data Type
TwoLeadECG	1772	82	ECG
ECGFiveDay	2772	132	ECG
GunPoint	3150	150	Motion
Wafer	3150	150	Sensor
Trace	5775	275	Sensor
StarLightCurve	21504	1024	Sensor

Table 4: Performance evaluation results (average Score)

Dataset	Proposed Approach	GI-Random	GI-Fix	GI-Select	Discord
TwoLeadECG	0.3951	0.2873	0.0629	0.1663	**0.4931**
ECGFiveDay	0.3903	0.2988	0.2671	0.105	**0.4794**
GunPoint	**0.4728**	0.3715	0.2411	0.056	0.4
Wafer	**0.3179**	0.2126	0.1382	0.248	0.309
Trace	**0.5718**	0.2022	0.3601	0.3408	0.2816
StarLightCurve	**0.9369**	0.6930	0.5301	0.8759	0.9161

Table 5: Performance evaluation results (HitRate)

Dataset	Proposed Approach	GI-Random	GI-Fix	GI-Select	Discord
TwoLeadECG	0.72	0.52	0.4	0.24	0.8
ECGFiveDay	0.8	0.44	0.36	0.24	0.8
GunPoint	0.68	0.56	0.44	0.12	0.68
Wafer	0.72	0.4	0.36	0.4	0.52
Trace	0.96	0.4	0.8	0.6	0.52
StarLightCurve	1.0	0.96	0.76	1.0	1.0

7.1 Performance Evaluation in Comparison to Baseline Methods

Since it is difficult to find annotated time series for anomaly detection, to evaluate our proposed method, we compile our own data using several real-world and synthetic time series datasets from the UCR Time Series Classification Archive [3].

7.1.1 Datasets. We chose six time series datasets with diverse characteristics from the archive. The properties of the data are shown in Table 3. The datasets originate from different application domains including medicine (ECG), 3D motion tracking (GunPoint), manufacturing (Wafer), synthetic sensor data (Trace) and astronomy (StarLightCurve), with different instance lengths (ranging from 82 to more than 1000).

The instances are labeled with class information. For each dataset, we treat all instances that belong to the first class as "normal" data, and all the remaining instances that belong to other classes as "anomalous." We first generate a normal time series by concatenating 20 randomly selected normal instances. Then an anomalous instance is randomly selected and planted into the generated normal time series at a random position between 40% and 80% of the series. In this manner, we generate 25 time series for each of the six datasets. One example of the generated time series for each of the six datasets are shown in Figure 7, with planted anomalous instances highlighted in red. All methods being compared are run to locate the planted anomalous instance in each time series, with sliding window length n equal to the instance length.

7.1.2 Performance Evaluation. Each of the tested methods returns top-3 ranked anomaly candidates (which are required
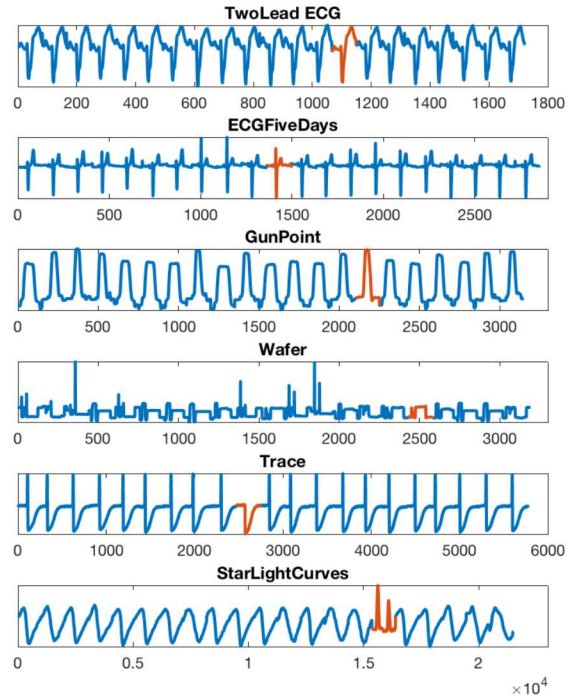

Figure 7: Examples of the test time series generated from six real-world and synthetic time series datasets. In each time series, the segment that belongs to a different class is highlighted in red.

to not overlap with each other) for each time series. We evaluate the performance of each method by quantifying the overlap between the discovered anomalies and the ground truth (the planted anomaly). Specifically, for each anomaly candidate, we compute a quantity called Score, defined as:

$$\text{Score} = 1 - \min\left(1, \frac{|\text{PredictLocation} - \text{GTLocation}|}{\text{GTLength}}\right), \quad (5)$$

where PredictLocation is the location of the anomaly candidate, while GTLocation and GTLength are the location and length of the planted anomalous instance. The maximum value, Score = 1, is obtained when PredictLocation exactly matches the ground truth. The better the overlap between the anomaly candidate and the ground truth, the higher the Score. If the anomaly candidate does not overlap with the ground truth at all, then Score = 0.

For each method and each time series, we only use the maximum Score achieved among the three anomaly candidates. For each method, we record the average Score value computed over the set of 25 time series generated per dataset; the average Score values are reported in Table 4. We also record the quantity called HitRate, which is the fraction of the anomaly candidates that overlap with the ground truth (i.e., satisfy the condition Score > 0); the HitRate values are reported in Table 5. Finally, we record the number of times the ensemble grammar induction method wins/ties/loses against a baseline method; these results are reported in Table 6.

7.1.3 Baselines. The proposed ensemble-based method is compared against four different baseline methods, which are described below.

- **Grammar Induction with Random Parameter Values** (GI-Random): The grammar-induction-based anomaly

detection approach with randomly selected parameter values w and a. The ranges from which w and a are chosen are the same as in the ensemble-based approach..

- **Grammar Induction with Fixed Generic Parameter Values** (GI-Fix): The grammar-induction-based anomaly detection approach with fixed parameter values $w = 4$ and $a = 4$. These are popular parameter values that can be used in most of datasets as reported in [20].
- **Grammar Induction with Selected Parameter Values** (GI-Select): The grammar-induction-based anomaly detection approach with w and a values selected via an optimization procedure described in [19], using 10% of the normal time series. The ranges from which w and a are chosen are the same as in the ensemble-based approach.
- **Time Series Discord** (Discord): The state-of-the-art approach that computes the one−nearest-neighbor (1-NN) distance for every subsequence in the time series; the subsequences with the largest 1-NN distances are labeled as anomalies. In the experiments, we use the latest Matrix-Profile-based implementation of this approach [23] to compute the 1-NN distances.

7.1.4 Results. The overall performance results measured by the average Score and HitRate are shown in Tables 4 and 5, respectively.

According to Table 4, the proposed approach achieves the highest average Score values in four out of six datasets, and the second-highest average Score values in two of the datasets. Compared with GI-Random, GI-Fix, and GI-Select, the proposed approach achieves higher average Score values in all six datasets. Compared with Discord, the proposed approach achieves a higher average Score value in four out of six datasets (GunPoint, Wafer, Trace and StarLightCurve). In two datasets (ECGFiveDays and TwoLeadECG), Discord outperforms the proposed approach.

According to Table 5, the proposed approach achieves the highest HitRate values (alone or shared) in five out of six datasets, and the second-highest HitRate value in one of the datasets. These results are consistent with those in Table 4 and indicate that the proposed approach can successfully locate an anomalous subsequence that strongly overlaps with the ground truth.

In addition, Figure 10 shows a detailed comparison summary of the proposed approach against all baselines. Each of the six rows corresponds to one of the datasets, and each of the four columns corresponds to one of the baselines. In each plot, a blue dot located at the point (x, y) denotes a pair of Score values (ensemble Score, baseline Score), computed from Eq. (5) for one of the 25 generated time series. A dot located in the lower triangle (highlighted in pink) of the plot corresponds to a win of the proposed approach (ensemble Score > baseline Score); a point located in the upper triangle corresponds to a loss (ensemble Score < baseline Score), and a point located on the diagonal corresponds to a tie (ensemble Score = baseline Score). The numbers of wins, ties, and losses of the proposed method against the baselines are shown in Table 6.

According to Figure 10, the proposed approach outperforms GI-Random, GI-Fix, and GI-Select. Moreover, in most of the tested datasets, the proposed method often detects ground truth anomalies that are completely missed by these variations of the grammar-induction-based approach (i.e., the respective baseline Score values are zero), while the opposite outcomes (ensemble Score values are zero) are quite rare. Compared with Discord, the proposed approach achieves similar performance. Moreover, in most of the tested datasets, cases where the ensemble-based approach discovers ground truth anomalies missed by Discord are more common than the opposite ones.

According to Table 6, compared with GI-Random, GI-Fix, and GI-Select, the proposed method wins in more than half of the time series in most datasets. Compared with Discord, the results are similar to those measured by Score and HitRate. Specifically, the proposed approach has more wins than losses in three datasets (GunPoint, Wafer, and Trace), loses more often than wins in two ECG datasets, and is in a virtual dead heat with Discord in the StarLightCurves dataset.

In summary, the experiments indicate that using an ensemble of parameter values can significantly improve the performance of the grammar-induction-based approach, compared to the variations of the method where a single combination of parameter values is selected. Also, the proposed method can achieve a competitive performance compared with the state-of-the-art discord approach. Importantly, while the latter has a quadratic time complexity, the former preserves a linear time complexity and hence is much more feasible for anomaly detection in large-scale data.

7.2 Effects of Parameter Value Ranges

In this subsection, we first evaluate how the ensemble grammar induction approach performs for different parameter value ranges determined by $w_{\max}$ and $a_{\max}$. Specifically, we evaluate the performance on the same time series that were used in Sec. 7.1. As the baseline for comparison, we use the best of the GI-Random, GI-Fix, and GI-Select methods for each dataset. We report the number of wins/ties/loses vs. the baseline for all tested parameter value ranges. We then evaluate how values of three hyper parameters — ensemble size N, ensemble selectivity τ, and sliding window length n — affect the result. In these experiments, we evaluate the performance based on Score and HitRate.

7.2.1 Effect of $w_{\max}$ and $a_{\max}$. We first test the proposed approach with values of $w_{\max}$ and $a_{\max}$ equal to each other and varying from 5 to 20. The results are shown in Table 7. According to the results, the smallest ranges ($w_{\max} = a_{\max} = 5$) lead to the worst performance. A possible explanation is that the ranges are too small to generate a sufficient number of high-quality rule density curves. The performance significantly improves for $w_{\max} = a_{\max} = 15$. However, increasing the range values beyond 15 is not useful; furthermore, in two datasets (TwoLeadECG and StarLightCurve) the performance slightly deteriorates when the ranges go up to 20, and in the GunPoint dataset the performance actually peaks for $w_{\max} = a_{\max} = 10$. The lack of performance improvement (or even some deterioration) at range values larger than 15 indicates that long and high-resolution SAX words may capture too much noise in data and, as a result, produce low-quality rule density curves. Nevertheless, the ensemble-based method still outperforms other approaches that rely on a single combination of parameter values.

7.2.2 Effect of $w_{\max}$. We also test the proposed approach with values of $w_{\max}$ varying from 5 to 20 and fixed $a_{\max} = 10$. The results are shown in Table 8. Once again, we observe that when the range is too small ($w_{\max} = 5$), the performance is the worst. The performance significantly improves for larger values of $w_{\max}$, with the peak-performance value depending on the dataset.

7.2.3 Effect of $a_{\max}$. In this experiment, we test the proposed approach with values of $a_{\max}$ varying from 5 to 20 and fixed $w_{\max} = 10$. The results are shown in Table 9. For $a_{\max} = 5$, the performance is subpar for the ECGFiveDays, GunPoint, and

Table 6: Wins/ties/losses of ensemble grammar induction against all baselines

Approach Dataset	TwoLeadECG	ECGFiveDays	GunPoint	Wafer	Trace	StarLightCurve
GI-Random	12/5/8	17/3/5	14/5/6	13/5/7	20/1/4	18/1/6
GI-Fix	17/7/1	13/5/7	15/4/6	17/6/2	14/1/10	24/0/1
GI-Select	14/5/6	18/5/2	16/8/1	9/8/8	14/3/8	17/0/8
Discord	8/4/13	9/1/15	14/7/4	12/5/8	18/1/6	12/0/13

Table 7: Wins/ties/losses of ensemble grammar induction against best GI baseline for different values of a_{max} and w_{max}

Approach	TwoLeadECG	ECGFiveDays	GunPoint	Wafer	Trace	StarLightCurve
$a_{max} = 5, w_{max} = 5$	1/12/12	8/9/8	3/9/13	3/14/9	4/11/10	2/0/23
$a_{max} = 10, w_{max} = 10$	12/5/8	13/5/7	14/5/6	9/8/8	14/1/10	17/0/8
$a_{max} = 15, w_{max} = 15$	14/4/7	17/2/6	13/4/8	13/7/5	15/0/10	18/0/7
$a_{max} = 20, w_{max} = 20$	12/4/9	17/2/6	13/4/8	13/7/5	15/0/10	17/1/7

Table 8: Wins/ties/losses of ensemble grammar induction against best GI baseline for different values of w_{max}

Approach	TwoLeadECG	ECGFiveDays	GunPoint	Wafer	Trace	StarLightCurve
$a_{max} = 10, w_{max} = 5$	5/9/11	6/8/11	5/6/14	7/9/9	4/10/11	1/0/24
$a_{max} = 10, w_{max} = 10$	12/5/8	13/5/7	14/5/6	9/8/8	14/1/10	17/0/8
$a_{max} = 10, w_{max} = 15$	10/5/10	18/3/4	11/6/8	18/3/4	15/0/10	19/0/6
$a_{max} = 10, w_{max} = 20$	12/4/9	18/2/5	10/4/11	14/3/8	16/0/9	20/0/5

Table 9: Wins/ties/losses of ensemble grammar induction against best GI baseline for different values of a_{max}

Approach	TwoLeadECG	ECGFiveDays	GunPoint	Wafer	Trace	StarLightCurve
$a_{max} = 5, w_{max} = 10$	11/5/9	8/8/9	7/8/10	12/7/6	11/5/9	1/1/23
$a_{max} = 10, w_{max} = 10$	12/5/8	13/5/7	14/5/6	9/8/8	14/1/10	17/0/8
$a_{max} = 15, w_{max} = 10$	11/6/8	13/6/6	13/4/8	8/8/9	16/0/9	15/0/10
$a_{max} = 20, w_{max} = 10$	11/4/10	14/5/6	13/4/8	9/9/7	15/0/10	12/1/12

Trace datasets, and it is terrible for the StarLightCurve dataset (as a rule, this dataset exhibits the worst performance when one or both range values are small). However, $a_{max} = 5$ actually results in the best performance for the Wafer dataset. For most datasets, the a_{max} values of 10, 15, and 20 produce very similar results.

The last two experiments indicate that having a larger range for w is more important than for a, which may indicate that a variation in the PAA size has a larger effect on the performance of the algorithm than a variation in the alphabet size, and hence w is a more important parameter to choose, in general.

Table 10: Performance (average Score) vs. N

Dataset	$N = 5$	$N = 10$	$N = 25$	$N = 50$
TwoLeadECG	0.3424	0.3488	0.3912	0.3951
ECGFiveDays	0.37	0.3882	0.4168	0.3903
GunPoint	0.3128	0.4629	0.4965	0.4728
Wafer	0.2308	0.2637	0.2839	0.3179
Trace	0.4767	0.5789	0.5994	0.5718
StarLightCurve	0.8244	0.7593	0.8676	0.9369

Table 11: Performance (HitRate) vs. N

Dataset	$N = 5$	$N = 10$	$N = 25$	$N = 50$
TwoLeadECG	0.52	0.6	0.72	0.72
ECGFiveDays	0.68	0.72	0.76	0.8
GunPoint	0.56	0.76	0.68	0.68
Wafer	0.44	0.64	0.6	0.72
Trace	0.76	0.96	0.96	0.96
StarLightCurve	1.0	1.0	1.0	1.0

7.2.4 Effect of Ensemble Size. We next evaluate the proposed method with the ensemble size N varying from 5 to 50 (recall that all other experiments used the fixed value $N = 50$). The average Score and HitRate of the proposed method for different ensemble sizes are shown in Tables 10 and 11, respectively.

We observe that the performance for a small ensemble size ($N = 5$) is worse than for a larger ensemble size ($N = 25$ or $N = 50$) for all datasets. In general, the Score and HitRate values increase as N grows, but these increases saturate when the ensemble size is large enough (e.g., when $N \geq 25$). The results indicate that selecting $N \geq 25$ may be suitable for most cases.

We also observe that the Score values are, in general, rather low for all methods, for most datasets except StarLightCurve. This could be due to the fact that the Score is normalized by the ground truth length (cf. Eq. (5)). However, the planted anomaly may only have a small section that differs from the "normal" data, and that may be what the algorithms detect. In such case, treating the entire planted instance as anomalous would indeed lower the Score. This explanation is later validated by the results in Table 13, which show that a shorter sliding window length results in higher Score values in some cases. Nevertheless, this is why we also use the HitRate, which is independent of the ground truth length, as an alternate measure.

7.2.5 Effects of Ensemble Selectivity. Next, we evaluate the proposed method with ensemble selectivity τ varying from 5% to 100%. In this experiment, the evaluation of the average Score value (which is computed over 25 time series) is repeated 20 times for each dataset and τ value. We then compute the mean and standard deviation over the set of 20 average Score values. The results are shown in Table 12. We observe that better performance is typically achieved for smaller τ values (e.g., from 5% for ECGFiveDays and Trace datasets to 20% for GunPoint and StarLightCurve datasets), while the standard deviation is typically smaller for τ values of 20% or 40%. Therefore, we recommend choosing τ around 20% since it combines a relatively high level of performance with a relatively small variance of results.

7.2.6 Effects of Sliding Window Length. In this experiment, we investigate how the proposed method performs when the sliding window length n is less than the ground truth anomaly

Table 12: Mean and standard deviation over 20 average Score values, vs. τ

Dataset	$\tau = 5\%$	$\tau = 10\%$	$\tau = 20\%$	$\tau = 40\%$	$\tau = 80\%$	$\tau = 100\%$
TwoLeadECG	0.4149 (0.04)	0.4196 (0.032)	0.4 (0.026)	0.3882 (0.027)	0.3354 (0.036)	0.3071 (0.032)
ECGFiveDays	0.425 (0.042)	0.41 (0.045)	0.38 (0.038)	0.37 (0.037)	0.35 (0.024)	0.32 (0.036)
GunPoint	0.488 (0.042)	0.50 (0.037)	0.505 (0.035)	0.488 (0.025)	0.43 (0.023)	0.412 (0.023)
Wafer	0.339 (0.05)	0.371 (0.042)	0.337 (0.027)	0.311 (0.027)	0.27 (0.032)	0.26 (0.037)
Trace	0.6136 (0.037)	0.6017 (0.035)	0.5972 (0.025)	0.5864 (0.024)	0.4997 (0.046)	0.4166 (0.042)
StarLightCurve	0.9057 (0.017)	0.9183 (0.016)	0.9327 (0.009)	0.9052 (0.012)	0.7359 (0.021)	0.628 (0.021)

Table 13: Performance (average Score) vs. n

Dataset	$n = 0.6n_a$	$n = 0.7n_a$	$n = 0.8n_a$	$n = 0.9n_a$	$n = n_a$
TwoLeadECG	0.4620	0.4605	0.4107	0.4259	0.3951
ECGFiveDays	0.4391	0.3691	0.3535	0.3797	0.3903
GunPoint	0.4373	0.4992	0.4680	0.4371	0.4728
Wafer	0.3095	0.4195	0.3389	0.2824	0.3179
Trace	0.5229	0.5911	0.5689	0.5852	0.5718
StarLightCurve	0.8624	0.8998	0.9216	0.9048	0.9369

Table 14: Performance (HitRate) vs. n

Dataset	$n = 0.6n_a$	$n = 0.7n_a$	$n = 0.8n_a$	$n = 0.9n_a$	$n = n_a$
TwoLeadECG	0.72	0.84	0.8	0.76	0.72
ECGFiveDays	0.96	0.8	0.84	0.72	0.8
GunPoint	0.84	0.68	0.72	0.64	0.68
Wafer	0.56	0.64	0.52	0.52	0.72
Trace	1.0	1.0	1.0	1.0	0.96
StarLightCurve	1.0	1.0	1.0	1.0	1.0

(a) RW time series

(b) ECG time series

(c) EEG time series

Figure 8: Scalability: Computation time vs. time series length.

length (denoted as n_a). Specifically, we test five different sliding window lengths equal to 60%, 70%, 80%, 90% and 100% of n_a. The average Score and HitRate values are shown in Tables 13 and 14, respectively. According to the results, while there exists some variation in the performance, the dependence on n is not significant, and the proposed method robustly outperforms the existing grammar-induction-based approaches in most cases.
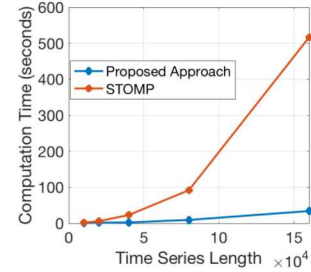
7.3 Scalability

We evaluate the scalability of the proposed ensemble-based approach by applying it to a 160,000 length random walk (RW) time series, ECG data [18] and electroencephalogram (EEG) data [13]. We compare the computation times for the proposed method and for the state-of-the-art discord discovery approach. In the experiment, we use STOMP [23], the latest Matrix-Profile-based algorithm, to detect discords. It has been shown that STOMP is both faster and more robust to different types of data compared to the original discord discovery method HOTSAX [9].
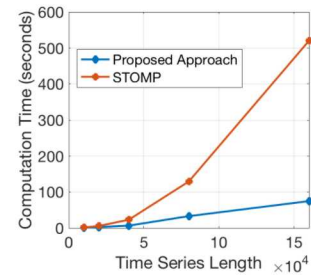
The scalability, evaluated as the computation time versus the time series length, is shown in Figure 8. We see that, as the time series length increases, the computation time grows significantly slower for the proposed method than for STOMP. At the largest time series length, the proposed approach is about one order of magnitude faster than STOMP, for all three types of time series data. We also find that the computation times for both approaches are roughly independent of the sliding window length.

7.4 Case Study: Anomaly Detection in Electric Power Usage Time Series

Interpreting the time dependence of electric power usage has many potential applications [21]. In this section, we show that the ensemble grammar induction method can find anomalies in large-scale electric power usage time series. We use 100 days of the fridge-freezer power usage data provided in [14], to evaluate the performance of the proposed method. The entire time series, which consists of approximately 600,000 points, is shown in Figure 9(a), and the first 20,000 points are shown in Figure 9(b). We run the proposed method with the sliding window length of 900, which is about the duration of one cycle (shown in red box in Figure 9(b)). The computation time is about one minute. Figures 9(c) and 9(d) show the two top-ranked anomaly candidates detected by the proposed method.

We see that the top-1 anomaly represents a cycle whose shape is unusual compared to the typical cycles shown in Figure 9(b). The top-2 anomaly represents an unusual event that contains normal cycles and short spikes. The two anomalies represent different, unusual power usage patterns. Since this time series is very long, and the anomalies have different lengths, using the state-of-the-art discord discovery approach would be time consuming. In contrast, the proposed method provides an efficient way to detect anomalies.
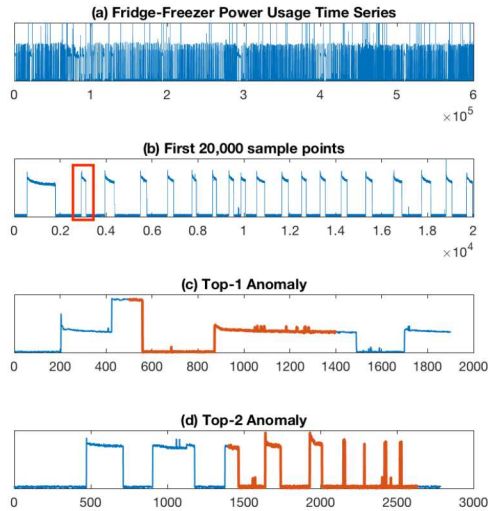
Figure 9: A 600,000 length fridge-freezer power usage time series, and two top-ranked anomalies detected in this series by ensemble grammar induction.

7.5 Detecting Multiple Anomalies

We also investigate the effectiveness of the proposed approach in detecting multiple anomalies in time series. In this experiment, we use the StarlightCurve dataset to generate 10 time series. Each of these time series is of length 43008 and contains two randomly selected and placed anomalies of length 1024. We evaluate the performance by the number of ground truth anomalies detected (a ground truth anomaly is considered detected if it overlaps with at least one of the top-3 ranked anomaly candidates). The proposed method performed well as it successfully identified both anomalies in nine time series and one of the two anomalies in one time series.

8 CONCLUSION

In this paper, we introduce a robust grammar-induction-based anomaly detection approach utilizing ensemble learning. Instead of using a particular combination of parameter values for anomaly detection, the proposed method generates the final result based on a set of results obtained using different parameter values. The experiments performed on datasets with known ground truth show that the proposed ensemble approach can outperform existing grammar-induction-based approaches with different criteria for selection of parameter values. We also show that the proposed approach, which has a linear time complexity with respect to the data size, can achieve performance similar to that of the state-of-the-art distance-based anomaly detection approach that has a quadratic time complexity.

ACKNOWLEDGMENTS

REFERENCES

[1] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. 2000. LOF: identifying density-based local outliers. In *ACM sigmod record*, Vol. 29. ACM, 93–104.

[2] Xiao-yun Chen and Yan-yan Zhan. 2008. Multi-scale anomaly detection algorithm based on infrequent pattern of time series. *J. Comput. Appl. Math.* 214, 1 (2008), 227–237.

[3] Hoang Anh Dau, Eamonn Keogh, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, Yanping, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, Gustavo Batista, and Hexagon-ML. 2018. The UCR Time Series Classification Archive. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.

[4] Len Feremans, Vincent Vercruyssen, Boris Cule, Wannes Meert, and Bart Goethals. 2019. Pattern-based anomaly detection in mixed-type time series. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*.

[5] Yifeng Gao, Qingzhe Li, Xiaosheng Li, Jessica Lin, and Huzefa Rangwala. 2017. TrajViz: A Tool for Visualizing Patterns and Anomalies in Trajectory. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 428–431.

[6] Yifeng Gao and Jessica Lin. 2018. Exploring variable-length time series motifs in one hundred million length scale. *Data Mining and Knowledge Discovery* 32, 5 (2018), 1200–1228.

[7] Yifeng Gao, Jessica Lin, and Huzefa Rangwala. 2016. Iterative Grammar-Based Framework for Discovering Variable-Length Time Series Motifs. In *Machine Learning and Applications (ICMLA), 2016 15th IEEE International Conference on*. IEEE, 7–12.

[8] C Sweetlin Hemalatha, Vijay Vaidehi, and R Lakshmi. 2015. Minimal infrequent pattern based approach for mining outliers in data streams. *Expert Systems with Applications* 42, 4 (2015), 1998–2012.

[9] Eamonn Keogh, Jessica Lin, and Ada Fu. 2005. Hot sax: Efficiently finding the most unusual time series subsequence. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*. Ieee, 8–pp.

[10] Yuan Li, Jessica Lin, and Tim Oates. 2012. Visualizing Variable-Length Time Series Motifs.. In *Proceedings of the 2012 SIAM International Conference on Data Mining*. SIAM, 895–906.

[11] Jessica Lin, Eamonn Keogh, Stefano Lonardi, Jeffrey P Lankford, and Donna M Nystrom. 2004. Visually mining and monitoring massive time series. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 460–469.

[12] Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. 2007. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and knowledge discovery* 15, 2 (2007), 107–144.

[13] Abdullah Mueen. 2013. Enumeration of time series motifs of all lengths. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*. IEEE, 547–556.

[14] David Murray, Jing Liao, Lina Stankovic, Vladimir Stankovic, Richard Hauxwell-Baldwin, Charlie Wilson, Michael Coleman, Tom Kane, and Steven Firth. 2015. A data management platform for personalised real-time energy feedback. In *Proceedings of the 8th International Conference on Energy Efficiency in Domestic Appliances and Lighting*. 1–15.

[15] Craig G. Nevill-Manning and Ian H. Witten. 1997. Identifying hierarchical strcture in sequences: A linear-time algorithm. *J. Artif. Intell. Res.(JAIR)* 7 (1997), 67–82.

[16] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. 2012. 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*. ACM, 262–270.

[17] Shebuti Rayana and Leman Akoglu. 2015. Less is more: Building selective anomaly ensembles with application to event detection in temporal graphs. In *Proceedings of the 2015 SIAM International Conference on Data Mining*. SIAM, 622–630.

[18] Pavel Senin, Jessica Lin, Xing Wang, Tim Oates, Sunil Gandhi, Arnold P Boedihardjo, Crystal Chen, and Susan Frankenstein. 2015. Time series anomaly discovery with grammar-based compression.. In *EDBT*. 481–492.

[19] Pavel Senin, Jessica Lin, Xing Wang, Tim Oates, Sunil Gandhi, Arnold P Boedihardjo, Crystal Chen, and Susan Frankenstein. 2018. Grammarviz 3.0: Interactive discovery of variable-length time series patterns. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 12, 1 (2018), 10.

[20] Pavel Senin, Jessica Lin, Xing Wang, Tim Oates, Sunil Gandhi, Arnold P Boedihardjo, Crystal Chen, Susan Frankenstein, and Manfred Lerner. 2014. GrammarViz 2.0: a tool for grammar-based pattern discovery in time series. In *Machine Learning and Knowledge Discovery in Databases*. Springer, 468–472.

[21] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, and Eamonn Keogh. 2016. Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View that Includes Motifs, Discords and Shapelets. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*. 1317–1322.

[22] Yue Zhao, Zain Nasrullah, Maciej K Hryniewicki, and Zheng Li. 2019. LSCP: Locally selective combination in parallel outlier ensembles. In *Proceedings of the 2019 SIAM International Conference on Data Mining*. SIAM, 585–593.

[23] Yan Zhu, Zachary Schall-Zimmerman, Nader Shakibay Senobari, Chin-Chia Michael Yeh, Gareth Funning, Abdullah Mueen, Philip Brisk, and Eamonn J. Keogh. 2016. Matrix Profile II: Exploiting a Novel Algorithm and GPUs to Break the One Hundred Million Barrier for Time Series Motifs and Joins. *2016 IEEE 16th International Conference on Data Mining (ICDM)* (2016), 739–748.

(a) Vs. GI-Random (TwoLeadECG)   (b) Vs. GI-Fix (TwoLeadECG)   (c) Vs. GI-Select (TwoLeadECG)   (d) Vs. Discord (TwoLeadECG)

(e) Vs. GI-Random (ECGFiveDays)   (f) Vs. GI-Fix (ECGFiveDays)   (g) Vs. GI-Select (ECGFiveDays)   (h) Vs. Discord (ECGFiveDays)

(i) Vs. GI-Random (Wafer)   (j) Vs. GI-Fix (Wafer)   (k) Vs. GI-Select (Wafer)   (l) Vs. Discord (Wafer)

(m) Vs. GI-Random (GunPoint)   (n) Vs. GI-Fix (GunPoint)   (o) Vs. GI-Select (GunPoint)   (p) Vs. Discord (GunPoint)

(q) Vs. GI-Random (Trace)   (r) Vs. GI-Fix (Trace)   (s) Vs. GI-Select (Trace)   (t) Vs. Discord (Trace)

(u) Vs. GI-Random (StarLightCurve)   (v) Vs. GI-Fix (StarLightCurve)   (w) Vs. GI-Select (StarLightCurve)   (x) Vs. Discord (StarLightCurve)
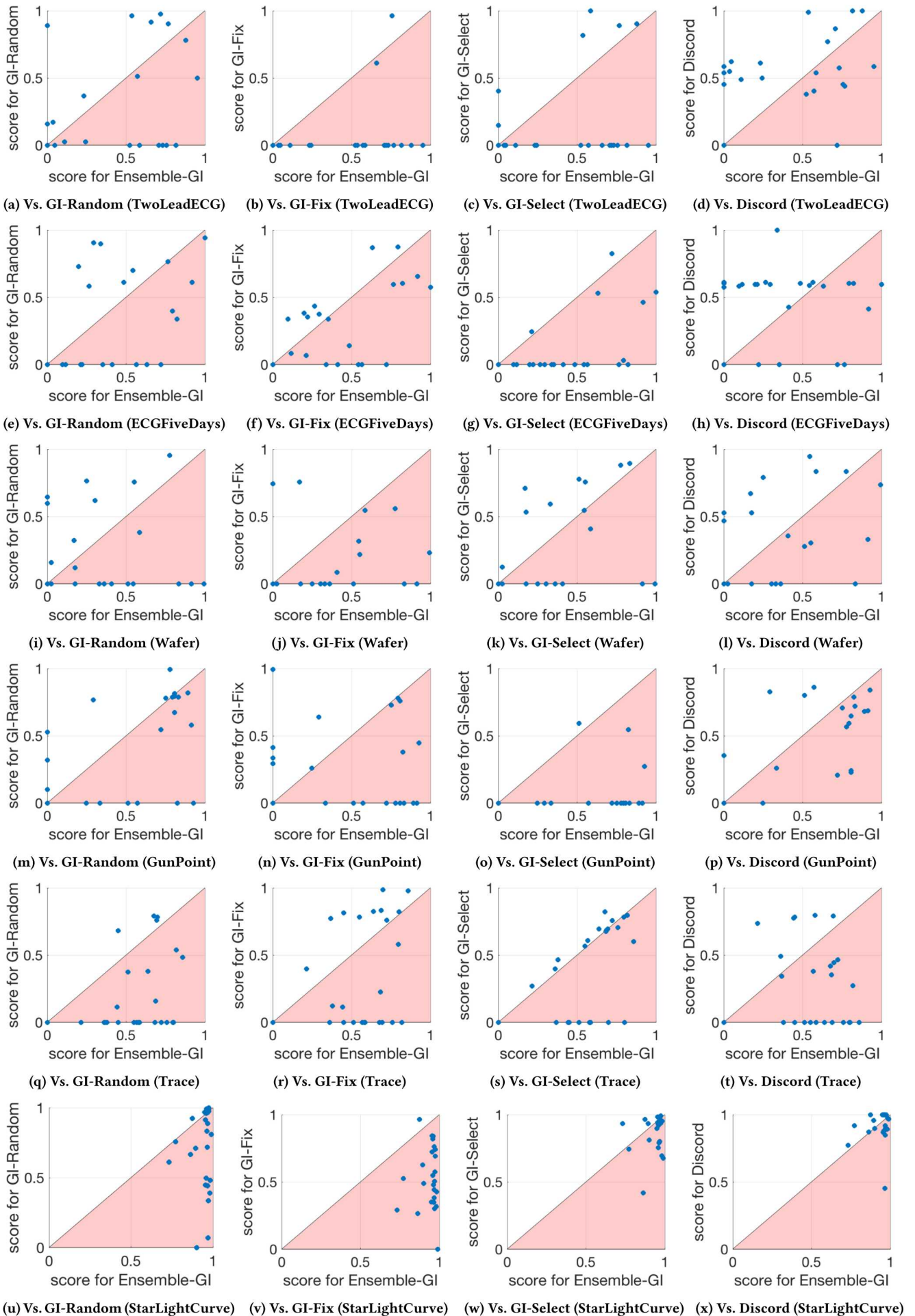
Figure 10: Summary of performance comparison of ensemble grammar induction against baseline methods.