ISSN 1995-0802, Lobachevskii Journal of Mathematics, 2024, Vol. 45, No. 3, pp. 1333–1347. © Pleiades Publishing, Ltd., 2024.

PaSTiLa: Scalable Parallel Algorithm for Unsupervised Labeling of Long Time Series

M. L. Zymbler[1]* and A. I. Goglachev[1]**

(Submitted by E. K. Lipachev)

[1] *South Ural State University, Chelyabinsk, 454080 Russia*

Received January 18, 2024; revised January 31, 2024; accepted February 13, 2024

Abstract—Summarization aims at discovering a small set of typical subsequences (patterns) in the given long time series that represent the whole series. Further, one can implement unsupervised labeling of the given time series by assigning each subsequence a tag that corresponds to its most similar pattern. In the previous research, we developed the PSF (Parallel Snippet-Finder) algorithm for the time series summarization on GPU, where a snippet is the given-length subsequence, which is similar to many other subsequences w.r.t. the bespoke distance measure MPdist. However, PSF is limited by the demand that the snippet length be predefined by a domain expert. In this article, we introduce the novel parallel algorithm PaSTiLa (Parallel Snippet-based Time series Labeling) that discovers snippets and produces the labeling of the given time series on an HPC cluster with GPU nodes. As opposed to its predecessor, PaSTiLa employs the automatic selection of the snippet length from the specified range through our proposed heuristic criterion. In the experiments on labeling quality over time series from the TSSB (Time Series Segmentation Benchmark) dataset, PaSTiLa outperforms state-of-the-art segmentation-based competitors in average F_1 score. In the case of long-length time series (typically more than 8–10 K points), PaSTiLa outruns the rivals. Finally, over the million-length time series, our algorithm demonstrates a close-to-linear speedup.

DOI: 10.1134/S1995080224600766

Keywords and phrases: *time series, summarization, labeling, snippet, MPdist, Snippet-Finder, parallel algorithm, GPU, CUDA, high-performance cluster*

INTRODUCTION

Over the last decade, a wide spectrum of subject domains have been faced with the processing of long time series generated by high-frequency sensors: Internet of Things, digital industry, personal healthcare, climate modeling and prediction of natural disasters, etc. Summarization is one of the basic time series mining problems, and aims to discover a small set of patterns (typical subsequences) that provide a concise representation the given long time series. Further, one can implement unsupervised labeling of the time series by assigning each subsequence a tag that corresponds to its most similar pattern. It is worth noting that the time series segmentation [4, 7, 20] is closely related to the unsupervised labeling topic since the segmentation aims at splitting the given time series into intervals that are semantically different from neighboring ones.

An apparent approach to summarization, the motif concept [16] is limited, since motifs cannot indicate the coverage, a fraction of the given time series, of the pattern found. Another approach, the shapelet concept [22] is limited as well, since it is supervised due to demand of training data. Recently introduced, the snippet concept [11] eliminates the limitations above. A time series snippet is the given-length subsequence, which is similar to many other subsequences w.r.t. the bespoke distance measure MPdist [6]. In turn, the similarity of two equal-length subsequences w.r.t. MPdist is proportional to the number of smaller-length sub-subsequences in them that are close to each other w.r.t. the normalized

*E-mail: mzym@susu.ru

**E-mail: goglachevai@susu.ru

Euclidean distance, regardless of the order of matching sub-subsequences. The original Snippet-Finder algorithm [11] provides unsupervised labeling of the given time series since all the subsequences that are similar to a snippet are exactly specified and counted, however, at a cubic time complexity concerning the time series length.

Our developed PSF (Parallel Snippet-Finder) algorithm [24] accelerates the original snippet discovery schema on GPU (graphics processing unit) and ensures acceptable performance. However, PSF, like its predecessor, is limited by the fact that the snippet length should be predefined by a domain expert, whose choice may not be optimal. Figure 2 illustrates the low- and high-quality labeling of a human's electrocardiogram with two classes of signals: a normal heartbeat and a myocardial infarction, depending on the snippet length chosen. Thus, PSF can be improved by the automatic selection of some optimal snippet length from the specified range.

The article pushes our previous research forward and contributes as follows:

- We introduce the novel parallel algorithm PaSTiLa (Parallel Snippet-based Time series Labeling) that discovers snippets and produces the labeling of the given time series on an HPC cluster with GPU nodes. As opposed to PSF, its predecessor, instead of using the snippet length predefined by a domain expert, PaSTiLa employs the automatic selection of the snippet length from the specified range through our proposed heuristic criterion.

- We carry out extensive experiments to evaluate our algorithm over real-world time series against state-of-the-art segmentation-based analogs [4, 7, 20]. In the experiments on labeling quality over 75 time series from the TSSB (Time Series Segmentation Benchmark) dataset [4], PaSTiLa outperforms competitors in average F_1 score. In the case of long-length time series (typically more than 8–10 K points), PaSTiLa outruns the rivals. Finally, over the millions-length time series, our algorithm outruns the above rivals and demonstrates a close-to-linear speedup. To facilitate the reproducibility of our study, we establish a repository [8] that contains the algorithm's source code, data, etc.

The remainder of the article is organized as follows. Section 1 briefly discusses related works. In Section 2, we introduce the notation and formal definitions, along with a short description of the Snippet-Finder and PSF (Parallel Snippet-Finder) algorithms our study is based on Section 3 introduces PaSTiLa, our novel parallel algorithm for automatic unsupervised labeling of long time series. In Section 4, we discuss the results of the experimental evaluation of PaSTiLa. Finally, in Conclusions, we summarize the results obtained and suggest directions for further research.

1. RELATED WORK

In this section, we briefly discuss several state-of-the-art approaches to the following topics closely related to our research: time series summarization, time series segmentation, and change point detection. Summarization aims at discovering a small set of patterns that can be used to represent and label the given time series. Segmentation aims at splitting the given time series into intervals that are semantically different from neighboring ones. Change point detection identifies the locations of shifts from one segment to another that are caused by operational state changes in the process being monitored. Despite the fact that segmentation and change point detection problems concern discovering boundary locations and not producing representative patterns, further, we employ them in the experimental evaluation of our approach.

A time series snippet [11] is the given-length subsequence, which is similar to many other subsequences w.r.t. the bespoke distance measure MPdist [6]. In turn, the similarity of two equal-length subsequences w.r.t. MPdist is proportional to the number of smaller-length sub-subsequences in them that are close to each other w.r.t. the normalized Euclidean distance, regardless of the order of matching sub-subsequences. Informally speaking, the snippet discovery in the time series is somewhat like K-medoids clustering [13] of the multidimensional points: the snippet and its nearest neighbors correspond to the medoid and points of the cluster, respectively, where a medoid is a point whose sum of dissimilarities to all the points in the cluster is minimal. Snippet-Finder [11] is the algorithm for the snippet discovery that provides all the subsequences that are similar to a snippet to be exactly specified and counted. Despite the fact that Snippet-Finder is unsupervised and demands only one

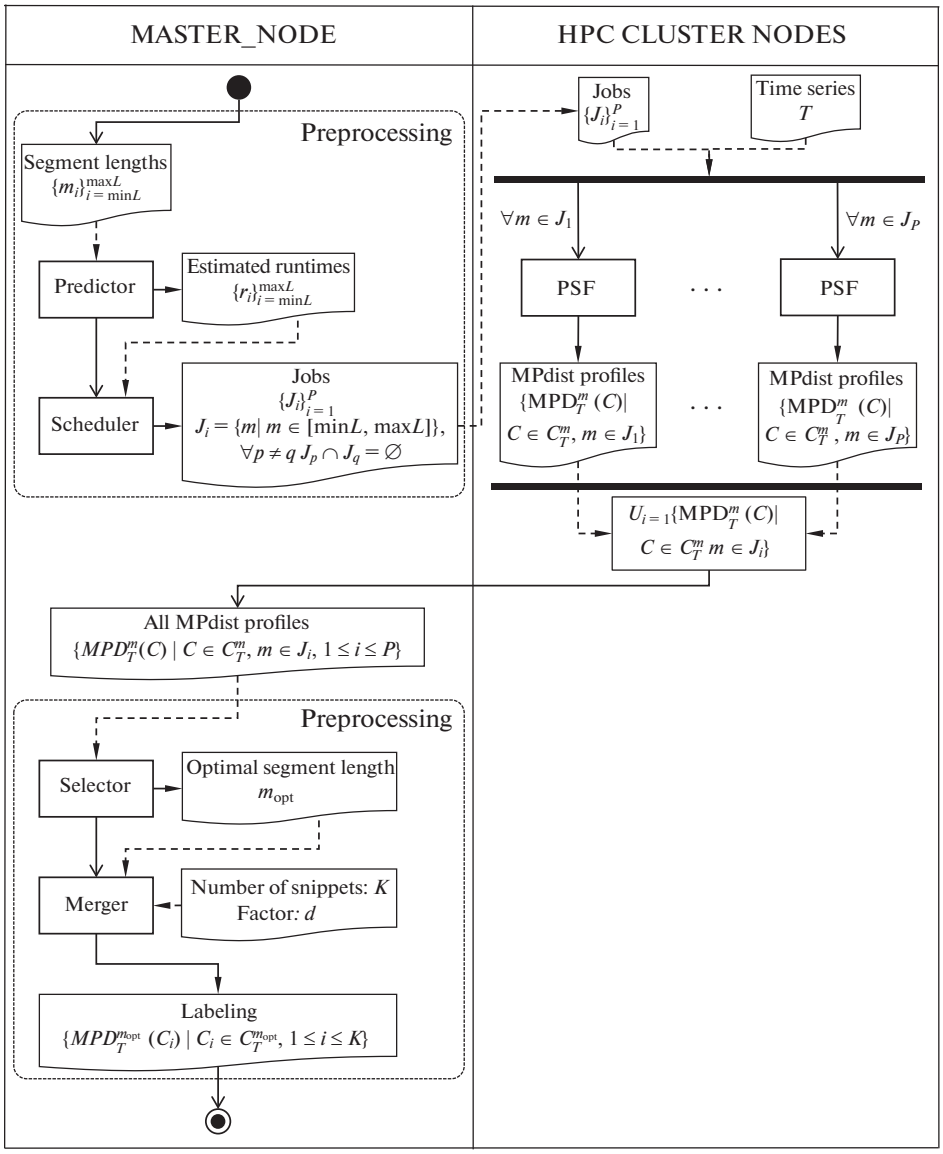


Fig. 1. Parallelization scheme of PaSTiLa.

parameter, the snippet length, its time complexity is cubic w.r.t. the time series length. The PSF (Parallel Snippet-Finder) algorithm [24] accelerates the original snippet discovery schema with GPU and ensures acceptable performance. However, PSF can be improved if, instead of the predefined snippet length, we employ the automatic selection of some optimal one from the specified range.

A time series motif [16] is a pair of the given-length subsequences that are very similar to each other. The MK algorithm [16] employs the Euclidean distance and triangular inequality to effectively prune unpromising candidates for motif. There are developments that accelerate MK on various parallel architectures [25, 26]. However, being chosen as representative patterns, motifs cannot indicate the coverage of each pattern.

A time series shapelet [22] is the given-length subsequence that is both the most similar one to most of the subsequences of a given class and the most dissimilar one from the subsequences of other classes. Since the shapelet assumes the subsequences of the given time series are pre-classified, such a concept cannot be considered unsupervised.

The ClaSP (Classification Score Profile) algorithm [4] hierarchically splits the given time series into two parts. A change point is determined by training a binary time series classifier for each possible

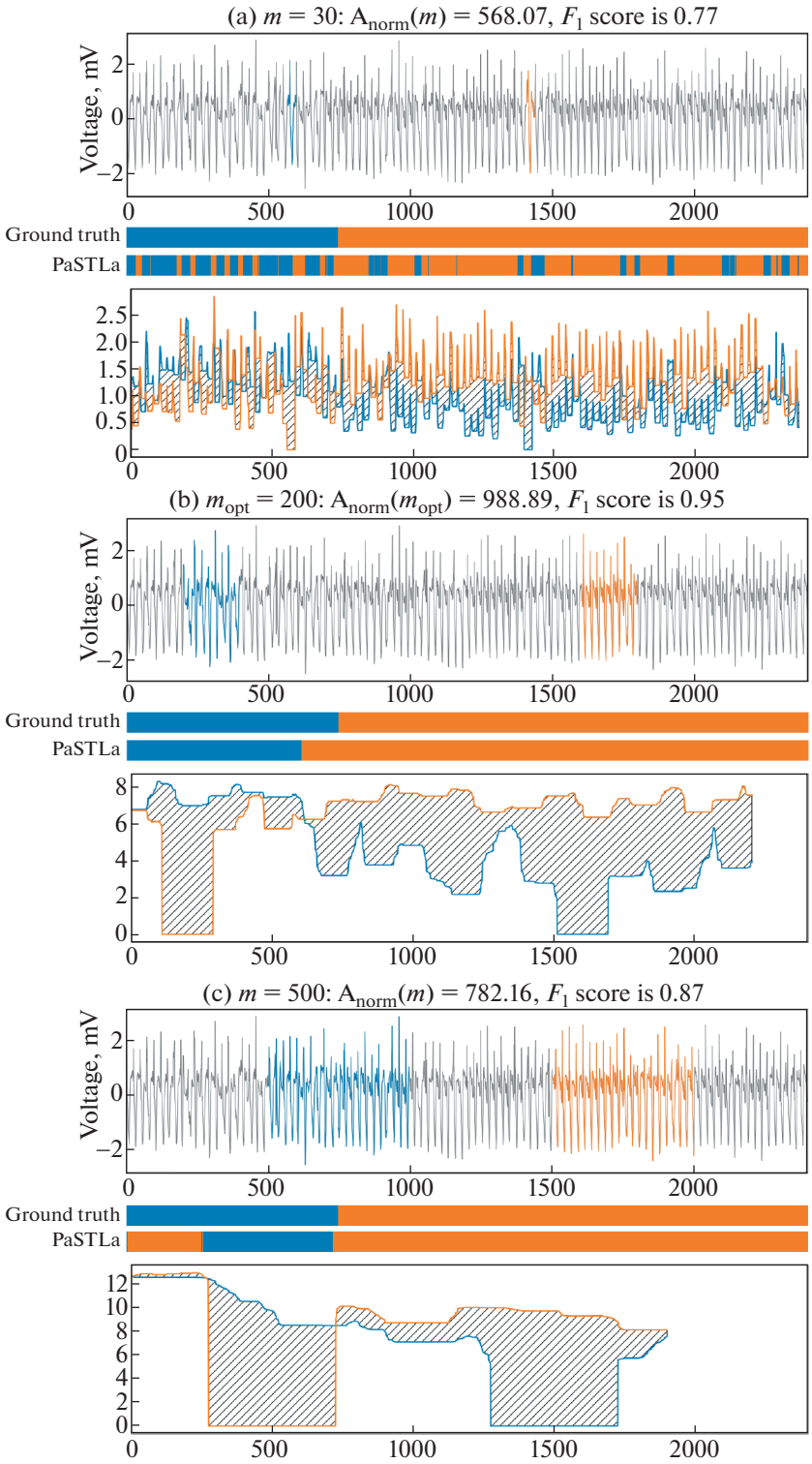


Fig. 2. An example of snippet selection (*top*: labeling, *bottom*: area between the MPdist-profile curves).

split point and selecting the one split that is best at identifying subsequences to be from either of the partitions. ClaSP learns its main two model parameters from the data using two bespoke algorithms.

The FLUSS (Fast Low-cost Unipotent Semantic Segmentation) [7] takes a subsequence length and the number of change points as parameters, based on the intuition that in the time series, subsequences in similar segments are more similar than subsequences that occur after a change point. FLUSS

produces a companion time series called the Arc Curve, which annotates the given time series with information about the likelihood of a regime change at each location. To do this, for the given time series, FLUSS calculates the matrix profile and matrix profile index [22]: the time series, where, respectively, ith element is the distance to and index of the nearest neighbor for ith subsequence. As a next step, a bespoke algorithm examines the Arc Curve obtained and decides how many regimes exist. However, the FLUSS's demand that the number of change points be known beforehand is hard to fulfill in practice.

The BinSeg (Binary Segmentation) algorithm [20] employs greedy sequential approach: first, one change point is detected in the complete input time series, then series is split around this change point, then the operation is repeated on the two resulting sub-series until a stopping criterion is met. The time complexity of BinSeg is $O(n \log n)$ w.r.t. the time series length. However, this low complexity comes at the expense of optimality: in general, BinSeg's output is only an approximation of the optimal solution. The problem is that the estimated change points are not estimated from homogeneous segments, and each estimate depends on the previous ones. Moreover, change points that are close are often imprecisely detected.

Concluding our review of related work, we can see that the snippet concept [11] is one of the most promising approaches to unsupervised time series summarization and labeling. Currently, the PSF (Parallel Snippet-Finder) algorithm [24] is the only parallel implementation of snippet discovery, to the best of our knowledge. However, PSF does not fit computer cluster architecture and can be improved by employing the automatic selection of some optimal snippet length from the specified range instead of the predefined one. Thus, in this article, we address the issues above.

2. PRELIMINARIES

Prior to introducing the proposed algorithm for unsupervised labeling of long time series, below, in Subsections 2.2.1 and 2.2.2, respectively, we first give basic notation and concepts our approach is based on (the matrix profile [22], MPdist distance measure [6], and snippet [11]) and then briefly describe our previous development, the PSF (Parallel Snippet-Finder) algorithm [24].

2.1. Notation and Basic Concepts

Time Series and Subsequence. A *time series* is a chronologically ordered sequence of real-valued numbers

$$T = \{t_i\}_{i=1}^{n}, \quad t_i \in \mathbb{R}.$$

The length of a time series, n, is denoted by $|T|$.

A *subsequence* $T_{i,\,m}$ of a time series T is its subset of m successive elements that starts at the ith position

$$T_{i,m} = \{t_k\}_{k=i}^{i+m-1}, \quad 1 \le i \le n-m+1, \quad 3 \le m \ll n.$$

Matrix Profile. For the given m-length query subsequence Q and time series T, a *distance profile* is a vector of the distances between Q and each subsequence of T

$$D_T^m(Q) = \{\mathrm{Dist}(Q, T_{i,\,m})\}_{i=1}^{n-m+1}, \tag{1}$$

where $\mathrm{Dist}(\cdot,\cdot)$ is a non-negative and symmetric function.

A subsequence $T_{i,\,m}$ is the *nearest neighbor* of an m-length query subsequence Q if

$$T_{i,m} = \arg\min D_T^m(Q).$$

For the given two n-length time series A and B, and the subsequence length m, a *matrix profile* P_{AB}^m is a vector of distances between each subsequence in A and its nearest neighbor in B

$$P_{AB}^m = \{\min D_B^m(A_{i,m})\}_{i=1}^{n-m+1}.$$

It is worth noting that, commonly, $P_{AB}^m \neq P_{BA}^m$. Aside snippets, there are diverse time series mining primitives that are based on the matrix profile concept: novelets [15], semantic motifs [10], chains [23], etc.

The MPdist Distance Measure. Let us have two m-length time series A and B, and the subsequence length ℓ (typically, it is taken from range $\lceil 0.3m \rceil \leq \ell \leq \lceil 0.8m \rceil$ [6])[1)]. Let us denote the concatenation of the matrix profiles P^{ℓ}_{AB} and P^{ℓ}_{BA} as P^{ℓ}_{ABBA}, $|P^{\ell}_{ABBA}| = 2(m - \ell + 1)$. Let $\text{SORTED}P^{\ell}_{ABBA}$ denotes the P^{ℓ}_{ABBA}'s version, where the elements are sorted in ascending order. Then, the MPdist distance between A and B is calculated as follows

$$\text{MPdist}_{\ell}(A, B) = \begin{cases} \text{SORTED}P^{\ell}_{ABBA}(k), & o|P^{\ell}_{ABBA}| > k \\ \text{SORTED}P^{\ell}_{ABBA}\big(2(m - \ell + 1)\big), & otherwise, \end{cases} \tag{2}$$

where $k = \lceil 0.1 \cdot m \rceil$. Here the parameter k is set to be equal to 5 percent of $2m$, which is the length of concatenation of A and B. If the subsequence length ℓ is close to the time series length m, then the maximum value of $\text{SORTED}P^{\ell}_{ABBA}$ is used as a result, since $|P^{\ell}_{ABBA}| < \lceil 0.1 \cdot m \rceil$ [6].

To calculate distance between subsequences, MPdist uses the Euclidean distance as the $\text{Dist}(\cdot, \cdot)$ function, which is defined as below

$$\text{ED}(X, Y) = \sqrt{\sum_{i=1}^{\ell} (x_i - y_i)^2}.$$

Despite the fact that MPdist does not obey the triangular inequality, it is robust to spikes, warping, linear trends, etc. [6].

Snippets. Hereinafter, without a loss of generality, we assume that the time series length n is a multiple of the subsequence length m (if n/m is not an integer, we pad the time series right to the end by zeros until the result of the division above becomes an integer). Next, let us represent T as a set of m-length *segments* (non-overlapped subsequences), and denote such a set as S^m_T

$$S^m_T = \{S_i | S_i = T_{m \cdot (i-1)+1, m}, \quad 1 \leq i \leq n/m\}.$$

A time series *snippet* is an actual segment of T. Let us denote a set of m-length snippets of T as C^m_T,

$$C^m_T = \{C_i | C_i \in S^m_T, \quad 1 \leq i \leq n/m\}, \tag{3}$$

where a time series snippet $C_i \in C^m_T$ is provided with the following attributes: an index, a set of nearest neighbors, and a fraction. We denote these attributes, respectively, as $C_i.index$, $C_i.NN$, and $C_i.frac$.

An *index* of a snippet $C_i \in C^m_T$ is a number j of a segment that corresponds to the snippet, i.e., $S_j = T_{m \cdot (j-1)+1, m}$.

Nearest neighbors of a snippet $C_i \in C^m_T$ is a set of subsequences that are the most similar to the corresponding segment w.r.t. the MPdist distance measure

$$C_i.NN = \{T_{j,m} | S_{C_i.index} = \arg \min_{1 \leq s \leq n/m} \text{MPdist}(T_{j,m}, S_s), \quad 1 \leq j \leq n - m + 1\}.$$

A *fraction* of a snippet $C_i \in C^m_T$ is a ratio of the number of the snippet's nearest neighbors to the total number of m-length subsequences in the time series

$$C_i.frac = \frac{|C_i.NN|}{n - m + 1}.$$

Snippets are ordered in descending order of their fraction

$$\forall C_i, C_j \in C^m_T : \quad i < j \Leftrightarrow C_i.frac \geq C_j.frac.$$

[1)] In this definition, we use m and ℓ instead of n and m to designate the length of the time series and subsequence, respectively, since below, MPdist is calculated between subsequences, not time series.

2.2. The Snippet-Finder Algorithm and its Parallelization

The Snippet-Finder algorithm [11] is given by a time series T, a segment length m, and an integer parameter K ($1 \le K \le n/m$), and should find top-K snippets $\{C_i\}_{i=1}^{K} \subset C_T^m$ by fraction, including all attributes of each snippet. Snippet-Finder performs as follows.

By analogy with the distance profile (1), Snippet-Finder employs the MPdist-profile, the vector $MPD \in \mathbb{R}^{n-m+1}$ of the MPdist-distances between the given m-length query subsequence Q and each subsequence of the time series T

$$MPD_T^m(Q) = \{\text{MPdist}(Q, T_{i,m})\}_{i=1}^{n-m+1}. \tag{4}$$

Let us denote a K-combination of S_T^m (i.e., a subset of K distinct elements of the set of m-length segments) and a set of all K-combinations as $\sigma_{K(S_T^m)}$ and $\mathcal{P}_{K(S_T^m)}$, respectively. Then, the function $ProfileArea$, defined as below, plays the role of the objective function in the snippet discovery

$$ProfileArea(\sigma_{K(S_T^m)}) = \sum_{i=1}^{n-m} \min_{S \in \sigma_{K(S_T^m)}} MPD_T^m(S)_i. \tag{5}$$

Such a function calculates an area under the curves of MPdist-profiles, in which query subsequences are the time series segments. Since in the case when all the segments are employed in the area under the curves, the area is exactly zero, i.e., $ProfileArea(\sigma_{n/m(S_T^m)}) \equiv 0$, Snippet-Finder selects a smaller number K of segments as snippets that approach $ProfileArea$ to zero

$$\{C_i\}_{i=1}^{K} = \arg \min_{\sigma_{K(S_T^m)} \in \mathcal{P}_{K(S_T^m)}} ProfileArea(\sigma_{K(S_T^m)}).$$

In the PSF (Parallel Snippet-Finder) algorithm [24], aiming at highest possible performance, while calculating MPdist between subsequences, instead of the Euclidean distance, we use the squared z-normalized Euclidean distance that is defined as follows

$$\text{ED}^2_{\text{norm}}(X, Y) = \text{ED}^2(\hat{X}, \hat{Y}), \quad \hat{x}_i = \frac{x_i - \mu_x}{\sigma_x}, \quad \mu_x = \frac{1}{\ell}\sum_{i=1}^{\ell} x_i, \quad \sigma_x = \sqrt{\frac{1}{\ell}\sum_{i=1}^{\ell} x_i^2 - \mu_x^2}.$$

In contrast with the original algorithm, where the calculation of an MPdist-profile for a segment and each subsequence of the time series is one serial step, in PSF [24], it is performed in several steps, where each step is parallelized with GPU. At the first step, for each segment $S \in S_T^m$, we calculate $ED_{\text{matr}} \in \mathbb{R}^{(m-\ell+1)\times(n-\ell+1)}$, the distance matrix of all ℓ-length subsequences of S and T

$$ED_{\text{matr}}(i, j) = \text{ED}^2_{\text{norm}}(S_{i,\ell}, T_{j,\ell}).$$

At the second step, we calculate the vector $allP_{BA} \in \mathbb{R}^{n-\ell+1}$ that stores column-wise minima of the matrix ED_{matr}

$$allP_{BA}(j) = \min_{1 \le i \le m-\ell+1} ED_{\text{matr}}(i,\, j).$$

At the third step, we calculate the matrix $allP_{AB} \in \mathbb{R}^{(m-\ell+1)\times(n-\ell+1)}$ that stores row-wise minima in the ℓ-length sliding windows of the matrix ED_{matr}

$$allP_{AB}(i, j) = \min_{j \le c \le j+m-\ell+1} ED_{\text{matr}}(i,\, c).$$

Next, for each segment, we obtain the matrix profile by concatenating each column of the $allP_{AB}$ matrix and all $(m-\ell)$-length subsequences from the vector $allP_{BA}$ and denoting the resulting structure as $P_{ABBA} \in \mathbb{R}^{2(m-\ell+1)}$

$$P_{ABBA}(T_{j,\ell}) = \{allP_{AB}(i, j)\}_{i=1}^{m-\ell+1} \{allP_{BA}(i)\}_{i=j}^{m-\ell+1}.$$

Finally, through the obtained matrix profiles, we calculate the MPdist distances between each segment and each m-length subsequence according to (2), then calculate the area under the curves (5) and discover snippets (3).

3. METHOD

Below, we introduce PaSTiLa (Parallel Snippet-based Time series Labeling), novel parallel algorithm for labeling long time series on GPU-based high-performance (HPC) clusters. In Subsection 3.3.1, we describe the parallelization scheme, whereas in Subsections 3.3.2 and 3.3.3, we discuss postprocessing techniques that improve the quality of labeling through automatic choice of the segment length and merging the similar activities, respectively.

3.1. Parallelization Scheme

In what follows, we assume that a high-performance cluster for the task at hand is homogeneous and consists of P nodes ($P > 1$) with the same number of graphics processors onboard for each node. Let us given the n-length time series T and the snippet length that ranges from $minL$ to $maxL$, where $minL < maxL \ll n$. Our approach supposes that T is replicated for each of the P cluster nodes.

Figure 1 depicts the parallelization scheme of PaSTiLa. One of the cluster nodes is claimed as a master to perform preprocessing and postprocessing, whereas the rest of the nodes perform parallel calculations. At the first step of preprocessing, for each segment length in the specified range, *Predictor* produces the estimated running time of the snippet discovery on a single cluster node. Next, based on the *Predictor*'s results, *Scheduler* completes preprocessing, creating for each cluster node a batch job that consists of one to $\lceil R/P \rceil$ ($R = maxL - minL + 1$) segment lengths to process, where all the jobs eventually provide the HPC cluster with a balanced load.

Predictor is implemented through the polynomial regression based on the fact that the serial Snippet-Finder algorithm has cubic time complexity w.r.t. the time series length [11]. To provide *Predictor* with the training data, we run the PSF algorithm [24] on a single cluster node over synthetic data generated through the Random walk model [17]. In such experiments, we varied the time series length from 10^3 to 10^6 with step 10^2 while changing the segment length from 5% to 25% of the time series with step 1%.

Scheduler treats the task of creating batch jobs for the cluster nodes as the multiway number partitioning problem [9], where a multiset of numbers is to be partitioned into a fixed number of subsets, such that the sums of the subsets are as similar as possible. Although the problem is NP-hard [9], there are various algorithms that solve it efficiently in many cases. To implement *Scheduler*, in our study, we employ the Karmarkar–Karp algorithm (or, the largest differenting method) [12] that has the time complexity of $O(R \log R)$.

During the calculation phase, GPUs at each node discover snippets of the lengths assigned through the PSF algorithm independently of those at the other nodes. After completing the calculations, each node sends the results (the snippets found and their MPdist-profiles) to the master node, which further performs postprocessing. Data exchanges across cluster nodes are implemented through MPI (Message Passing Interface) [19].

Postprocessing is performed in two steps, where, respectively, *Selector* determines the optimal snippet length in the specified range according to our proposed heuristic criterion (see Subsection 3.3.2) and *Merger*, taking the optimal snippet length found, improves the labeling quality through combining activities represented by snippets that are the most similar w.r.t. MPdist (see Subsection 3.3.3).

3.2. Unsupervised Selection of the Snippet Length

In this study, we propose a heuristic criterion to automatically choose the segment length in the specified range. The criterion employs all MPdist-profiles (4), where a query subsequence is one of the snippets found that are part of the input for postprocessing. According to the criterion, we should determine the segment length at which the area between the curves of such MPdist-profiles will be maximum. Let us denote the optimal segment length as m_{opt}, then it can be found as below

$$m_{\text{opt}} = \arg \max_{minL \le m \le maxL} A_{\text{norm}}(m), \quad A_{\text{norm}}(m) = \frac{A(m)}{A_{\text{max}}(m)}, \tag{6}$$

$$A(m) = \sum_{\substack{1 \le p \ne q \le K \\ C_p, C_q \in C_T^m}} \sum_{i=1}^{n-m+1} |MPD_T^m(C_p)_i - MPD_T^m(C_q)_i|, \quad A_{\text{max}}(m) = \max_{\substack{1 \le i \le n-m+1 \\ C_p \in C_T^m}} MPD_T^m(C_p)_i.$$

Table 1. Empirical evaluation of the criterion (6) over the ECG time series (an excerpt of numerical data)

Snippet length, m	Area between MPdist profiles, $A_{\text{norm}}(m)$	Labeling quality, F_1 score
30	568.09	0.77
100	968.65	0.86
150	897.09	0.92
200	**988.89**	**0.95**
250	942.46	0.89
350	895.64	0.90
500	782.16	0.87

Indeed, formula (6) involves the set of MPdist-profiles $\{MPD_T^m(C_i)\}_{i=1}^K$, where query subsequences are snippets $\{C_i\}_{i=1}^K \subset C_T^m$. In the numerator, the area between the above curves is calculated as the sum of the absolute differences between their correspondent points, which is taken for every pair of snippets found. The denominator plays the role of the normalizing factor since the area above decreases when the segment length increases.

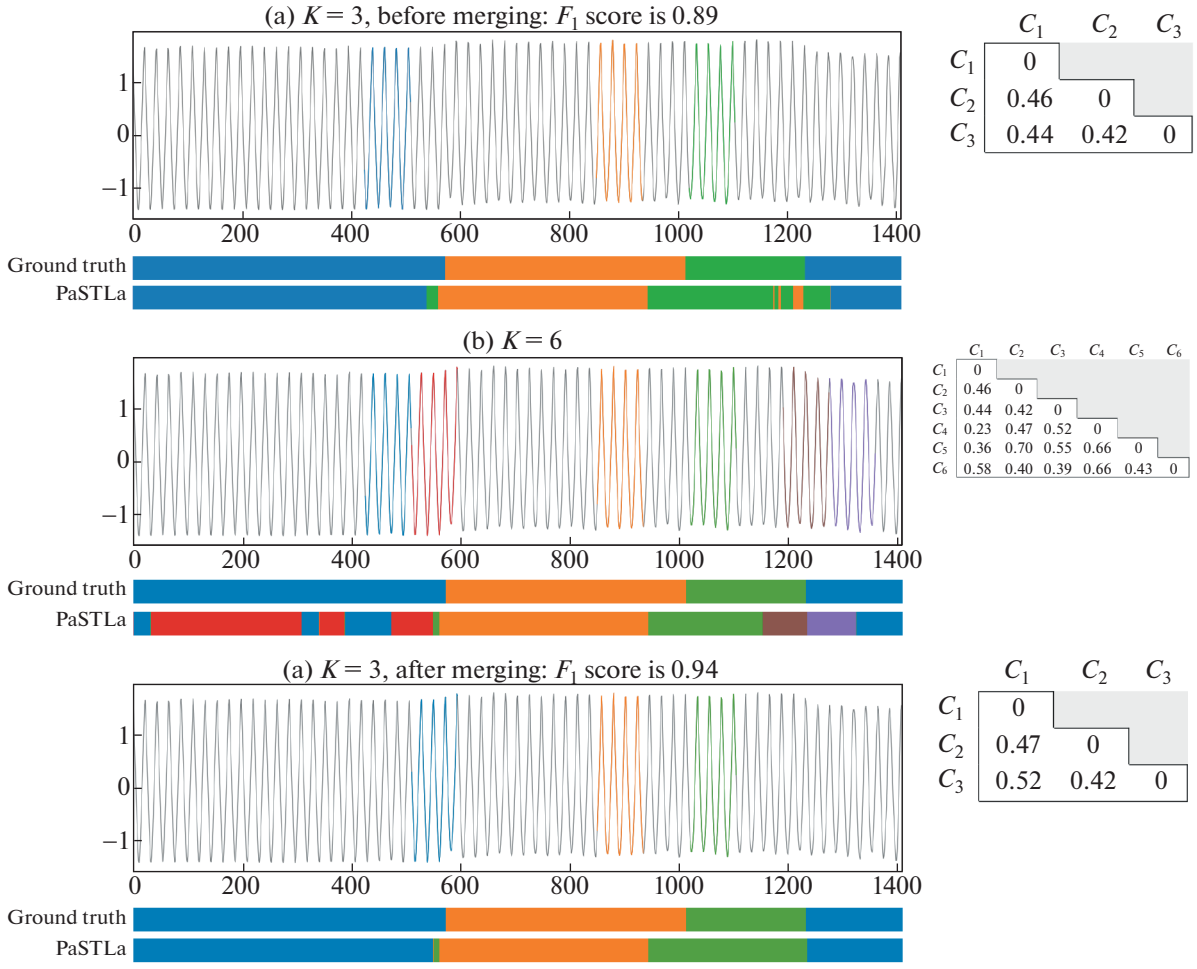
The criterion is based on our observation that the more the snippets found are dissimilar from each other, the larger the area between the above-mentioned curves (despite such a property being performed frequently, we are not claiming that this follows a strict pattern). Our observation was evaluated over the TSSB (Time Series Segmentation Benchmark) dataset [4], which is a collection of 75 different-length time series from diverse domains, where each time series represents some subject's activities, where the number of activities varies in the range 2..7, and the subject changes activity from one to another up to 6 times. Table 1 depicts an excerpt of the evaluation over the ECG time series included in TSSB, where snippets represent two classes: a normal heartbeat and a myocardial infarction.

In Fig. 2, we visualize three cases given in Table 1: for $m = 30$, $m = 200$, and $m = 500$ (see Figs. 2a, 2b, and 2c, respectively). As can be seen, the value $m_{\text{opt}} = 200$ provides the biggest values of A_{norm} and F_1 score.

3.3. Merging Similar Snippets

To improve the quality of labeling, we propose the following technique. At first, having obtained the snippet length from *Selector*, instead of the given number of snippets to be found, K, through PSF, we discover many times more snippets, dK, where integer d is a user-defined parameter ($2 \leq d \leq \lceil n/K \cdot m_{\text{opt}} \rceil$). Next, we calculate MPdist distance for every pair of snippets found. Finally, we merge every two closest snippets until it results in exactly K snippets. For the pair of merged snippets, we choose the resulting snippet as one with the greatest fraction. The nearest neighbors of two merged snippets are combined and obtain the same label.

In Fig. 3, we illustrate the proposed technique over a time series from the ADIAC dataset [2, 3]. In Fig. 3a, we depict the time series labeling produced by our algorithm, where the number of snippets, $K = 3$, and the matrix of MPdist-distances between the snippets were found; initially, labeling quality w.r.t. the F_1 score is 0.89. Figure 3b shows the labeling and distance matrix for the greater number of snippets, $K = 2 \cdot 3 = 6$. As can be seen from the distance matrix, the pairs of the closest snippets, sorted by the MPdist distance between them in ascending order, are C_1 and C_4, C_1 and C_5, and C_3 and C_6. In each pair above, the snippets are to be merged into the former snippet in the pair since it has a bigger fraction. Finally, Fig. 3c depicts the labeling and distance matrix after the merging. As can be seen, the labeling quality has improved to 0.94 w.r.t. the F_1 score, and final snippets are more distinct from each other than at the initial stage.



	C_1	C_2	C_3
C_1	0		
C_2	0.46	0	
C_3	0.44	0.42	0

	C_1	C_2	C_3	C_4	C_5	C_6
C_1	0					
C_2	0.46	0				
C_3	0.44	0.42	0			
C_4	0.23	0.47	0.52	0		
C_5	0.36	0.70	0.55	0.66	0	
C_6	0.58	0.40	0.39	0.66	0.43	0

	C_1	C_2	C_3
C_1	0		
C_2	0.47	0	
C_3	0.52	0.42	0

Fig. 3. An example of snippet merging (*left*: labeling, *right*: MPdist distance matrix for snippets).

4. EXPERIMENTAL EVALUATION

To evaluate the proposed algorithm, we carried out experiments over various real-world time series. We designed the experiments to be easily reproducible with our repository [8], which contains the algorithm's source code and all the datasets used in this work. Below, we describe the experimental setup and discuss the results of the experiments, in Subsections 4.4.1 and 4.4.2, respectively.

4.1. Experimental Setup

Goals. In the experiments, we evaluated the quality and performance of labeling PaSTiLa provides, in comparison with the rival algorithms mentioned above: FLUSS [7, 14], ClaSP [4, 5], and BinSeg [20, 21]. In addition, we assessed the algorithm's speedup, i.e., its ability to decrease the running time when the same-length time series is processed on the increasing number of GPUs.

Measures. To measure the quality of labeling, we employ the commonly used F_1 score, which is defined as the harmonic mean of precision and recall

$$\mathrm{F}_1 = 2\frac{\mathrm{Precision}\mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}, \quad \mathrm{Precision} = \frac{TP}{TP + FP}, \quad \mathrm{Recall} = \frac{TP}{TP + FN},$$

where TP, FP, TN, and FN denote, respectively, the number of true positive, false positive, true negative, and false negative points w.r.t. the pre-labeled time series. For the segmentation-based algorithms, we calculate the F_1 score over the predicted change points in comparison with the ground truth ones w.r.t. margin of 1% of the time series length [4].

Table 2. Hardware platform of the experiments

Feature	GPU	CPU
Brand and product line	NVIDIA Kepler	Intel Sandy Bridge
Model	K20X	E5-2660
Number of cores	2688	8
Core frequency, GHz	0.732	2.2
RAM, Gb	6	64
Peak performance (double precision), TFLOPS	1.31	0.282

The algorithm's performance is interpreted as its running time. For each experiment, we ran PaSTiLa ten times and took the average value as the final running time. The speedup of our algorithm employing k graphics processors is calculated as $S(k) = t_1/t_k$, where t_1 and t_k are the running times of PSF [24] (i.e., predecessor of PaSTiLa) on one GPU and PaSTiLa on k GPUs, respectively.

Datasets. To evaluate the quality and performance of labeling, we employ the above-mentioned TSSB dataset [4] (see Subsection 3.3.2), which was also used by the authors of the above rival algorithms to evaluate the segmentation quality. To assess our algorithm's speedup, we use the Solar Power time series [18], which contains more than 7.39 million points representing the daily solar power production in Australia recorded per every 4 seconds starting from August 1, 2019.

Hardware. We carry out our evaluation on up to 64 nodes of the Lobachevsky supercomputer (University of Nizhny Novgorod, Russia) [1], where the node characteristics are summarized in Table 2. Our algorithm runs on GPU(s) of the node(s), whereas each rival algorithm being serial runs on one core of CPU.

4.2. Results and Discussion

Labeling quality. In Fig. 4, for all competitors, we show the quality of labeling as F_1 score boxplots over all time series from the TSSB dataset. As can be seen, PaSTiLa outperforms the segmentation-based competitors BinSeg, FLUSS, and ClaSP. In addition, we assess if our proposed snippet selection scheme improves the labeling quality. As can be seen, for the segment length liberately chosen as 5% or 25% of the time series length, PSF, the parallel version of the original snippet discovery algorithm, is inferior to PaSTiLa, where the segment length is calculated according to formula (6).

In addition, in Fig. 5, we depict the labeling of the Solar Power time series produced by PaSTiLa and the best segmentation-based competitors. Despite the fact that this time series does not have ground truth labeling, it can be seen that PaSTiLa is more accurate than rival algorithms in determining the day-night cycles: in fact, two snippets found represent increasing and decreasing solar power production, respectively.

Performance and speedup. Figure 6 depicts the experimental results on performance of labeling over the TSSB dataset for all competitors. To illustrate the insights found, we split TSSB into two groups w.r.t. the length of time series: short ($2000 < n \leq 8000$), and long ($8000 < n \leq 27000$), where 56 and 19 time series are included in the groups, respectively. For each group, we show the total running time of labeling all the time series therein on 2, 4, 8, 16, 32, and 64 single-GPU nodes, respectively. It can be seen that over small-length time series (see Fig. 6a), BinSeg outruns both serial FLUSS and ClaSP, and parallel PaSTiLa when it is running on four or fewer nodes. The reason is that our algorithm does not fit small-length time series since for such data, the running time spent for exchanges between cluster nodes becomes comparable with the rest calculations. In case of long-length time series (see Fig. 6b), PaSTiLa outruns the above rivals when it is running on eight or more nodes.

In Fig. 7, we show the performance and speedup of our algorithm over the Solar Power time series. PaSTiLa, as expected, significantly outruns its serial rivals over the millions-length time series (see Fig. 7a). PaSTiLa demonstrates close-to-linear speedup, which slightly decreases starting from 32 GPUs our algorithm is running on (see Fig. 7b). The reason is the algorithm's overhead on one-time

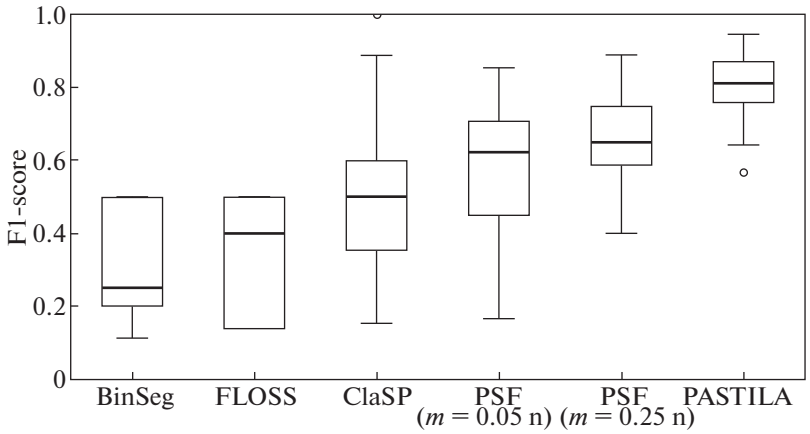


Fig. 4. Labeling quality over the TSSB dataset.

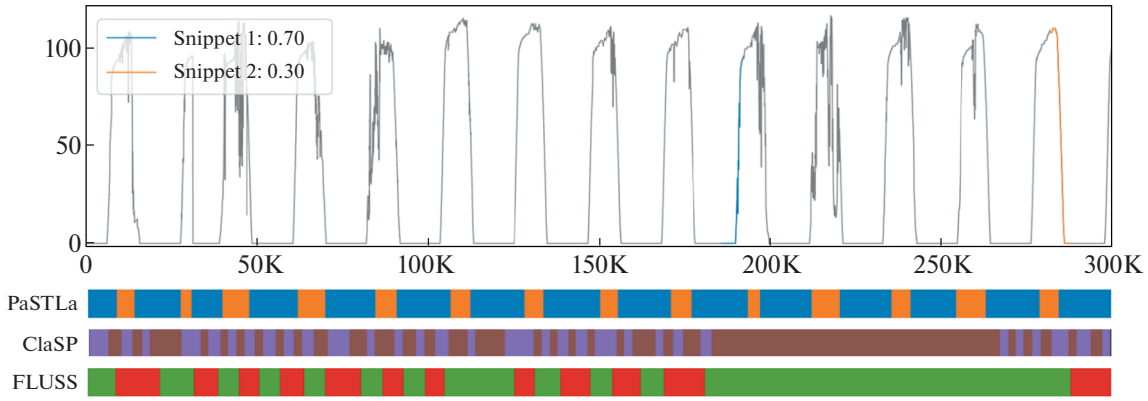


Fig. 5. Labeling the Solar Power time series.

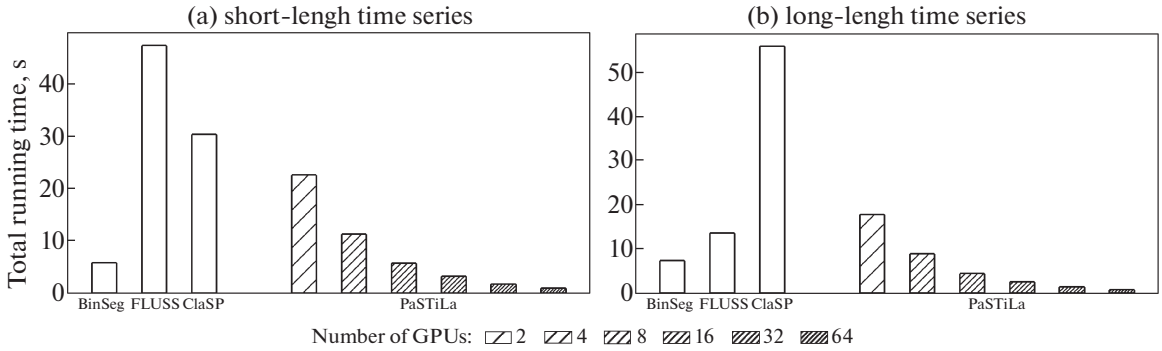


Fig. 6. Performance of PaSTiLa over the TSSB dataset.

data exchange, the impact of which increases starting from the aforementioned number of GPUs. Our algorithm's performance and speedup are weakly proportional to the number of GPUs installed on a cluster node.

CONCLUSIONS

In this article, we addressed the problems of unsupervised summarization and labeling of long time series arised in a wide spectrum of subject domains: Internet of Things, digital industry, personal healthcare, climate modeling and prediction of natural disasters, etc. Summarization aims to discover

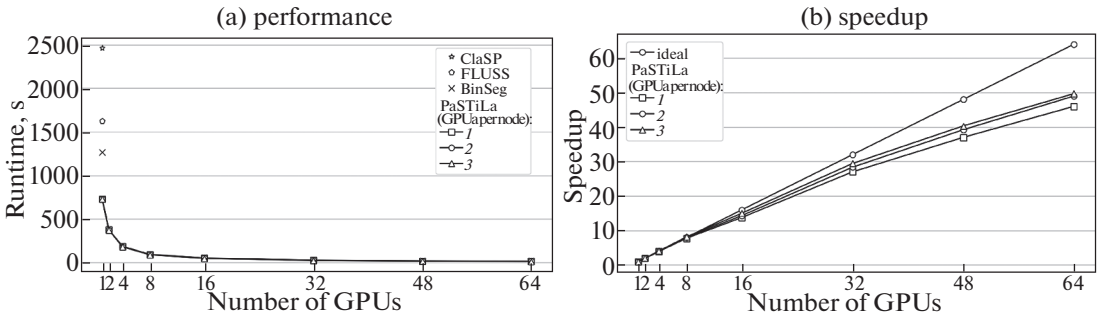


Fig. 7. Performance and speedup of PaSTiLa over the Solar Power time series.png.

a small set of patterns (typical subsequences) that provide a concise representation the given long time series. Further, labeling of the time series can be implemented by assigning each subsequence a tag that corresponds to its most similar pattern.

Our study is based on the snippet concept [11], where a snippet is the given-length subsequence, which is similar to many other subsequences of the given time series w.r.t. the normalized Euclidean distance-based measure MPdist [6]. Our previously developed PSF (Parallel Snippet-Finder) algorithm [24] accelerates the original snippet discovery schema on GPU. However, in PSF, like in its predecessor, the snippet length should be predefined by a domain expert. In this article, we introduce the novel parallel algorithm PaSTiLa (Parallel Snippet-based Time series Labeling) that performs the snippet discovery and labeling of the given time series on an HPC cluster with GPU nodes, employing the automatic selection of the snippet length from the specified range through our proposed heuristic criterion.

PaSTiLa performs according to the scenario that, for the task at hand, an HPC cluster is homogeneous with the same number of GPUs onboard for each node, and the given time series is replicated for each of the cluster nodes. Then, one of the cluster nodes is claimed as a master to perform preprocessing and postprocessing, whereas the rest of the nodes perform parallel calculations. At the first step of preprocessing, for each segment length in the specified range, *Predictor* produces the estimated running time of the snippet discovery on a single cluster node. *Predictor* is implemented through the polynomial regression, where the training data are obtained through runs of PSF on a single cluster node over synthetic time series with varied the lengths of both the time series and segment. Next, based on the *Predictor*'s results, *Scheduler* employs the Karmarkar–Karp algorithm and, for each cluster node, creates a batch job consisting of segment lengths to process, where all the jobs eventually provide the HPC cluster with a balanced load. Further, at each cluster node, for each length in the node's job, GPUs discover snippets through the PSF algorithm independently of the other nodes. The resulting snippets and their MPdist-profiles are sent through MPI to the master node for postprocessing. At the first step of postprocessing, *Selector* determines the segment length at which the area between the curves of such MPdist-profiles will be maximum. Next, the segment length found through such a criterion is given to *Merger* that discovers many times more snippets than the given input number of the subject's activities and then combines every two closest snippets w.r.t. the MPdist distance and their labels until it results in exactly the given number of snippets.

We carried out extensive experiments to evaluate our algorithm against state-of-the-art segmentation-based analogs: FLUSS [7], ClaSP [4], and BinSeg [20]. In the experiments on labeling quality over the TSSB dataset [4], which consists of 75 time series from diverse domains with varying both numbers of activities and their changes from one to another, PaSTiLa outperforms competitors in average F_1 score. As for labeling performance, over small-length time series (typically less than 8–10 K points), PaSTiLa is inferior to serial competitors since for such data, the running time spent for exchanges between cluster nodes becomes comparable with the rest calculations. In the case of long-length time series, PaSTiLa outruns the rivals. In addition, we assessed our algorithm's scalability over the Solar Power time series [18], which contains more than 7 million points, where PaSTiLa demonstrates a close-to-linear speedup compared to the PSF running time on a single cluster node and is more accurate than rival algorithms in determining the day-night cycles.

Finally, we establish a repository [8] that contains the algorithm's source code and datasets involved in the experiments to facilitate the reproducibility of our study.

Our further research might elaborate on the application of PaSTiLa in the development of deep learning model(s) for online human activity recognition through analysis of time series from wearable devices.

ACKNOWLEDGMENTS

This work was financially supported by the Russian Science Foundation (grant no. 23-21-00465).

REFERENCES

1. Lobachevsky Supercomputer. https://hpc-education.unn.ru/en/resources. Accessed Jan 12, 2023.
2. J. M. H. du Buf, H. R. Shahbazkia, A. Ciobanu, M. Bayer, S. Droop, R. Head, S. Juggins, S. Fischer, H. Bunke, M. H. F. Wilkinson, J. B. T. M. Roerdink, J. L. Pech-Pacheco, and G. Cristóbal, "Diatom identification: A double challenge called ADIAC," in *Proceedings of the 10th International Conference on Image Analysis and Processing ICIAP 1999, September 27–29, 1999, Venice, Italy* (IEEE Comput. Soc., 1999), pp. 734–739. https://doi.org/10.1109/ICIAP.1999.797682
3. H. A. Dau, E. Keogh, K. Kamgar, C. C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, H. B. Yanping, N. Begum, A. Bagnall, A. Mueen, G. Batista, and Hexagon-ML Collab., The UCR Time Series Classification Archive. https://www.cs.ucr.edu/ eamonn/time_series_data_2018/. Accessed Jan 12, 2023.
4. A. Ermshaus, P. Schäfer, and U. Leser, "ClaSP: Parameter-free time series segmentation," Data Min. Knowledge Discov. **37**, 1262–1300 (2023). https://doi.org/10.1007/S10618-023-00923-X
5. A. Ermshaus, P. Schäfer, and U. Leser, ClaSPy: A Python Package for Time Series Segmentation. https://github.com/ermshaua/claspy/. Accessed Jan 12, 2023.
6. S. Gharghabi, S. Imani, A. J. Bagnall, A. Darvishzadeh, and E. J. Keogh, "An ultra-fast time series distance measure to allow data mining in more complex real-world deployments," Data Mining Knowledge Discov. **34**, 1104–1135 (2020). https://doi.org/10.1007/s10618-020-00695-8
7. S. Gharghabi, C. M. Yeh, Y. Ding, W. Ding, P. Hibbing, S. LaMunion, A. Kaplan, S. E. Crouter, and E. J. Keogh, "Domain agnostic online semantic segmentation for multi-dimensional time series," Data Mining Knowledge Discov. **33**, 96–130 (2019). https://doi.org/10.1007/S10618-018-0589-3
8. A. Goglachev and M. Zymbler, PaSTiLa: Parallel algorithm for unsupervised labeling of long time series on multi-GPU clusters. https://github.com/goglachevai/PaSTiLa. Accessed Jan 12, 2023.
9. R. L. Graham, "Bounds on multiprocessing timing anomalies," SIAM J. Appl. Math. **17**, 416–429 (1969). https://doi.org/10.1137/0117039
10. S. Imani and E. J. Keogh, "Matrix profile XIX: Time series semantic motifs: A new primitive for finding higher-level structure in time series," in *Proceedings of the 2019 IEEE International Conference on Data Mining, ICDM 2019, Beijing, China, November 8–11, 2019* (IEEE, 2019), pp. 329–338. https://doi.org/10.1109/ICDM.2019.00043
11. S. Imani, F. Madrid, W. Ding, S. E. Crouter, and E. J. Keogh, "Introducing time series snippets: A new primitive for summarizing long time series," Data Mining Knowledge Discov. **34**, 1713–1743 (2020). https://doi.org/10.1007/s10618-020-00702-y
12. N. Karmarkar and R. M. Karp, "The differencing method of set partitioning," Technical Report No. UCB/CSD-83-113 (EECS Dep. Univ. California, Berkeley, CA, 1983).
13. L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis* (Wiley, Chichester, 1990). https://doi.org/10.1002/9780470316801
14. S. M. Law, "STUMPY: A powerful and scalable Python library for time series data mining," J. Open Source Software **4** (39), 504 (2019). https://doi.org/10.21105/joss.01504
15. R. Mercer and E. J. Keogh, "Matrix profile XXV: Introducing novelets: A primitive that allows online detection of emerging behaviors in time series," in *Proceedings of the IEEE International Conference on Data Mining, ICDM 2022, Orlando, FL, Nov. 28–Dec. 1, 2022* (IEEE, 2022), pp. 338–347. https://doi.org/10.1109/ICDM54844.2022.00044
16. A. Mueen, E. J. Keogh, Q. Zhu, S. Cash, and M. B. Westover, "Exact discovery of time series motifs," in *Proceedings of the SIAM International Conference on Data Mining, SDM 2009, April 30–May 2, 2009, Sparks, NV* (SIAM, 2009), pp. 473–484. https://doi.org/10.1137/1.9781611972795.41
17. K. Pearson, "The problem of the random walk," Nature (London, U.K.) **72**, 342 (1905). https://doi.org/10.1038/072342a0
18. G. Rakshitha, C. Bergmeir, G. Webb, M. Abolghasemi, R. Hyndman, and P. Montero-Manso, Solar Power Dataset (4 Seconds Observations). https://doi.org/10.5281/zenodo.4656027. Accessed 2020.

19. M. Snir, "Technical perspective: The future of MPI," Commun. ACM **61** (10), 105 (2018). https://doi.org/10.1145/3264415
20. C. Truong, L. Oudre, and N. Vayatis, "Selective review of offline change point detection methods," Signal Process **167**, 107299 (2020). https://doi.org/10.1016/J.SIGPRO.2019.107299
21. C. Truong, L. Oudre, and N. Vayatis, ruptures: A Python Library for Off-line Change Point Detection. https://github.com/deepcharles/ruptures/. Accessed Jan 12, 2023.
22. C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H. A. Dau, D. F. Silva, A. Mueen, and E. J. Keogh, "Matrix profile I: all pairs similarity joins for time series: A unifying view that includes motifs, discords and shapelets," in *Proceedings of the IEEE 16th International Conference on Data Mining ICDM 2016, December 12–15, 2016, Barcelona, Spain* (IEEE, 2016), pp. 1317–1322. https://doi.org/10.1109/ICDM.2016.0179
23. Y. Zhu, M. Imamura, D. Nikovski, and E. J. Keogh, "Introducing time series chains: A new primitive for time series data mining," Knowledge Inf. Syst. **60**, 1135–1161 (2019). https://doi.org/10.1007/S10115-018-1224-8
24. M. Zymbler and A. Goglachev, "Fast summarization of long time series with graphics processor," Mathematics **10**, 1781 (2022). https://doi.org/10.3390/math10101781
25. M. Zymbler and Y. Kraeva, "Discovery of time series motifs on Intel many-core systems," Lobachevskii J. Math. **40**, 2124–2132 (2019). https://doi.org/10.1134/S199508021912014X
26. M. Zymbler and Y. Kraeva, "Parallel algorithm for time series motif discovery on graphic processor," Vestn. YuUrGU, Ser.: Vychisl. Mat. Inform. **9** (3), 17–34 (2020). https://doi.org/10.14529/cmse200302