Supplement for "Scalable Order-Preserving Pattern Mining"

Anonymous

I. Omitted Proofs

A. Proof of Lemma 5

Proof. We first show that, for any string w over a totally ordered alphabet of size σ, there exist only up to $2\sigma + 1$ possibilities for $\mathsf{LastCode}(w \cdot a)$ over all letters a.

Let us start with the trivial case, where w is the empty string and so $\mathsf{LastCode}(w \cdot a) = \mathsf{LastCode}(a) = (\bot, \bot)$, for any a. In this case, the above upper bound clearly holds.

For the rest of the proof, we assume that w is nonempty. We have the following two cases:

1) If letter a belongs to the alphabet of w, then $\mathsf{LastCode}(w \cdot a)$ is equal to (x_a, x_a), where x_a is the *single position* of the rightmost occurrence of a in w. These are σ possibilities: one for every a.
2) Otherwise let the letters of w in the sorted order be equal to a_j for $j \in [0, \sigma)$, and let x_j be the position of the rightmost occurrence of a_j in w. There exists a single value $j \in [-1, \sigma)$ such that $a_j < a < a_{j+1}$, where $a_{-1} = -\infty$ and $a_\sigma = \infty$. Then $\mathsf{LastCode}(w \cdot a)$=$(x_j, x_{j+1})$, where $x_{-1} = x_\sigma = \bot$. Note that, in particular, only elements x_j can occur in $\mathsf{LastCode}$. These are $\sigma + 1$ possibilities: $(\bot, x_0), (x_0, x_1), (x_1, x_2), \ldots, (x_{\sigma-1}, \bot)$.

The upper bound of $2\sigma + 1$ follows by combining cases 1 and 2 above. By the above upper bound, any branching node of OPST cannot have more than $2\sigma + 1$ outgoing edges. □

B. Proof of Lemma 6

Proof. We first prove that an explicit non-branching node in $\mathsf{OPST}(w)$ can appear on a root-to-leaf path with witness suffix w' only right before the first occurrence of some letter.

Let xb be a witness substring of a node directly below an explicit non-branching node u, where x is a string and b is a letter. For the explicit non-branching node to be constructed there must exist a branching node v such that $\mathsf{SufLink}(v) = u$. Let the witnesses of any two children of v be equal to $a'x'b'$ and $a''x''b''$, respectively, where x', x'' are strings and a', a'', b', b'' are letters. We have:

- $a'x' \approx a''x''$, since both those witnesses are represented by the same node v.
- $a'x'b' \not\approx a''x''b''$, since the two children of v are distinct nodes – recall that node v is branching.
- $x'b' \approx xb \approx x''b''$, since Locus of both $x'b'$ and $x''b''$ (witnesses of children of v with their first letter deleted) must be equal to the single child of u – recall that node u is explicit but non-branching so it has only one child.

Notice that for two equal-length strings $w_1 \not\approx w_2$, there must exist two positions i, j such that the relationship ($<$, $=$, $>$) between $w_1[i]$ and $w_1[j]$ is different than the relationship between $w_2[i]$ and $w_2[j]$. Hence the relationship between a' and b' has to be different than the relationship between a'' and b'' as the relationship of all the other pairs of the letters of $a'x'b'$ and $a''x''b''$ is fixed by the two listed equivalences. This proves that b' cannot appear in x' as if that was the case the relationship between a' and b' (as a letter of x') would be fixed and the same as the relationship between a'' and b'' – a contradiction. Thus b does not appear in x since $xb \approx x'b'$.

Now, since the explicit non-branching node on a root-to-leaf path can only appear just above the first occurrence of a letter, there are only at most σ such nodes. Clearly, this also implies that there can only be at most σ explicit non-branching nodes between any two branching nodes. □

C. Proof of Lemma 7

Proof. We mimic the proof of the complexity of McCreight's algorithm [1]. Therein, the bound on the number $k(w)$ of moves was proved based on the *node depth* of the nodes u and v: when going from u to its branching parent u', the node depth decreases by 1, and then upon using the suffix link of u' it can again decrease by at most 1. Thus we can do at most $2n$ moves upward, and since the node depth is bounded by n (the maximal node depth in the tree), at most $3n$ moves downward.

In the OP setting, by Lemma 6 and the above analysis of McCreight's algorithm, the node depth can decrease by at most $\sigma + 2$ when going upwards (which also bounds the possible number of moves) due to the explicit non-branching nodes that do not correspond to any explicit or branching node on the path from root to $v = \mathsf{SufLink}(u)$. This means that we can make at most $n(\sigma+2)$ moves upward and at most $n(\sigma+2)+n$ moves downward; that is, $k(w) \leq n(2\sigma + 5) = \mathcal{O}(n\sigma)$. □

II. Alternative OPST Construction Algorithm

The OPST construction algorithm presented in the main paper runs in $\mathcal{O}(n\sigma \log \sigma)$ time using $\mathcal{O}(n)$ space. In theory, this bound is not satisfactory when σ is large; e.g., $\sigma = \Theta(n)$. We next present an $\mathcal{O}(n(\log \sigma + \log^2 \log n))$-time and $\mathcal{O}(n)$-space construction algorithm. As elaborated in the Introduction of the main paper, this algorithm is of theoretical interest.

Let us start by introducing *dynamic weighted ancestor* (DWA) queries on node-weighted trees and providing a lemma explaining how they can be performed efficiently in theory [2].

DWA queries are essential for an alternative, more efficient, computation of OPST suffix links.

Lemma 1 (DWA queries [2])**.** *Let N be an integer such that $w = \Omega(\log N)$, where w is the machine word-size. There exists a data structure which maintains a node-weighted tree T with weights in $[N]$ in $\mathcal{O}(|T|)$ space and supports the following operations in $\mathcal{O}(\log^2 \log N)$ time:*

- *WeightedAncestor(v, g): return the highest ancestor of v with weight at least g;*
- *Insert a leaf with weight g and v as a parent;*
- *Insert a node with weight g by subdividing the edge joining v with its parent.*

The weight of any node in T must be greater than any of its ancestors; and the root has weight 0*.*

We employ DWA queries to construct suffix links during the construction of OPST. Intuitively, DWA queries let us retrieve any implicit or explicit ancestor of a node of OPST and thus we can use them to arrive at an appropriate string depth efficiently. We apply Lemma 1 using *string depth* as node weights, so $N := n$. Algorithm ComputeSuffixLink(u) below replaces ComputeSuffixLinkSimple(u) in the main paper.

Algorithm 1 ComputeSuffixLink(u)

Require: The DWA data structure given by Lemma 1.

1: $d \leftarrow$ Depth(u)
2: $\ell \leftarrow$ GetLeaf(Witness$(u) + 1$)
3: $v \leftarrow$ WeightedAncestor$(\ell, d-1)$
4: **if** Depth$(v) > d - 1$ **then**
5: $\quad v \leftarrow$ CreateNode$(v, d-1)$
6: SufLink$(u) \leftarrow v$
7: **return** v

Let d be the string depth of node u. Instead of locating the nearest ancestor of u that has a suffix link, following its suffix link, and going down to reach string depth $d-1$, we employ DWA queries: we first find the leaf node with label Witness$(u) + 1$ (thus deleting one letter from the left) and from thereon trigger a DWA query to reach string depth $d-1$. Line 3 of ComputeSuffixLink(u) takes $\mathcal{O}(\log^2 \log n)$ time by Lemma 1. All other lines of the algorithm take $\mathcal{O}(1)$ time. The total time *for all* calls of ComputeSuffixLink(u) is thus $\mathcal{O}(n \log^2 \log n)$ time. We thus obtain the following result.

Theorem 1. *For any string w of length n over a totally ordered alphabet of size σ, OPST(w) can be constructed in $\mathcal{O}(n(\log \sigma + \log^2 \log n))$ time using $\mathcal{O}(n)$ space.*

III. Experimental Evaluation: Details And Additional Results

A. Baseline algorithms

The baseline algorithms BA-CP and BA-MP take $\mathcal{O}(nk \log k)$ time, where k is the length of the longest frequent pattern in w. They start with computing the PrefCode of all the substrings of w in the order of increasing lengths. Given PrefCode$(w[i\,..\,j-1])$ and the ordered balanced binary search tree on $w[i], \cdots, w[j-1]$ (i.e., the nodes of the tree are these letters) we can compute the PrefCode$(w[i\,..\,j])$ in $\mathcal{O}(\log(j-i))$ time by finding the location of $w[j]$ in the order using the tree. We can then easily update the tree by adding the element $w[j]$ and removing the element $w[i]$ allowing for the computation of PrefCode$(w[i+1\,..\,j+1])$. Each time we compute the hash of this PrefCode$(w[i\,..\,j])$ in $\mathcal{O}(1)$ time (using Karp-Rabin hash [3], and knowing the hash of PrefCode$(w[i\,..\,j-1])$), and store it in the array $A[i][j]$. We can now group all the substrings with the same hash using of a hash-table. This way we know for each substring $w[i\,..\,j]$ if it is frequent and can easily check if the pattern $w[i\,..\,j-1]$ is right-closed or right-maximal.

Once we reach length $k+1$ when no pattern is frequent we can stop the computation, and hence we only consider the $\mathcal{O}(nk)$ substrings of length at most $k+1$ spending $\mathcal{O}(\log k)$ time for each one.

Since it is enough to store the array A and the hash tables only for the two next lengths of the substrings (i.e., upon processing the length-$(\ell+1)$ substrings we get to know if the length-ℓ ones are closed/maximal, and when we start processing the length-$(\ell+2)$ substrings we have already reported the patterns of length ℓ and will never access them again) the solution requires only $\mathcal{O}(n)$ words of space using a hash table of $\mathcal{O}(n)$ size and $\mathcal{O}(1)$-time operations [4].

B. Details of the datasets

In this section, we provide the details of the 6 real-world, large-scale datasets used in our experiments. See Table I in the main paper for links to these datasets except ECG that is proprietary and cannot be made public for privacy reasons. The datasets that we discuss below except ECG are also available at https://bit.ly/4bRlO9G.

HOU is derived from the Individual household electric power consumption dataset (IHEPC) which comprises approximately 2.07 million measurements of electric power consumption for a single house located in Sceaux (7km from Paris, France). These measurements were recorded with a one-minute sampling rate over a period of almost 4 years, spanning from December 2006 to November 2010 (a total of 47 months). To construct a large-scale dataset, we concatenated the measurements from 3 sub-meterings.

SOL is derived from the records of solar power production in 2006, with a sampling rate of every 10 minutes from 137 synthetic solar photovoltaic power plants in Alabama State, US. It was preprocessed by [5] and we concatenated the data from all 137 power plants.

ECG is a dataset derived from collected ECG responses during exercise-induced pain in 40 participants. During the experiment, participants sat on a chair with their elbow at a 90º angle and their wrist joint 20 cm above a table surface. They were instructed to hold their Baseline Weight (calculated based on the heaviest weight they could lift) in an isometric contraction for as long as possible. The ECG and Time to

Exhaustion (TTE) of participants was measured, based on the duration of the weight-holding task. Additionally, participants' Pain Intensity [6] and their Rating of Perceived Exertion were recorded for every minute during the exercise [7]. As mentioned in the main paper, we used ECG in two ways. In the context of experiments about mining patterns from a single string, we concatenated the data from all participants producing a string with length $n = 22,973,535$ and alphabet size $\sigma = 31,731$. In the context of experiments about pattern-based clustering, we used ECG as is (i.e, as a collection of strings, one for each participant). The average and maximum length of the strings was $574,338$ and $1,241,745$, respectively, and the alphabet sizes of the strings are in $[6270, 30248]$.

TRA is derived from 48 months hourly data (2015-2016) provided by the California Department of Transportation. The data describes the road occupancy rates (between 0 and 1) measured by different sensors on San Francisco Bay area freeways. The data was preprocessed by [5] and we concatenated the data from different sensors.

TEM is derived from temperature measurements, captured using an 8×8 infrared array sensor (Panasonic Grid-EYE) over a period of 3 weeks in 2018, located in Bucharest, Romania. Data were recorded at a frequency of 1 Hz, with each frame comprising 64 temperature values in degrees Celsius, corresponding to the 64 cells of the sensing grid. We concatenated the data in the order they were recorded.

WHA is derived from underwater microphone signals, specifically analyzing the energy within the 360-370 Hz frequency band, as released by the National Centers for Environmental Information. It contains passive acoustic recordings of cetaceans collected from areas within the Pacific Islands Region from April 2005 to September 2019. It was preprocessed by [8].

TABLE I: UCR clustering dataset characteristics

Dataset	Sequence Length	# of Sequences	# of Ground Truth Clusters
CinCECGTorso (CCT)	1639	40	4
Wafer (WAF)	152	1000	2

We used 2 time series datasets from different domains for clustering and visualization, obtained from the UCR Archive [9]. Their characteristics are shown in Table I. The sequences in each dataset are of equal length. CCT has two ground truth clusters and WAF four (the time series in each cluster have the same label).

C. Clustering Quality Measures

In the following, we discuss the three widely adopted measures of clustering quality we used, namely Normalized Mutual Information (NMI) [10], Homogeneity (h)[11], and the Rand Index (RI) [12].

A clustering is a partition of a collection of strings into sets called clusters. We consider two clusterings C, C' with sizes $|C|$ and $|C'|$, respectively, and a collection of N strings. C is the ground truth clustering. NMI, h, and RI quantify how similar is C to the ground truth clustering. The values in all these measures are in $[0, 1]$, where a higher value indicates that C' is closer to the ground truth clustering.

The NMI between C and C' is defined as:

$$\text{NMI}(C, C') = \frac{2 \cdot \sum_{i=1}^{|C|} \sum_{j=1}^{|C'|} \frac{n_{ij}}{N} \log\left(\frac{n_{ij/N}}{n_i \cdot \hat{n}_j / N^2}\right)}{-\sum_{i=1}^{|C|} \frac{n_i}{N} \log \frac{n_i}{N} - \sum_{j=1}^{|C'|} \frac{\hat{n}_j}{N} \log \frac{\hat{n}_j}{N}},$$

where n_i denotes the number of strings in the ith cluster in C, $\hat{n}_j$ denotes the number of strings in the jth cluster in C', and n_{ij} denotes the number of clusters belonging both in the ith cluster in C and in the jth cluster in C'.

Homogeneity (h) measures how much the strings in a cluster of the clustering C' agree with those in a cluster of the ground truth clustering C. A cluster in C' is considered homogeneous when it consists entirely of strings belonging to one cluster of C. $h(C, C')$ is defined as:

$$h(C, C') = 1 - \frac{-\sum_{i=1}^{|C|} \sum_{j=1}^{|C'|} \frac{n_{ij}}{N} \log\left(\frac{n_{ij/N}}{\hat{n}_j / N}\right)}{-\sum_{i=1}^{|C|} \frac{n_i}{N} \log \frac{n_i}{N}},$$

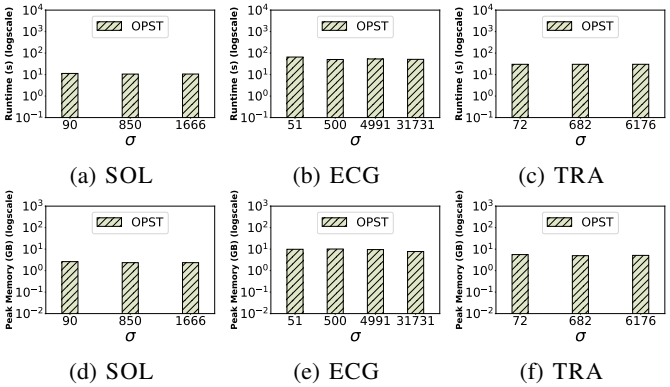
where n_i, n_{ij}, $\hat{n}_j$, and N are as defined above.

The Rand Index (RI) measures similarity between the clustering C' and the ground truth clustering C by counting the number of pairs of strings that are placed in the same clusters in C and C', in different clusters in C and C', or in the same clusters in one of the C and C' and in different clusters in the other. $\text{RI}(C, C')$ is defined as follows:

$$\text{RI}(C, C') = \frac{a + b}{a + b + c + d},$$

where:

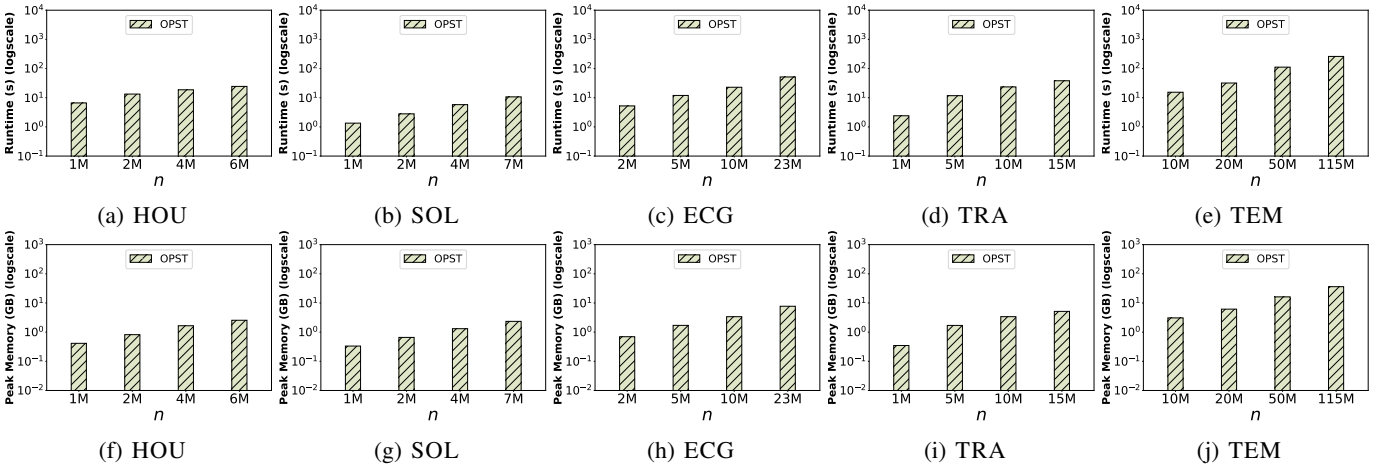
- a is the number of pairs of strings that are in the same cluster in C and C'.
- b is the number of pairs of strings that are in different clusters in both C and C'.
- c is is the number of pairs of strings that are in the same cluster in C but in different clusters in C'.
- d is is the number of pairs of strings that are in different clusters in C but in the same cluster in C'.



(a) SOL (b) ECG (c) TRA

(d) SOL (e) ECG (f) TRA

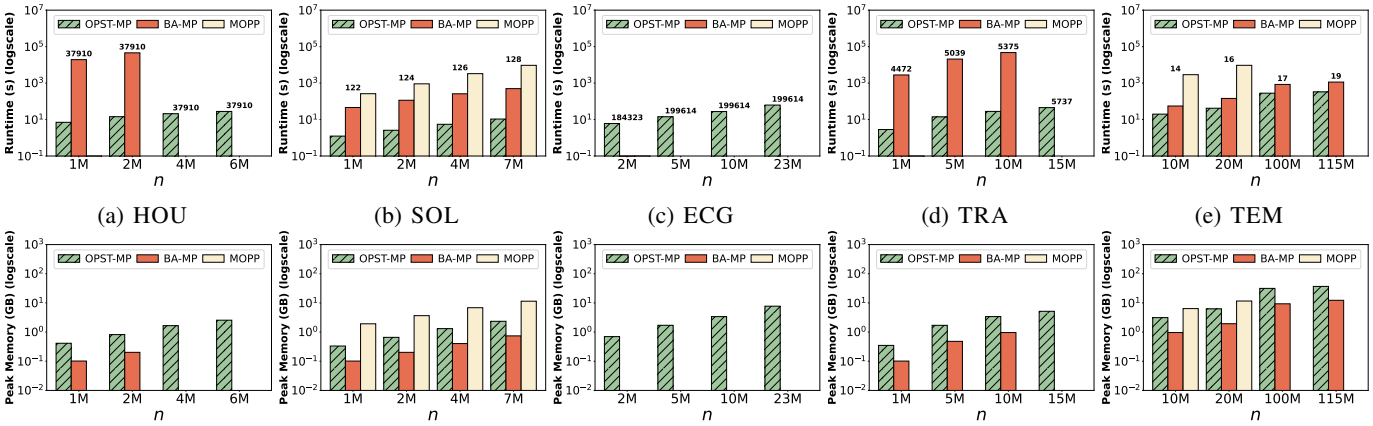
Fig. 1: OPST: (a - c) Runtime and (d - f) respective peak memory consumption for varying σ



(a) HOU (b) SOL (c) ECG (d) TRA (e) TEM

(f) HOU (g) SOL (h) ECG (i) TRA (j) TEM

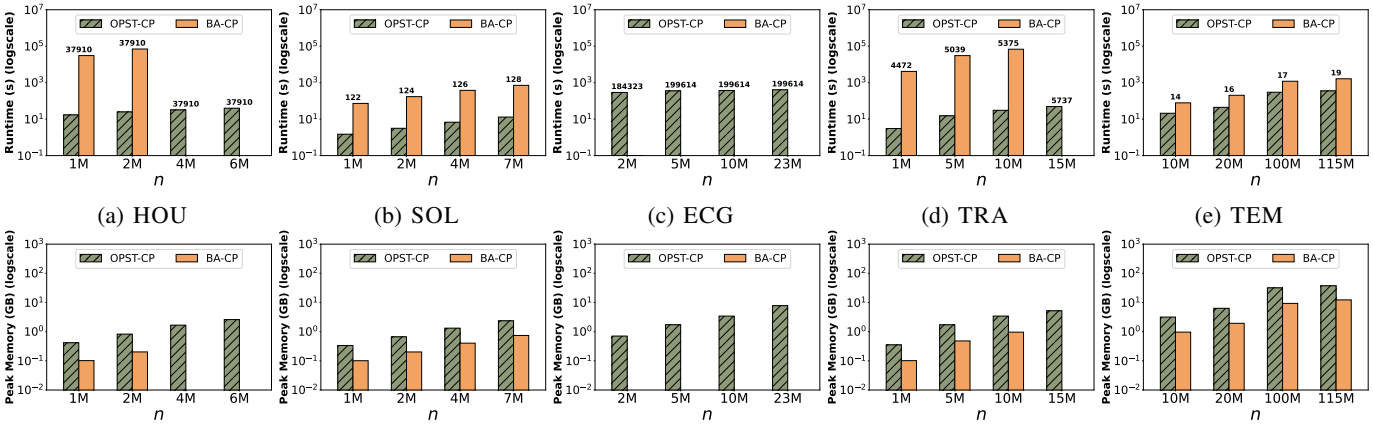
Fig. 2: OPST: (a - e) Runtime and (f - j) respective peak memory consumption for varying n



(a) HOU (b) SOL (c) ECG (d) TRA (e) TEM

(f) HOU (g) SOL (h) ECG (i) TRA (j) TEM

Fig. 3: OPST-MP, BA-MP and MOPP: (a - e) Runtime and (f - j) respective peak memory consumption for varying n. Missing bars for BA-MP and MOPP indicate that they *did not finish within 24 hours.* The value above each pair of bars in (a) - (e) represents the maximum length k of all τ-maximal τ-frequent OP patterns. τ is set to 3 for HOU and TEM, and $\tau = 10$ for the other datasets.



(a) HOU (b) SOL (c) ECG (d) TRA (e) TEM

(f) HOU (g) SOL (h) ECG (i) TRA (j) TEM

Fig. 4: OPST-CP and BA-CP: (a - e) Runtime and (f- j) respective peak memory consumption for varying n. Missing bars for BA-CP indicate that it *did not finish within 24 hours.* The value above each pair of bars in (a) - (e) represents the maximum length k of all closed τ-frequent OP patterns. τ is set to 3 for HOU and TEM, and $\tau = 10$ for the other datasets.

D. Additional experiments results

Figs. 1 to 4 show the impact of σ and n on the runtime and peak memory consumption (construction space) of all methods using the datasets that are not included in our main paper. The results are consistent with those presented in the main paper. The experiments of runtime and peak memory consumption against σ on HOU and TEM were not conducted because σ for HOU and TEM is only 88 and 76, respectively. The values are too small to make meaningful discretization by rounding the raw data to different decimal places.

REFERENCES

[1] E. M. McCreight. "A Space-Economical Suffix Tree Construction Algorithm". In: *J. ACM* (1976), pp. 262–272.

[2] T. Kopelowitz and M. Lewenstein. "Dynamic weighted ancestors". In: *SODA*. 2007, pp. 565–574.

[3] R. M. Karp and M. O. Rabin. "Efficient randomized pattern-matching algorithms". In: *IBM journal of research and development* 31.2 (1987), pp. 249–260.

[4] M. A. Bender et al. "Iceberg Hashing: Optimizing Many Hash-Table Criteria at Once". In: *J. ACM* (2023), 40:1–40:51.

[5] G. Lai et al. "Modeling long-and short-term temporal patterns with deep neural networks". In: *SIGIR*. 2018, pp. 95–104.

[6] D. B. Cook et al. "Naturally occurring muscle pain during exercise: assessment and experimental evidence". In: *Med Sci Sports Exerc.* 29.8 (1997), pp. 999–1012.

[7] G. Borg. *Introduction to Time Series and Forecasting*. 1998.

[8] M. Ceccarello and J. Gamper. "Fast and Scalable Mining of Time Series Motifs with Probabilistic Guarantees". In: *Proceedings of the VLDB Endowment* (2022), pp. 3841–3853.

[9] H. A. Dau et al. *The UCR Time Series Classification Archive*. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/. Oct. 2018.

[10] N. X. Vinh, J. Epps, and J. Bailey. "Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance". In: *JMLR* 11 (2010), pp. 2837–2854.

[11] A. Rosenberg and J. Hirschberg. "V-Measure: A conditional entropy-based external cluster evaluation measure". In: *EMNLP-CoNLL*. 2007, pp. 410–420.

[12] W. M. Rand. "Objective criteria for the evaluation of clustering methods". In: *Journal of the American Statistical Association* (1971), pp. 846–850.