SAND in Action: Subsequence Anomaly Detection for Streams

Paul Boniol
EDF R&D; Univ. de Paris
paul.boniol@edf.fr

John Paparrizos
University of Chicago
jopa@uchicago.edu

Themis Palpanas
Univ. de Paris; IUF
themis@mi.parisdescartes.fr

Michael J. Franklin
University of Chicago
mjfranklin@uchicago.edu

ABSTRACT

Subsequence anomaly detection in long data series is a significant problem. While the demand for real-time analytics and decision making increases, anomaly detection methods have to operate over streams and handle drifts in data distribution. Nevertheless, existing approaches either require prior domain knowledge or become cumbersome and expensive to use in situations with recurrent anomalies of the same type. Moreover, subsequence anomaly detection methods usually require access to the entire dataset and are not able to learn and detect anomalies in streaming settings. To address these limitations, we propose SAND, a novel online system suitable for domain-agnostic anomaly detection. SAND relies on a novel steaming methodology to incrementally update a model that adapts to distribution drifts and omits obsolete data. We demonstrate our system over different streaming scenarios and compare SAND with other subsequence anomaly detection methods.

PVLDB Reference Format:
Paul Boniol, John Paparrizos, Themis Palpanas, and Michael J. Franklin. SAND in Action: Subsequence Anomaly Detection for Streams. PVLDB, 14(1): XXX-XXX, 2020. doi:XX.XX/XXX.XX

1 INTRODUCTION

Large collections of data series[1] are becoming a reality in a large variety of scientific domains and there is an increasing need for systems to efficiently and accurately analyze them [3, 17, 19, 22]. Moreover, many domains require methods to handle streams and adapt to drifts and changes in data distribution. Anomaly or outlier detection is a well-studied problem [4, 25, 30] with applications to several scientific domains [18]. For the case of data series, we are interested in identifying *anomalous subsequences*, where outliers are not single values, but rather a sequence of values. This difference matters, because even when all values in a sequence taken independently from one another are within the healthy range of values, the *sequence* of these same values may be anomalous (e.g., the shape of the subsequence may not be normal). Therefore, subsequence anomaly detection permits the user to detect early potentially abnormal events that would have otherwise been unnoticed.

Considering that several real-world cases are based on continuous streams of data, anomaly detection methods need to take place in real-time. Thus, such methods have to deal with drifts in data distribution. As an example, Figure 1(a) depicts acceleration on the x-axis of a device positioned on the chest of a human performing different actions [26]. We observe that the data characteristics corresponding to subsequences are different for Nordic walking (Figure 1(b)) and rope jumping (Figure 1(c)). As changes of actions happen in real-time, the detection of abnormal subsequences (e.g., the red subsequence in Figure 1(c)) needs to adapt to such changes.

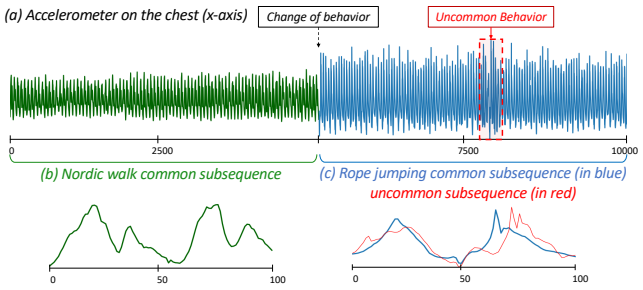

Figure 1: (a) Accelerometer variation positioned on chest (y-axis) while (b) Nordic walking and (c) rope jumping.

In the non-streaming case, several methods have been proposed. For example, the discord-based methods identify as anomalies the subsequences with the largest distances to their nearest neighbor [12, 24, 30]. Alternative approaches estimate anomalies by embedding subsequences into trees [13], sets [5, 6], or directed graphs [7] to capture the different behaviors of the data series, such that anomalies (i.e., rare events) are easy to discriminate. The set-based and the directed graph-based approaches have been shown to outperform the previous state-of-the-art methods [6, 7]. Nevertheless, in the case of streaming data series, among all previous methods for subsequence anomaly detection, only discords methods [30]) and tree-based methods [14] can be used. The remaining methods cannot adapt to distribution changes and learn new data characteristics, both of which are required in data streams.

We address the aforementioned problems and propose a novel system suitable for subsequence anomaly detection in data streams. This system is based on SAND [8], which builds a data set of subsequences representing the different behaviors of the data series. These subsequences are weighted using statistical characteristics such as their cardinality (i.e., how many times the subsequence occurred) and their temporality (i.e., the time difference this subsequence has been detected for the last time). SAND enables this data structure to be updated from one batch to another while being able to compute an anomaly score at every timestamp. Thus, SAND proposes a solution to the subsequences anomaly detection task on streaming sequence data. In this paper, we demonstrate the benefit of a system using SAND to solve subsequence anomaly detection over the data stream. We explore three different scenarios to illustrate how our method achieves the latter problem.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

[1]If the dimensions are ordered by time then we refer to data series as *time series*. We will use the terms *sequence*, *data series*, and *time series* interchangeably.

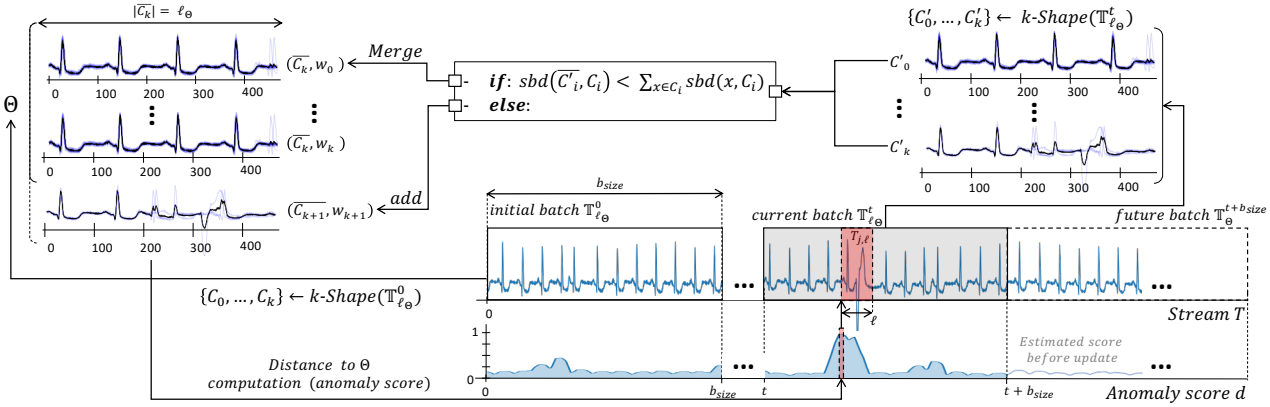

Figure 2: SAND computation framework.

2 SAND APPROACH

We formulate a method for subsequence anomaly detection that operates over data streams.

[Data series and data stream] We focus on the analysis of ordered sequences of measurements. We distinguish between sequences with fixed size (data series) and unbounded evolving sequences (data streams). We formally defined them as follow: a data series $T \in \mathbb{R}^n$ is a sequence of real-valued numbers $t_i \in \mathbb{R}$ $[T_0, T_1, ..., T_n - 1]$; $|T|$ is defined as the length of T. We are interested in local section of the data series, called subsequences. A subsequence $T_{i,\ell} \in \mathbb{R}^\ell$ of a data series T is a subset of contiguous values from T of length ℓ (usually $\ell \ll |T|$) starting at position i: $T_{i,\ell} = [T_i, T_{i+1}, ..., T_{i+\ell-1}]$. We define the set of all subsequences of length ℓ in a given data series T: $\mathbb{T}_\ell = \{T_{i,\ell} | \forall i.0 \leq i \leq |T| - \ell + 1\}$. In the specific case of data streams, the total size of the data series is not known and potentially infinite. Moreover, one can wait for a given number of points before analyzing them. In this case, we define this quantity as a batch. For a given timestamp of arrival t, we note a batch $\mathbb{T}^t_\ell = \{T_{t,\ell}, ..., T_{t+b_{size}-\ell,\ell}\}$ a ordered set of subsequences of length ℓ of size $|\mathbb{T}^t_\ell| = b_{size}$. $\mathbb{T}^0_\ell$ is the initial batch.

[Data series clustering] Since the approach used in our system is based on clustering, we first introduce the related basic elements. Formally, given a set of observations (or subsequences which are the topic of this paper), clustering algorithms aim to partition this set into k distinct clusters, such that the within-cluster sum of squared distances is minimized. For a given set of subsequences $\mathbb{T}_\ell$, we note $\mathbb{C} = \{C_0, ..., C_k\}$ the optimal partition of k cluster C_i with $\forall C_i, C_j \in C, C_i \cap C_j = \emptyset$. We note $\bar{C_i}$ the centroid of cluster C_i.

The k-means algorithm solves this partitioning problem using the z-normalized Euclidean Distance, ED. Formally, given two sequences, A and B, of the same length, ℓ, we can calculate their ED, as follows [9, 16, 27–29]: $ED(A, B) = \sqrt{\sum_i^l (\frac{A_{i,1}-\mu_A}{\sigma_A} - \frac{B_{i,1}-\mu_B}{\sigma_B})^2}$, where μ and σ represent the mean and standard deviation of the sequences.

However, ED-based algorithms cannot capture the necessary property of alignment in data series (i.e., ED is not a competitive distance measure in terms of accuracy [23]). Recently, k-Shape (clustering algorithm based on *Shape-Based-Distance* (*SBD*)) has shown state-of-the-art performance in data-series clustering [20, 21]. This distance uses cross-correlation to find the appropriate alignment between two sequences. The k-shape centroid computation corresponds to an optimization problem in which we are computing the minimizer (i.e., sequence) of the sum of squared distances to all other sequences using the *SBD*. Formally, for a given cluster C_i, the centroid computation is defined as follows:

$$\bar{C_i} \leftarrow \underset{\bar{C_i}}{argmax} \frac{(\bar{C_j})^T . M . \bar{C_i}}{(\bar{C_i})^T . \bar{C_i}} \quad \text{with: } M = Q^T . S_i . Q,\ Q = I - \frac{1}{|\bar{C_i}|} O, \text{ and } S_i = \sum_{A \in C_i} A . A^T \tag{1}$$

In practice, the centroid corresponds to the eigenvector of the largest eigenvalue of the real symmetric matrix M.

Following previous studies [5, 6], clustering can summarize the underlying patterns in data and, therefore, can be used to extract the recurring behavior in datasets for anomaly detection purposes. Clusters with a large number of subsequences characterized a highly recurrent pattern/behavior. Thus, a subsequence with a high distance with a large cluster might correspond to an unusual pattern and potentially abnormal. We formulate the following problem:

Problem 1 (Streaming Subseq. Anom. Detection). *Given a data stream T, arriving in batches $\mathbb{T}^t_\ell$ (with b_{size} the size of the batches) and a targeted anomaly subsequence length ℓ, return $\mathcal{S}_A$, a set containing the η most abnormal subsequences of length ℓ in $\mathbb{T}^t_{\ell_{N_M}}$.*

In this work, we focus on the *Top-k* anomalies; using instead a threshold ϵ to detect anomalies is a straightforward extension.

2.1 SAND Framework

In this section, we briefly present SAND, our solution for unsupervised subsequence anomaly detection in data streams that our SubStream system is based on. Overall, we compute and update a weighted set of subsequences over time. Figure 2 depicts the general framework of the model, which we summarize below:

[Model Creation] We start by computing our model on the initial batch. We first select subsequences candidates (of length $\ell_\Theta > \ell$) and then perform the k-Shape clustering algorithm. These clusters are then scored and stored in memory. Formally, the model is defined as $\Theta = \{(\bar{C_0}, w_0), (\bar{C_1}, w_1)..., (\bar{C_k}, w_k)\}$ with $\bar{C_i}$ the different clusters. The corresponding weight w_i are computed as $w_i = \frac{|C_i|^2}{\sum_{\bar{C_j} \in \Theta} sbd(\bar{C_i}, \bar{C_j})}$. we denote C_i the subsequences set, and

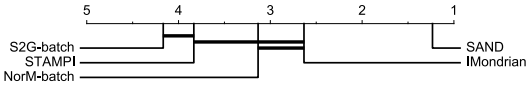

Figure 3: Critical difference diagrams using a Wilkoxon pair-wised signed rank test (with $\alpha = 0.05$) on both single and multiple normality datasets.

$\bar{C}_i$ its centroid. Nevertheless, storing C_i implies an infinite storage need for unlimited streams. Thus, in practice, we do not store C_i.
[Model update] After each new batch, we compute a new clustering on the newest batch. We then need to merge the two clustering (initial and new) partitions. The following steps are then performed:

- *Matching Strategy:* We then match the new cluster with the most similar one in the current model stored in memory. The matching procedure is based on a distance threshold that corresponds to the intra-cluster average distance. Formally, for a given cluster C_i, the threshold is defined as $\tau_{c,i} = \sum_{T_{j,\ell} \in C_i} SBD(T_{j,\ell}, \bar{C}_i)$. If the distance between an existing cluster and a new cluster is smaller than the latter threshold, we merge the two clusters. In the other case, we create a new cluster.
- *Centroid Update:* We then compute the new centroid of two merged clusters without keeping in memory the raw subsequences of these two clusters. This mechanism is a novel technical extension of k-Shape for streaming scenarios. In practice, for a given cluster C_i, we store only the matrix S_i in Equation 1 (of fixed size whatever the number of subsequences). In practice, for a cluster C_i' to be merged with a cluster C_i, we update S_i as $S_i \leftarrow S_i + \sum_{T_{j,\ell_\Theta} \in C_i'} T_{j,\ell_\Theta}.T_{j,\ell_\Theta}^T$. We finally update the weights for each cluster (new, merged, or unchanged).

[Score Computation] At any time, one can compute the anomaly score on the current batch using the current model stored in memory. For a given subsequence $T_{j,\ell} \in \mathbb{T}_\ell$ in the current batch (of length $\ell < \ell_\Theta$), we compute $d_j = \sum_{\bar{C}_i} w^i * min_{x \in [0, \ell_\Theta - \ell]} \{dist(T_{j,\ell}, (\bar{C}_i)_{x,\ell})\}$. We incrementally learn the mean and the standard deviation to compute the anomaly score such that the anomaly detection is adapted to the current batch.

Figure 3 depicts the critical difference diagram computed using a Wilcoxon pair-wised signed-rank test (comparing SAND with STAMPI [11], IMondrian Forest [14], a batch adaptation of NormA [6] and Series2Graph [7] which operate independently on each new arriving batch) on a benchmark of datasets containing data series from the MIT-BIH Supraventricular Arrhythmia Database (MBA) [10, 15] and Simulated Engine Disks data (SED) from the NASA Rotary Dynamics Laboratory [2] (and combinations of them to simulate distribution drifts). The results show that SAND significantly outperforms the online state-of-the-art methods [8].

3 SAND IN ACTION: SYSTEM OVERVIEW

We now describe the system using SAND. The GUI is a standalone web application, developed using Python 3.6 and the Dash framework [1]. Figure 4 displays the different frames of the GUI. The mainframe is shown in Figure 4(a). Once the user opens the web application, they can upload a dataset (as well as the anomaly annotations, if available). At first, only the initial batch is displayed. When new batches arrive, by default only the current and the three newest batches are displayed (as in Figure 4(a.1)). Nevertheless, the user can navigate through older sections of the data series. Moreover, if annotations are provided, they will be colored in red. The user can then change the values of ℓ and the batch size b_{size} by clicking in the SAND dropdown in the navigation bar. Then, by clicking on the initialization button, the anomaly score on the initial batch will be displayed under the data series plot. Moreover, the model Θ (the centroids of the model) is depicted in Figure 4(a.2). By clicking on one of the centroids, the user can visualize the corresponding weight and the time distribution of the subsequences contained in the selected centroid (Figure 4(a.4)). Once the model is initialized, it is ready to be updated with new batches arriving. The user then may (i) decide to manually add the next batch (by clicking on the "by batch" button) or (ii) have the system process batches continuously (by clicking on the "continuous" button). When a new batch arrives, both the data series and the anomaly score plots are updated. The model is also updated, and the new clusters from the current batch that need to be merged with an existing cluster will be aligned with it (under the label "cluster to be merged" as in Figure 4(a.3)). The new clusters appear under “clusters to create.”

The user can also run other anomaly detection methods: STAMPI [11] and Isolation Mondrian Forest [14] (IMondrian). Their anomaly scores will be shown together with the SAND anomaly scores (Figure 4). If annotations are provided, performance analysis can be done by clicking on the performance button: a new frame will appear (Figure 4(b)) displaying accuracy and time execution evaluations. For accuracy evaluation, we compute the precision (the number of correctly detected anomaly points divided by the total number of anomalous points, depicted in Figure 4(b.1)), recall (the number of correctly detected anomaly points divided by the total number of anomalous points, depicted Figure 4(b.2)), and $F1 = \frac{2*Recall*Precision}{Recall+Precision}$ (Figure 4(b.3)). These three accuracy measures are computed for thresholds moving between the minimal and the maximal value of the anomaly score (subsequences that have a score above a given threshold are marked as an anomaly). We then use either the maximal value of the F1 score for the best threshold (Figure 4(b.5)), or the Area Under the F1 Curve computed for all possible thresholds (Figure 4(b.6)) as global accuracy measures. Finally, the average execution time per batch for every method is summarized in another bar plot (Figure 4(b.4))).

4 DEMONSTRATION SCENARIOS

This demonstration has four goals: (i) showcase the effectiveness of SAND and compare it to competing approaches in term of anomaly detection accuracy; (ii) enable the user to understand and interpret the intermediate steps of the method SAND using the uploaded data series, by visualizing the centroids and the evolution of the weights while new batches arrive; and (iii) challenge the user to identify anomalies by navigating the data and SAND’s parameters.
[Scenario 1: Effectiveness] This scenario begins with a long data series (200,000 points with 108 anomalies of length approximately equal to 100 points each) representing the concatenation of the 806 and 820 records from the MIT-BIH Supraventricular Arrhythmia Database (MBA) [10, 15] (as illustrated in Figure 4(a)). We will first run SAND by displaying the intermediate, inner steps, and computing and displaying the anomaly scores. Then, we will run

(a) Screenshot SAND system main frame

(b) Screenshot when performance button is pressed

Figure 4: SAND system GUI main frame.

the competing approaches and display their anomaly scores as well. Overall, we will show that SAND is both faster and more accurate.

[Scenario 2: System Internals] The second scenario will allow the user to examine the SAND framework inner steps. Directly through the mainframe, the user will be able to visualize both the Θ model centroids (by being able to zoom and move), the centroids weights, and the time distribution of the subsequences inside the corresponding clusters. Then, while new batches arrive, the user will also be able to inspect the cluster matching operated automatically by SAND. These inner steps facilitate the understanding of the score computation that is computed and displayed at each batch.

[Scenario 3: Discovering Distribution Drifts] The third scenario will focus on the analysis of the discovered anomalies and distribution drift. This task will go beyond the anomaly detection task. We will use the visualization tools provided by our system to understand the impact of distribution drift on the model Θ, and more generally, how SAND is handling distribution drift. The user will notice a drastic change in the time distribution of the centroids (as in Figure 4(a.4)) and the creation of centroids with new shapes. This scenario will help the user to understand the challenging task of anomaly detection over data streams with distribution drifts.

5 CONCLUSIONS

We demonstrate a system that uses the novel SAND framework for streaming unsupervised subsequence anomaly detection. This system permit the user to follow the computational steps involved in SAND framework and compare easily the accuracy and run time performances of SAND and other state-of-the-art methods.

Acknowledgments Work supported by EDF R&D and ANRT.

REFERENCES

[1] Dash documentation. https://dash.plotly.com/.
[2] A. Abdul-Aziz, M. R. Woike, N. C. Oza, B. L. Matthews, and J. D. lekki. Rotor health monitoring combining spin tests and data-driven anomaly detection methods. *Structural Health Monitoring*, 2012.
[3] A. J. Bagnall, R. L. Cole, T. Palpanas, and K. Zoumpatianos. Data series management (dagstuhl seminar 19282). *Dagstuhl Reports*, 9(7), 2019.
[4] V. Barnet and T. Lewis. *Outliers in Statistical Data*. John Wiley and Sons, Inc., 1994.
[5] P. Boniol, M. Linardi, F. Roncallo, and T. Palpanas. Automated Anomaly Detection in Large Sequences. In *ICDE*, 2020.
[6] P. Boniol, M. Linardi, F. Roncallo, T. Palpanas, M. Meftah, and E. Remy. Unsupervised and Scalable Subsequence Anomaly Detectionin Large Data Series. *VLDBJ*, 2021.
[7] P. Boniol and T. Palpanas. Series2graph: Graph-based subsequence anomaly detection for time series. *PVLDB*, 13(11), 2020.
[8] P. Boniol, T. Palpanas, J. Paparrizos, and M. J. Franklin. SAND: Streaming Subsequence Anomaly Detection. *PVLDB*, 2021.
[9] B. Y. Chiu, E. J. Keogh, and S. Lonardi. Probabilistic discovery of time series motifs. In *SIGKDD 2003*, pages 493–498, 2003.
[10] G. et al. Physiobank, physiotoolkit, and physionet. *Circulation*.
[11] Y. Z. et al. Matrix profile II: exploiting a novel algorithm and gpus to break the one hundred million barrier for time series motifs and joins. In *ICDM 2016*.
[12] M. Linardi, Y. Zhu, T. Palpanas, and E. J. Keogh. Matrix Profile Goes MAD: Variable-Length Motif And Discord Discovery in Data Series. In *DAMI*, 2020.
[13] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *ICDM*, ICDM, 2008.
[14] H. Ma, B. Ghojogh, M. N. Samad, D. Zheng, and M. Crowley. Isolation mondrian forest for batch and online anomaly detection, 2020.
[15] G. B. Moody and R. G. Mark. The impact of the mit-bih arrhythmia database. *IEEE Engineering in Medicine and Biology Magazine*, 2001.
[16] A. Mueen, E. J. Keogh, Q. Zhu, S. Cash, and M. B. Westover. Exact discovery of time series motifs. In *SDM 2009*.
[17] T. Palpanas. Data series management: The road to big sequence analytics. *SIGMOD Rec.*, 44(2):47–52, 2015.
[18] T. Palpanas and V. Beckmann. Report on the first and second interdisciplinary time series analysis workshop (ITISA). *SIGMOD Rec.*, 48(3), 2019.
[19] J. Paparrizos and M. J. Franklin. GRAIL: Efficient Time-Series Representation Learning. *PVLDB*, 12(11):1762–1777, 2019.
[20] J. Paparrizos and L. Gravano. k-Shape: Efficient and Accurate Clustering of Time Series. In *SIGMOD*, pages 1855–1870, 2015.
[21] J. Paparrizos and L. Gravano. Fast and accurate time-series clustering. *TODS*, 42(2):1–49, 2017.
[22] J. Paparrizos, C. Liu, B. Barbarioli, J. Hwang, I. Edian, A. J. Elmore, M. J. Franklin, and S. Krishnan. Vergedb: A database for iot analytics on edge devices. In *CIDR*, 2021.
[23] J. Paparrizos, C. Liu, A. J. Elmore, and M. J. Franklin. Debunking four long-standing misconceptions of time-series distance measures. In *SIGMOD*, page 1887–1905, 2020.
[24] P. Senin, J. Lin, X. Wang, T. Oates, S. Gandhi, A. P. Boedihardjo, C. Chen, and S. Frankenstein. Time series anomaly discovery with grammar-based compression. In *EDBT*, 2015.
[25] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online outlier detection in sensor data using non-parametric models. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, 2006.
[26] D. van Kuppevelt, V. van Hees, and C. Meijer. Pamap2 dataset preprocessed v0.3.0, July 2017.
[27] J. Wang, A. Balasubramanian, L. M. de la Vega, J. Green, A. Samal, and B. Prabhakaran. Word recognition from continuous articulatory movement time-series data using symbolic representations. In *SLPAT*.
[28] C. Whitney, D. Gottlieb, S. Redline, R. Norman, R. Dodge, E. Shahar, S. Surovec, and F. Nieto. Reliability of scoring respiratory disturbance indices and sleep staging. *Sleep*, November 1998.
[29] D. Yankov, E. J. Keogh, J. Medina, B. Y. Chiu, and V. B. Zordan. Detecting time series motifs under uniform scaling. In *ACM*.
[30] 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 *ICDM*, pages 1317–1322, 2016.