Online Temporal Data Mining and Learning: Pursuing Enhanced Efficiency and Robust Algorithms

Sheng Zhong
The University of New Mexico

Follow this and additional works at: https://digitalrepository.unm.edu/cs_etds

Part of the Artificial Intelligence and Robotics Commons, Databases and Information Systems Commons, Data Science Commons, and the Geophysics and Seismology Commons

Recommended Citation

This Dissertation is brought to you for free and open access by the Engineering ETDs at UNM Digital Repository. It has been accepted for inclusion in Computer Science ETDs by an authorized administrator of UNM Digital Repository. For more information, please contact disc@unm.edu.
Sheng Zhong

Candidate

Computer Science

Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

Abdullah Mueen

Chair

Shuang Luan

Member

Bruna Jacobson

Member

Jessica Lin

Member
Online Temporal Data Mining and Learning: Pursuing Enhanced Efficiency and Robust Algorithms

by

Sheng Zhong
B.E., Electronic Science, University of Electronic Science and Technology of China, 2016
B.S., Computer Engineering, University of New Mexico, 2016

DISSERTATION
Submitted in Partial Fulfillment
of the Requirements for the Degree of
Doctor of Philosophy
Computer Science
The University of New Mexico
Albuquerque, New Mexico

May, 2024
Online Temporal Data Mining and Learning: Pursuing Enhanced Efficiency and Robust Algorithms

by

Sheng Zhong

B.E., Electronic Science, University of Electronic Science and Technology of China, 2016
B.S., Computer Engineering, University of New Mexico, 2016
Ph.D., Computer Science, University of New Mexico, 2024

ABSTRACT

Time series data mining and learning serve as a cornerstone across various domains, including finance, healthcare, and science. Recent advancements in network and sensor technologies have ignited an increasing interest in real-time temporal data mining and learning techniques. Various tasks benefit from these techniques, such as environmental monitoring, event detection, anomaly identification, and forecasting. However, these techniques still face significant challenges in the online environment settings, encompassing aspects like efficiency, accuracy, robustness, and scarcity of labeled data. This dissertation presents four innovative solutions: FilCorr, DCT-MASS, FewSig, and BitLINK to overcome these challenges. We evaluate each method and showcase their practical significance through applications in earthquake early warning systems, aftershock sequence detection, and blockchain address identification. We hope this research will enhance our understanding of the challenges and opportunities in online temporal data mining and learning, ultimately leading to more efficient and robust algorithms.
# TABLE OF CONTENTS

| List of Figures | vii |
| List of Tables  | vii |

## 1 Introduction

1.1 FilCorr: Filtered and Lagged Correlation on Streaming Time Series . . 2
1.2 MASS: Distance Profile of a Query over a Time Series . . . . . . . . . . 3
1.3 FewSig: Online Few-Shot Time Series Classification . . . . . . . . . . . 3
1.4 BitLink: Temporal Linkage of Address Clusters in Bitcoin Blockchain . . 4

## 2 FilCorr: Filtered and Lagged Correlation on Streaming Time Series 5

2.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
2.2 Background and Notation . . . . . . . . . . . . . . . . . . . . . . . . . . 8
2.3 Related Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
2.4 FilCorr: Filtered Lagged Correlation . . . . . . . . . . . . . . . . . . . . 12
   2.4.1 Lagged Correlation . . . . . . . . . . . . . . . . . . . . . . . . . . 13
   2.4.2 Lagged Correlation with Filtering . . . . . . . . . . . . . . . . . . 14
   2.4.3 Computational Complexity . . . . . . . . . . . . . . . . . . . . . . 18
   2.4.4 Extensions to Our Implementation . . . . . . . . . . . . . . . . . . 19
   2.4.5 Discussion on Data Independence . . . . . . . . . . . . . . . . . . 20
   2.4.6 Managing Spurious Correlation . . . . . . . . . . . . . . . . . . . . 20
2.5 Experimental Evaluation . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
   2.5.1 Sanity Check . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
   2.5.2 Setup . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
   2.5.3 Efficiency . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
   2.5.4 Comparison to Existing Method . . . . . . . . . . . . . . . . . . . . 23
   2.5.5 Parameter Sensitivity . . . . . . . . . . . . . . . . . . . . . . . . . 26
2.6 Case Studies . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
   2.6.1 Seismic Event Monitoring . . . . . . . . . . . . . . . . . . . . . . . 27
   2.6.2 Motion Monitoring . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
   2.6.3 Electrophysiological Monitoring . . . . . . . . . . . . . . . . . . . 32
2.7 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33

## 3 MASS: Distance Profile of a Query over a Time Series 35
3.1 Introduction ...................................................... 35
3.2 Related Work .................................................... 36
3.3 Background and Notation ...................................... 38
3.4 MASS: Mueen’s Algorithm for Similarity Search .......... 40
  3.4.1 MASS V1 .................................................. 41
  3.4.2 MASS V2 .................................................. 42
  3.4.3 MASS V3 .................................................. 43
  3.4.4 MASS V4 .................................................. 45
  3.4.5 Algorithmic Complexity .................................. 47
3.5 Extensions of MASS ............................................ 48
  3.5.1 Weighted Distance Profile ............................... 48
  3.5.2 Absolute Distance Profile ............................... 49
  3.5.3 Correlation Profile ...................................... 50
  3.5.4 Partial Correlation Profile ............................. 50
  3.5.5 Multivariate Time Series ............................... 52
  3.5.6 Discussion ................................................ 52
3.6 Comparison with Indexing Solution ......................... 52
3.7 Utility in Application Domains ................................ 53
  3.7.1 Survey on applications .................................. 53
  3.7.2 Robots ................................................... 53
  3.7.3 Solar Power .............................................. 54
  3.7.4 Seismology .............................................. 55
3.8 Utility in Algorithms .......................................... 55
3.9 Conclusion ...................................................... 59

4 FewSig: Online Few-Shot Time Series Classification .... 60
  4.1 Introduction .................................................. 60
    4.1.1 Motivation .............................................. 61
  4.2 Related Work ................................................ 62
    4.2.1 Semi-supervised learning on time series .............. 62
    4.2.2 Few-shot learning ..................................... 63
    4.2.3 Aftershock Detection .................................. 64
  4.3 Online Few-shot Time Series Classification ............. 64
    4.3.1 Distance/Dissimilarity as features .................. 66
    4.3.2 Selective model ....................................... 68
    4.3.3 General time series classifier ...................... 69
LIST OF FIGURES

2.1 (left) A M7.8 earthquake in Gurkha, Nepal, on Apr-25, 2015. (right) The signals recorded at three different stations at different times due to the spatial separation of sensors. Data collected from IRIS [54].

2.2 The top row shows raw waveforms with no clear seismic signal. The bottom row shows filtered waveforms of the corresponding raw waveforms in the top row. Note the decrease in absolute value in the y-axis and the increased visibility of the signal. The filter band is 0.4Hz to 3Hz. The average lagged correlation of all three pairs of raw signals is zero. The average lagged correlation of all three pairs after filtering is 0.53.

2.3 Examples of various windows on a stream, where \( m = 4, l = 3, \) and \( \text{step} = 2 \).

2.4 Running example of how to derive the filtered frequency window and filtered basic window.

2.5 Comparison between Naive algorithm (left) and FilCorr (right) for computing a correlation between two basic windows. The keep non-zero operation that bridges Naive and FilCorr only needs to perform once to initiate the FilCorr, and then it can update the filtered frequency window from the previous one.

2.6 Butterworth second order bandpass filter 3-7Hz.

2.7 Example of the Ringing effect on two uncorrelated seismic signals and how the multiplication by a Hamming window can address the false high correlation between them.

2.8 Offline performance of FilCorr and naive algorithm.

2.9 Online performance of FilCorr and naive algorithm. The left figure shows the results varying the sampling rate and fixed values for the remaining parameters. The right figure shows the results under different output rates. In both figures, the total number of stream pairs are computed as all possible combinations of any two streams.

2.10 Comparison with ParCorr fixing \( \text{step} = 20 \). The vertical red line shows the crossing point between ParCorr and FilCorr with lag = 0, and the vertical blue line shows the crossing point between ParCorr and naive algorithm with lag = 0.
2.11 Comparison with ParCorr varying the step from 5 to 20 observations. The vertical red line shows the crossing point between ParCorr and FilCorr with lag = 0; the arrow points the point where the output rate is 6Hz. The vertical blue line shows the crossing point between ParCorr and naive algorithm with lag = 0, and the arrow points to the output rate as 7Hz.

2.12 Results for the parameter sensitivity test considering the online scenario.

2.13 Different recommendation levels on FilCorr based on the application requirements. Neutral means there are no apparent benefits (No loss either) when compared with time domain implementation. Approximation means results are derived from fewer frequency components.

2.14 Main components of Seisviz, a real-time system for seismic event monitoring.

2.15 Pairwise correlation among 29 stations at Yellowstone, WY over different times given an M6.5 earthquake occurred in Challis, Idaho, at 23:52:30 2020-03-31(UTC).

2.16 Motion sensor locations and examples of time series generated by each sensor in the three axes.

2.17 Cross-correlation between pairs of dancers over all body parts considering the use and absence of filter and lag.

2.18 Lag-correlated pairs of channels present in one group and not present in the other.

3.1 Convolving a time series with a reversed and padded query produces necessary sliding dot products (results from step 2, 3, 4) for distance profile.

3.2 Valid convolution produces necessary information for distance profiling in half space and time required by full convolution.

3.3 (left) Splitting the long time series into segments of length $K$. (right) The best value of $K$ depends on the hardware. In this example, $K = 2^{16}$ produces the fastest execution.

3.4 The stopwatch time in seconds for different algorithms and different input lengths when $m = 100, k = 2^{15}$. The platform has an I9-9900k CPU with 128GB RAM, the Matlab version is 2021b.
3.5 The query corresponds to the accelerometer signal of two steps taken by a SONY robot walking on the carpet. The accompanying weighted vector preserves the periods where the robot’s foot contacts the carpet and ignores periods where the foot is in the air. We applied MASS to calculate the weighted distance profile of the query on the full sequence of the robot’s movements over time (yellow time series), and the red time series is the result distance profile. In this profile, lower values correspond to a closer match with the query signal. We use a grey-shaded area to denote the time when the robot is moving on the carpet based on the ground truth data. The weighted distance profile matches this ground truth, as multiple subsequences in this period have a weighted distance below the threshold.

3.6 The top plot illustrates the query pattern which represents the typical power consumption of a house during idle or vacant periods. This pattern is characterized by its cyclic and consistent nature with minimal variations, which is typical of a household when it is not actively being used. The middle plot exhibits the power consumption time series data of a household over an extended period. The fluctuations and spikes indicate varying levels of activity and usage within the household. In the bottom plot, the distance profile calculated by the MASS algorithm is presented. The dashed line represents a distance threshold set at 22.5. Periods, where the distance profile falls below this threshold, are shaded in gray and labeled as Vacancy. These intervals signify times when the household’s power consumption pattern closely matches the idle or vacant query pattern, suggesting that the house was likely unoccupied during these times.
3.7 Two seismograph traces are presented at the top, the red trace denotes
the mainshock, while the blue trace represents an aftershock. These
traces are extracted from a streaming time series recorded at a seismic
station running at 40 Hz. The displayed data includes the 30 seconds
before and after the arrival time of the seismic event, which has been
identified by human analysts. Initially, when compared directly, the two
sequences exhibit a 0% correlation, indicating no direct similarity between
the mainshock and aftershock as captured by the seismographs. However,
we use MASS to compute the distance profile of a query, which is a 40-
second subsequence from the mainshock, against the 60-second sequence of
the aftershock. The bottom figure showcases the 80% correlation between
the query and the best-matching subsequence from the aftershock data.

3.8 Execution time of MASS-Discord in comparison to traditional discord
discovery algorithm on three real datasets.

3.9 Execution time comparison between MASS and a naive approach when
applied to complete-linkage clustering on streaming time series. The
comparison spans three different real-world datasets robots, solar data,
and seismographs. The MASS algorithm demonstrates a considerable
reduction in execution time compared to the naive method, maintaining
a consistent lead as the subsequence length increases. The MASS-based
approach is around 20 times faster on solar datasets and approximately
15 times faster on robot and seismograph datasets.

3.10 The top graph shows the time series of the first dimension of the Winding
dataset with various segments color-coded to represent different clusters
identified by Algorithm 11. The lower graphs provide expanded views of
identified clusters, showing the consistency within each cluster despite the
noisy environment. The subsequence length is set to 100.

4.1 Online classification with self-training.

4.2 Two real aftershock signals show the highest correlation of 80.84% when
one is shifted in time relative to the other to correct for human error in
picking, and near 0% correlation if not shifted.

4.3 The DTW alignment between the middle segments of signals in Figure
4.2, the warping band is limited to 3 points. The blue signal exhibits
warping due to the variation in wave propagation from the event origin to
the seismometer. The correlation after correcting warping is 89.85%.
4.4 Distance features for a time series \( t \).

4.5 Proposed general time series classifier.

4.6 Critical difference diagram of 5 models on the 68 UEA&UCR benchmark datasets. \( \rho = 0.05 \) for Nemenyi test.

4.7 Mean F1 scores of 30 trails on 68 datasets comparison between FewSig and Wei’s model (left), SSTSC (right).

4.8 Critical difference of FewSig with the different number of votes on 68 datasets, \( vote = 2 \) yield highest rank.

4.9 Critical difference diagram of FewSig with four different \( tFPR \) values.

4.10 F1 score comparison between labeled sets \( L \) with the different number of positive instances for 68 datasets.

4.11 The distribution of origins for events in the LEB and the local catalog \([73]\).

4.12 Left figure shows the geographical distribution of origins for extracted arrivals at MKAR. The right figure shows some example waveforms from the BHZ channel at MKAR. The first five aftershocks are in red. They were filtered with a 0.4Hz to 10Hz Butterworth bandpass filter.

4.13 Online performance for classifying the Nepal aftershock sequence at MKAR. F1 scores of different models are shown on the solid curves. A point on a curve shows the score when testing the events at and before the time on the x-axis. The accumulated number of testing aftershocks and non-aftershocks are represented by the light green and blue shaded areas respectively. Both SUCCESS\(_{OL}\) and SSTSC\(_{OL}\) have F1 scores of zero throughout. We only use Z channel time series data for SSTSC\(_{OL}\) since it only supports univariate time series.

4.14 Online performance for classifying the Nepal aftershock sequence at MKAR. TPR and FPR scores of FewSig and Wei\(_{OL}\) are shown on the solid and dotted curves.

4.15 The origins of 26 false positives (red) and 37 false negatives (blue) obtained by FewSig overall.

4.16 The origin distribution of events in LEB and the local catalog \([77]\). The events in LEB are selected by limiting the origins to the red rectangular region and limiting the origin time from 2017-09-08T04:49:17.000000Z to 2018-03-08T04:24:000000Z.
4.17 Origins for extracted arrivals at TXAR station

4.18 Online performance of each model while classifying the Chiapas aftershock sequence at TXAR.

4.19 Online performance for classifying Chiapas aftershock sequence at TXAR.

TPR, FPR scores of FewSig and Wei OL are shown on the solid and dotted curves.

4.20 The origins of 10 false positives (red) and 13 false negatives (blue) obtained by FewSig overall.

5.1 Temporal activity of Bitcoin addresses owned by BitPay, addresses with the same color are in the same cluster which is formed by utilizing the multi-input heuristic. The x-axis indicates the timestamp of the first receiving transaction, while the y-axis corresponds to the timestamp of the last spending transaction for each address.

5.2 The time series of daily receiving transaction count for address clusters owned by BitPay. On the left, a clear temporal linkage signature is depicted, demonstrating the transition from flushed cluster C1 to carry-on cluster C2. The right graph presents the transaction count time series for four clusters (C1, C2, C3, C4) tagged by WalletExplorer.com, alongside two clusters (M1, M2) that have been identified through our BitLINK.

5.3 Transaction graph of clusters owned by HelixMixer

5.4 Temporal visualization of address flushing across various Bitcoin services labeled by WalletExplorer. The red time series represents flushed clusters, while the blue time series indicates the carry-on clusters as identified by WalletExplorer. The top row illustrates the flushing pattern, with the bottom row providing a detailed view of the transition period. The Y-axis shows the daily count of receiving transactions by a cluster. The X-axis shows years in the top row and days/months in the bottom row.

5.5 The architecture for the BitLINK, the output dimension from each layer is reported in the top figure.

5.6 Data augmentation process utilizing Bitcoin on-chain data. For demonstrative purposes, we focus on the augmentation of the red time series at the second split point.

5.7 The blue curve shows raw times, and the red curve shows trend time series for BTC-e. The Y-axis is the daily receiving transaction count. Vertical lines indicate the start and end of each non-zero sub-sequence.
5.8 Three receiving time series of three clusters owned by CrimeNetwork.cc, tagged by WalletExplorer. Both the blue time series and green time series are temporally linked with the same red one.

5.9 Left: 80% unlabeled dataset for training, 20% for validation, using WalletExplorer and manually labeled data as testing. Right: Full unlabeled dataset for training, WalletExplorer data as validation, manually labeled data for testing.

5.10 Confusion matrix for the testing results of BitLINK on W.E. and M.L. datasets when the best G-mean is achieved on the validation set.

5.11 Left: TPR, FPR results for each service when tested with BitLINK. Right: G-mean of each service with respect to TS2Vec and BitLINK.

5.12 The time series for 35 address clusters owned by HelixMixer identified by the W.E, which are shown in color. 50 additional address clusters linked by BitLINK shown in black. Click Here for a high-resolution version.

5.13 The time series of daily counts of receiving transactions for 5 address clusters owned by Primedice identified by the WalletExplorer, which are shown in color. 10 additional address clusters linked by BitLINK shown in black.

5.14 The left figure displays the time series data for Bitcoin Fog and its most frequent receiver. On the right, there are six chained clusters chain linked from the most frequent receiver. These linked clusters also demonstrate a notable correlation with Bitcoin Fog. The correlation between them is calculated using a normalized DTW distance with a 2-day warping band.
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Symbols and definitions.</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>Capabilities of FilCorr and related work. The symbol ✓ represents a</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>claimed capability; – represents extendable capability, and × represents</td>
<td></td>
</tr>
<tr>
<td></td>
<td>unknown.</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>Time complexity in Big-O notation</td>
<td>47</td>
</tr>
<tr>
<td>3.2</td>
<td>FLOPs comparison when ( m = 100, k = 2^{15} ) for V3</td>
<td>47</td>
</tr>
<tr>
<td>4.1</td>
<td>Symbols and notations.</td>
<td>65</td>
</tr>
<tr>
<td>4.2</td>
<td>Average running time in seconds. ( k ) is the number of candidate parameter</td>
<td>73</td>
</tr>
<tr>
<td>5.1</td>
<td>Features utilized in different tools for analyzing Bitcoin address clusters.</td>
<td>87</td>
</tr>
<tr>
<td>5.2</td>
<td>Symbols and notations.</td>
<td>93</td>
</tr>
<tr>
<td>5.3</td>
<td>Data sets size</td>
<td>94</td>
</tr>
<tr>
<td>5.4</td>
<td>Testing results on the W.E. dataset when G_mean of validation score is</td>
<td>98</td>
</tr>
<tr>
<td></td>
<td>highest on the M.L. dataset. The numbers are the mean values for five</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rounds.</td>
<td></td>
</tr>
<tr>
<td>5.5</td>
<td>Testing results on the M.L. dataset when G_mean of validation score is</td>
<td>98</td>
</tr>
<tr>
<td></td>
<td>highest on the W.E. dataset. The numbers are the mean values for five</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rounds.</td>
<td></td>
</tr>
<tr>
<td>5.6</td>
<td>BitLINK performance in adding more information about three services.</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>The BitLINK columns show added information to what exists in the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WalletExplorer columns.</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

As the digital realm grows, the influx of time-dependent data from diverse sources creates opportunities and challenges for industries and researchers. Accurate and timely decision-making, supported by vast amounts of real-time data, can positively influence numerous sectors, including finance, healthcare, safety, energy management, etc. However, the exponential growth in data generation has made the processing and analysis of this information increasingly challenging. As a result, online temporal data mining and learning have emerged as essential techniques for addressing these challenges. This dissertation explores these rapidly evolving areas, focusing on the development and application of state-of-the-art methods, models, and algorithms.

Numerous state-of-the-art data mining algorithms, such as Trillion DTW [101] and MASS [86], have significantly improved the efficiency of time series mining tasks. However, directly applying these offline techniques to online environments may prove inadequate, as they do not account for the unique challenges posed by online tasks. Online scenarios require algorithms or models that can swiftly process and analyze continuously updated data points, keeping pace with the incoming data stream. Additionally, online data mining algorithms must exhibit high scalability to cope with increasing data volume and maintain robustness in noisy conditions.

In response to these challenges, we first introduce FilCorr, a model-free method for real-time event detection and monitoring. FilCorr can compute all pairwise Pearson correlations among hundreds of high-speed streaming time series in real-time. We also propose a modified version of the MASS algorithm, utilizing the Discrete Cosine Transform to compute the distance profiles. This adaptation eliminates the use of complex numbers found in the original MASS, enhancing stability for online computations.

On the other hand, numerous offline models, such as Hive-COTE [78], ROCKET [27],
and different time series classification algorithms, have achieved significant results on static datasets. Deep neural network-based models also demonstrate strong performance when provided with sufficient static training data. Yet, these models generally underperform in online settings due to limited data availability and the dynamic nature of data patterns over time. In contrast, our FewSig model, an online few-shot learning model, enables accurate identification of novel events with minimal training data and adapts to evolving data autonomously, which is ideal for dynamic online environments. Furthermore, we introduce BitLink, a self-supervised learning model tailored for assembling time series data where labeling is scarce, enhancing the efficacy and applicability in minimally supervised online settings.

The following are the abstracts of the proposed works.

1.1 FilCorr: Filtered and Lagged Correlation on Streaming Time Series

Monitoring systems have hundreds or thousands of distributed sensors gathering and transmitting real-time streaming data. The early detection of events in these systems, such as an earthquake in a seismic monitoring system, is the base for essential tasks such as warning generations. To detect such events, it is usual to compute pairwise correlation across the disparate signals generated by the sensors. Since the data sources (e.g., sensors) are spatially separated, it is essential to consider the lagged correlation between the signals. Besides, many applications require processing a specific band of frequencies depending on the event’s type, demanding a pre-processing step of filtering before computing correlations. Due to the high speed of data generation and the large number of sensors in these systems, the operations of filtering and lagged cross-correlation need to be efficient to provide real-time responses without data losses. This work proposes a technique named FilCorr that efficiently computes both operations in one single step. We achieve an order of magnitude speedup by maintaining frequency transforms over sliding windows. Our method is exact, devoid of sensitive parameters, and easily parallelizable. Besides our algorithm, we also provide a publicly available real-time system named Seisviz that employs FilCorr in its core mechanism for monitoring a seismometer network. We demonstrate that our technique is suitable for several monitoring applications, such as seismic signal monitoring, motion monitoring, and neural activity monitoring.
1.2 MASS: Distance Profile of a Query over a Time Series

Given a long time series, the distance profile of a query time series computes distances between the query and every possible subsequence of a long time series. MASS (Mueen’s Algorithm for Similarity Search) is an algorithm to efficiently compute distance profile under z-normalized Euclidean distance. MASS is recognized as a useful tool in many data mining works. However, complete documentation of the increasingly efficient versions of the algorithm does not exist.

In this work, we formalize the notion of a distance profile, describe four versions of the MASS algorithm, show several extensions of distance profiles under various operating conditions, describe how MASS improves performances of existing data mining algorithms, and finally, show utility on a few real datasets from seismology, robotics, and power grids.

1.3 FewSig: Online Few-Shot Time Series Classification

Seismic monitoring systems sift through seismograms in real-time, searching for target events, such as underground explosions. In this monitoring system, a burst of aftershocks (small earthquakes that occur following a large earthquake for days to years) can be a source of confounding signals. Such a burst of aftershock signals can overload the human analysts of the monitoring system. To reduce the load, at the onset of a sequence of events (e.g., aftershocks), a human analyst can label the first few of these events and start an online classifier to filter out subsequent aftershock events. We propose an online few-shot classification model FewSig for time series data for the above use case. The framework of FewSig consists of a selective model to identify the high-confidence positive events that are used for updating the models and a general classifier to label the remaining events. Our specific technique uses a two-level decision tree as the selective model and a general classifier model based on distance metric learning with Neighborhood Component Analysis (NCA). The algorithm demonstrates surprising robustness when tested on univariate datasets from the UEA/UCR archive. Furthermore, we show two real-world earthquake events where the proposed algorithm reduces the human effort in monitoring applications by filtering out the aftershock events.
1.4 BitLink: Temporal Linkage of Address Clusters in Bitcoin Blockchain

In the Bitcoin blockchain, addresses that share a transaction as common spenders can potentially belong to the same entity (e.g., a gambling service) and are termed as an address cluster. However, links (i.e., trust relationship) between disjoint address clusters of common spenders can be established when one address cluster is abandoned entirely and another cluster is generated soon after. We propose exploiting these synchronous actions of address clusters to link the clusters across time.

We train a deep neural network model with the temporal signatures generated from unlabeled data in a self-supervised manner. We utilize this model to test whether two clusters show synchronous temporal signatures that belong to the same entity. We apply the model to 26 wallets collected from WalletExplorer.com. We demonstrate three successful cases of linking address clusters of three major services (HelixMixer, Primedice, and Bitcoin Fog) to complete their transaction history until the current block. This enabled us to answer questions related to the revenue and expenditures of these services and create informative aggregate statistics.
Chapter 2

FilCorr: Filtered and Lagged Correlation on Streaming Time Series

2.1 Introduction

Large monitoring systems such as a network of seismic stations or forest fire detection systems typically have hundreds of distributed sensors gathering and transmitting real-time and streaming data. An event in these systems, like an earthquake or an abnormal higher temperature, creates dynamic responses at these sensors. The responses can be arbitrarily lagged because of sensors’ spatial separation and be limited to a specific band of frequencies depending on the type of event. For these systems, real-time event detection is essential for decision-making or alert generation. We demonstrate an algorithm to correlate streaming data generated from distributed sensors in real-time to detect events.

A concrete application where our algorithm can be employed is seismic monitoring. In this application, when an earthquake happens, seismometers (i.e., seismic sensors) across the earthquake region observe the wave at varying times for varying duration, while the signals recorded at these sensors are often correlated. For better understanding, in Figure 2.1, we show a magnitude 7.8 event in Gurkha, Nepal, on April 25, 2015. Three waveforms recorded at three stations (marked red in the map) show a high correlation when the lag due to propagation delay is considered. The seismic wave generated in Nepal reaches Japan in about eight minutes and Australia in about eleven minutes.

In addition, filtering is a mandatory operation in seismic signal processing to remove undesired noise from data for events. We illustrate the importance of filtering in
Figure 2.1: (left) A M7.8 earthquake in Gurkha, Nepal, on Apr-25, 2015. (right) The signals recorded at three different stations at different times due to the spatial separation of sensors. Data collected from IRIS [54].

Figure 5.7. The raw waveforms rarely demonstrate a correlation between events. In contrast, waveforms filtered between 0.4Hz to 3Hz achieve a higher correlation. For many applications, mainly those involving digital signal processing, certain properties of data are made explicit when the signal is represented in the frequency domain [111]. Thus, filtering is also essential to extract descriptive features for machine learning models.

Figure 2.2: The top row shows raw waveforms with no clear seismic signal. The bottom row shows filtered waveforms of the corresponding raw waveforms in the top row. Note the decrease in absolute value in the y-axis and the increased visibility of the signal. The filter band is 0.4Hz to 3Hz. The average lagged correlation of all three pairs of raw signals is zero. The average lagged correlation of all three pairs after filtering is 0.53.

In a monitoring system that consumes streaming data from hundreds of sensors, computing lagged correlation (or asynchronous correlation) can be challenging because of the fast data rate, a large number of stations, and the necessity for accurate correlation values. Although the problem has been studied for decades, none of the existing methods such as BRAID [104], COLR-tree [6], or StatStream [155], can monitor one hundred seismometers at a 10Hz rate on a single workstation, a usual setting for many systems.
The main reason for the failure of these methods is the data-dependent pruning, projection, or indexing technique. Seismic traces are mostly white noise (except when events happen), stressing the algorithms falling behind the streams. In contrast, none of these algorithms are amenable to requirements of data pre-processing, such as filtering. Hence, pre-processing must always be done before correlation computation, resulting in a loss of efficiency.

In this chapter, we propose an algorithm, **FilCorr** – Filtered Lagged Correlation, to merge filtering and lagged correlation computation to extract *data-independent* efficiency. Our algorithm efficiently maintains frequency information over the stream and calculates correlation in the frequency domain. The technique is *exact*, *devoid of sensitive parameters* and *easily parallelizable*. We experimentally show that our technique is suitable for monitoring applications where the lagged correlation of filtered time series is required, such as seismic event monitoring, motion monitoring, and neural activity monitoring. This chapter introduces a new real-time seismic events monitoring system (SeisViz) that exploits the FilCorr algorithm to monitor high-speed data streams. Our implementation can monitor one hundred seismic stations at a 10Hz rate without any delay. A demonstration of Seisviz capturing past earthquakes is available on our website [149], and the system for real-time monitoring is available at [www.seisviz.com](http://www.seisviz.com). This extended version explains algorithmic details to support its implementation using different programming languages. We perform new experiments concerning the parameter sensitivity of FilCorr and discuss their parallelization on multiple processing units and how FilCorr can work with other types of filters. Additionally, we introduce two new case studies on neuroimaging and motion-capture data, demonstrating the applicability of FilCorr.

The main contributions of this chapter are summarized below:

- We demonstrate a working system to cross-correlate hundreds of high-speed streams in real-time at 10Hz speed using a conventional workstation;
- We merge digital filters and correlation computation in one combined step to achieve time and space efficiency;
- We show three case studies in which such high-speed lagged correlation help to detect events of interest.

The rest of the chapter is organized as follows. In Section 2.2, we present the general definitions related to streaming time series and notation employed throughout the sections. In Section 2.3, we discuss related work. In Section 2.4, we introduce our proposed algorithm FilCorr. In Section 2.5, we discuss the performance of our proposed algorithm.
In Section 2.6, we present three case studies in which FilCorr is employed to compute the lagged correlation of filtered data. Finally, our conclusions and future works are presented in Section 2.7.

### Table 2.1: Symbols and definitions.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s^x$</td>
<td>Streaming time series with ID $x$</td>
</tr>
<tr>
<td>$N$</td>
<td>Total number of streams that a system is processing</td>
</tr>
<tr>
<td>$s^x[t]$</td>
<td>The observation value of stream $s^x$ at time $t$</td>
</tr>
<tr>
<td>$f^x$</td>
<td>Sampling rate of stream $s^x$</td>
</tr>
<tr>
<td>$f_s$</td>
<td>Lower bound frequency of the bandpass filter in Hz</td>
</tr>
<tr>
<td>$f_t$</td>
<td>Upper bound frequency of the bandpass filter in Hz</td>
</tr>
<tr>
<td>$m$</td>
<td>Length of basic windows in number of observations</td>
</tr>
<tr>
<td>$l$</td>
<td>Lag size in number of observations</td>
</tr>
<tr>
<td>$w_t^x$</td>
<td>The basic window in $s^x$ at time $t$, includes $m$ observations: ${s^x[t - m + 1] \ldots s^x[t]}$</td>
</tr>
<tr>
<td>$w_{t}^{x}$</td>
<td>The filtered basic window, filtered version of $w_t^x$ in the time domain</td>
</tr>
<tr>
<td>$W_t^x$</td>
<td>The frequency window, discrete Fourier transformed version of $w_t^x$</td>
</tr>
<tr>
<td>$W_{t</td>
<td>f_s}^x$</td>
</tr>
<tr>
<td>$s_t^x$</td>
<td>The sliding window in $s^x$ at time $t$, covers observations in range ${s^x[t - l - m + 1] \ldots s^x[t]}$ includes $(l + 1)$ basic windows: ${w_{t-l}^x \ldots w_{t}^x}$</td>
</tr>
<tr>
<td>$w_{t}^{x}[j]$</td>
<td>$j$th element of the basic window $w_t^x$, $w_{t}^{x}[j] = s^x[t - m + 1 + j]$</td>
</tr>
<tr>
<td>$lcorr_{xy}$</td>
<td>Lagged correlation value between $s_t^x$ and $s_t^y$ at time $t$</td>
</tr>
<tr>
<td>$corr_{x,y}$</td>
<td>Correlation value of $w_{t}^{x}$ and $w_{t}^{y}$</td>
</tr>
<tr>
<td>step</td>
<td>Gap between two successive correlation computation in number of observations. If step = 10 and the first output is $lcorr_{11}^{x,y}$ then the next output is $lcorr_{11}^{x,y}$</td>
</tr>
</tbody>
</table>

### 2.2 Background and Notation

We define time series as a sequence of observations, and they are in the form of real numbers measuring a quantity at a fixed sampling rate. A streaming time series is an unbounded sequence of observations generated at a fixed rate. Table 2.1 shows the symbols and definitions considered in our discussions. The relationships of main variables are also illustrated in Figure 2.3. We use $s^x$ to represent a particular streaming time series $x$. We define the basic window as a continuous segment of a time series using $w^x$. A basic window of size $m$ from time series $s^x$ at time $t$ is denoted by $w_t^x$ which contains observations from...
Figure 2.3: Examples of various windows on a stream, where \( m = 4 \), \( l = 3 \), and \( \text{step} = 2 \).

\[ s^x[t - m + 1] \] to \( s^x[t] \). We can extract at most \( n - m + 1 \) basic windows of length \( m \) from a long time series of length \( n >> m \). Adjacent basic windows are not independent of each other, overlapping basic windows are trivially close in the high dimensional space.

**Filtering:** An essential pre-processing task for time series required by many applications. Filters are most commonly used to remove undesirable components from a time series. There are many types of filters useful in various domains. All filters have response functions convolved with a signal to apply the filter. Such response functions contain relative weight for each frequency. In an analytical form, the weights are non-zero for all frequencies. However, many weights are significantly smaller than the rest in practice, allowing us to cut off and keep only the necessary frequencies. Most applications employ a band that focuses on the need of the application. For example, seismic monitoring uses up to 10Hz [107], and EEG monitoring uses a gamma activity band between 30Hz-50Hz [21].

In this chapter, we assume that the given filter band is a contiguous set of frequencies, and all frequencies are equally important. This is also known as a box filter or ideal bandpass filter. Our proposed method is extendable to a non-contiguous set of frequencies with varying weights, i.e., other types of filters. We define the frequency window \( W^x_t \) as a window that contains all the frequency components of the basic window \( w^x_t \), and each \( W^x_t[k] \) where \( k = 0, 1, 2, \ldots, m - 1 \) are defined in Equation 2.1. Note \( i = \sqrt{-1} \), \( W^x_t[k] \) is a complex number.

\[
W^x_t[k] = \sum_{j=0}^{m-1} w^x_t[j]e^{-\frac{i2\pi}{m}kj} \tag{2.1}
\]

We use \( f^x \) in Hz to represent the sampling rate of the streaming time series \( s^x \) and the filtered basic window \( w^x_t' \) to denote the filtered version of \( w^x_t \). If we apply a box filter with a band from \( f_s \) to \( f_t \) in Hz \((0 < f_s \leq f_t)\), then \( w^x_t' \) can be derived from Equation 2.2.

\[
w^x_t'[j] = \frac{1}{m} \left( \sum_{k=\lfloor \frac{j}{m} \rfloor}^{\lfloor \frac{j}{m} + \frac{m}{m} \rfloor} W^x_t[k]e^{\frac{i2\pi}{m}kj} + \sum_{k=\lfloor \frac{m-j}{m} \rfloor}^{\lfloor \frac{m-j}{m} + \frac{m}{m} \rfloor} W^x_t[k]e^{\frac{i2\pi}{m}kj} \right) \tag{2.2}
\]
Figure 2.4 shows a running example of how to derive the \( W_t^x f_t \) and \( w_t^x \). Note that there is a corner case when \( m \) is even and \( f_t = \frac{f_s}{2} \), then \( w_t^x[j] \) is equals to \( \frac{1}{m} \left( \sum_{k=\lceil m f_s / f_t \rceil}^{\lfloor m f_s / f_t \rfloor} W_t^x[k] e^{i \frac{2 \pi k j}{m}} \right) \). The Fourier transform of \( w_t^x \) will only contains non-zero components that corresponding to frequency from \( f_s \) to \( f_t \) in \( W_t^x \). If \( f_s = 0 \) and \( f_t = \frac{f_s}{2} \) (Nyquist frequency), then \( w_t^x \) is identical to the basic window \( w_t^x \).

![Figure 2.4: Running example of how to derive the filtered frequency window and filtered basic window](image)

**Correlation computation in the frequency domain:** If we are given two basic windows \( w_t^x \) and \( w_t^y \), Pearson’s correlation coefficient between them is defined in the time domain as Equation (2.3):

\[
\text{corr}_{t_1,t_2}^{x,y} = \frac{\sum_{j=0}^{m-1} w_t^x[j] w_t^y[j] - m \mu(w_t^x) \mu(w_t^y)}{m \sigma(w_t^x) \sigma(w_t^y)} \tag{2.3}
\]

To compute the correlation in the frequency domain, we can exploit Parseval’s theorem [94, 92], which is expressed as Equation (2.4) for the discrete Fourier transformation:

\[
\sum_{j=0}^{m-1} |w_t^x[j]|^2 = \frac{1}{m} \sum_{k=0}^{m-1} |W_t^x[k]|^2 \tag{2.4}
\]

Based on Equation (2.4), we can compute Equation (2.3) in the frequency domain: \( \mu(w_t^x) \) equals \( \frac{W_t^x[0]}{m} \) since \( W_t^x[0] = \sum w_t^x[j] \). \( \sigma(w_t^x) \) equals \( \sqrt{\frac{\sum w_t^x[j]^2}{m} - \left[ \mu(w_t^x) \right]^2} \) in the time domain and the \( \sum w_t^x[j]^2 \) term can be computed in the frequency domain by directly applying Equation (2.4). Since the discrete Fourier transform is a linear operation, Parseval’s theorem can also be expressed as Equation (2.5) with two signals.

\[
\sum_{j=0}^{m-1} |w_t^x[j] - w_t^y[j]|^2 = \frac{1}{m} \sum_{k=0}^{m-1} |W_t^x[k] - W_t^y[k]|^2 \tag{2.5}
\]

Based on Equation (2.5), we can get Equation (2.6) to compute \( \sum_{j=0}^{m-1} w_t^x[j] w_t^y[j] \). Thus, we show how to compute each term of Equation (2.3) in the frequency domain.
\[
\frac{1}{2m} \left( \sum_{k=0}^{m-1} |W_x[k]|^2 + \sum_{k=0}^{m-1} |W_y[k]|^2 - \sum_{k=0}^{m-1} |W_x[k] - W_y[k]|^2 \right)
\] (2.6)

**Intuition:** Computing correlation values in the frequency domain is more efficient when we have a bandpass filter. Based on Equation 2.2, the computation of correlation in the frequency domain only takes time \(O(B)\), where \(B\) is the bandwidth in a number of frequency components.

**Problem Statement:** Given \(N\) streaming time series, a frequency band \((f_s, f_t)\), and a maximum allowable lag \(l\), compute the Pearson’s correlation coefficients for all pairs of streaming time series over a sliding window up to the given lag \(l\).

We argue in this chapter, with the empirical case study, this problem is very practical. In most domains, filtering can get rid of unwanted signals to compute correlation on right signals; and a reasonable maximum lag always exists, beyond which no correlation coefficient is meaningful.

### 2.3 Related Work

Lagged correlation is a problem that has been studied for decades [155, 104, 6]. However, none of the existing methods consider a set of features required by modern applications that monitor hundred of sensors in real-time. Based on these requirements and FilCorr properties, we categorize the related works to our dissertation according to four groups, as follows.

**Lagged Correlation Computation:** Computing lagged correlation, or cross-correlation, on offline data is a fundamental operation that is benefited from Fast Fourier Transform. Researchers have proposed an online algorithm for lagged correlation computation named BRAID [104]. However, the method samples frequency coefficients in a logarithmic manner to approximate the correlation value. Moreover, the method computes correlations over the entire stream, unlike our method that computes over a sliding window in real-time. For lagged correlation computation, one may consider offline indexing methods such as iSAX [109] and COLR-tree [6]. However, such offline indexes suffer from many modification (insert or delete) operations over correlation computation.

**Real-time Correlation Computation:** Efficient all-pair correlation computation for streaming data is no longer an active research problem. StatStream [155] is a technique that exploits a few Fourier coefficients to prune improbable pairs quickly. Similarly, ParCorr [137] performs random projections to prune improbable pairs without redundancy due to the sliding window. Hardware-based techniques are often used for computation
at MHz to GHz rate \[50\]. However, it is necessary to note that none of these methods consider lagged correlation after filtering.

**Frequency Domain Correlation Computation:** Frequency domain features are widely employed for various tasks; however, we consider our work a pre-process engine for downstream machine learning tasks, unlike many related works that use frequency domain features in machine learning tasks. For instance, Random Interval Spectral Ensemble (RISE) \[14\] uses spectral features for classification tasks, and various techniques from \[132\] utilize frequency features for deep learning. Many algorithms also exploit frequency as a computation space for dimension reduction purposes, such as the techniques proposed by \[85\] \[107\] calculate the correlations in the frequency domain. However, most of the works consider the offline nature of computation.

**Filtered Correlation Computation:** In this category, our method is unique. To the best of our knowledge, no work exploits filtering operation to extract efficiency in correlation computation. It is somewhat surprising considering the widespread filtering usage for processing real-time data captures. The novelty in our technique is that the speedup is not data-dependent, unlike any of the aforementioned work.

In Table 2.2, we present a comparison of the main capabilities of FilCorr concerning state-of-the-art algorithms for cross-correlation computation over multiple streaming time series. Some capabilities are not demonstrated but are trivially achievable with simple extensions of the algorithms. FilCorr comprehensively covers capabilities across several existing works, making it unique in the suite of correlation computing algorithms.

Table 2.2: Capabilities of FilCorr and related work. The symbol ✓ represents a claimed capability; – represents extendable capability, and × represents unknown.

<table>
<thead>
<tr>
<th></th>
<th>FilCorr</th>
<th>ParCorr [137]</th>
<th>BRAID [104]</th>
<th>COLR-Tree [6]</th>
<th>StatStream [155]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data-independent</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Filtering</td>
<td>✓</td>
<td>–</td>
<td>×</td>
<td>×</td>
<td>–</td>
</tr>
<tr>
<td>Lagged correlation</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Real-time</td>
<td>✓</td>
<td>✓</td>
<td>–</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Exact</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>–</td>
<td>✓</td>
</tr>
<tr>
<td>Parallel</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

### 2.4 FilCorr: Filtered Lagged Correlation

The primary motivation of FilCorr is the need for systems to compute lagged correlation of filtered high frequency streaming time series from distributed sensors, such as seismic
event monitoring. These systems require monitoring hundreds of sensors responsible for generating data at high speed. Surprisingly, none of the existing methods from the literature can deal with all of these requirements.

In this section, we present our solution in detail. For simplicity but without loss of generality, we assume all streams have the same sampling rate $f$, no discontinuity (no data loss during the transmission) and the observations are all aligned, which means they all have timestamps in set $\{1, 2, 3..., t - 2, t - 1, t, \ldots\}$.

### 2.4.1 Lagged Correlation

We use $lcorr_{xy}^t$ to denote the lagged correlation value at time $t$ between $s^x$ and $s^y$. It is defined in Equation 2.7.

$$lcorr_{xy}^t = \text{Max}(\text{corr}_{xy}^{t_i}, \text{corr}_{xy}^{t_i_t}); t_i \in [t - l, t] \quad (2.7)$$

$lcorr_{xy}^t$ is the largest Pearson’s correlation coefficient value among the correlations between the most recent basic windows at time $t$ and all the previous basic windows from time $t - l$ to $t$. Such a strategy can cover all possible cases when an event appears in different streams at different time.

**Case 1:** No lag. The event appears in both streams at the same time. $lcorr_{xy}^t$ equals to $corr_{xy}^{t_i}$ which is the correlation value between basic window $w_i^x$ and $w_i^y$.

**Case 2:** The event appears in $s^y$ first then in $s^x$ at time $t$, such scenario will be captured by computing correlation between $w_i^x$ and $w_i^y$. $t_i \in [t - l, t - 1]$.

**Case 3:** Similar to case 2, the event appears in $s^x$ first then in $s^y$ at time $t$. Then $lcorr_{xy}^t$ will be located among correlations between $w_i^y$ and $w_i^x$. $t_i \in [t - l, t - 1]$.

To combine all cases, we define the sliding window $s_i^x$ and $s_i^y$. Each sliding window will cover all $(l + 1)$ basic windows that are necessary to compute $lcorr_{xy}^t$. The relations are shown in Figure 2.3.

To represent this process with code, we have line 3 to line 10 in Algorithm 1 and line 5 to line 11 in Algorithm 2. It takes the last basic window from one sliding window and computes correlation with all basic windows in another sliding window and vice versa. It returns the largest value as the value of the lagged correlation between the two sliding windows.
2.4.2 Lagged Correlation with Filtering

The following discussions are based on the ideal bandpass filter, which has frequency response values equal to 1 for frequencies in the band, and 0 for frequencies outside the band. For any other known frequency response, we can apply FilCorr trivially. Before diving into the details of algorithms, Fig. 2.5 illustrates the high-level procedures to compute the correlation between two filtered basic windows for both Naive and FilCorr algorithms.

**Naive approach:** A straightforward approach to filter time series and compute lagged correlation. For the filtering, the naive approach will transform each basic window to the frequency domain, filter out frequency components outside the filter band before transforming back to the time domain. This process shows in filter function with lower cutoff frequency as $f_s$ and higher cutoff frequency as $f_t$ in Algorithm 1.

We define the *filtered basic window* $w'_x$, as the filtered version of $w_x$. $w'_x$ and $w'_y$ have the same length. Unlike two successive basic windows, $w'_x$ and $w'_{x+1}$ are independent and do not share the same overlapping values.

Once all the filtered windows in $s'_x$ and $s'_y$ are calculated, we can compute $lcorr_{xy}^t$ by calling the function $LagFilterCorr$ in Algorithm 1 with parameters $(s'_x, s'_y, l, m, f, f_s, f_t)$. The $oneCorr$ function is based on Equation 2.8 derived from Equation 2.3 by substituting $w'_x$ for $w_x$. 

---

Figure 2.5: Comparison between Naive algorithm (left) and FilCorr (right) for computing a correlation between two basic windows. The keep non-zero operation that bridges Naive and FilCorr only needs to perform once to initiate the FilCorr, and then it can update the filtered frequency window from the previous one.
\[
\text{corr}_{t_1, t_2}^{x,y} = \frac{\sum_{j=0}^{m-1} w_{t_1}^x[j] w_{t_2}^y[j] - m \mu(w_{t_1}^x) \mu(w_{t_2}^y)}{m \sigma(w_{t_1}^x) \sigma(w_{t_2}^y)}
\]  

(2.8)

We can avoid some redundant computation by storing sums, means and standard deviations of all filtered windows which can be computed from Equation 2.9, 2.10, 2.11. Thus, only \(\sum_{j=0}^{m-1} w_{t_1}^x[j] w_{t_2}^y[j]\) needs to be computed for each pair of filtered windows. In our implementation, only one copy of a filtered window \(w_t^x\) and its statistics are stored.

\[
\sum w_t^x = \sum_{j=0}^{m-1} w_t^x[j]
\]  

(2.9)

\[
\mu(w_t^x) = \frac{\sum w_t^x}{m}
\]  

(2.10)

\[
\sigma(w_t^x) = \sqrt{\frac{\sum w_t^x[j]^2}{m} - [\mu(w_t^x)]^2}
\]  

(2.11)

**Proposed approach:** We propose combining the filtering and correlation computation in one step to speedup the computation of lagged correlation on filtered time series. Such an approach can improve not only time efficiency but also space efficiency.

We define the frequency window \(W_t^x\), which is the Fourier transformed version of basic window \(w_t^x\). \(W_t^x|f_s\) as the filtered frequency window which only contains half coefficients corresponding to frequencies from \(f_s\) to \(f_t\) in \(W_t^x\). \(W_t^x|f_s\) will only keep one half of elements to save memory space since elements in \(W_t^x\) are complex conjugate symmetric. The main idea of our approach is to incrementally updates filtered frequency window \(W_t^x|f_s\) from the previous filtered frequency window \(W_{t-1}^x|f_s\) instead of calculating from basic window \(w_t^x\). If all filtered frequency windows in a pair of sliding windows are computed, then the lagged correlation coefficients can be directly computed using the frequency components based on Equations 2.4 and 2.6. Thus, our proposed method avoids computing repeated Fourier transforms and inverse Fourier transforms on every basic windows and perform correlation computation directly in the frequency domain on fewer data. The length of \(W_t^x|f_s\) is \(\left\lfloor \frac{f_t-f_s}{f_m} m \right\rfloor + 1\), where \(f\) is the sampling rate of the streams.

To maintain frequency components upon receiving a new observation, the algorithm removes the quantities for each frequency that contributed by the first observation \(w_{t-1}^x[0]\) of basic window \(w_{t-1}^x\) and adds the quantities for each frequency that brought by the new observation \(w_t^x[m-1]\) of basic window \(w_t^x\). To account for the slide of the window, the algorithm updates \(k^{th}\) coefficient by multiplying \(e^{i \frac{2\pi k}{m}}\), for \(\left\lfloor \frac{f}{f_m} m \right\rfloor \leq k \leq \left\lfloor \frac{f^2}{f_m} m \right\rfloor\). This process is applied to each of the frequencies from \(f_s\) to \(f_t\). The steps are precisely
Algorithm 1: Naive

1 Function LagFilterCorr(s^x_t, s^y_t, l, m, f, f_s, f_t)
2     \textbf{for} \ t_i = t - l : t - 1 \textbf{do}
3         w^x_{t_i} \leftarrow \text{filter}(w^x_t, f, f_s, f_t)
4         w^y_{t_i} \leftarrow \text{filter}(w^y_t, f, f_s, f_t)
5     \textbf{end for}
6     curMax \leftarrow \text{oneCorr}(w^x_{t_i}, w^y_{t_i}, m) \ // \text{case 1}
7     \textbf{for} \ t_i = t - l : t - 1 \textbf{do}
8         w^x_{t_i} \leftarrow \text{filter}(w^x_t, f, f_s, f_t)
9         w^y_{t_i} \leftarrow \text{filter}(w^y_t, f, f_s, f_t)
10        \text{tmp1} \leftarrow \text{oneCorr}(w^x_{t_i}, w^y_{t_i}, m) \ // \text{case 2}
11        \text{tmp2} \leftarrow \text{oneCorr}(w^x_{t_i}, w^y_{t_i}, m) \ // \text{case 3}
12     \textbf{end for}
13     curMax = \max(curMax, \text{tmp1, tmp2})
14 \textbf{return} curMax

Function filter(w^x_t, f, f_s, f_t)

16     LB = \lfloor mf_s/f \rfloor
17     UB = \lfloor mf_t/f \rfloor
18     W^x_t \leftarrow \text{FFT}(w^x_t)
19     W^x_t[0 : LB - 1] \leftarrow 0
20     W^x_t[UB + 1 : m - UB - 1] \leftarrow 0
21     W^x_t[m - LB + 1: m - 1] \leftarrow 0
22 \textbf{return} \text{IFFT}(W^x_t)

Function oneCorr(w^x_{t_i}, w^y_{t_i}, m)

24     \textbf{return} \sum_j \mu(w^x_{t_i}[j]) \mu(w^y_{t_i}[j]) / \sigma(w^x_{t_i}) \sigma(w^y_{t_i})

represented by \textit{filter} function in Algorithm 2.

Once filtered frequency windows \{W^x_{t-1|f_s}, \ldots, W^x_{t|f_t}\} and \{W^y_{t-1|f_s}, \ldots, W^y_{t|f_t}\} are ready, then the lagged correlation \textit{lcorr}^{xy}_t can be computed by calling Algorithm 2. The function \textit{oneCorr} in Algorithm 2 will be called to compute each \textit{corr}^{xy}_{t_i,t_2}. Note that the correlation coefficients calculated from filtered frequency windows are \textit{exactly} the same as the coefficients calculated from filtered windows in the naive algorithm. The exactness of our algorithm is directly derived from Parseval’s theorem described earlier in Section 2.2.

To explain the function \textit{oneCorr} in Algorithm 2, we describe how Equation 2.8 is evaluated using a filtered frequency window instead of a filtered basic window. In the following, we show how each term from Equation 2.8 can be expressed using frequency terms in detail.

Based on the Equation 2.10, we can calculate the mean of filtered basic window with Equation 2.12.
\[ \mu(w_f^x) = \begin{cases} \frac{W_f^x[0]}{m} & \text{if } f_s = 0 \\ 0 & \text{if } f_s > 0 \text{ (DC term is filtered)} \end{cases} \quad (2.12) \]

Based on Parseval’s theorem in Equation 2.4, we can compute the \( \sum w_f^x[j]^2 \) in Equation 2.13. We multiply a constant 2 to include the symmetric part of \( W_f^x[f_s] \). This also applies to Equation 2.14, 2.15. There are two special cases, one case is when the \( f_s = 0 \) and another case is when the length of a basic window is an even number and the \( f_t \) equals to the Nyquist frequency which is one half of the sampling rate \( f \). For both cases, \( W_f^x[\lceil \frac{m}{2} \rceil] \) or \( W_f^x[0] \) need to be subtracted after we multiply 2 since there is no symmetric value to them. Both values need to be subtracted if conditions for both cases hold.

\[ \sum_{j=0}^{m-1} w_f^x[j]^2 = \frac{2}{m} \sum |W_f^x[f_t][k]|^2 \quad (2.13) \]

Then the standard deviations can be calculated as below if \( f_s > 0 \).

\[ \sigma(w_f^x) = \sqrt{\frac{2}{m^2} \sum |W_f^x[f_t][k]|^2} \quad (2.14) \]

Lastly, the product term \( \sum (w_f^x[j]w_f^y[j]) \) can be expressed in Equation 2.15 based on Equation 2.6.

\[ \frac{2}{2m} \left( \sum |W_f^x[f_t][k]|^2 + \sum |W_f^y[f_t][k]|^2 - \sum |W_f^x[f_t][k] - W_f^y[f_t][k]|^2 \right) \quad (2.15) \]

Finally, we can derive the equation that appears in line 16 of Algorithm 2 when \( f_s > 0 \). For the case when \( f_s = 0 \), it only needs to include the \( \mu(w_f^x) \) term in the computations since it is no longer equals zero.

In order to control the output rate, we use the parameter \( \text{step} \). FilCorr can output all-pair correlations upon receiving the next set of observations in the streams. However, the output rate does not necessarily have to be the same as the input rate. If the current sliding window is \( s_f^x \), we can slide to \( s_f^x + \text{step} \), where \( \text{step} \) is the number of observations in a stream the algorithm gathers before outputting the next set of pairwise correlations. When \( \text{step} = 1 \), the algorithm outputs at the same rate as the input. The larger the \( \text{step} \), the slower the output rate.
Algorithm 2: FilCorr

1 Function LagFilterCorr($s^x_l$, $s^y_l$, $l$, $m$, $f$, $f_s$, $f_t$)
2 \[ LB \leftarrow m[f_s/f] \]
3 \[ UB \leftarrow m[f_t/f] \]
4 \[ W^x_{t|f_s} \leftarrow \text{filter}(W^x_{t-1|f_s}, m, LB, UB, w^x_{t-1}[0], w^x_{t}[m-1]) \]
5 \[ W^y_{t|f_s} \leftarrow \text{filter}(W^y_{t-1|f_s}, m, LB, UB, w^y_{t-1}[0], w^y_{t}[m-1]) \]
6 \[ \text{curMax} \leftarrow \text{oneCorr}(W^x_{t|f_s}, W^y_{t|f_s}, m) \quad \text{// case 1} \]
7 \[ \text{for } t_i = t - l : t - 1 \text{ do} \]
8 \[ W^x_{t_i|f_s} \leftarrow \text{filter}(W^x_{t_i-1|f_s}, m, LB, UB, w^x_{t_i-1}[0], w^x_{t_i}[m-1]) \]
9 \[ W^y_{t_i|f_s} \leftarrow \text{filter}(W^y_{t_i-1|f_s}, m, LB, UB, w^y_{t_i-1}[0], w^y_{t_i}[m-1]) \]
10 \[ \text{tmp1} \leftarrow \text{oneCorr}(W^x_{t_i|f_s}, W^y_{t_i|f_s}, m) \quad \text{// case 2} \]
11 \[ \text{tmp2} \leftarrow \text{oneCorr}(W^x_{t_i|f_s}, W^y_{t_i|f_s}, m) \quad \text{// case 3} \]
12 \[ \text{curMax} = \max(\text{curMax}, \text{tmp1}, \text{tmp2}) \]
13 end
14 return \text{curMax}
15 end
16
17 Function \text{filter}(W^x_{t|f_s}, m, LB, UB, d, a)
18 \quad //d is the element that will be deleted
19 \quad //a is the element that will be added
20 \quad \text{for } q \text{ from } 0 : UB - LB \text{ do} \]
21 \quad \[ k \leftarrow q + LB \quad //k is the index of W^x_t \]
22 \quad \[ W^x_{t+1|f_s}[q] = e^{\frac{2\pi k}{m}} (W^x_{t|f_s}[q] - d \cdot e^{-i\frac{2\pi mk}{m}} + a \cdot e^{-i\frac{2\pi mk}{m}}) \]
23 \quad end
24 return $W^x_{t+1|f_s}$
25 end
26
27 Function \text{oneCorr}(W^x_{t_1}, W^y_{t_2}, m)
28 \quad //Assume 0 < f_s < f_t < f/2
29 \quad return $\frac{\sum_q |W^x_{t_1}[q]|^2 + \sum_q |W^y_{t_2}[q]|^2 - \sum_q |W^x_{t_1}[q]| - W^y_{t_2}[q]|^2}{2\sqrt{\sum_q |W^x_{t_1}[q]|^2 \sum_q |W^y_{t_2}[q]|^2}}$
30 end
31

2.4.3 Computational Complexity

FilCorr algorithm contains two stages of computation, the initial stage and the streaming processing stage. In the initial stage, FilCorr utilizes the FFT algorithm to transform the beginning basic window to the frequency window, then keeping the components within the filter band to get the filtered frequency window. The total time complexity for this stage is $O(m \log m + B)$ in which the FFT algorithm contributes $O(m \log m)$ \cite{32, 36}, and filtering contributes $O(B)$, $m$ is the length of the basic window and $B$ is the number of elements within the filter band which is $1 + \lfloor \frac{m-f_s}{f_t} \rfloor$. The cost for this one-time operation can be amortized among the following streaming processing steps, thus yielding amortized
time complexity $O\left(\frac{m\log m + B}{\text{len}(\text{stream})}\right)$, this value is near zero in the long run. Therefore, we do not count this cost to the overall time complexity.

The time complexity for streaming processing of one pair time series at any step is composed of two parts: i) filtering and ii) correlation computation. For the filtering, FilCorr only takes $O(B)$ time based on the `filter` function in Algorithm 2. The worst case is $O(m)$ when $B$ is close to $m$. Filtering in the naive algorithm will take $O(m\log m)$ to transfer the frequency and time domain when applying FFT and IFFT algorithm. As for the cost of computing lagged correlation coefficients, the `oneCorr` function in both algorithms will be executed $l$ times. Each iteration of the naive algorithm will take $O(m)$ based on the equation in line 42 while the FilCorr only takes $O(B)$ time.

The combined time complexity depends on the number of pairs of time series in the system. We assume all permutations of $O(N)$ time series. Thus, the time complexity to compute one lagged correlation value for naive is $O(Nm\log m + lmN^2)$, and $O(BN + lBN^2)$ for FilCorr, $O(mN + lmN^2)$ in the worst case. In practice, the speedup is more because the number of possible lags is much less than the window size, and the frequency band is much smaller than the number of observations in the signal.

The naive algorithm’s space complexity is $O(Nlm)$ to maintain all the filtered basic windows in the most recent sliding window for all streams. The space complexity of FilCorr is $O(NlB)$.

### 2.4.4 Extensions to Our Implementation

Our proposal can be easily extended to execute in parallel and to employ different digital filters. In this section, we discuss the properties of FilCorr that provide such flexibilities.

**Parallelization:** Since lagged correlation computation for one pair of streams is independent of other pairs, we can utilize multiple processing units (i.e., thread, core, processor, etc.) to expand the capacity to calculate multiple pairs in parallel. Each pair will maintain their sliding windows for the streams $s^x$ and $s^y$. In order to achieve the best real-time performance, one filtered frequency window with its statistics can be accessed by all pairs to save more memory and avoid redundant computation.

**Different filters:** Our method can adapt to other types of digital filters; for instance, the Butterworth filter illustrated in Figure 2.6. We can directly apply this filter on top of the box filter by changing each frequency weight. In this way, we can still save time from computing unnecessary frequencies and keep the filter property within the bandwidth. This operation will not change the overall time complexity since it is a linear operation to factor the weights in all frequencies. Note that incrementally updating the
DFT coefficients is also a linear time operation. Our method is not limited to the digital frequency-domain filters. A time-domain digital or analog filter can also be applied for streams passing through as pre-processing then passing into our method for correlation computation on the targeted frequencies.

Figure 2.6: Butterworth second order bandpass filter 3-7Hz.

2.4.5 Discussion on Data Independence

The general idea of FilCorr is to exploit the use of digital filters to achieve faster pairwise correlation computation. Besides removing undesired noise, filters allow computing the correlation between time series in a reduced dimensional space provided by the frequency domain. It turns the time cost of our method data-independent and will be affected by neither the sparsity nor the similarity. This cost is related to the lag and the bandwidth assumed by the filter, typically much smaller than the original series. It is also important to note that unlike other methods that speedup the correlation computation by pruning improbable pairs and provide approximate results, our approach efficiently calculates all possible lagged pairs and provides exact results.

2.4.6 Managing Spurious Correlation

Filtered correlation can occasionally be spurious because of Ringing effect [43]. It occurs due to spectral leakage when the length of the basic window is mutually prime with the period of the signal. For instance, in Figure 2.7(a), we show two uncorrelated signals (correlation = 0.14) obtained from different seismometers. If we filter such signals by a box filter, we obtain a false high correlation of 0.70 in the resulting signals, as illustrated in Figure 2.7(d). This high correlation is mainly caused by similar oscillations at the edges of both signals. This effect can be addressed by “windowing” (i.e., multiplying with a window function) the time series before converting them to the frequency domain. The resulting signals after the process of windowing using a Hamming window (Figure 2.7(b)) are illustrated in Figure 2.7(c). Finally, the filtered signals after the windowing process
are illustrated in Figure 2.7(e). We note that the correlation of the original signals is reduced to 0.13, eliminating the previously observed false high correlation.

Figure 2.7: Example of the Ringing effect on two uncorrelated seismic signals and how the multiplication by a Hamming window can address the false high correlation between them.

Windowing on each basic window data will change the filter process since we could not maintain the filtered frequency window incrementally. Thus, we need to apply the filter function from the naive algorithm.

2.5 Experimental Evaluation

All our experiments are reproducible and the supplementary material such as code, data, and additional results are publicly available on our supporting website [149]. FilCorr is exact and deterministic, and the efficiency is not data-dependent.

2.5.1 Sanity Check

Before the experimental evaluation and comparisons with existing methods, we show a sanity check to demonstrate that our approach is fast enough to be employed in a real-time system that is capable of monitoring hundreds of sensors with a high sampling rate. To demonstrate this claim, we develop a system named Seisviz (www.seisviz.com) that will render the lagged correlation values computed by FilCorr in real-time. It has been used successfully for monitoring a seismic network at Yellowstone, WY, USA. This network has 30 stations, where each station has up to six channels. Each channel represents an individual stream of observations at 100Hz. We obtain 670 pairs if we pair the streams from different stations by the same channel type. We demonstrate videos of detected earthquakes in real-time by Seisviz on our website [149]. In Section 2.6.1 we return to the discussion about our findings on seismic data and a detailed system implementation description.
2.5.2 Setup

All experiments are performed on a desktop computer with an Intel i9-9900k (8 cores) processor, 32GB of memory, running a Linux operation system. As FilCorr is data-independent, we use synthetic data for various experiments. The performance on real-world data will be discussed in the case studies presented in Section 2.6.

We create two testing scenarios to evaluate the performance of naive and FilCorr algorithm: offline and online. In the offline scenario, all observations are available beforehand, so the system will use its full power to compute until it finishes computation on all data. For the online scenario, the observations are generated in a streaming fashion with the speed of sampling rate.

In the offline scenario, we measure the total execution time, including I/O operations. In the online scenario, we seek to find the maximum number of streams that the system can compute their pairwise lagged correlation without any delay. To measure this, we create an ideal environment where all the streams have an equal length and the same sampling rate specified by the parameter $f$. We consider the system capable of processing this number of streams if the finish time is ahead of the expected next computation time. There is no need to measure at each step because if the number of streams exceeds the system capability, then the extra time to finish computation will accumulate at each step and reflect at the finish time.

For both offline and online, we run ten trials to confirm an algorithm’s capability on a certain number of streams to account for random events in the operating system.

2.5.3 Efficiency

In Figure 2.8, we show the execution time of FilCorr and the naive algorithm in the offline scenario. We consider three filter bands (5Hz, 25Hz, and 50Hz), and five sampling rates between 100Hz to 300Hz with increments of 50Hz. The results testify to the time complexity discussed in Section 2.4.3. The naive algorithm’s execution time remains on the same level for the same sampling rate no matter the bandwidth size. This is because the number of observations used for correlation computation of the naive algorithm does not change along with the filter bandwidth in the time domain. On the contrary, the narrower the band the fewer frequency components for FilCorr to compute, so less execution time. Another conclusion we can draw from the figure is that the growth rate for FilCorr is much slower than the naive for a fixed bandwidth when the sampling rate is increasing. This is because the basic window length is calculated based on the time which is 10 seconds in here. For FilCorr, the number of frequency components in the
filter band remains the same since $B$ is defined as $1 + \lfloor \frac{L}{f} m \rfloor$, $m$ equals $10f$ so $B$ equals $1 + \lfloor 10(f_t - f_s) \rfloor$. The only extra cost for FilCorr is coming from the longer lag. However, for the naive, it is affected by both longer lag and more observations for computing the correlation; thus, it increases at a much higher rate than FilCorr.

In Figure 2.8 we show the number of pairs that each algorithm can process without delay in the online scenario. We vary the input sampling rate and output rate for both algorithms. In all experimental settings, FilCorr can process (up to $4 \times$) more sensors than the naive algorithm. The performance gap increases with higher input or output rate. For other filter bands, the general performance trends hold.

In Figure 2.9, we show the number of pairs that each algorithm can process without delay in the online scenario. We vary the input sampling rate and output rate for both algorithms. In all experimental settings, FilCorr can process (up to $4 \times$) more sensors than the naive algorithm. The performance gap increases with higher input or output rate. For other filter bands, the general performance trends hold.

Figure 2.8: Offline performance of FilCorr and naive algorithm.

Figure 2.9: Online performance of FilCorr and naive algorithm. The left figure shows the results varying the sampling rate and fixed values for the remaining parameters. The right figure shows the results under different output rates. In both figures, the total number of stream pairs are computed as all possible combinations of any two streams.

2.5.4 Comparison to Existing Method

Based on our previous comparison shown in Table 2.2, we argue that FilCorr is a comprehensive method for streaming correlation computation. However, although not an ideal match in capabilities, we identify ParCorr [137] as the most recent baseline with state-of-the-art performance. ParCorr calculates pairwise correlation in parallel with the Apache Spark system based on randomly projected sketches. Note that ParCorr does not
compute lagged correlation. The following experiments are conducted on the same setup as previous experiments.

In order to favor ParCorr’s implementation in our comparison, we perform all the experiments in the offline scenario. To offset the Spark system’s costs, we conduct another set of experiments as offsets. Each offset experiment will only process one time series with window size as 1, step size as 1, and the length of this time series depends on the actual corresponding experiment parameters. Our goal is to make sure the sliding window in both offset experiments and actual corresponding experiments move the same number of times. All the experiment results here are adjusted based on the results of the corresponding offset experiment.

Since the ParCorr is data-dependent, we use two sets of synthetic data with 2,000 observations in each targeted to emulate the best-case and worst-case scenarios for ParCorr. The cost of ParCorr depends on the number of pairs it can prune without computing the correlation coefficients. Our first synthetic dataset contains sequences of uniformly distributed random numbers, which is expected to contain only uncorrelated pairs. Thus, a random noise dataset is the best data for ParCorr where it can prune all possible pairs. To further boost ParCorr’s performance, we use a high correlation threshold (candThreshold) for better pruning. The purpose of this is to guarantee that no two series will lead to actual correlation computation for ParCorr. Besides, we also set the parameter candThreshold with a high value in ParCorr as double insurance. On the contrary, the second dataset is a sinusoid that is expected to have all possible pairs of streams to be highly correlated. In this case, ParCorr computes correlation for all possible pairs, failing to prune and demonstrating the worst-case performance. For FilCorr and Naive, the pairs are generated based on all possible combinations from all the streams.

We show the performance comparison in Figure 2.10. The light grey shaded area represents the range of performance by ParCorr. The worst-case performance (on sinusoid data) is illustrated by the superior grey line with solid circles, and the best-case (on random data) by ParCorr is illustrated by the inferior grey line with solid boxes. The actual performance of ParCorr on any other dataset should be in between the worst-case and best-case lines. Figure 2.10 (zoom-out) shows that the time spent by FilCorr for various lags is well inside the shaded area. To be fair to ParCorr, when we consider the synchronous correlation (lag = 0), FilCorr is more efficient than the best-case of ParCorr up to around 700 streams as shown in Figure 2.10 (zoom-in). Therefore, we recommend FilCorr on a single desktop computer when the number of streams is less than 700, instead of using a parallel system.

In the second experiment, we fix the total number of streams at 800 and vary the
Figure 2.10: Comparison with ParCorr fixing $step = 20$. The vertical red line shows the crossing point between ParCorr and FilCorr with lag = 0, and the vertical blue line shows the crossing point between ParCorr and naive algorithm with lag = 0.

$step$ from 5 to 20 observations, which correspond to the output rate of 20Hz to 5Hz, respectively. We note in the results shown in Figure 2.11 that the execution time for all methods increases when the $step$ is getting smaller to compute more correlation coefficients for a higher output rate. However, ParCorr increases at a higher rate compared to FilCorr. The zoom-in figure shows that FilCorr with lag = 0 has better performance than the best-case of ParCorr when the output rate is higher than 6Hz.

Figure 2.11: Comparison with ParCorr varying the $step$ from 5 to 20 observations. The vertical red line shows the crossing point between ParCorr and FilCorr with lag = 0; the arrow points the point where the output rate is 6Hz. The vertical blue line shows the crossing point between ParCorr and naive algorithm with lag = 0, and the arrow points to the output rate as 7Hz.
2.5.5 Parameter Sensitivity

In this section, we discuss the algorithms’ sensitivity to three design parameters: i) lag, ii) filter bandwidth, and iii) window size. We consider the online scenario and measure the performance between the naive algorithm and FilCorr. We fix the sampling rate at $f = 100\text{Hz}$ and $step = 10$ observations for all experiments. The results are presented in Figure 2.12.

![Figure 2.12: Results for the parameter sensitivity test considering the online scenario.](image)

Figure 2.12 shows that doubling either the window size or the lag size has similar effects, where the number of pairs the algorithm can handle shrinks to half for both FilCorr and naive. The performance of FilCorr approaches the naive when the bandwidth increases.

2.6 Case Studies

This section discusses different scenarios that can benefit from FilCorr. We present three case studies with data from diverse domains, such as seismic signals captured by seismometers, motion signals captured by accelerometers, and brain activity signals from neuroimaging.

Before we dive into case studies, we would like to make a general recommendation about the usage of FilCorr in different domains. In general, any application that needs pairwise Pearson’s correlation values for various purposes can utilize FilCorr. However, some cases can enjoy the full benefits from FilCorr while some are negligible. To simplify the decision-making of a domain user, we draw a flowchart in Figure 2.13.
the performance of FilCorr to the naive approach, we set different recommendation levels based on the need for filtering and execution speed.

![Flowchart](image)

Figure 2.13: Different recommendation levels on FilCorr based on the application requirements. Neutral means there are no apparent benefits (No loss either) when compared with time domain implementation. Approximation means results are derived from fewer frequency components.

### 2.6.1 Seismic Event Monitoring

We have deployed a system for monitoring seismic events in which FilCorr is one of the core components. We designed such a system, named Seisviz\(^1\), by following the separation of concerns design principle (SoC) \[93\]. The main components of Seisviz are illustrated in Figure 2.14.

![Diagram](image)

Figure 2.14: Main components of Seisviz, a real-time system for seismic event monitoring.

The system has four modules: i) data collector, ii) the Kafka cluster, iii) the FilCorr computing unit, and iv) the Seisviz web server. Such a modular design is useful for developing and maintaining the system, while robustness and scalability are improved because failures can easily be tracked to one of the modules, and scaling each module is

\(^1\)The system is publicly available at [www.seisviz.com](http://www.seisviz.com)
easier than scaling the whole system. There are two data pipelines in the system, one path originates at the seismic sensors and ends in the FilCorr computing unit. The other path originates at the FilCorr computing unit and ends in the Seisviz server. The seismic observations are fetched from the Incorporated Research Institutions for Seismology - IRIS data center to the Kafka cluster, then consumed by the FilCorr computing unit. The correlation values are computed by the FilCorr computing unit and saved in the Kafka cluster, and finally consumed by the Seisviz web server.

The Kafka cluster was used for temporary data storage and data distribution. Kafka has essential features that meet our requirements, such as processing streaming data in real-time, storing a certain amount of historical data in a durable way, and allow multiple consumptions by several applications and systems. Besides, it provides extra benefits, including fault tolerance and high availability, which can enhance the reliability and the scalability of our system.

A few key points are worth discussing when using Kafka for time series data. The first thing is the order of observations since it is natural to keep each observation in a timely order or index-based order for time series; however, no such order can be maintained if we use multiple Kafka topics with one partition or one topic with multiple partitions to store observations from a time series. In other words, there is no guarantee that the order for each observation arriving at the consumer is the same order when they are generated. So the downstream services need to restore the order. This is because Kafka can only guarantee the records from the same topic partition will arrive at the consumer in the same order as they were appended to the partition but not for the records across many partitions. This leads to another approach that uses a topic with only one partition to store observations from a time series. Such an approach can bring time efficiency to the downstream services as they no longer need to restore each observation order. However, this may cause a performance penalty when consuming a large number of records at a time compared with the approach using a topic with multiple partitions on several nodes in a cluster. We choose the latter approach for simplicity, and our system scale is not reaching that performance bottleneck.

Secondly, Kafka only supports millisecond precision for timestamp, generating losses when the period of a stream is less than a millisecond. To solve this, we first look at the structure of the Kafka record. Each record has three attributes: key, value, timestamp. The key attribute is free in our case, so we can combine timestamp and key attributes to store the time information of an observation; time components after millisecond can be saved in the key attribute. This approach can support the precision level up to nanoseconds, which is sufficient for virtually any streaming time series problem.
As previously stated, our system considers IRIS as a data source. IRIS provides a protocol called SeedLink for users to receive real-time data. The streams of time series from IRIS are sent out in the form of batches containing a certain number of observations. Although SeedLink is based on the TCP/IP protocol that guarantees packets transmission in order and without any loss, the packets’ order may not be consistent with observations’ time order. To address this, we develop a collector module to save the streams in the Kafka cluster and, most importantly, recover the original data streams in time order. This process will encounter three different scenarios: 1) There is a time gap between data batches; in other words, a segment of observations of a time range is missing. 2) Two batches are overlapping. 3) There is a time shift among observations; if the time of one observation is shifted, then the time duration between this observation and the previous observation is no longer consistent with the sampling rate. This time shift has to be smaller than the sampling period to distinguish this scenario from the first scenario. Once we have all the necessary observations, we can start computing the filtered lagged correlation values using the FilCorr algorithm.

The back-end server will read the computed correlation value from the Kafka cluster, group the results based on the timestamp, and then send the results in a streaming manner to the front end. The front-end website will render the correlation value on a colored line between two points on the map, as illustrated in Figure 2.15. The point represents the location of a station, which consists of various seismometers. The correlation value is depicted by the color and transparency of the line between two stations. Since there are usually multiple seismometers at one station, if there are multiple pairs of streams between two stations, then only the correlation with the highest value will be rendered at the moment.

Figure 2.15: Pairwise correlation among 29 stations at Yellowstone, WY over different times given an M6.5 earthquake occurred in Challis, Idaho, at 23:52:30 2020-03-31(UTC).
In Figure 2.15, we illustrate the propagation of a seismic event of magnitude 6.5 that occurred at Challis, Idaho, on March 03, 2020 and which was observed by our system about 300 miles away in the stations at Yellowstone. In this representation, each red location symbol represents a station, and a colored line represents the lagged correlation between two stations. Different colors and levels of alpha reflect the lagged correlation value. The correlation values from 0 to 0.5 to 0.9 are mapped from green to yellow to red. A black edge represents a correlation value greater than 0.9. We notice that the correlated edges are appearing between station pairs as the earthquake wave reaches them. Similarly, edges become uncorrelated when the wave has passed through.

In summary, our system is monitoring a seismic network with 30 stations in Yellowstone, Wyoming. There is a total of 98 streams, which can form a total of 670 pairs. Our system is computing the correlation of these 670 pairs at a 10Hz rate. Correlation coefficients can capture earthquake propagation through a region in real-time, which can easily be converted to a detector with the rule: “If more than Q% of pairs of stations are highly correlated (> 0.8), an earthquake is propagating.” The utility of such a detector is massive for early warning systems because seismic wave propagate at 8 km/s, which is much slower than electronic signals carrying the warning.

As we previously have shown in Figure 5.7, filtering is essential for seismic data in order to remove noise and obtain a representation in the frequency domain that better describes the signals. For this case study, we consider a 20 seconds window size, a box bandpass filter with a cutoff frequency of 3Hz and 7Hz, and a lag of 10 seconds.

Although the Seisviz website shows the results with a few minutes lag, we still claim our system is real-time since the delay occurs before the data arrive at our system. The delay is typically caused at the origin (i.e., seismometer) and during the transmission process. The delay time varies and is beyond our control. Seisviz waits initially to accumulate enough data to be able to calculate the first set of correlations. After that, the system processes (i.e., flows data through the modules and computes the correlations) at a faster rate than the rate of streams.

### 2.6.2 Motion Monitoring

In recent decades, the emergence of low-cost wearable devices made it easier to measure body parts’ motion by accelerometers and gyroscope sensors. One interesting use of motion data is in interactive systems that monitor the movement synchronization between different users in real-time. Synchronous motion is at the heart of many art forms,
including dance and music, sports such as swimming and gymnastics, and electronic games. The monitoring of a group of users is useful for physical activities, choreography design, precise movements evaluation, and other tasks.

In this case study, we demonstrate how our method monitors motions in real-time for dance sessions, where slight irregularities are expected between the participants. For evaluation, we consider the Dancestix data [97], which has four participants who performed a Lady Gaga’s song with the same choreography with 21 dance steps following the 581 beats of a metronome at 2Hz. As illustrated in Figure 2.16, each participant has four three-axis accelerometer sensors placed on the hip, wrist, arm, and leg. The data was recorded at 100Hz, totaling time series with 29,255 observations for each sensor in a given axis, while we filter the data between 1Hz and 3Hz.

![Figure 2.16: Motion sensor locations and examples of time series generated by each sensor in the three axes.](image)

Ideally, any pair of dancers should show a strong correlation, while a slight deviation would suggest quality degradation. Since dance motions are repetitive, the allowable lag in detecting correlation must not be more than the duration of a dance step. In Figure 2.17, we show the resulting pairwise correlation between overall body parts of dancers, considering a 2 seconds window (200 observations), a lag of 50 and 0 (no lag) observations, with a box filter (1-3Hz) and without filtering. The location of the dance steps is represented by the vertical lines identified by letters (A-T).

For all pairs in Figure 2.17, we note that when we consider the lagged correlation of filtered data, higher values are observable over time. The correlations are close to zero when we do not consider filter and lag, demonstrating the importance of such features for this problem. From these correlations, we can carry out real-time analysis to identify fatigue, off-rhythm dancers, etc. For example, we see that dancer 3 is significantly less correlated to all other participants while performing step M.
2.6.3 Electrophysiological Monitoring

Parkinson’s disease is the second-most common neurodegenerative disorder that affects 2-3% of the population older than 65 years of age [96]. As the disease progresses, people may have difficulty walking and talking. It is caused by a significant decrease in dopamine production, a neurotransmitter that helps us to automatically carry out voluntary body movements without the need for thinking about every single movement that our muscles make. In the absence of dopamine, particularly in a small brain region called the substantia nigra, the individual’s motor control is lost [63].

An essential tool for monitoring the disease’s progress and its effects is the electroencephalogram (EEG) [55]. An EEG records electrical activity produced by the brain via small noninvasive sensors attached to the scalp in different regions, offering a direct measure of neuronal activities. The study of the correlation between EEG signals from different parts of the brain has been an active research area in the last years [87] [18] [58]. In general, a high correlation between the signals from different electrodes indicates similar brain activity, and a low correlation indicates that the brain activity at the different measurement sites is relatively independent [16]. However, patients with conditions such as Parkinson’s related dementia and Alzheimer’s disease often exhibit different behavior and a slow oscillatory brain activity compared with healthy subjects [58]. In EEG data, it is expected a lag due to the time spent to transfer the signals from one brain region

Figure 2.17: Cross-correlation between pairs of dancers over all body parts considering the use and absence of filter and lag.
to another. Also, the use of filters is essential to identify the frequencies that compose the signal. In this direction, for better analysis, it is essential to consider the lagged correlation of filtered EEG data, as proposed in this chapter.

In this case study, we consider a dataset of 25 patients with Parkinson’s disease and an equal number of people in the control group [21]. Subjects performed a task of identifying novel sounds in a sequence of known sounds. EEG was recorded continuously by a 64-channel system with a sampling rate of 500Hz. Here we only consider the EEG responses to the novel sounds. We first average all the trials and participants in one group then calculate pairwise lagged correlation across the channels. The results are for the one second window, 60ms lag containing 30 observations, and a box filter with a band from 0Hz to 20Hz.

Figure 2.18: Lag-correlated pairs of channels present in one group and not present in the other.

In Figure 2.18 we only show the pairs that are correlated only with lag and not correlated without lag. The left figure shows the correlated pairs of channels with lagged correlation values greater than 0.5 in the patient group that are not correlated in the control group, and the right figure shows the lagged correlated pairs in the control group but not in the patient group. We observe the front-back lag-correlation in the patient group, while the control group does not show any correlation in the frontal lobe. We do not claim any neurological significance of this finding. However, the difference between patient and control groups suggests that filtered and lagged correlation can support EEG analysis as a diagnostic tool.

2.7 Conclusion

This chapter demonstrates an algorithm, FilCorr, to compute filtered and lagged correlation over streaming time series. FilCorr combines filtering and cross-correlation
computing operations in one step to obtain the lagged correlation between streaming time series efficiently. FilCorr is faster than the state-of-the-art ParCorr algorithm that computes correlation in parallel. We show three case studies where the algorithm achieves promising results towards greater societal impacts. We also provide a publicly available real-time system named Seisviz that employs FilCorr in its core mechanism for monitoring a seismometer network.
Chapter 3

MASS: Distance Profile of a Query over a Time Series

3.1 Introduction

Time series is a sequence of observations made in time order. Given a query time series, the similarities or distances of the query to all possible subsequences of a time series constitute a distance profile of the query. Computing distance profile is a fundamental task in time series data mining and has been utilized in many existing works [143][151][152]. For example, to compute the Matrix Profile of a time series, the STAMP algorithm [142] repeatedly computes the distance profiles of subsequences of the given series. However, even though a rising interest in profiling time series data [65] has been observed, the literature does not present a formal and comprehensive understanding of the algorithms for distance profiling of a query over a time series.

In this chapter, we define the time series distance profile under various operating requirements and provide detailed discussions on the algorithms, extensions, utility and use cases. We describe four algorithms to compute the distance profile under Euclidean distance. The algorithms are incrementally efficient and uniquely useful. We describe four different extensions of the distance profile: weighted Euclidean distance profile, un-normalized Euclidean distance profile, correlation profile and partial correlation profile for dual queries. We present faster algorithms for time series discord discovery and time series subsequence clustering exploiting distance profiles than the traditional search-and-prune algorithms. We finally show three novel use cases of distance profiles in the domains of seismology, power consumption, and robotics.
3.2 Related Work

The distance profile serves as a foundational element in time series data mining and machine learning. It measures the similarities between a query sequence and subsequences within a longer time series. This measurement is integral to a range of critical tasks, including motif discovery, anomaly detection, and classification. MASS significantly boosts the efficiency of distance profile computation without compromising the precision of the results. The subsequent sections will provide an overview of the wide-ranging applications of the MASS algorithm, illustrating its significance in diverse data mining and machine learning tasks.

Pattern recognition: A fundamental application of the MASS algorithm lies in its ability to identify specific patterns that are related to certain events or characteristics. An example of this is its use in recognizing the actions of electrical appliances within a household by computing distance profiles of unique power consumption patterns against smart meter measurements [133]. The authors claim that this approach boosted by MASS can be carried out in (near) real-time using edge computing. Additionally, MASS can also help with real-time defect detection during metal additive manufacturing [22].

Matrix profile calculation for motif discovery: When identifying motifs without the knowledge of the specific pattern to query, the matrix profile (MP) [142] becomes an invaluable tool. The construction of MP involves determining the distance profile for every possible subsequence within a time series. Here, the MASS algorithm is crucial, serving as a key component in efficient MP calculation. It is integrated into algorithms such as STAMP [142] and SCRIMP++ [153], where it significantly enhances their performance. Although not directly incorporated in STOMP [154], the optimization methods used for calculating dot products in MASS versions 3 and 4 are still influential and relevant.

Domain-specific variations of matrix profile: Beyond the standard matrix profile, the MASS algorithm facilitates the development of specialized profiles tailored to specific fields of study. Notable examples include: similarity matrix profile (SiMPle) for cover song identification [114, 113], this variant leverages MASS to analyze and compare musical pieces, demonstrating its effectiveness in the field of musicology. In-phase matrix profile applied in EEG data analysis [126]. Radius profile employed for identifying repeating subsequences useful in various analytical, MASS aids in recognizing patterns that recur within time series. Analog ensemble profile used in meteorological analysis [139], MASS contributes to the creation of analog ensembles, enhancing the accuracy and depth of weather-related predictions and studies.
Classification and clustering: Many time series classification and clustering methods that rely on the distance profile can benefit from MASS. For instance, Abdoli et al. leveraged MASS to classify chicken behaviors [3]. They utilized a labeled subsequence representing a specific behavior as a query. MASS computed the distance profile against a streaming time series of chicken movements within a certain period. Subsequences similar to the query were classified accordingly. Heo et al. demonstrated the application of MASS in music. They used the algorithm to calculate the distance profile for individual songs in a test set against a composite time series created by concatenating all songs from a training set [49]. Lin et al. developed improved embeddings for classification tasks based on the distance profiles computed by MASS, showcasing its potential in refining data representation for better classification accuracy [75]. Emerging research continues to explore the capabilities of MASS in handling vast volumes of time series data for classification, One study [75] demonstrated MASS enables classification on time series with billions of samples, underscoring its scalability and efficiency.

Prediction: A key use of MASS in prediction involves comparing recent observations with historical data to identify similar past scenarios and use them to forecast near-future states. This approach is particularly effective when a substantial amount of historical data is available, which inherently demands more computational power. MASS demonstrates its significant utility in this process, for instance, MASS was employed to compare current weather data against a dataset containing the past 20 years’ historical observations for solar power prediction [139]. Further research indicates that MASS can enhance resolution by up to 400%, transforming hourly forecasts into more precise 15-minute intervals [140] which is crucial for applications requiring high-resolution data and timely decision-making. The accuracy can be further improved by incorporating data from various sources, such as a sensor network. MASS’s adaptability to different data dimensions makes it well-suited for analyzing complex, multi-source datasets [138, 35].

Anomaly detection: While prediction with MASS is about forecasting future states based on historical data, its use in anomaly detection serves a different purpose. In this context, the MASS algorithm is adept at identifying patterns that are unprecedented or deviate from the norm. In environments where data is continually updated and immediate response is required, MASS’s ability to quickly and accurately identify anomalies is invaluable. MASS also excels in analyzing vast datasets where manual detection of anomalies would be impractical or impossible. Referenced studies [76, 8, 62, 71, 22] highlight the above scenarios where MASS has been successfully applied in anomaly detection, demonstrating its versatility and effectiveness in this area.
3.3 Background and Notation

In this section, we define the necessary terms and concepts to describe the algorithms in Section 3.

**Definition 1 (time series):** A time series \( T \) of length \( n \) is an ordered sequence of real numbers \( T[i] \) measured in equally spaced time, in which \( T = (T[1], T[2], \ldots, T[n]) \).

**Definition 2 (subsequence):** A subsequence \( T_{i,L} \) is a continuous segment of length \( L \) from a time series \( T \) starting from position \( i \). \( T_{i,L} = (T[i], T[i+1], \ldots, T[i+L-1]) \), where \( 1 \leq i \leq n - L + 1 \). For a time series of length \( n \), there can be a total of \( \frac{n(n+1)}{2} \) subsequences of all possible lengths.

**Definition 3 (query):** A time series \( Q \) of length \( m \), which is searched within a time series \( T \) of length \( n >> m \).

**Definition 4 (distance profile):** Given a time series \( T \) and a query \( Q \) the distance profile is another sequence \( D \) of length \( n - m + 1 \) such that \( D[i] = dist(T_{i,m}, Q) \).

Here, \( dist \) is a distance function that defines the distance between two equal-length time series. Typical distance functions include Euclidean distance, Pearson’s correlation coefficient, cosine similarity and angular distance. Some distance functions can compare two unequal length time series such as dynamic time warping (DTW) [66], longest common subsequence (LCSS) [127] and move-split-merge (MSM) [120]. One can consider more variations of the distance profile by allowing the distance function more flexibility, such as by removing the end-point constraint [112]. However, the definition of distance profile is not limited to how the distance function operates on the pair of time series. The distance functions can also differ in their ranges. Closed ranges increase the utility of the distance profile (as we describe later) by allowing meaningful aggregation operations such as MIN and MAX. The distance function can be discontinuous by generally treating all dissimilar subsequences in the same way by assigning \( \infty \) distance allowing us to speed up computation.

In most parts of this chapter, we consider z-normalized Euclidean distance as our distance measure without any discontinuity. The z-normalized Euclidean distance between two time series \( x \) and \( y \) of length \( m \) is defined in Equation 3.1. Here \( X \) is the normalized time series, and \( x \) is the original time series.

\[
dist(x, y) = \sqrt{\sum_{i=1}^{m} (X[i] - Y[i])^2} \tag{3.1}
\]

where \( X[i] = \frac{x[i] - \mu_x}{\sigma_x} \) for \( i = 1, 2, \ldots, m \). \( \mu_x \) is the mean and \( \sigma_x \) is the sample standard
deviation of \(x\). This distance function is bounded between zero and 2\(m\) \[33\]. A simple set of steps can lead us to the following working formula for z-normalized Euclidean distance \[154\].

\[
dist(x, y) = \sqrt{2m(1 - \frac{\sum_i x[i]y[i] - m\mu_x\mu_y}{m\sigma_x\sigma_y})} \tag{3.2}
\]

\textbf{Algorithm 3:} \texttt{BruteForce}(\texttt{T}, \texttt{Q})

\begin{algorithm}
\begin{algorithmic}[1]
\State \textbf{Input:} A time series \(T\) of length \(n\) and a query \(Q\) of length \(m\)
\State \textbf{Output:} \(D\), the distance profile of \(Q\) in \(T\)
\State /*Index starting at 1 */
\State \(D[1:n-m+1] \leftarrow 0\) \hspace{1em} //\(D\) has length \(n-m+1\)
\State \(Q \leftarrow \texttt{zNorm}(Q)\)
\For {\(i \leftarrow 1:n-m+1\)}
\State \(T' \leftarrow \texttt{zNorm}(T_{i,m})\)
\State \(D[i] \leftarrow \sqrt{\sum_{j=1}^{m} (T'[j] - Q[j])^2}\)
\EndFor
\end{algorithmic}
\end{algorithm}

The Algorithm \[3\] describes the brute force way to compute the distance profile. The algorithm scans the time series \(T\) once. At each position, the algorithm normalizes the subsequence \(T_{i,m}\) and computes the distance to the normalized query. The algorithm saves all distances in the array \(D\). The computational complexity of the algorithm is \(O(nm)\). Precisely, the algorithm needs \(2m\) arithmetic operations at each iteration. To avoid \(2m\) operations in each iteration, we can exploit just-in-time normalization \[100\] that computes and stores two arrays of cumulative sums that can be used to obtain normalized distances. Since we normalize the query before scanning \(T\), the working formula can be further simplified by assigning \(\mu_y = 0\) and \(\sigma_y = 1\)

\[
dist(x, y) = \sqrt{2m(1 - \frac{\sum_i x[i]y[i]}{m\sigma_x})} \tag{3.3}
\]

To exploit this simplified formulation, we need to compute dot products over sliding windows and obtain the standard deviation of the sliding window just in time. We use the following working formula for standard deviation.

\[
\sigma_x[i] = \sqrt{\frac{1}{m} \sum_{j=i}^{i+m-1} x[j]^2 - (\frac{1}{m} \sum_{j=i}^{i+m-1} x[j])^2} \tag{3.4}
\]

The function \texttt{movstd} from Algorithm \[4\] demonstrate how to compute the \(\sigma\) by visiting
each element \( T[i] \) once. This is achieved by computing the array of cumulative sums \( S \) and an array of cumulative sums of squares, \( S_2 \), over the time series \( T \).

---

**Algorithm 4: JustInTime(\( T, Q \))**

**Input:** A time series \( T \) of length \( n \) and a query \( Q \) of length \( m \)

**Output:** \( D \), the distance profile of \( Q \) in \( T \)

```
1 \( D[1 : n - m + 1] \leftarrow 0 \)
2 \( Q \leftarrow \text{zNorm}(Q) \)
3 \( \sigma_T \leftarrow \text{movstd}(T, m) \)
4 \textbf{for} \( i \leftarrow 1 : n - m + 1 \) \textbf{do}
5 \hspace{1em} \( D[i] \leftarrow \sqrt{2\times(m-\sum_{j=1}^{m}(T[i+j-1]*Q[j])/\sigma_T[i])} \)
6 \textbf{end}
7 \textbf{Function} \text{movstd}(T, m)
8 \hspace{1em} \( N \leftarrow \text{length}(T) \)
9 \hspace{1em} \( S[1 : N + 1] \leftarrow 0 \)
10 \hspace{1em} \( \sigma[1 : N - m + 1] \leftarrow 0 \)
11 \hspace{1em} \textbf{for} \( i \leftarrow 2 : N + 1 \) \textbf{do}
12 \hspace{2em} \( S[i] \leftarrow S[i - 1] + T[i - 1] \)
13 \hspace{1em} \textbf{end}
14 \hspace{1em} \( S_2 \leftarrow S^2 \)
15 \hspace{1em} \textbf{for} \( i \leftarrow m + 1 : N + 1 \) \textbf{do}
16 \hspace{2em} \( j \leftarrow i - m \)
17 \hspace{2em} \( \sigma[j] \leftarrow \sqrt{\frac{1}{m}(S_2[i] - S_2[j]) - (\frac{1}{m}(S[i] - S[j]))^2} \)
18 \hspace{1em} \textbf{end}
19 \hspace{1em} \textbf{return} \( \sigma \)
20 \textbf{end}
```

Although the Algorithm 4 does not reduce the overall time complexity of Algorithm 3, there is a \( 2 \times \) speedup that we consider as the baseline algorithm that one would consider for computing distance profile.

### 3.4 MASS: Mueen’s Algorithm for Similarity Search

We describe an \( O(n \log n) \) algorithm to compute the distance profile under z-normalized Euclidean distance. We claim the proposed approach is faster than the baseline approach (Algorithm 4) that has a time complexity of \( O(nm) \) since \( m > \log n \) holds for most real-world applications. The core idea is to use convolution operation. We explain an intuitive example and then formally describe the algorithm.
3.4.1 MASS V1

We apply two optimizations to achieve $O(n \log n)$. First, we use convolution to compute the dot products in Equation 3.3. Second, we utilize the properties of the Discrete Fourier Transform to compute the convolution.

![Figure 3.1: Convolving a time series with a reversed and padded query produces necessary sliding dot products (results from step 2, 3, 4) for distance profile.](image)

If $x$ and $y$ are vectors of polynomial coefficients, convolving them is equivalent to multiplying the two polynomials. An example of convolving two vectors $x$ and $y$ of size four is shown in Figure 3.1.

We exploit convolution operation to compute sliding dot products between a query ($Q$) and subsequences of a time series $T$. To achieve that, we reverse the query and pad the query with zeros to match the length of the time series $T$. Consider a small time series $T$ of length four and a query $Q$ of length two. The convolution operation between $T$ and reversed and padded $Q$ produces the three sliding dot products of $Q$ over $T$. In addition, a few useless values are also produced, including some trailing zeroes. Thus, one convolution operation provides all sliding dot products of $Q$ in $T$.

To compute the convolution operation in $O(n \log n)$, we utilize the convolution theorem \[12\], which states the Fourier transform of a convolution between $x$ and $y$ equals the pointwise multiplication of their Fourier transform. To avoid the time domain aliasing \[45\] from multiplying DFTs and maintain the integrity of the convolution results, we need to zero-pad \[67\] $x$ and $y$. This process can be summarized in Equation 3.5 and the detailed padding operations are described in line 4 and 6 in MASS V1 (Algorithm 5).

\[
\text{conv}(x, y) = \text{IDFT} (\text{DFT} (\text{pad}(x)) \cdot \text{DFT} (\text{pad}(y))) \quad (3.5)
\]

MASS V1 exploits Fourier Transform, convolution theorem and cached cumulative sums for sliding standard deviation to compute the distance profile. The overall time
Algorithm 5: MASS V1(T,Q)

Input: A time series T of length n and a query Q of length m
Output: D, the distance profile of Q in T

1. $D[1 : n - m + 1] \leftarrow 0$
2. $Q \leftarrow \text{zNorm}(Q)$
3. $Q \leftarrow \text{reverse}(Q)$
4. $Q[m + 1 : 2 \times n] \leftarrow 0$ //Tail padding zeroes
5. $\sigma_T \leftarrow \text{movstd}(T, m)$
6. $T[n + 1 : 2 \times n] \leftarrow 0$ //Tail padding zeroes
7. $\text{dotPs} \leftarrow \text{frequencyConv}(T, Q, n, m)$
8. $D \leftarrow \text{normEuclidean}(\text{dotPs}, \sigma_T)$

Function frequencyConv($T, Q, n, m$)

10. $T F \leftarrow \text{FFT}(T)$
11. $Q F \leftarrow \text{FFT}(Q)$
12. $D P = T F \times Q F$ //element-wise products
13. return $\text{IFFT}(D P)[m : n]$

Function normEuclidean($\text{dotPs}, \sigma$)

16. return $\sqrt{2 \times (m - \text{dotPs} / \sigma)}$

The complexity is $O(n \log n)$ when using the Fast Fourier Transform algorithm (FFT) [26]. This can be seen easily because each of the lines in the algorithm, except the call to frequencyConv in line 7, is a linear operation with a worst-case time complexity of $O(n)$. The Algorithm MASS V1 is rather a simplified description of the exact algorithm. There are corner cases that need separate handling. If a subsequence is a constant time series, the standard deviation is zero, causing divide by zero errors. To avoid such cases, MASS needs to check the standard deviations ahead of the division operation in Line 16. In some applications, the query $Q$ comes from the time series $T$, resulting in trivial matches [142] that must be excluded by setting $\infty$ as distance values in the distance profile. For brevity, we omit these corner cases in the Algorithm 5.

3.4.2 MASS V2

Convolution defined using DFT produces more useless numbers than the necessary ones. We can improve MASS V1 by reducing the padding size. We define an operation named valid convolution shown in Equation 3.6. We demonstrate MASS V2 in Algorithm 6.

$$\text{valid.conv}(x, y) = \text{IDFT} (\text{DFT}(x) \times \text{DFT}(\text{pad}(y)))$$ (3.6)
Figure 3.2: Valid convolution produces necessary information for distance profiling in half space and time required by full convolution.

**Algorithm 6: MASS V2(T, Q)**

- **Input:** A time series T of length n and a query Q of length m
- **Output:** D, the distance profile of Q in T

```plaintext
1. D[1 : n - m + 1] ← 0
2. Q ← zNorm(Q)
3. Q ← reverse(Q)
4. Q[m + 1 : n] ← 0  // only padding query
5. σ_T ← movstd(T, m)
6. dotPs ← frequencyConv(T, Q, n, m)
7. D ← normEuclidean(dotPs, σ_T)
```

This operation only needs to tail pad y with zeros to match the length of x. Therefore, the output size is immediately reduced to half of that of a full convolution. This reduction does not change the overall time complexity of the algorithm; however, a 2× speedup can be observed based on this simple change. Figure 3.2 depicts a valid convolution operation that slides the query over the time series. Note that valid_conv(x, y) is not symmetric and it is different from valid_conv(y, x).

### 3.4.3 MASS V3

When a time series T cannot fit in the computer memory, MASS V2 will not work as defined. However, the distance profile can be computed in batches and concatenated to produce the final distance profile. In addition, it is well explored and understood that the FFT algorithm can benefit from aligning the input along the word boundaries in the
Algorithm 7: MASS_V3(T, Q)

Input: A time series T of length n, a query Q of length m and a given batch size k that ≥ m

Output: D, the distance profile of Q in T

1. D[1 : n−m+1] ← 0
2. Q ← zNorm(Q)
3. Q ← reverse(Q)
4. Q[m+1 : k] ← 0
5. T'[1 : k] ← 0
6. for i ← 1 : k−m+1 : n−m+1 do
    j ← i+k−1
    T'[1 : k] ← T[i : j]
    if j > n then //handle the last batch
        j ← n
        T'[1 : k] ← 0
        T'[1 : j−i+1] ← T[i : j]
    end
7. σ_T ← movstd(T', m)
8. dotPs ← frequencyConv(T', Q, j−i+1, m)
9. D[i:j−m+1] ← normEuclidean(dotPs, σ_T)

computer memory [1]. Moreover, the latest work [59] shows that when the input size satisfies N = 2^n the performance of FFT can be further improved by around 25%. To achieve the power of 2 input sizes and process large time series by parts, we describe the MASS V3 (Algorithm 7). In section 3.5, we show that MASS V3 has an average 31% speed improvement over MASS V2 and an average 96% improvement over our baseline just-in-time approach.

We start by describing the splitting process. We split T into segments of length that is a suitable power of two fitting in the memory. Subsequent segments must overlap m−1 observations to ensure we can concatenate the resulting distance profiles produced by MASS V2. The last segment can be of arbitrary length, as needed for the input time series. See Figure 3.3 for the splitting process.

We explore the sensitivity of the segment size K on the overall performance of MASS V3. Figure 3.3 shows how the execution time of our MATLAB implementation changes when increasing K. The best batch size must vary among systems and must not have any impact on the accuracy of the output.

The linear space complexity of the algorithm ensures extreme parallelism. We can
use GPUs to speed up MASS V3 by simply storing the data in GPU shared memory and using FFT operations that can exploit parallel processing on a GPU.

### 3.4.4 MASS V4

Time series similarity search is a time domain operation focusing only on the real parts of the output produced by a DFT-based fast convolution operation. Although theoretically, the output of the convolution operation must not have any imaginary part, we observe complex numbers with leakage in the imaginary part due to round-off errors. This can be eliminated by using real data FFT based on Hermitian symmetry. However, this solution comes with certain trade-offs. It sacrifices the simplicity of the transformation, and the space efficiency of in-place transformation [1, 38], and limits the potential for parallel computing using multiple graphics processing units [3, 4]. In addition, the complex numbers will inevitably bring in additional data structure and algorithms for related operations, which may increase the overhead cost, for instance, implementation on dedicated hardware (e.g., FPGA). We introduce a Discrete Cosine Transformation (DCT) based version of MASS that only uses the real parts of the complex numbers, and guarantees zero leakage to the imaginary parts.

The convolution between \( x \) and \( y \) can be computed with DCT type-1 of element-wise multiplication of DCT type-2 transformed \( x \) and \( y \) with zero-padding. The process is shown in Equation 3.7. The detailed padding procedures described in function \texttt{DCTPadding} from Algorithm 8.

\[
\text{conv}(x, y) = \text{DCT}_1(\text{DCT}_2(\text{pad}(x))). \ast \text{DCT}_2(\text{pad}(y)) \quad (3.7)
\]

Unlike in MASS V2, where the \texttt{valid_conv} does not pad the \( x \), MASS V4 pads both
Algorithm 8: MASS.V4(T,Q)

Input: A time series T of length n, a query Q of length m and a given batch size that satisfies \( k \geq \lfloor (3m + 1)/2 \rfloor \)

Output: D, the distance profile of Q in T

1. \( D[1 : n - m + 1] \leftarrow 0 \)
2. \( Q \leftarrow zNorm(Q) \)
3. for \( i \leftarrow 1 : k - m + 1 : n - m + 1 \) do
4.   \( j \leftarrow i + k - 1 \)
5.   if \( j > n \) then
6.     \( j \leftarrow n \)
7.   \( T' \leftarrow T[i : j] \)
8.   \( \sigma_T \leftarrow \text{movstd}(T', m) \)
9.   \( \text{dotP}_s \leftarrow \text{DCTDotProduct}(T', Q, j - i + 1, m) \)
10.  \( D[i : j - m + 1] \leftarrow \text{normEuclidean}(\text{dotP}_s, \sigma_T) \)

Function DCTDotProduct(T', Q, n', m)

12. \( T_{pad}, Q_{pad}, si \leftarrow \text{DCTPadding}(T', Q, n', m) \)
13. \( N \leftarrow \text{length}(T_{pad}) \)
14. \( T_c \leftarrow \text{DCT\_type2}(T_{pad}) \) //Orthogonal DCT applied here
15. \( Q_c \leftarrow \text{DCT\_type2}(Q_{pad}) \)
16. \( \text{dotP}_s[1 : N + 1] \leftarrow 0 \)
17. \( \text{dotP}_s[1 : N] \leftarrow T_c \ast Q_c \)
18. \( \text{dotP}_s[1] \leftarrow \text{dotP}_s[1] \ast \sqrt{2} \)
19. \( \text{dotP}_s \leftarrow \text{DCT\_type1}(\text{dotP}_s) \)
20. \( \text{dotP}_s[1] \leftarrow \text{dotP}_s[1] \ast 2 \)
21. \( \text{return} \ \sqrt{2N} \ast \text{dotP}_s[si : si + n' - m] \)

Function DCTPadding(T', Q, n', m)

22. \( p_2 \leftarrow \lfloor (n' - m + 1)/2 \rfloor \)
23. \( p_1 \leftarrow p_2 + \lfloor (m + 1)/2 \rfloor \)
24. \( p_3 \leftarrow 0 \)
25. \( p_4 \leftarrow n' - m + p_1 - p_2 \)
26. \( T_{pad}[1 : n' + p_1] \leftarrow 0 \) //padding p1 zeros at head of T'
27. \( T_{pad}[1 + p_1 : n' + p_1] \leftarrow T' \)
28. \( Q_{pad}[1 : m + p_2 + p_4] \leftarrow 0 \)
29. \( Q_{pad}[1 + p_2 : 1 + p_2 + m - 1] \leftarrow Q \)
30. \( \text{start\_index} \leftarrow p_1 - p_2 + 1 \)
31. \( \text{return} \ T_{pad}, Q_{pad}, \text{start\_index} \)

x and y. Hence, the batch size, K, to slice the time series T, must be selected considering the padded data for computing the DCT. We omit this calculation in Algorithm 8 for simplicity. However, we provide well-documented code for MASS V4 that automatically selects the best K on a given system. One worth mentioning note is that the DCT transformations in function DCTDotProduct are orthogonal transforms. One can think of using non-orthogonal transforms with scaling factors, however, we leave it as future work.
at this point.

### 3.4.5 Algorithmic Complexity

The performances of the MASS algorithms have never been documented before. In this section, we provide computational complexity, theoretical FLOPs (Floating Point Operations) count and stopwatch timing of MASS running on various data sizes.

#### Table 3.1: Time complexity in Big-O notation

<table>
<thead>
<tr>
<th>Algo.</th>
<th>JIT</th>
<th>V1</th>
<th>V2</th>
<th>V3</th>
<th>V4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td>(O(nm))</td>
<td>(O(n \log n))</td>
<td>(O(n \log n))</td>
<td>(O(\frac{n-k}{k-m}k \log k))</td>
<td>(O(\frac{n-k'}{k'-m}k \log k))</td>
</tr>
</tbody>
</table>

Table 3.1 shows the time complexity in Big-O notation. Our baseline approach JustInTime takes \(O(nm)\). V1 and V2 decrease the \(m\) term to \(\log n\) since for most real-world applications \(m > \log n\). V1 has a larger constant coefficient than V2 due to extra padding. For V3, the number of loops is \(\lceil \frac{n-k}{k-m+1} + 1 \rceil\) \(k\) is batch size and for each loop, FFT operations take \(O(k \log k)\) of length \(k\) input. V4 shares the same time complexity as V3, although the batch size \(k'\) for V4 is generally smaller than the \(k\) for V3.

#### Table 3.2: FLOPs comparison when \(m = 100\), \(k = 2^{15}\) for V3

<table>
<thead>
<tr>
<th>(n)</th>
<th>(2^{20})</th>
<th>(2^{22})</th>
<th>(2^{24})</th>
<th>(2^{26})</th>
<th>(2^{28})</th>
<th>(2^{30})</th>
<th>(2^{32})</th>
</tr>
</thead>
<tbody>
<tr>
<td>JIT</td>
<td>2.233E8</td>
<td>8.934E8</td>
<td>3.574E9</td>
<td>1.429E10</td>
<td>5.718E10</td>
<td>2.287E11</td>
<td>9.148E11</td>
</tr>
<tr>
<td>V1</td>
<td>2.508E8</td>
<td>1.098E9</td>
<td>4.774E9</td>
<td>2.062E10</td>
<td>8.855E10</td>
<td>3.786E11</td>
<td>1.612E12</td>
</tr>
<tr>
<td>V2</td>
<td>1.263E8</td>
<td>5.527E8</td>
<td>2.401E9</td>
<td>1.036E10</td>
<td>4.450E10</td>
<td>1.902E11</td>
<td>1.612E12</td>
</tr>
<tr>
<td>V3</td>
<td>7.320E7</td>
<td>2.841E8</td>
<td>1.130E9</td>
<td>4.515E9</td>
<td>1.805E10</td>
<td>7.220E10</td>
<td>2.888E11</td>
</tr>
<tr>
<td>V4</td>
<td>1.072E8</td>
<td>4.200E8</td>
<td>1.677E9</td>
<td>6.704E9</td>
<td>2.681E10</td>
<td>1.072E11</td>
<td>4.289E11</td>
</tr>
</tbody>
</table>

Table 3.2 shows FLOPs for various time series lengths when query length \(m\) and batch size \(k\) are fixed. For the JustInTime algorithm, we count the total number of additions and multiplications; both are counted as 1 FLOP. For DFT-based MASS, We use the state-of-the-art FLOPs formula for computing the real data FFT given by [59]. For DCT-based MASS, we applied the FLOPs formula in work [108], which is the follow-up work of [59] to compute the FLOPs count for DCT. In Table 3.2, we see V1 has around 12% more operation counts than JIT when \(n = 2^{20}\) and 76% when \(n = 2^{32}\). However, in Figure 3.4, we see V1 is 88% faster than JIT since additional overheads are not counted as FLOPs count, including memory operations, execution of branches and implementation of underlying libraries[37]. The V3 executes the smallest FLOPs counts among all the
algorithms. V4 executes 46% to 48% more FLOPs than V3 due to the extra padding with zeros and most importantly, the DCT always executes more FLOPs than FFT. The performance shown in Figure 3.4 matches the observations made in Table 3.2. Overall, V3 is the fastest in all three metrics: time complexity, stopwatch time and FLOPs count.

The code used for this experiment is available on this project site.

![Figure 3.4](image_url) The stopwatch time in seconds for different algorithms and different input lengths when \( m = 100, k = 2^{15} \). The platform has an I9-9900k CPU with 128GB RAM, the Matlab version is 2021b.

### 3.5 Extensions of MASS

#### 3.5.1 Weighted Distance Profile

Often practitioners have prior knowledge about the importance of various segments in the query. To exploit that knowledge, we consider setting a weight vector \( w = w[1], w[2], \ldots, w[m] \) that modifies the distance function by weighing each squared error differently, as shown in Equation 3.8. We can then expand the distance function in the sum of product form, as shown in Equation 3.9. An application demonstrating the usage of the weighted distance profile is available in Section 3.7.2.
\[
\text{weight_dist}(x, y) = \sqrt{\sum_{i=1}^{m} w[i](x[i] - \mu_x)\frac{1}{\sigma_x} - y[i]^2} 
\]

\[
\frac{1}{\sigma_x^2} \sum_{i=1}^{m} w[i]x[i]^2 - 2\mu_x w[i]x[i] - 2\sigma_x w[i]x[i]y[i] + \mu_x^2 w[i] + 2\mu_x \sigma_x w[i]y[i] + \sigma_x^2 w[i]y[i]^2 \quad (3.9)
\]

In the above summation, the left three terms have \(x[i]\)'s, hence they need to calculate sliding dot products. The remaining terms on the right are free of \(x_i\), hence they are pre-calculated before convolving. The first two terms are calculated by taking the sliding dot products of the weight vector over the time series \(x\) and its squared form \(x^2\). The term \(w[i]x[i]y[i]\) is calculated by taking the element-wise dot product \(w \circ y\) first and then calculating the sliding dot product over the \(x\). Note that vectors \(w\) and \(y\) are of identical size, hence, we can perform element-wise dot product.

### 3.5.2 Absolute Distance Profile

A common variation of the distance profile looks for the absolute distance between the query and the subsequences. This approach is particularly useful in two scenarios: 1) Instances where the absolute magnitudes of data points are crucial often do not require normalization. For example in aviation, particularly for flight trajectory altitude time series data, the absolute altitude values during crucial phases like takeoff and landing are more important than the normalized shape. Altitude variations must be analyzed as such to distinguish different geographical locations. Similarly, for solar power prediction involving Global Horizontal Irradiation (GHI) time series analysis also benefits from this, as absolute irradiation values directly impact power generation predictions [140]. 2) When data is pre-normalized using methods other than z-normalization, direct computation of the Euclidean distance is often adequate. An example is in cover song identification tasks [114, 113], where MASS computes the absolute distance profile on chroma-based features. These features undergo normalization using Chroma Energy Normalized Statistics (CENS), eliminating the need for additional z-normalization.

Calculating the distance profile without normalization is simpler. Only the moving sum of squares is needed in addition to the sliding dot product over the \(x\). The distance values can be calculated using Equation 3.10.

\[
\text{absolute_dist}(x, y) = \sqrt{\sum_{i=1}^{m} (x[i]^2 - 2x[i]y[i] + y[i]^2)} 
\]

49
3.5.3 Correlation Profile

The relationship between z-normalized Euclidean distance and Pearson’s correlation enables simple transformations from one another \[105\].

\[
\text{corr}(x, y) = 1 - \frac{\text{dist}(x, y)^2}{2m} \tag{3.11}
\]

Therefore, if we have a distance profile \(D\) for a query \(Q\) and a time series \(T\), we can produce a correlation profile in one scan. An application demonstrating the usage of the correlation profile is available in Section 3.7.4.

3.5.4 Partial Correlation Profile

Distance profiles are useful in computing the partial correlation between two time series conditioned on any subsequence of a longer time series. The partial correlation coefficient between two variables \(x\) and \(y\), conditioned on a variable \(z\) is defined in Equation \[3.12\].

\[
\rho_{xy,z} = \frac{\rho_{xy} - \rho_{xz}\rho_{yz}}{\sqrt{1 - \rho_{xz}^2}\sqrt{1 - \rho_{yz}^2}} \tag{3.12}
\]

Correlation profiles can be extended to compute the partial correlation profile between two queries with respect to a long conditional time series \(z\). The computation can be done by computing the correlation profiles of \(x\) and \(y\) as queries in the time series \(z\). This provides \(\rho_{xz}\) and \(\rho_{yz}\) in the above equation. Since \(\rho_{xy}\) is constant for queries \(x\) and \(y\), the partial correlation profile can, thus, be calculated from the two correlation profiles.

Partial correlation is crucial in scenarios where it’s necessary to measure the correlation between two variables while controlling the effects of a third variable. This measurement is especially relevant in cases where external factors influence the correlation between primary variables. For instance, in the analysis of ECG patterns, considering additional factors such as air pressure or body posture may be needed. Omitting such factors can lead to inaccurate correlations. To illustrate the significance of the partial correlation profile, we employed a 1-NN classifier, comparing its accuracy using both partial and standard correlation profiles. The methodology is detailed in Algorithm 9. We tested on the “ECG200” dataset from the UCR repository [9], which includes data classified into two categories: ‘normal’ and ‘ischemia’, with samples from both male and female subjects. We initially computed the standard correlation profile between the testing and training datasets, employing labels from the nearest neighbors for classification. This standard approach yielded an accuracy of 88%. However, when incorporating gender
as a factor through partial correlation, the accuracy improved. For each test case, we computed the partial correlation with respect to all training set cases to accommodate the absence of explicit gender information in the dataset. The minimum correlation value was then selected, as specified in line 15 of Algorithm 9. This incorporation of gender via partial correlation led to a 3% increase in accuracy compared to the standard correlation approach.

**Algorithm 9:** Partial_Corr_Classification($X_{train}$, $X_{test}$, $Y_{train}$, $Y_{test}$)

**Input:** $X_{train}$ is the training data with size $N_{train} \times m$. $X_{test}$ is the testing data with size $N_{test} \times m$. $m$ is the time series length of each case.

**Output:** Accuracy of the 1-NN classifier with the partial correlation profile.

1. $Train_{TS} \leftarrow \text{concat}(X_{train})$ //concatenating rows in $X_{train}$
2. $mask \leftarrow [1 : m : N_{train} * m – m + 1]$
3. $M[1 : N_{test}, 1 : N_{train}] \leftarrow 0$
4. for $i \leftarrow 1 : N_{Test}$ do
5. \hspace{1em} $x_{test} \leftarrow X_{test}[i,:]
6. \hspace{1em} C_1 \leftarrow \text{MASSCorr}(Train_{TS}, x_{test})
7. \hspace{1em} C_1 \leftarrow C_1[mask]
8. \hspace{1em} for $j \leftarrow 1 : N_{Train}$ do
9. \hspace{2em} $x_{train} \leftarrow X_{train}[j,:]
10. \hspace{2em} c_1 \leftarrow C_1[j]
11. \hspace{2em} C_2 \leftarrow \text{MASSCorr}(Train_{TS}, x_{train})
12. \hspace{2em} C_2 \leftarrow C_2[mask]
13. \hspace{2em} partial_corr \leftarrow (c_1 - C_1 .* C_2)./\sqrt{(1 - C_1^2) .* (1 - C_2^2)}$
14. \hspace{2em} partial_corr[j] \leftarrow \text{Inf}
15. \hspace{2em} M[i,j] \leftarrow \text{min}(partial_corr)$
16. \hspace{1em} end
17. \hspace{1em} end
18. $LOC \leftarrow \text{argmax}(M)$ //get index of maximum element in each row.
19. $\hat{Y} \leftarrow Y_{train}[LOC]$
20. $a \leftarrow Y_{test} - \hat{Y}$
21. $acc \leftarrow \text{sum}(a \neq 0)$
22. return $acc$

**Function MASSCorr($x$, $y$)**

23. \hspace{1em} $D \leftarrow \text{MASS}(x,y)$
24. \hspace{1em} $C \leftarrow 1 - D^2./\text{length}(y)$ //element-wise operations, Equation 3.11
25. \hspace{1em} return $C$
26. \hspace{1em} end
3.5.5 Multivariate Time Series

In certain applications, the analysis of multivariate time series data becomes essential. One common approach for computing the distance profile of multivariate time series is first computing the individual profiles of each dimension, and then integrating them into a single, unified profile. The integration process typically involves merging these profiles using a weighting factor \[139, 95\]. The weighting factor can either be assigned uniformly across all dimensions or determined through a learning process from the data.

3.5.6 Discussion

All the versions of distance profiles can be efficiently computed using any version of the MASS algorithm with little modifications, which are highlighted in Equations 3.9, 3.10, 3.11, 3.11. A key advantage of using MASS for these calculations is its consistency in both space and time complexity across different types of correlation profiles. The exact time complexity is available in Table 3.1.

3.6 Comparison with Indexing Solution

Distance profiling is different from indexing or index-based solutions to search for the nearest neighbor. This is an important distinction to be made, hence, demands a separate section.

An index is built on a large amount of data (typically larger than memory) in order to search the nearest neighbor (or k-nearest neighbors) of any given query very efficiently \[109, 20\]. There are several parameters involved in building and using an index: the time to build the index, the time to search for one query, and the number and types of queries searched. The goal is to search for a query interactively, while the time to create an index is generally long because of the involvement of the disk. Most indexing works consider a wide variety of queries to demonstrate generalizability.

In practice, if an index is already built and only a few neighbors are needed for each query, distance profiling is not suitable. In contrast, if the data and the query both change frequently, distance profiling is suitable to offset the overhead of index creation. What is the largest sized data that we can profile at interactive speed? Roughly, profiling a one billion long time series takes less than a minute on an off-the-shelf computer. We argue that any time series subsequence database less than one billion observations does not warrant indexing.
3.7 Utility in Application Domains

Figure 3.5: The query corresponds to the accelerometer signal of two steps taken by a SONY robot walking on the carpet. The accompanying weighted vector preserves the periods where the robot’s foot contacts the carpet and ignores periods where the foot is in the air. We applied MASS to calculate the weighted distance profile of the query on the full sequence of the robot’s movements over time (yellow time series), and the red time series is the result distance profile. In this profile, lower values correspond to a closer match with the query signal. We use a grey-shaded area to denote the time when the robot is moving on the carpet based on the ground truth data. The weighted distance profile matches this ground truth, as multiple subsequences in this period have a weighted distance below the threshold.

The utility of a distance profile comes from the knowledge of the entire distribution of distances between a query and a time series. When the frequency of the matches is important and variable for different queries, the distance profile is a great tool to exploit. We show three use cases of distance profiles in three different domains: robotics, seismology and power grid management.

3.7.1 Survey on applications

3.7.2 Robots

The accelerometer on a foot of a Sony AIBO robot records the walking cycles of the robot. The accelerometer readings show signatures produced by the surface via the reactive force on the foot. We take two cycles of a robot walking on a carpet and use a weighted distance profile by having zero weightage for the segment when the foot is in the air. The distance profile shows matches when the robot was walking on a carpet.
Figure 3.6: The top plot illustrates the query pattern which represents the typical power consumption of a house during idle or vacant periods. This pattern is characterized by its cyclic and consistent nature with minimal variations, which is typical of a household when it is not actively being used. The middle plot exhibits the power consumption time series data of a household over an extended period. The fluctuations and spikes indicate varying levels of activity and usage within the household. In the bottom plot, the distance profile calculated by the MASS algorithm is presented. The dashed line represents a distance threshold set at 22.5. Periods, where the distance profile falls below this threshold, are shaded in gray and labeled as Vacancy. These intervals signify times when the household’s power consumption pattern closely matches the idle or vacant query pattern, suggesting that the house was likely unoccupied during these times.

MASS can be used to create a distance profile of a power consumption time series [79]. The total power consumption of a household contains patterns of the idle state of the house (i.e. when nobody is in the house). If the idle state pattern of a house is known, the distance profile of the power consumption time series easily provides the number of days (or percentage of days) the household was vacant. In figure 3.6 we show a power consumption time series for six months and an idle state pattern. We also show the distance profile, which can be thresholded at 22.8 to calculate the proportion of time the house was vacant. In this example, 18.2% of the time, the house was vacant.
Figure 3.7: Two seismograph traces are presented at the top, the red trace denotes the mainshock, while the blue trace represents an aftershock. These traces are extracted from a streaming time series recorded at a seismic station running at 40 Hz. The displayed data includes the 30 seconds before and after the arrival time of the seismic event, which has been identified by human analysts. Initially, when compared directly, the two sequences exhibit a 0% correlation, indicating no direct similarity between the mainshock and aftershock as captured by the seismographs. However, we use MASS to compute the distance profile of a query, which is a 40-second subsequence from the mainshock, against the 60-second sequence of the aftershock. The bottom figure showcases the 80% correlation between the query and the best-matching subsequence from the aftershock data.

3.7.4 Seismology

MASS can be used to match aftershocks to each other when a large earthquake happens. Consider the two seismographs in Figure 3.7 recorded at the station MKAR in Kazakhstan. The aftershock (shown in blue) does not align properly with the red major earthquake (mainshock) if we use human-annotated wave arrival time [150]. In this case, the arrival times are more than 250ms apart, resulting in a poor correlation (0.08) coefficient. When we use sliding (or shifting) Euclidean distance [84], the two aftershocks show an 80% correlation that confirms the closeness of the sources. MASS is a great tool to compute sliding Euclidean distance in $O(n \log n)$ time as opposed to the naive nested-loop algorithm.

3.8 Utility in Algorithms

Distance profiles are good data structures to redesign and speed up existing data mining algorithms. We pick Time Series Discord [141] and clustering [102] to demonstrate the utility.

Time series discord is the subsequence in a time series that has the furthest nearest neighbor. The subsequence, whose nearest neighbor is the most dissimilar, is the time series
**Algorithm 10: MASS_Discord(T, m, best_so_far)**

**Input:** A time series $T$ of length $n$ and a discord length $m$

**Output:** discordLocation and discordDistance in $T$

1. $q \leftarrow T[1:m]$
2. $i \leftarrow 1$
3. $iLoc \leftarrow [1, 2, \ldots, n - m + 1]$
4. discordDistance $\leftarrow$ best_so_far
5. discordLoc $\leftarrow 0$
6. **while** $i < n - m + 1$ **do**
7.   $D \leftarrow$ MASS($T, q$)
8.   $D[max(i - m + 1, 1) : min(i + m - 1, n)] \leftarrow \infty$
9.   **if** minimum($D$) $>$ discordDistance **then**
10.      discordDistance $\leftarrow$ minimum($D$)
11.      discordLoc $\leftarrow i$
12. **end**
13.  **for** $j \leftarrow 1 : n - m + 1$ **do**
14.     **if** $D[j] <$ discordDistance **then**
15.        $iLoc[j] \leftarrow -1$
16. **end**
17. **end**
18.  $i \leftarrow$ Next positive index in $iLoc$
19.  $q \leftarrow T[i : i + m - 1]$
20. **end**

---

Figure 3.8: Execution time of MASS_Discord in comparison to traditional discord discovery algorithm on three real datasets.

discord. Traditional discord discovery algorithms exploit pruning and early abandoning strategies when computing individual distances. However, distance profiles can merge
many of these distance computations and speed up the search for discord.

In Algorithm 10, we show how distance profiles can be used to prune unpromising subsequences and hop over regions of repetitive segments of the time series. The efficiency of the algorithm depends on data characteristics in the same way traditional algorithms depend. However, a traditional algorithm computes one distance at a time and occasionally prunes candidate subsequences. The MASS_Discord computes one distance profile at a time and prunes as many candidates as possible. Thus, MASS_Discord spends less time in distance computation and more time in pruning. We see a sizable speed-up over traditional pruning strategies when tested on the three real datasets described in the previous section. In Figure 3.8, we vary the length of the discord and measure the time in seconds to find the discord by both traditional and MASS_Discord algorithms.

![Figure 3.9: Execution time comparison between MASS and a naive approach when applied to complete-linkage clustering on streaming time series. The comparison spans three different real-world datasets robots, solar data, and seismographs. The MASS algorithm demonstrates a considerable reduction in execution time compared to the naive method, maintaining a consistent lead as the subsequence length increases. The MASS-based approach is around 20 times faster on solar datasets and approximately 15 times faster on robot and seismograph datasets.](image)

The task of clustering within a single time series stream involves grouping subsequences from the stream in a manner where the selected subsequences are non-overlapping and may have gaps between them. This approach, based on the principle that clustering of time series from a single stream of data requires ignoring some of the data \[102\].

Contrasts with the clustering of individual time series, which requires computing distances among independent time series of the same length. For time series stream clustering, it is essential to consider the distance between all possible pairs of subsequences. One method is to maintain all pairwise distances in a matrix, but this approach has significant space complexity, \(O(n^2)\) where \(n\) is the length of the time series. For instance, a time series with a length of 46,340 would require approximately 16GB of RAM for the distance matrix. Thus, an algorithm with \(O(n)\) space complexity is more practical for
Algorithm 11: MASS_Complete-linkage_Clustering(T, m, \( \delta \))

\begin{verbatim}
Input: A time series T of length n, subsequence length m and a threshold of maximum distance in a cluster.
Output: C
//First compute matrix profile.
1 MP, MP I ← SCRIMP ++ (T, m)
//iLoc maintains valid index in MP.
2 iLoc ← ones(n − m + 1)
3 for i ← 1 : n − m + 1:
4   C[i] ← {};
5 while sum(iLoc) > 1 and min(MP) < \( \delta \):
6   cmi ← argmin(MP)
7   n1 ← min(cmi, MP I[cmi])
8   n2 ← max(cmi, MP I[cmi])
9   nc ← n1
10  merge(C, m, n1, n2, iLoc)
11  md, mdi ← Update(T, m, C, n1, iLoc)
12  MP[n1] ← md
13  MP I[n1] ← mdi
//Update elements in MP, MPI whose NN location is not valid.
14  for k ← 1 : n − m + 1:
15    if iLoc[k] ≠ 0 and (iLoc[MP I[k]] == 0 or MP I[k] == nc):
16      MP[k] ← n1:
17      MP I[k] ← Update(T, m, C, n1, iLoc)
18  MP[ILOC == 0] ← inf
19  MP I[ILOC == 0] ← inf
20 return C
\end{verbatim}

\begin{verbatim}
Function merge(C, m, n1, n2, iLoc)
if empty(C[n1]) and empty(C[n2]):
  C[n1] ← \{n1, n2\}
  iLoc[n1−m+1 : n1+m+1] ← 0
  iLoc[n2−m+1 : n2+m+1] ← 0
  iLoc[n1] ← 1
elif empty(C[n1]) and !empty(C[n2]):
  C[n1] ← C[n2] \cup \{n1\}
  C[n2] ← \{
  iLoc[n1−m+1 : n1+m+1] ← 0
  iLoc[n1] ← 1
  iLoc[n2] ← 0
elif !empty(C[n1]) and empty(C[n2]):
  C[n1] ← C[n1] \cup \{n2\}
  iLoc[n2−m+1 : n2+m+1] ← 0
else:
  C[n1] ← C[n1] \cup C[n2]
  C[n2] ← \{
  iLoc[n2] ← 0

Function Update(T, m, C, ti, iLoc)
//ti is the target index in MP pending to be updated.
if empty(C[ti]):
  //ti is a sub-sequence
  D ← MASS(T, T[ti:ti+m−1])
  D[ti−m+1 : ti+m−1] ← Inf
  else:
    //ti is a cluster
    D ← Inf(n−m+1)
    for k ← 1: length(C[ti]):
      si ← C[ti][k]
      cur_dist ← MASS(T, T(si:si+m−1))
      //element-wise maximum
      D ← max(D, cur_dist)
    D[ti] ← Inf
    for i ← 1 : length(D):
      if !empty(C[i]) and i ≠ ti:
        for k ← 1: length(C[i]):
          D[i] ← max([D[i], D(C[i][k])])
      D[iLoc == 0] ← Inf
      [dim, dmi] ← min(D)
      return dim, dmi
\end{verbatim}

real-world applications.

We propose a complete-linkage clustering method for streaming time series that utilizes only \( O(n) \) space, leveraging the MASS in Algorithm 11. The process starts by computing the matrix profile using the SCRIMP++ algorithm [153] (line 1). In each iteration of the While loop (line 5), a new cluster is formed or two existing clusters are merged based on the matrix profile, as defined in the merge function (starting at line 20). Any changes in the clusters trigger the update function (starting at line 39), which updates certain matrix profile cells. Here, we use MASS to compute the distance profile, and then employ a MAX aggregator to update cluster distances. This process requires only \( n \) auxiliary space since only one distance profile is maintained at a time. Overall, the space requirement to
store $iLoc, MP, MPI, D$ is $4n$.

We benchmarked the performance of our MASS based algorithm against a naive implementation that uses the \texttt{pdist} function in MATLAB to only compute the necessary pairwise distances. This comparison was conducted by varying the length of the subsequence and measuring the processing time in seconds for both algorithms on three real-world datasets. Figure 3.9 demonstrates that MASS based approach is 20 times faster on solar dataset and around 15 times faster on robot and seismograph datasets.

In Figure 3.10 we show the results of clustering an industrial dataset. The data comes from an industrial wire winding process [15]. Note that the data has significant non-uniform noise, including spikes and dropouts. Although ground truth data is not available for verification, the clustering results visually demonstrate a clear distinction between the different clusters, suggesting that the applied method effectively captures the inherent similarities within clusters and differences across clusters.

### 3.9 Conclusion

We define the distance profile of a query over a time series and provide a series of algorithms to compute distance profiles under Euclidean distance and its variants. We discuss the performance of these algorithms both quantitatively and qualitatively. We demonstrate the utility of distance profiles as a tool for data mining algorithms in various real applications. The chapter serves as the first complete documentation of distance profiling algorithms, which had only partially been discussed in articles and web pages.
Chapter 4

FewSig: Online Few-Shot Time Series Classification

4.1 Introduction

In offline semi-supervised few-shot classification, a classification model is learned from a small number of positive instances, an arbitrary number of negative instances, and a sufficiently large number of unlabeled instances. In online few-shot classification, the unlabeled instances are time ordered and are only available to the model when they occur. The model is then incrementally updated with unlabeled data over time, unlike PU-learning [90] and other semi-supervised learning approaches [117, 131, 135] that exploit unlabeled data all at once. Moreover, in the online setting, a human expert labels only the first few instances from a stream instead of a few of the most representative instances from a large pool of unlabeled instances.

In addition to the small number of training instances, online few-shot learning poses two key challenges. First, the online classification process requires each test instance to be classified before the next instance arrives. This imposes a serious efficiency constraint, challenging computationally expensive algorithms for this task. Second, it must be determined whether the newly classified positive instances are classified with sufficiently high confidence that they should be added to the training set before potentially re-training the model.

In this chapter, we propose a two-level framework. In the first level, a classifier exploits a pre-computed distance matrix under dynamic time warping (DTW) [100] distance to identify the high-confidence positive instances from unlabeled instances by bounding the false-positive rate at a maximum and adding them to the training set. In the
second level, an ensemble classifier based on distance metric learning with Neighborhood Component Analysis (NCA) is trained with Focal Loss to tackle class imbalance. This classifier evaluates the unclassified instances from the first level. We demonstrate that this framework is significantly more accurate than existing semi-supervised algorithms in the online few-shot setting.

4.1.1 Motivation

We consider online few-shot classification for time series data with application to seismic monitoring. Seismic monitoring is an online task essential for national security and public safety. Current seismic monitoring systems are not fully automated and require human analysts to review the information produced by algorithms to ensure accuracy. The amount of time an analyst takes to review a block of data (i.e., time series) is driven mainly by the number of events in that block and the amount of manual work needed to form each event completely. Large events can take longer to review as they are observed at more stations, and many of these arrivals may not be detected and associated automatically.

For example, prior to the 2011 Tohoku Earthquake and Tsunami event (the strongest earthquake recorded in Japanese history), the Late Event Bulletin (LEB [2]) of the International Data Center (IDC www.ctbto.org) averaged 120 events per day with approximately 2,000 time-defining associated arrivals (i.e., seismic signals). In the immediate aftermath of Tohoku, the LEB contained 830 events per day with approximately 20,000 time-defining associated arrivals. This alone is a 7× to 10× increase in the analyst workload [98]. In addition, the standard STA/LTA [7, 70] detectors become less effective in detecting rapid aftershocks, requiring the analysts to add more signals manually and associate them to these aftershocks.

With an increasing streaming workload, possible mediations are increasing the number of analysts (i.e., resource) and/or delay reporting (i.e., admit vulnerability). A real-time aftershock detector can reduce the manual workload significantly if it does not admit more errors than humans. However, such a detector poses several computational challenges. First, there is no training data until the main event happens. The old/historical events and their aftershocks are worthless because their epicenters are unlikely to be the same as the epicenter of the current event. Hence, historical signals do not bear any more similarity to the aftershocks than the similarity they bear to any other earthquakes. Second, the waveforms of the same event vary when observed at different stations because the waves take different paths through the earth. However, the current number of stations on Earth
is too little to capture enough training data for a station-agnostic model. Therefore, the classical computer vision approach\cite{122} that trains an offline model and adapts it to new events is not an option.

To address these challenges, we aim to exploit the similarities of the main earthquake and the first several aftershocks to the later aftershock signals and do so in real-time. The similarity of the aftershock signals is due to the source characteristics and the propagation paths from the sources to the stations, which are the two factors that control the waveforms, being similar. We must use only a few training instances of the aftershocks (i.e., the positive class) that we can collect shortly after the main shock. The detector must work in real-time to reduce human workload as well as to improve itself by learning from recent events. In other words, each event must be classified before the next one arrives.

The main contributions of this work are summarized below:

- We develop a novel online few-shot time series model (\textit{FewSig}), that can be trained on a few positive signals, and adapt to the new unlabeled signals iteratively.

- We evaluate \textit{FewSig} on 68 datasets and compare them with online versions of existing semi-supervised time series classification algorithms. The comprehensive evaluation details the parameter sensitivity, efficiency, and effectiveness of the proposed model.

- We evaluate the online performance of \textit{FewSig} for the aftershock detection task on two earthquakes, the 2015 Gorkha earthquake in Nepal (Mw 7.8) and the 2017 Chiapas earthquake in Mexico (Mw 8.2).

### 4.2 Related Work

#### 4.2.1 Semi-supervised learning on time series

Semi-supervised learning (SSL) methods have been proposed to avoid poorly generalizable models due to insufficient labeled data to train a supervised model. Wei’s Algorithm\cite{131} and DTWD\cite{23} are examples of self-training methods, a well-known semi-supervised learning approach. Souza et al.\cite{118} perform clustering to select the most representative instances from an unlabeled dataset to be labeled by an expert and then perform label propagation to classify the remaining instances. Nguyen et al.\cite{90} proposed a solution based on clustering and self-training, considering only positive and unlabeled instances. Marussy et al. proposed SUCCESS\cite{72}, a model based on constrained hierarchical clustering. Xu et al. proposed\cite{136} based on the graph-theoretic SSL algorithm. SSSL\cite{129}
performs self-training with shapelet classification on unlabeled instances. Several state-of-the-art deep-learning models have shown dominance in recent works. For example, Jawed et al. proposed a multi-task learning network (MTL) \[57\] to jointly train the ConvNet with classification and forecasting by sharing latent representations. SemiTime \[31\] shows better results than MTL on some datasets, the model learns past-future temporal relations from the unlabeled data, and the backbone feature extractor is shared with the TSC module which is trained on the labeled dataset. SSTSC \[135\] increases the richness of the temporal context by splitting a time series into past-anchor-future, making the model learn a higher-quality semantic context from the unlabeled dataset.

Even with good results in the semi-supervised scenario, it is worth mentioning that none of these works are adequate for the online few-shot learning setting proposed in this work, in which an unlabeled dataset does not exist to help train the initial model.

4.2.2 Few-shot learning

A wide range of applications encounters inherent constraints such as privacy and safety issues, ethical issues, or prohibitive costs of manual analysis, which require training a supervised model using a limited number of labeled instances per class. The paradigm of few-shot learning \[130\] has been designed to tackle this challenge. Existing models, such as MAML \[33\] and ProtoNet \[116\], exploit task-agnostic knowledge acquired during the meta-training stage for faster learning in new tasks, assuming the data for the training and testing tasks are from the same distribution \[91, 44\]. However, aftershock sequences inherently present out-of-distribution datasets. The origins of different aftershock sequences are seldom in the same region, and a single event may not manifest any similarity across seismometers located in different places due to path variation. The complexity of the Earth and the sparse distribution of seismometers make this task exceedingly challenging for these models.

The complexity further amplifies when considering cross-domain learning \[44\]. As noted by Gao et al. in work \[44\], the performance of the same meta-learning model can drastically differ across various target domains. This is also true for time series data, as indicated by \[89\], where the meta-learning approach did not significantly differ from the DTW-based 1-NN classifier when the meta-training and meta-testing data exhibit different distributions.

In contrast, **FewSig** is designed to train with a few initial positive instances and some negative instances from the target task, eliminating the need for either pre-training or meta-learning.
4.2.3 Aftershock Detection

The existing aftershock detection works can be grouped into two categories, one is template matching [105], and the other one is deep neural networks [25, 146]. The template matching techniques have high interpretability, but the performance relies on the quality and the quantity of the templates, which are very challenging to acquire in a short period after the main shock. For instance, Seiscorr [105] needed hundreds of aftershocks following a large earthquake, which occurred over the first six hours of data. That requirement makes this method inadequate for online monitoring applications.

The problem is exacerbated in deep learning solutions. Supervised deep neural networks suffer when the training set has only a few instances of aftershocks which leads to a long waiting time to collect enough data. Consider Zhang et al. [146], where authors trained on thousands of aftershocks from the first 25 days to achieve only a 67.5% F1-score at the highest on the last five days of aftershocks. In comparison, FewSig achieves the high nineties even with a few positive instances shortly after the main shock. Realizing this caveat in using historical data, other researchers [25] have developed PNN (paired neural network) to learn a similarity function from historical events, however, the trained model is yet to be evaluated on real earthquake events, while we assess on multiple large scale real seismic events. The problem is too complex to be solved using available historical data, which prevents us from exploiting attractive alternatives such as rule-based classifiers, support vector machines, etc.

4.3 Online Few-shot Time Series Classification

In this section, we formulate the problem and then introduce the technical framework of FewSig. Table 5.2 summarizes the notations and abbreviations.

We define the online few-shot classification task as follows; there are three datasets, a labeled set $L$, an online testing set $O$, and a training set $T$. $L = \{(t^1, y^1), (t^2, y^2), ..., (t^l, y^l)\}$ with true labels available, only a few (i.e., five) times series in $L$ belong to $c^+$, and the remaining belong to $c^-$; $O$ is the online testing set that saves all the new time series pending to be classified; $T$ is the training set for training a model. Initially, $O$ is empty, and the model is trained on $T$, and $T = L$. During the online testing phase, each new time series $t^j$ will be added to $O$, $O = O \cup \{t^j\}$, and the model predicts its label $\hat{y}^j$. Finally, the model performance is evaluated by comparing the predicted label with the true label for all time series in $O$. During the online evaluation process, the model can be retrained at any moment with $T$ that contains $L$ and any number of time series from
Table 4.1: Symbols and notations.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t^i$</td>
<td>Time series $i$</td>
</tr>
<tr>
<td>$t^i_{s,m}$</td>
<td>A sub-sequence of length $m$ in $t^i$ that contains $t^i[s : s + m - 1]$ $m \leq$ length of time series, assuming equal length for all time series.</td>
</tr>
<tr>
<td>$y^i$</td>
<td>True label for $t^i$</td>
</tr>
<tr>
<td>$\hat{y}^i$</td>
<td>Predicted label for $t^i$</td>
</tr>
<tr>
<td>$T$</td>
<td>Training set.</td>
</tr>
<tr>
<td>$N$</td>
<td>Size of the training set $T$, which is the number of time series in $T$</td>
</tr>
<tr>
<td>$L$</td>
<td>Labeled set</td>
</tr>
<tr>
<td>$O$</td>
<td>Online testing set</td>
</tr>
<tr>
<td>$d_{i,j}$</td>
<td>Distance between $t^i$ and $t^j$.</td>
</tr>
<tr>
<td>$D^i$</td>
<td>Distance feature array of $t^i$, $D^i = [d_{i,1}, d_{i,2}, ..., d_{i,N}]$</td>
</tr>
<tr>
<td>$c^+$</td>
<td>Positive class</td>
</tr>
<tr>
<td>$c^-$</td>
<td>Negative class</td>
</tr>
<tr>
<td>$d_{i,k}^+$</td>
<td>Average distance between $t^i$ and $k$ nearest neighbors from class $c^+$</td>
</tr>
<tr>
<td>$d_{i,k}^-$</td>
<td>Average distance between $t^i$ and $k$ nearest neighbors from class $c^-$</td>
</tr>
<tr>
<td>tFPR</td>
<td>Target FPR for the selective model</td>
</tr>
</tbody>
</table>

$O$ that have already been classified. Note that $\hat{y}^i$ is immutable, which means the model cannot leverage the current time series to update the previously predicted results.

We propose a general framework shown in Figure 4.1 for this task. There are two models: a selective model to identify the high-confidence positive time series and a general classifier model to re-evaluate the rest of the time series, the high-confidence positive time series will be added to $T$ for retraining. Initially, both models are trained on $T$ and $T = L$. For each new time series $t^j$, the solid orange path in Figure 4.1 represents the case when $t^j$...
is identified as a high-confidence positive by the selective model, $T$ will be expanded with \{(t^j, c^+)\} and both models will be retrained on the updated $T$ and finally $\hat{y}^j = c^+$; The green dotted path demonstrates the other case when $t^j$ is not a high-confidence positive, the general classifier will evaluate $t^j$ independently for the second time and yield the final label $\hat{y}^j$.

The framework does not add an explicit stopping criteria for the growth of the training set, $T$. This is rather an empirical choice because the frequency of aftershocks is the highest immediately after the main shock and decreases rapidly over days. Thus, although the training set is growing, the time between events to exploit the training set is also increasing. Hence, we avoid limiting the growth of the $T$. If event frequency stays high for longer, we recommend limiting the training set to maintain the required rate.

4.3.1 Distance/Dissimilarity as features

![Figure 4.2: Two real aftershock signals show the highest correlation of 80.84% when one is shifted in time relative to the other to correct for human error in picking, and near 0% correlation if not shifted.](image)

We use sliding Dynamic Time Warping (DTW) as a shift and warping invariant distance metric to compute the distance among time series. The sliding Euclidean distances effectively correct alignment errors [84]. Still, it is very sensitive to small mismatches [64] due to signal warping. We use sliding DTW distances to address two sources of errors in the time series. The first is due to errors in alignment. For example, human analysts often pick the onset time of a seismic phase’s arrival in seismograms, and any picking error can alter results dramatically, as shown in Figure 4.2. The second
error is due to the inherent complexity of the system being monitored. For example, the
earth’s non-uniformity changes the waves’ propagation from neighboring events. Figure
4.3 shows how DTW deals with warping by allowing one to many alignments to correct
tiny variations in the time series.

We formalize \(d_{i,j} \) as the distance between \(t_i^i\) and \(t_j^j\) in Equation 4.1. \(d_{i,j}\) is the smallest
DTW distance between two subsequences \(t_{i}^{i,q,m}\) in \(t_{i}^{i}\) and \(t_{j}^{j,q,m}\) in \(t_{j}^{j}\) for given \(q, m, s_{1}, s_{2}, r\).
We perform a similarity search with a sliding window to find this subsequence pair. The
search process defined in Equation 4.2 is the same approach employed in [100]. We take
one subsequence from a time series as a query and search for the most similar one in
another time series and vice versa, \(d_{i,j}\) is the minimum distance during this process. \(r\) in
Equation 4.2 is the Sakoe-Chiba Band [81], \(q, m\) defines the query sub-sequence, and \(s_{1}, s_{2}\)
defines the search boundaries. We recommend readers to tutorials [82, 81] for details of
these parameters.

\[
d_{i,j} = \min \left\{ SS(t^i, t^j, s_1, s_2, q, m, r), \quad SS(t^j, t^i, s_1, s_2, q, m, r) \right\} \quad (4.1)
\]

\[
SS(t^i, t^j, s_1, s_2, q, m, r) = \min \left\{ DTW(t_{i}^{i,q,m}, t_{j}^{j,q,m}, r), s_1 \leq s \leq s_2 \right\} \quad (4.2)
\]

We perform a search with some candidate values to find the optimal parameter
configuration \(\{k^*, q^*, m^*, s_1^*, s_2^*, r^*\}\) that yields the lowest loss defined in Equation 4.4
for a given dataset. The training set only contains instances from \(L\) with true labels
available at this stage. \(d_{i}^{k^*}\) defined in Equation 4.3 is the mean value of the \(k\) smallest

\[
d_{i}^{k^*} = \frac{\sum_{k} \text{smallest}(\{d_{i,j} | j \neq i \& t^i \in T \& y^j = c^+\})}{k} \quad (4.3)
\]

\[
\text{DistLoss} = \sum_{t^i \in T} \begin{cases} 
\max(0, d_{i}^{k^*} - d_{i}^{k^-})/m & \text{if } y^i = c^+ \\
\max(0, d_{i}^{k^-} - d_{i}^{k^*})/m & \text{if } y^i = c^-
\end{cases} \quad (4.4)
\]
distances between \( t^i \) and other positive time series in \( T \); \( d^{k^-}_i \) is defined in a similar way. To handle the bias of different query lengths \( m \), each distance is normalized by \( m \) when computing the loss. The optimal parameter configuration remains constant during the online evaluation phase. We set \( k^* = 1 \) for simplicity, the candidate values for the query are the middle subsequences with a length of 80% to 90% of the entire time series, the searching boundaries are a few seconds wider than the query on both ends, and the Sakoe-Chiba Band in a few samples range, i.e., from 0 to 10. More parameter candidates will increase the model initiation time, and some domain knowledge can be applied here for a smaller set of parameter candidates that yield faster model launching time.

### 4.3.2 Selective model

$$\hat{y}^i = \begin{cases} c^+ & \text{if } d^{k^+}_i < h^*_+ & d^{k^-}_i >= h^*_- \text{ other cases} \\ c^- \end{cases}$$  \( (4.5) \)

The selective model must have a very low FPR since any false-positive instance can have long-term detrimental effects. We propose a model that classifies events based on the conditions expressed in Equation 4.5. Two thresholds \( h^*_+ \) and \( h^*_- \) are established based on the training set for a given \( tFPR \) (target FPR) value.

We demonstrate the learning process with the entire \( T \) of size \( N \). We enumerate all possible split values, \( \{h^i_+, 0 \leq i \leq N\} \) and \( \{h^i_-, 0 \leq i \leq N\} \). \( h^i_+ \) and \( h^i_- \) are defined in Equation 4.6, \( d^{k^+_i}, 1 \leq i \leq N \) is the \( i \)th smallest value in the sorted \( d^{k^+_i} \) array for all \( t^i \in T \), \( \delta \) is a very small number. The learning process can be summarized in Equation 4.7 as finding two thresholds \( h^*_+, h^*_- \) such that the FPR is closest to \( tFPR \) while TPR is highest when evaluated with the entire training set. \( t^i \) will be classified as positive only when \( d^{k^+_i} < h^*_+ \) and \( d^{k^-}_i \geq h^*_- \).

$$h^i_+ = \begin{cases} ds^{k^+_i} - \delta & i == 0 \\ (ds^{k^+_i} + ds^{k^-}_i)/2 & 0 < i < N \\ ds^{k^-}_N + \delta & i == N \end{cases}$$  \( (4.6) \)

$$h^*_+, h^*_- = \arg \max \{h^i_+, h^i_-\} TPR(1 - |FPR - tFPR|)$$  \( (4.7) \)

The optimal \( k^* \) in both \( d^{k^+_i} \) and \( d^{k^-}_i \) are selected from a candidate set \( k = \{1, 2, ..., 8\} \) with highest F1 score by performing Leave-One-Out evaluation on \( T \). Once \( k^* \) is estimated, \( h^*_+, h^*_- \) are selected again with loss function in Equation 4.7 on the entire training set \( T \). Note that the \( k^* \) needs to be re-calculated each time when retraining the model.
4.3.3 General time series classifier

We utilize the distance feature $D^i$ to represent $t^i$ as shown in Figure 4.4 for our proposed general classifier enlightened by the existing time series classification works [64, 39, 56]. Each element in $D^i$ is the distance between $t^i$ and a time series in $T$. The length of $D^i$ is dependent on the size of $T$. If a new time series $t^j$ is added to $T$, then all the time series in $T$ need to update the features by appending $d_{i,j}$ at the tail. The feature for each upcoming new time series is computed with the latest $T$.

Jain pointed out in [56] that not all the dimensions in $D^i$ contribute equally during the classification, thus we consider removing less significant dimensions and boosting significant dimensions by performing Neighborhood Component Analysis (NCA [41]) with Focal-Loss [68]. This is achieved by learning a linear transformation matrix $A$ such that the 1-NN classifier performs well under this transformed space. We define the learning process in Equations 4.8, 4.9, 4.10, 4.11. $FL$ is Focal loss, $p_{t_i}$ is the probability that $t^i$ is correctly classified. $\alpha \in [0, 1]$ is a balancing factor for addressing the class imbalance. $\gamma$ is the focusing parameter that can make learning more focused on the hard misclassified positive ones rather than numerous simple negative ones. We use $\alpha = 0.5, \gamma = 2$ recommend in the original work [68] across all the experiments.
\[ A^{*}_{M \times N} = \arg\min_{A_{M \times N}} \sum_{t \in T} F L(t_i) \] (4.8)

\[ F L(t_i) = -\alpha(1 - p_{t_i})^\gamma \log p_{t_i} \] (4.9)

\[ p_{t_i} = \begin{cases} p_i & \text{if } y_i = c^+ \\ 1 - p_i & \text{if } y_i = c^- \end{cases} \] (4.10)

\[ p_i = \sum_{y_i = c^+, j \neq i} e^{-\|A^i - A^j\|^2} \sum_{k \neq i} e^{-\|A^i - A^k\|^2} \] (4.11)

\[ M = \{a \lfloor \frac{N}{20} \rfloor, a \in \{1, 2, 3, ..., 19\}\} \] (4.12)

Instead of using only one matrix \( A \in \mathbb{R}^{M \times N} \), \( N \) is number of time series in \( T \), and \( M \) is the reduced feature dimension, we learn nineteen \( A \) with various dimensions \( M = \{M_1, M_2, ..., M_{19}\} \). \( M \) can be estimated based on the best Cross Validation performance on the training set or based on a certain level of randomness. For this work, we applied the same approach described in [56] that can be expressed in Equation [4.12]. Each matrix \( A \) is trained separately, and each space will have an independent 1-NN classifier trained with transformed distance features of \( T \). The final predicted label \( \hat{y}_i \) is voted on among the 19 1-NN classifiers. \( vote = v \) means the \( \hat{y}_i = c^+ \) as long as there are \( v \) 1-NN classifiers classifying \( t_i \) as \( c^+ \). The final structure with our proposed model is shown in Figure 4.5.

### 4.4 Experimental Evaluation

We compare FewSig with four semi-supervised models adapted for the online few-shot classification settings on 68 datasets from the UEA/UCR Time Series Repository [9] covering various domains. All our experiments are reproducible, the source code, data, and additional results are available on our supporting website [13]. We performed all the experiments on an AMD EPYC 7402 server (24 cores) with a 4xRTX3090 GPU and 128GB RAM.

#### 4.4.1 Models for comparison

We select three traditional semi-supervised time series classification models: Wei’s model [131], DTWD [23], SUCCESS [72] and one state-of-the-art semi-supervised model.
SSTSC [135] based on deep learning. Next, we briefly describe the original algorithms and how we adopt them in the online setting.

Wei’s model uses a one-nearest-neighbor (1-NN) classifier as the base model. Initially, $L$ contains only a few labeled positive time series and $T = L$. To expand $T$, the algorithm iteratively selects the high-confidence positive time series from the unlabeled set measured by the nearest neighbor Euclidean distance. When iteration stops, all time series in $T$ are labeled as $c^+$, and the remaining ones in the unlabeled set are $c^-$. We take the following procedures to adapt the model for the online setting: 1) Initially $T = L$, $L$ includes both $c^+$ and $c^-$. 2) The 1-NN classifier will classify each new time series $t^j$, if $\hat{y}^j = c^+$, $t^j$ will be added to $T$ and retrain the model. 3) If the model is updated, then all the previous $t^i \in O$ that are not in $T$ will be reconsidered whether to add in $T$. 4) The model will be retrained immediately after $T$ is expanded in step 3, and step 3 is repeated. 5) The iteration will stop when $T$ is not expanded in step 3.

DTWD applied a similar approach for augmenting $T$ via self-training on an unlabeled set. It employs a new distance measure and a one-class classifier. The distance measure is the ratio of DTW distance to Euclidean distance. The one-class classifier relies on the entire unlabeled set to pick the optimal parameters however, this is impractical in the online scenario. Thus, we consider Wei’s approach with the distance measure employed by DTWD.

SUCCESS is based on the constrained single-linkage hierarchical agglomerative clustering algorithm with DTW distance. One constraint when linking the instances is that instances from $L$ can not be linked. The final label of a cluster is decided by the majority class of instances in $L$. The unlabeled instances in a cluster share the same cluster label. Then a 1-NN classifier will be trained for testing. To work in the online setting, we first utilize SUCCESS to classify a new time series $t^j$, then rebuild the cluster hierarchy with $L$ and the existing $O$ including $t^j$, and finally retrain the 1-NN classifier.

SSTSC learns an encoder that can capture the temporal context based on the unlabeled dataset in a self-supervised manner. This encoder will be used as a backbone for the supervised TSC module that is trained on the labeled dataset $L$. We consider the same approach employed for SUCCESS to adapt SSTSC for the online setting.

### 4.4.2 Online experimental protocols

We select 68 out of 128 univariate time series datasets from the UEA/UCR repository [9]. The selection is based on the size of the datasets, which should be less than 800 time series since FewSig trains on five positive time series, and increasing the number of test
instances does not significantly impact the performance. Hence, we chose a cut-off for the datasets. For each dataset, we re-assign the labels, the minority class is considered as $c^+$ for unbalanced datasets, and a randomly chosen class as $c^+$ for balanced datasets. All other classes are considered as $c^-$.

We randomly order all the time series in a dataset due to the absence of event time. $L$ consists of the first five positive time series and the first half of the negative time series; $O$ consists of the rest of the time series. The time series in $O$ is fed to the model sequentially in assigned order to simulate online scenarios. The scores are computed by comparing the $\hat{y}_t$ with $y_t$ for all $t \in O$. To diminish the random bias, we perform 30 trials for each dataset, and we verify that none of the trials share more than two positive time series in $L$ with other trials. The final scores of a model are averaged among 30 trials.

There is no parameter configuration for Wei’s model, DTWD, and SUCCESS. For SSTSC, we use the same configuration described in the original paper [13]. We use the following settings for FewSig across all datasets: $t_{FPR}=1\%$, $\alpha=0.5$, $\gamma=2$, and SDG with $lr=0.02$, the final F1 score is averaged when $vote=\{1,2,3,4\}$ separately.

### 4.4.3 Experimental Results

#### 4.4.3.1 Ranking comparison

![Critical difference diagram of 5 models on the 68 UEA&UCR benchmark datasets. $\rho=0.05$ for Nemenyi test.](image)

Figure 4.6: Critical difference diagram of 5 models on the 68 UEA&UCR benchmark datasets. $\rho=0.05$ for Nemenyi test.

We conduct the Friedman test and Nemenyi post-hoc test on all 68 datasets across five models. The ranking is computed with the mean F1 scores of 30 trials per dataset. We show the critical difference diagram in Figure 4.6 in which FewSig ranks at the top with a significant statistical difference to the rival models. The average F1 for 68 datasets between FewSig and Wei’s model, SSTSC are shown in Figure 4.7. FewSig has a higher F1 for most of the datasets in both cases. The results for each dataset are available on the supporting website [13].
Figure 4.7: Mean F1 scores of 30 trails on 68 datasets comparison between FewSig and Wei’s model (left), SSTSC (right).

Table 4.2: Average running time in seconds. $k$ is the number of candidate parameter sets to find the optimal one.

<table>
<thead>
<tr>
<th>Model</th>
<th>Initial time</th>
<th>Ave. Time per $t_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Distance</td>
<td>Train</td>
</tr>
<tr>
<td>Wei’s</td>
<td>1.49</td>
<td>0.0004</td>
</tr>
<tr>
<td>DTWD</td>
<td>5.01</td>
<td>0.0005</td>
</tr>
<tr>
<td>SUCCESS</td>
<td>3.517</td>
<td>0.0003</td>
</tr>
<tr>
<td>SSTSC</td>
<td>0</td>
<td>26.78</td>
</tr>
<tr>
<td>FewSig</td>
<td>8.571*k</td>
<td>14.46</td>
</tr>
</tbody>
</table>

4.4.3.2 Time complexity

We present the average running time of 68 datasets for each model in Table 4.2. We consider the time of initial training on $L$, the time for inferring each instance in $O$, and the time for model retraining. Since the model does not need to be retrained on each instance $t_j \in O$ for Wei’s model, DTWD, and FewSig, the time is amortized on the entire testing set.

Although FewSig performs significantly more accurately, FewSig is not as efficient as existing methods. This highlights that FewSig trades a bit of speed to gain accuracy. However, FewSig takes several minutes to initiate and processes roughly one event per second. This speed is sufficient for our target domain because the shortest time between successive events is 16 seconds in our online testing sets for both Nepal and Chiapas aftershock sequences.
4.4.3.3 Parameter sensitivity

In this section, we discuss the FewSig sensitivity to three design parameters: i) different vote numbers for the general classifier, ii) tFPR, and iii) the initial number of positive time series in \( L \).

![Figure 4.8: Critical difference of FewSig with the different number of votes on 68 datasets, vote = 2 yield highest rank.](image)

![Figure 4.9: Critical difference diagram of FewSig with four different tFPR values.](image)

![Figure 4.10: F1 score comparison between labeled sets \( L \) with the different number of positive instances for 68 datasets.](image)

i) Figure 4.8 shows the F1-score ranking of FewSig with the different number of votes for the ensemble. If two classifiers agree on a positive instance, it gives the best overall results across all 68 datasets. ii) Figure 4.9 shows there is a significant difference
in ranking when tFPR goes from one extreme (0.1%) to the other (2%). iii) In Figure 4.10 we show the F1 comparison with the different number of positive time series in $L$. The figure shows that doubling this number from 5 to 10 has more impact than doubling from 10 to 20.

4.5 Aftershock detection

In this section, we evaluate FewSig performance for the online aftershock detection on two major earthquakes Mw 7.8 Nepal (Gorkha) earthquake in 2015 and the Mw 8.2 Chiapas earthquake in 2017.

4.5.1 Data preparation

We use the Late Event Bulletin (LEB) [2], which provides information about events such as time, location, and magnitude. The seismograms (i.e. time series data) are collected from the International Monitoring System (IMS), operated by the Preparatory Commission of the Comprehensive Nuclear-Test-Ban Treaty Organization (CTBTO). For both major earthquakes, we use local catalogs to confirm the aftershocks. We use the catalog from McNamara et al. [73] for the Nepal earthquake, and the catalog from the Mexican Servicio Sismológico Nacional (SSN) [77] for the Chiapas earthquake. We describe our data preparation procedures in the following steps.

Aftershock association We select the ground truth aftershock events by joining the local catalog with LEB. We call this process aftershock association. The association process is necessary to double confirm that an event was truly an aftershock. We use each aftershock event from the local catalog as a query to search for an associated event in the LEB. The join condition for an association is that the events must be within 200km of their Great-circle distance and their origin times are within 5 seconds. We break ties by taking the events that are closest in time.

Non-aftershock selection We consider all the seismic events in the LEB that are outside the geographical region of aftershock events as non-aftershocks. For this experiment, we only select non-aftershock events with similar origin-to-station distances compared to the aftershock events to create challenging cases for the classifier.

Arrival selection We get signal arrival times for the aftershock and non-aftershock events at a target seismometer in the LEB. For this experiment, we only use P phase signals, as P is the most common phase in the LEB.
Waveform processing Based on the arrival time, we extract three 60-second time series for each event from the continuous waveforms of three broadband channels BHZ (vertical), BHN (north-south), BHE (east-west) of a single seismometer. Thus, the time series contains 30 seconds of pre-arrival and 30 seconds of post-arrival signal. This time window is large enough to capture the initial compressional seismic waves generated by any regional earthquake. If the waveforms are sampled at 40Hz, the length of each waveform is 2,400 real numbers. Following conventional seismic signal preprocessing techniques, we remove the linear trend, and the mean value of each waveform, then taper the waveforms before filtering. We applied a 0.4Hz to 10Hz second-order Butterworth bandpass filter in both directions to cancel the phase shift. Next, we compute the Signal to Noise Ratio (SNR) on the filtered waveforms as the ratio of the standard deviation of the signal part (post-30 seconds) over the noise part (pre-30 seconds). For the experiments, we only use waveforms that have $\text{SNR} \geq 2$.

4.5.2 Features for multidimensional time series

$$SS(i, j, s_1, s_2, q, m) = \min \{ \sum_{\beta \in \{Z,N,E\}} \alpha_{\beta}^j DTW(t_{s,m}^{i\beta}, t_{q,m}^{j\beta}, r), s_1 \leq s \leq s_2 \}$$

(4.13)

$$\alpha_{\beta}^i = \frac{\max(||t_{i\beta}^{z}||)}{\max(||t_{i\beta}^{z}||) + \max(||t_{i\beta}^{n}||) + \max(||t_{i\beta}^{e}||)}, \beta \in \{Z, N, E\}$$

(4.14)

To accommodate three channel/dimension time series data, we modify Equation 4.2 to Equation 4.13 by combining three weighted distances of time series from the same channel. $t_{i\beta}^{z}, \beta \in \{Z, N, E\}$ represents the time series from $\beta$ channels for the event $i$. Equation 4.14 defines $\alpha_{\beta}^i$ which are the proportional weights of each channel.

We applied the same procedures described in section 3.1 to select the optimal parameters $\{q^*, m^*, s_1^*, s_2^*, r^*\}$ for Equation 4.13. In here we fix $q = 5$, $m = 50$, $s_1 = 4$ and $s_2 = 6$ seconds and compute the loss value defined in Equation 4.4 for each $r \in \{0, 1, ..., 10\}$ in number of points. We report $\{q^*, m^*, s_1^*, s_2^*, r^*\}$ applied in the result sections.

4.5.3 Experimental Settings

We reproduced the scenario when an earthquake happens to evaluate the real-world performance of $\text{FewSig}$. Initially, we have the labeled set containing the non-aftershocks before the main shock and the first five aftershocks. All the later events are in the online testing set $O$ without knowing their labels. The model will sequentially classify each event in $O$ based on their event time. Finally, we show the performance of each model over time regarding TPR, FPR and F1 scores.
Parameters for FewSig are constant for all the experiments: \( t_{FPR} = 0.5\% \), \( vote = 2 \), \( learning\ rate(\text{lr}) = 0.02 \), \( epoch = 250 \), \( \gamma = 2 \), \( \alpha = 0.5 \). The warping banding for computing the DTW varies at each station. We report the exact number in corresponding sections.

### 4.5.4 Results for the 2015 Nepal Earthquake

Figure 4.11: The distribution of origins for events in the LEB and the local catalog [73]. The events in the LEB for the 2015 Nepal earthquake are selected by limiting the origins to the red rectangular region and limiting the origin time from 2015-04-25T06:11:24.290000Z to 2016-05-14T22:45:53.330000Z.

The origins of the events in LEB and the local catalog [73] are shown in Figure 4.11. The MKAR seismic station has the highest number of recorded arrivals in LEB among other IMS stations for the ground truth aftershock events. Thus we fetch all the seismograms for the experiment at MKAR.

We extracted 217 aftershock arrivals and 1182 non-aftershock arrivals at the MKAR based on the conditions described in the data preparation section, each arrival has three corresponding time series from three channels. Next, we put 5 aftershocks and 705 non-aftershocks in \( L \), and 212 aftershocks and 477 non-aftershocks in \( O \) based on the event time. Figure 4.12 shows the origins of the extracted arrivals. We use \( q = 5 \), \( m = 50 \), \( s_1 = 4 \), \( s_2 = 6 \) seconds and \( r = 1 \) point for Equation 4.13.

We demonstrate the performance of FewSig and reference models over time regarding F1, TPR, and FPR in Figure 4.13 and 4.14. We can conclude that FewSig consistently leads the other models by around 0.2 on F1, 20% on TPR, and essentially maintains the
Figure 4.12: Left figure shows the geographical distribution of origins for extracted arrivals at MKAR. The right figure shows some example waveforms from the BHZ channel at MKAR. The first five aftershocks are in red. They were filtered with a 0.4Hz to 10Hz Butterworth bandpass filter.

Figure 4.13: Online performance for classifying the Nepal aftershock sequence at MKAR. F1 scores of different models are shown on the solid curves. A point on a curve shows the score when testing the events at and before the time on the x-axis. The accumulated number of testing aftershocks and non-aftershocks are represented by the light green and blue shaded areas respectively. Both SUCCESS\_OL and SSTSC\_OL have F1 scores of zero throughout. We only use Z channel time series data for SSTSC\_OL since it only supports univariate time series.

The F1-score of FewSig rapidly increases when more aftershocks are selected for the training set. F1 reaches 89% at the peak point after 157.2 hours since the main shock, then it gradually decreases when there are more non-aftershock events. Finally, the F1-score levels out at 0.85, the TPR does so at 82.55%, and the FPR at 5.4% for FewSig. During the online training process, FewSig is retrained 127 times. The origins of false positives and false negatives obtained by FewSig are demonstrated in Figure 4.15. False positives are harmful because human analysts will have to spend time correcting such errors if indeed they detect them. If they don’t detect them, then target events could be
Figure 4.14: Online performance for classifying the Nepal aftershock sequence at MKAR. TPR and FPR scores of FewSig and Wei OL are shown on the solid and dotted curves.

Figure 4.15: The origins of 26 false positives (red) and 37 false negatives (blue) obtained by FewSig overall.

missed. The scattering of false positives in multiple locations suggests FewSig making unbiased mistakes.

4.5.5 Results for the 2017 Chiapas Earthquake

To test the universality of FewSig, we further examine the 2017 Chiapas aftershock sequence. We use the catalog of the Mexican Servicio Sismológico Nacional (SSN) [77] as the local catalog. The selection criteria we made on this catalog are: 1. The period from 2017-09-08 to 2018-03-08. 2. The latitude is from 14 to 17 and the longitude is from -96 to -93. The origins of events in the local catalog and LEB are shown in Figure 4.16. We choose the station TXAR for this earthquake since it has the most number of arrivals.

We balanced the number of non-aftershocks in L and O due to insufficient non-aftershock events after the main shock in the LEB. Then we have 5 aftershocks and 280
non-aftershocks in $L$, and 134 aftershocks and 280 non-aftershocks in $O$ after balancing. The distribution of origins is shown in Figure 4.17.

Figures 4.18 and 4.19 are analogous to Figures 4.13 and 4.14, but for the Chiapas earthquake, FewSig outperforms the reference models by achieving an overall 0.91 F1 score, 90.3% TPR, and 3.57% FPR. During the online training process, FewSig is retrained 86 times. The overall false positives in Figure 4.20 are scattered and suggest FewSig is making unbiased errors.

### 4.6 Conclusion

In this chapter, we present an online few-shot classifier for time series data. Our method only requires a few labeled positive instances and some labeled negative instances and can be gradually enhanced with the new unlabeled instances automatically. FewSig
outperforms existing methods on 68 datasets from the UEA/UCR repository and achieves adequate performance for the online aftershock detection task when evaluated with two real-world earthquakes. Our model is lightweight and does not rely on a large labeled dataset which makes it applicable for online seismic monitoring systems to reduce the workload of analysts.
Figure 4.19: Online performance for classifying Chiapas aftershock sequence at TXAR. TPR, FPR scores of FewSig and Wei OL are shown on the solid and dotted curves.

Figure 4.20: The origins of 10 false positives (red) and 13 false negatives (blue) obtained by FewSig overall.
Chapter 5

BitLINK: Temporal Linkage of Address Clusters in Bitcoin Blockchain

5.1 Introduction

Bitcoin is an anonymous and decentralized electronic cash system, employs hashed cryptographic keys as addresses for transactions. Digital services, particularly within the gambling and financial sectors, that leverage Bitcoin as a payment method often manage a multitude of Bitcoin addresses to facilitate their operations. To manage funds efficiently across these addresses, such services commonly consolidate several addresses into a single transaction. This is made possible by the multi-input and multi-output capabilities inherent to the Bitcoin protocol. Nevertheless, while this method increases efficiency, it simultaneously compromises anonymity. The aggregation of multiple input addresses in a single transaction typically signifies either singular control by an entity or management among an alliance of trusted entities. Prior research [69, 74, 47] has employed the multi-input heuristic in a recursive process to construct an address cluster, suggesting shared ownership of the involved addresses.

We have observed a strategic behavior among some entities, particularly those involved in mixing services, where they periodically abandon an address cluster and promptly establish a new one. This behavior appears to be a precautionary measure to prevent forming a large, easily traceable cluster using the multi-input heuristic. Thereby minimizing the risk of their activities being uncovered. Figure 5.1 illustrates this practice through the activities of an entity known as BitPay. We show four address clusters of
Figure 5.1: Temporal activity of Bitcoin addresses owned by BitPay, addresses with the same color are in the same cluster which is formed by utilizing the multi-input heuristic. The x-axis indicates the timestamp of the first receiving transaction, while the y-axis corresponds to the timestamp of the last spending transaction for each address.

BitPay labeled by WalletExplorer which is a public website providing tagged Bitcoin addresses. Initially, BitPay managed an address cluster C1, which was active from July 2011 to February 2012. Abruptly, in February 2012, almost all spending transactions from C1 ceased, and a new cluster, C2, emerged and began to accumulate Bitcoin. This pattern of simultaneous termination and initiation of transactions in distinct clusters is what we define as address flushing. Subsequently, BitPay transitioned from C2 to a missing cluster, M1, which later was flushed into cluster C3 in February 2015.

Notably, WalletExplorer did not detect two intermediate clusters, M1 and M2, indicating the inherent limitations of its manual clustering methodology. This method initially acquires an address from the service (register as a user) and then employs the multi-input heuristic recursively to cluster other addresses. The manual approach can be prohibitively costly if conducted repeatedly due to address flushing and is not scalable to the vast number of address clusters operated by various services on the Bitcoin blockchain. Our work introduces an automated framework designed to efficiently identify carry-on clusters subsequent to address flushing events. While human analysts may still be required to validate the most promising candidates, this framework significantly enhances efficiency, reducing the verification process from nearly one billion cases to a handful of cases compared to the manual approach. This substantial increase in efficiency not only optimizes current analytical processes but also has the potential to accumulate more labeled data, thereby facilitating continuous refinement and evolution of the system in the future.

Identifying untagged clusters of a service can provide valuable financial information. Although achieving full disclosure may be impossible, even partial clusters owned by a service can reveal useful knowledge. For instance, identifying any address clusters related
to mixing services, such as HelixMixer, which was the first bitcoin mixer penalized by FinCEN for violating anti-money-laundering laws \[24, 60\], can provide a lower bound of how many BTC have been laundered, at the minimum. Revealing addresses associated with darknet markets, such as SilkRoad, which was the biggest anonymous drug black market \[52, 53\], may assist law enforcement authorities.

As discussed in Section 2, existing heuristics and tools for address identification are inadequate for tracking entities that flush their addresses to obfuscate their activities. Our approach focuses on the temporal patterns of transaction activities as we observed flushing events exhibit a distinct temporal linkage signature. By plotting the time series of a daily number of receiving transactions for a given cluster, we can observe a pattern where two time series intersect, as illustrated in Figure 5.2(left). Furthermore, the transaction patterns remain consistent before and after a flushing event, providing another indicator for the flushing event between clusters.

![Temporal linkage signature for flushing](image)

Figure 5.2: The time series of daily receiving transaction count for address clusters owned by BitPay. On the left, a clear temporal linkage signature is depicted, demonstrating the transition from flushed cluster C1 to carry-on cluster C2. The right graph presents the transaction count time series for four clusters (C1, C2, C3, C4) tagged by Wallet-Explorer.com, alongside two clusters (M1, M2) that have been identified through our BitLINK.

In this study, we introduce BitLINK, a novel temporal heuristic designed to trace Bitcoin address clusters after a flushing activity through distinctive temporal linkage signatures. BitLINK posits that two clusters displaying distinctive temporal linkage signatures can be connected as part of the same entity. Leveraging this heuristic, we implement a deep neural network model capable of examining potential linkages between a known flushed cluster and another cluster on the blockchain. The objective is to accurately identify the carry-on cluster that succeeds the flushed one.

This model is trained in a self-supervised manner as it obviates the need for labeled data, which is a significant advantage in the context of the inherent anonymity of Bitcoin.
In addition, the generalization capability of the neural network makes it suitable for the dynamic nature of flushing events observed in the blockchain, as detailed in Section 5.3.2. Leveraging this trained model allows us to successfully identify previously undetected clusters, such as M1 and M2 in Figure 5.1. The temporal patterns of these clusters, illustrated in Figure 5.2 (right), reveal a matching growth trend in the number of receiving transactions.

The main contributions of this chapter are summarized below:

- This study is the first to formally define and discuss flushing behavior observed in the Bitcoin blockchain. We propose a novel temporal heuristic, BitLINK, paired with a self-supervised learning model to trace a flushed address cluster.

- We provide three case studies on three services: HelixMixer, Primedice and Bitcoin Fog. We use BitLINK to link all known address clusters of these services to new and verified address clusters, for all the time until the current block.

- We linked clusters from 60 services listed on WalletExplorer. The visualization of these clusters is available at http://www.bitlinkwallet.com/.

5.2 Related Work

The problem of Bitcoin address clustering has been a subject of research since the inception of Bitcoin. Despite the extensive studies in this area, the phenomenon of address flushing has not been explicitly discussed in the existing literature. Furthermore, none of the methodologies can effectively handle flushing behavior, which is crucial for improving performance in clustering analysis. This gap highlights the innovative aspect of our work.

**Entity identification by clustering:** Bitcoin address clustering is a key approach to identifying entities behind Bitcoin addresses. Typically, addresses within a single cluster are believed to be controlled by one entity. The fundamental technique is the multi-input (common-input) heuristic, introduced in the Bitcoin white paper [88]. Numerous adaptations and advancements of this heuristic have since been developed. The change (shadow) address heuristic [69 74 148], designed to identify addresses that collect change from transaction outputs, often falls short during address flushing events due to the lack of direct transactions among flushed clusters and the carry on clusters detailed in Figure 5.3. Specialized heuristics, such as the coinbase (mining) [47] and mixing heuristics [42], target specific activities like mining or mixing services but do not effectively distinguish between different entities within these categories. The
Figure 5.3: Transaction graph of clusters owned by HelixMixer

state-of-the-art multi-heuristics [18] proposed by He et al. combines the above heuristic rules and the Louvain community detection method for clustering.

Our methodology enhances the traditional common-input heuristic by integrating a novel temporal dimension, an aspect that remains unexplored in existing literature. Our approach maintains broad applicability in contrast to domain-specific analyses, which often depend on identifiable features or expert knowledge of particular Bitcoin services such as mixing services [110, 103, 10, 134] or mining operations [99, 123].

Table 5.1: Features utilized in different tools for analyzing Bitcoin address clusters.

<table>
<thead>
<tr>
<th>Tools</th>
<th>Heuristics</th>
<th>TX network Statistics</th>
<th>Asynchronous Time series</th>
</tr>
</thead>
<tbody>
<tr>
<td>BitIodine</td>
<td>multi-input &amp; change address</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>BitConeView</td>
<td>multi-input</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>BitConduite</td>
<td>multi-input</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>BitLINK</td>
<td>multi-input</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Entity identification tools:** Various tools have been developed to assist Bitcoin blockchain analysis. BitIodine [119] is a notable framework for constructing a pipeline for analysis. However, its ability is limited by its reliance on traditional heuristics and
lacks the integration of temporal data analysis. Other frameworks like Bitconeview [29], BitConduite [124], and BitExTract [144] similarly concentrate on transaction graphs that do not incorporate time series signatures. The state-of-the-art Bitcoin analysis framework, BitAnalysis [121] incorporates a temporal view section for clusters achieved through the use of Dynamic Time Warping (DTW) to compare clusters active during the same period. However, it falls short of analyzing clusters that are active in different periods.

**5.3 Methods**

5.3.1 Temporal Signature for Address Flushing

The temporal linkage signature, as initially illustrated in Figure 5.2 (Left), is further exemplified through real-world cases demonstrated in Figure 5.4. This figure showcases the temporal linkage patterns of six distinct entities. A commonality across these examples is the presence of two intersecting time series: a decreasing red time series representing the flushed cluster, and an increasing blue time series indicating the carry-on cluster. We also observed variations of the flushing behaviors among entities. Ranging from the instantaneous flushing observed in entities like Bitfinex and Bter to more protracted
transitions over weeks, as noted in AntPool, LocalBitcoins, and CrimeNetwork. A particularly high correlation between two address clusters managed by Cryptsy was also noted, underscoring the variability of flushing dynamics.

Another key insight from these observations is the stability in the daily count of receiving transactions on the entity level before and after a flushing event. This stability suggests that customer behaviors remain largely unaffected by changes in receiving addresses and their changes (i.e., address flushing) of a service.

The identification of a temporal linkage signature between two address clusters, as outlined by our BitLINK heuristic, is a robust indicator of a flushing event. This observation enables us to trace clusters operated by the same entity after flushing activities.

5.3.2 Self-supervised Learning

Learning the temporal linkage signature, as illustrated in Figure 5.4, poses a significant challenge, primarily due to the intrinsic privacy features of blockchain technology that limit the availability of labeled data. This challenge is compounded by the diverse nature of flushing behaviors exhibited by distinct entities.

A promising approach to circumvent the scarcity of labeled data is through self-supervised learning. This methodology constructs supervised learning tasks from unlabeled data based on the intrinsic nature (i.e., temporal pattern) of the data. Inspired by one of the pre-training tasks used in BERT [28], which involves determining whether one sentence follows another, our self-supervised task aims to ascertain whether one time series follows another. This allows us to exploit the unlabeled data within the Bitcoin blockchain to generate pseudo-labeled datasets, as detailed in Section 5.3.4. The essence of this task lies in its simplicity and effectiveness in identifying temporal linkage signatures since the pattern of receiving transactions remains consistent before and after a flushing event.

5.3.3 Model architecture

A visualization of the network structure can be found in Figure 5.5. The architecture consists of a convolutional neural network (ConvNet) that can extract intricate features with minimal manual feature engineering. We follow a classical VGG design style [115] which constructs the network by piling up blocks of ConvNets. In summary, the model has three ConvNet blocks paired with three fully connected layers (MLP). Each block is equipped with multiple convolution kernels complemented by a max pooling layer, each
5.3.4 Data Augmentation

Figure 5.6 offers a visual demonstration of five steps to generate the pseudo-labeled dataset from Bitcoin on-chain data. We share the code to prepare the data from the raw blockchain data to the pseudo-labeled dataset on our support website [13].

**Step 1 - Cluster Generation:** We use Bitcoin Core [17] to synchronize Bitcoin on-chain data from its inception on January 03, 2009, to October 16, 2022, covering blocks 0 to 758,967. Then we employed BlockSci [61] for data parsing and clustering addresses using the common-input heuristic. This process initially identified 786,969,200 common input clusters. Due to computational constraints, this number was refined to 455,731 clusters, prioritizing those containing either more than 100 addresses or exceeding 400 transactions. Clusters already identified in the WalletExplorer database and those within manually labeled datasets, designated for testing purposes, were excluded.

**Step 2 - Time Series Generation:** We generate daily transaction time series $t_{ri}$ and $t_{si}$ capturing receiving and spending activities for each cluster. Transactions are counted at the block level first and then aggregated daily. A Bitcoin transaction is counted when its output or input address is contained in $c_i$. Transactions are counted once, regardless of multiple addresses from $c_i$ being in the same transaction. The complete time series spans 5,035 days based on our on-chain data. Although the augmentation process primarily focuses on the receiving transactions ($t_{ri}$), we also construct a corresponding spending time series ($t_{si}$) for each pair in the pseudo-labeled dataset. This inclusion of spending...
Figure 5.6: Data augmentation process utilizing Bitcoin on-chain data. For demonstrative purposes, we focus on the augmentation of the red time series at the second split point. Activities serves to enrich the dataset, providing additional dimensions of information that enhance the performance of our model.

Figure 5.7: The blue curve shows raw times, and the red curve shows trend time series for BTC-e. The Y-axis is the daily receiving transaction count. Vertical lines indicate the start and end of each non-zero sub-sequence.

**Step 3 - Time series filtering:** Transactional data can be sporadic as illustrated in Figure 5.7 where 261 sub-sequences of non-zero values emerge beyond the main service activation phase. We define a sub-sequence as a windowed time series data that has no zero value and cannot be expanded further. We only want to keep the sub-sequence corresponding to the main activate period of a cluster to enhance the efficiency of the following steps. To achieve this, we employed a second-order Butterworth lowpass filter,
setting the cut-off frequency at 0.015Hz and the time series frequency at 1Hz. This allowed us to extract the time series trend as shown by the red curve in Figure 5.7. We then evaluated the trend’s first-order derivative to identify the top 5 local minimums and maximums. Any sub-sequences that lacked extrema were filtered out (setting to 0).

**Step 4 - Time series segmentation:** For each time series of clusters that spanned more than 10 days, we determined split points positioned 15 days apart and ensured a 5-day buffer from both subsequence ends. Then we carefully eliminate points that might coincide with potential real flush activities. This is achieved by restricting the variance in mean values between the prefix and suffix segments. The difference between prefix and suffix means should not exceed 50% of the prefix mean. Furthermore, the prefix mean must exceed a threshold of 25. These mean calculations used a window up to 16 days in length.

**Step 5 - Positive Pair Generation:** For each split location, the time series was divided into a prefix (subsequence before the split point) and a suffix (subsequence after the split point). Then both subsequences are padded with zeros to match the length of the original time series to form a positive pair. Figure 5.6 illustrates a positive pair formed at the second split position. One cluster can yield the same number of positive pairs as the number of split positions. However, each prefix in the positive pair can form many negative pairs, as described in Step 5.

**Step 6 - Negative Pair Generation:** For each positive pair generated from the above step, we take the prefix and pair it with the complete time series of other clusters to form negative pairs. To create maximum confusion for the model, we first remove easy cases and then prioritize the hard cases. The easy cases are the pairs where the suffix has no overlapping with the prefix or has excessive overlapping. This is determined by a set of hyperparameters set empirically, including Pearson correlation value between the suffix and prefix, and the relative positions between prefix and suffix. The hard negative pairs are selected based on the Euclidean distance between the negative suffixes to the positive suffix. No more than 500 negative suffixes will be selected for each prefix.

### 5.3.5 Focal-Loss

Owing to the significant imbalance between positive and negative pairs, we employ Focal-Loss [68] during training. Defined in Equation 5.1, the loss function uses $\alpha \in [0, 1]$ as a balancing coefficient to manage this imbalance. Additionally, $\gamma$ operates as the focusing parameter, accentuating the learning on challenging misclassified positive instances over the numerous trivial negative ones.
\[ FL(\text{pair}(c^i, c^j)) = \begin{cases} \alpha(1 - p^{(i,j)})^\gamma \log(p^{(i,j)}) & \text{if } y^{(i,j)} = 1 \\ \alpha(1 - p^{(i,j)})^\gamma \log(1 - p^{(i,j)}) & \text{if } y^{(i,j)} = -1 \end{cases} \] (5.1)

### 5.4 Experimental Evaluation

#### 5.4.1 Problem Formulation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>(c^i)</td>
<td>address cluster (i) cluster is formed based on the multiple-input heuristic rule.</td>
</tr>
<tr>
<td>(t_r^{c^i})</td>
<td>Time series of the daily count of receiving transactions for cluster (c^i).</td>
</tr>
<tr>
<td>(t_s^{c^i})</td>
<td>Time series for daily count of spending transactions for cluster (c^i).</td>
</tr>
<tr>
<td>(p^{(i,j)})</td>
<td>Probability that cluster (c^j) is temporally linked to cluster (c^i). Probability that pair ((c^i, c^j)) is positive.</td>
</tr>
<tr>
<td>(y^{(i,j)})</td>
<td>True label for pair ((c^i, c^j)). (y^{(i,j)} = \pm 1)</td>
</tr>
<tr>
<td>(\hat{y}^{(i,j)})</td>
<td>Model predicted label for pair ((c^i, c^j))</td>
</tr>
</tbody>
</table>

In this section, we use the notations in Table 5.2 to formulate the problem. An address cluster contains the non-zero number of blockchain addresses that are spent together in some transactions. The membership to a cluster, \(c^i\), is a transitive relation - if \(c^i\) and \(c^j\) are spent together in one transaction, and \(c^j\) and \(c^k\) are spent together in another transaction, then \(\{c^i, c^j, c^k\}\) are in one address cluster. This entails that the address clusters are disjoint. This work aims to identify one or more address clusters that can be temporal linked with \(c^i\) if there is a flushing event. We formulate this as a classification problem: *given a pair of clusters, \(c^i\) and \(c^j\), determine the likelihood that two clusters can be temporally linked.* In other words, the predicted label is positive if it shows a temporal linkage signature; otherwise, it is negative.

This formulation is amenable to training a self-supervised model without any labeled data, enabling us to exploit the whole of the Bitcoin blockchain for training.

#### 5.4.2 Data Description

**pseudo-labeled dataset:** Following the steps in Section 5.3.4, we generated 71K positive pairs and 5,933k negative pairs for self-supervised learning. Detailed numbers are in Table
Table 5.3: Data sets size

<table>
<thead>
<tr>
<th></th>
<th># of Pos. pairs</th>
<th># of Neg. pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>pseudo-labeled set</td>
<td>71,883</td>
<td>5,933,212</td>
</tr>
<tr>
<td>WalletExplorer labeled set (W.E.)</td>
<td>65</td>
<td>59,066</td>
</tr>
<tr>
<td>Manually labeled set (M.L.)</td>
<td>25</td>
<td>32,426</td>
</tr>
</tbody>
</table>

5.3 The data generation process does not involve any synthetic components, except for the randomly selected split points for the time series. This approach helps us create data that closely resembles real-world scenarios, which in turn guarantees the correctness of our approach.

WalletExplorer labeled dataset (W.E.): WalletExplorer [128] has been an essential resource for various studies on the Bitcoin blockchain [156, 51]. Out of the 275 services listed on WalletExplorer, 49 services have multiple address clusters. Nonetheless, not all of these cluster pairs exhibit flushing behavior, some manifest parallel usage patterns, while others serve as cold wallets for balance maintenance, among other purposes. We manually selected and verified the cluster pairs that have flushing behavior. We finalized 65 positive pairs from 26 services. The goal of this work is to locate the carry-on cluster from a flushed cluster so the negative pairs are generated to simulate this process. So each prefix from a positive pair will form a negative pair to all the other clusters on the blockchain. To reduce the number of negative pairs, we make some assumptions that the cluster for the negative suffix should have at least 100 addresses or 400 transactions. The suffix cluster and the prefix cluster need to share at least one sender or receiver. This is based on the assumption that the clusters owned by the same entity should share some common senders and receivers [147]. In total, we finalized 65 positive pairs from 26 services and more than 59,000 negative pairs (data available on the support website [13]).

Manually labeled dataset (M.L.): We manually found several clusters associated with two services: HelixMixer and Primedice. While these clusters were missed by WalletExplorer, our manual review of all the time series (conducted before the development of BitLINK) revealed strong address flushing patterns, and they form a cluster chain starting from a labeled cluster by WalletExplorer and ending with a confirmed cluster which was verified by using off-chain information that had unintentionally been leaked. We discuss this in Section 5.6.

Each data pair \((c^i, c^j)\) consists of four time series: \(tr^i, tr^j, ts^i, ts^j\). These series are vertically stacked to generate a \(4 \times 32\) matrix, serving as the input to the model. Notably, while the full-time series is 5035 days long, we extract a sub-sequence of 32 days,
positioning the flushing point at its center.

5.4.3 Setup

All experiments are reproducible with resources available on our anonymous support website [13]. We conducted experiments on a single RTX3090 GPU. We adopted weight initialization from [46] and trained BitLINK using Stochastic Gradient Descent (SGD) with a learning rate of 0.02. For the focal loss, we followed [68] by setting $\gamma = 2$ and the balancing ratio $\alpha$ to 0.98, based on the ratio of the number of positive pairs to the number of negative pairs in the pseudo-labeled dataset.

![Figure 5.8: Three receiving time series of three clusters owned by CrimeNetwork.cc, tagged by WalletExplorer. Both the blue time series and green time series are temporally linked with the same red one.](image)

Our primary evaluation metric is the top-5 accuracy, commonly used in computer vision to discern multiple objects in an image. Given the possibility of multiple carry-on clusters due to parallel time series (see Figure 5.8), our model must recognize all potential matches. We also track the top-1 accuracy and other metrics TPR, FPR, and G_mean defined in Equation (5.2) When computing the Top-k accuracy, we only consider the ranking of $p^{(i,j)}$ for each candidate cluster $c^j$. For computing TPR, FPR, and G_mean, $\hat{y}^{(i,j)} = 1$ if $p^{(i,j)} > 0.5$, and -1 otherwise.

$$G_{\text{mean}} = \sqrt{TPR \times (1 - FPR)}$$ (5.2)

5.4.4 BitLINK Performance

To evaluate the learning capability of our model on unlabeled data, we allocate 80% of the pseudo-labeled dataset for training and reserve the remaining 20% for validation, while using the W.E. (WalletExplorer labeled set) and the M.L. (Manually labeled set) for testing. It is important to note that the clusters used as prefixes in the training and validation sets are mutually exclusive to prevent information leakage. The training and
validation curve, as illustrated in Figure 5.9 (Left), shows rapid convergence, indicating the model is capable of effectively learning the artificially created temporal linkage signature. However, a higher validation score implies reduced generalizability, which can lead to lower performance on real-world datasets. This is due to the diversity of actual flushing behavior encountered in real-world scenarios. Given that our primary objective is to detect temporal linkage signature, a task closely aligned with this self-supervised learning task. We can leverage either the W.E. or M.L. dataset to establish an early stopping point. This approach is intended to mitigate the risk of overfitting, ensuring the model remains applicable and effective in practical scenarios.

To test the generalizability of the model, we leverage the entire pseudo-labeled dataset for self-supervised training, while the W.E. and M.L. datasets are used exclusively for testing purposes. The training and testing performance, measured by the G mean score after each epoch, is plotted in Figure 5.9 (Right). As we do not fine-tune the model with labeled data (i.e., no supervised phase), the testing accuracy observed across the training epochs serves as a crucial indicator of the model’s real-world performance when trained solely on the unlabeled dataset.

The notable correlation between the testing curves of the W.E. and M.L. datasets allows us to use either one for validation and the other one for testing, particularly for implementing early stopping to prevent overfitting. For example, using W.E. for validation and M.L. for testing, we observe that the testing score at the peak of the validation set is similar to the score at the tenth epoch – the point of maximum performance for the testing set. This observation remains consistent when the roles of the W.E. and M.L.
datasets are swapped between validation and testing.

5.4.5 Performance Comparsion

We compare with two state-of-the-art models SemiTime\cite{31} with semi-supervised learning and TS2Vec with self-supervised learning on time series data.

SemiTime\cite{31} and SSTSC \cite{135} are semi-supervised models for time series classification. They train encoders via self-supervised learning and then fine-tune the encoder with an MLP on labeled datasets. The self-supervised approach involves splitting time series into non-overlapping segments and predicting if segments come from the same series. Data augmentation techniques like magnitude and time warping\cite{125} ensure sufficient training pairs. We used the SemiTime encoder and relation head due to its two-segment input, whereas SSTSC uses three. The encoder has four convolution blocks, each with a 1D kernel convolution layer, batch normalization, and ReLU. The relation head features two linear layers with batch normalization and ReLU in between. To ensure a fair comparison, we used the same training data for our model, and we replaced the BCE loss with Focal loss. Additionally, we adjusted the first convolution layer from one to two channels to accommodate both receiving and spending transaction series.

TS2Vec \cite{145} stands as a leading self-supervised learning model for time series data, surpassing prior benchmarks like TS-TCC \cite{30} and T-Loss \cite{34} in classification tasks. TS2Vec also adopts a two-phase approach, initially trains an encoder unsupervised, then transitions to fine-tuning and testing using the representations from the encoder. The self-supervised approach relies on dual augmented context representations of two overlapping sub-sequences as positive pairs. The encoder formulates a latent vector at every timestamp given two overlapping segments. Vectors from the same timestamp across overlapping segments are positive, while those from different timestamps or different time series at identical timestamps are considered negative. The encoder utilizes a series of residual convolutional blocks. For our purposes, we input the complete time series data into the encoder without explicit segmenting. After training, we leverage the instance-level representations from the encoder to train a classifier, adopting the same MLP structure as BitLINK. We use the same training set to train the classifier, and the training data for the encoder is fundamentally the same data as well. SemiTime was trained with these parameters for enhanced performance. While the TS2Vec encoder used default parameters, its classifier shared settings with BitLINK.

Table 5.4 shows the testing performance on the W.E. dataset at the epoch when the best validation G\textsubscript{mean} is achieved on the M.L. dataset. BitLINK outperforms its
Table 5.4: Testing results on the W.E. dataset when G_mean of validation score is highest on the M.L. dataset. The numbers are the mean values for five rounds.

<table>
<thead>
<tr>
<th>Models</th>
<th>Top5 Acc.</th>
<th>Top1 Acc.</th>
<th>TPR</th>
<th>FPR</th>
<th>G_mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>SemiTime</td>
<td>0.7169</td>
<td>0.5231</td>
<td>0.56308</td>
<td>0.0083</td>
<td>0.7613</td>
</tr>
<tr>
<td>TS2Vec</td>
<td>0.8954</td>
<td>0.56</td>
<td>0.8646</td>
<td>0.0034</td>
<td>0.9282</td>
</tr>
<tr>
<td>BitLINK</td>
<td>0.9323</td>
<td>0.7539</td>
<td>0.95538</td>
<td>0.0170</td>
<td>0.9691</td>
</tr>
</tbody>
</table>

Table 5.5: Testing results on the M.L. dataset when G_mean of validation score is highest on the W.E. dataset. The numbers are the mean values for five rounds.

<table>
<thead>
<tr>
<th>Models</th>
<th>Top5 Acc.</th>
<th>Top1 Acc.</th>
<th>TPR</th>
<th>FPR</th>
<th>G_mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>SemiTime</td>
<td>0.84</td>
<td>0.464</td>
<td>0.2</td>
<td>0.0001</td>
<td>0.4427</td>
</tr>
<tr>
<td>TS2Vec</td>
<td>1</td>
<td>0.728</td>
<td>0.99</td>
<td>0.0041</td>
<td>0.9939</td>
</tr>
<tr>
<td>BitLINK</td>
<td>0.96</td>
<td>0.96</td>
<td>0.96</td>
<td>0.0014</td>
<td>0.9791</td>
</tr>
</tbody>
</table>

counterparts in several key metrics. Table 5.5 shows the performance when tested on the M.L. dataset, BitLINK performs comparably to TS2Vec with a notable improvement in FPR but a slight decrement in TPR, caused by the misclassification of a single positive pair within the dataset. hence, There is no statistically significant difference between BitLINK and TS2Vec on the M.L. dataset.

Figure 5.10: Confusion matrix for the testing results of BitLINK on W.E. and M.L. datasets when the best G_mean is achieved on the validation set.

We show the BitLINK prediction results in the confusion matrix in Figure 5.10. The TPR and FPR for each of the 26 services in the WalletExplorer dataset are shown in Figure 5.11(Left). A comparison between TS2Vec and BitLINK on each of these services
Figure 5.11: Left: TPR, FPR results for each service when tested with BitLINK. Right: G_mean of each service with respect to TS2Vec and BitLINK.

is shown in Figure [5.11](Right). BitLINK is better in G_mean on all but one service.

5.4.6 Discussion

Based on the results in Table 3,4, BitLINK demonstrates a near 95% top-5 accuracy. This capability is particularly beneficial for manually chaining clusters detailed in Section 5.5 to achieve robustness, marking a significant improvement over the manual approach that requires analysts to sift through nearly a billion clusters. BitLINK effectively narrows down the options to the top 5 candidates, simplifying and enhancing the verification process.

While there is room for improvement, we are committed to further enhancing the BitLINK model. Our goal is to develop a fully automated system capable of chaining multiple clusters with guaranteed robustness. We believe that the facilitated manual exploration by BitLINK in its current phase will aid in gathering more data, which is crucial for advancing the model. Acquiring such data, especially in the context of Bitcoin anonymity, remains one of the most significant challenges.

5.5 Chain of Linked Clusters

Clusters can be sequentially linked, resulting in their transaction time series forming a seamless chain over time, as illustrated in Figure [5.2](right). To achieve this, we deploy BitLINK iteratively. Starting with an initial cluster as a prefix or suffix, BitLINK ranks potential suffixes or prefixes, respectively. We then manually inspect the top-ranked
candidates, selecting the cluster that stands out. This selected cluster then serves as the new prefix (or suffix), and the cycle continues.

The manual inspection is more of a design choice than a requirement. Chaining could be done automatically using the top-1 candidates produced by BitLINK at each iteration. However, the manual inspection step allows room for prior information and subjective preferences. For example, if the top candidate has participated in a CoinJoin \[40, 80, 19\] transaction, we exclude that candidate cluster because off-chain arrangements are beyond the scope of our work. Another example is that multiple top candidates may exhibit the same pattern. Manual inspection reduces confidence in those clusters even if BitLINK ranks them high. In general, we observe that 70% of the linkages are formed using the top-1 candidate, and 90% of the links are formed using the top-5 candidates.

Table 5.6: BitLINK performance in adding more information about three services. The BitLINK columns show added information to what exists in the WalletExplorer columns.

<table>
<thead>
<tr>
<th></th>
<th>HelixMixer</th>
<th>Primedice</th>
<th>Bitcoin Fog</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WalletExplorer</td>
<td>BitLINK</td>
<td>WalletExplorer</td>
</tr>
<tr>
<td># of clusters</td>
<td>35</td>
<td>50</td>
<td>5</td>
</tr>
<tr>
<td># of address</td>
<td>246,116</td>
<td>200,845</td>
<td>204,797</td>
</tr>
<tr>
<td>BTC received</td>
<td>188,051</td>
<td>127,715</td>
<td>129,274</td>
</tr>
<tr>
<td>BTC spent</td>
<td>188,034</td>
<td>127,713</td>
<td>129,266</td>
</tr>
<tr>
<td>BTC received in USD</td>
<td>$57,778,646</td>
<td>$59,592,710</td>
<td>$58,099,361</td>
</tr>
<tr>
<td>BTC spent in USD</td>
<td>$57,772,326</td>
<td>$60,000,959</td>
<td>$58,055,279</td>
</tr>
</tbody>
</table>

5.6 Case Study

We present three successful cases of linking address clusters for three services. We argue that such chains of linked clusters are better discovered by BitLINK. We describe the results of chain discovery on three services in Table 5.6.

5.6.1 HelixMixer

HelixMixer is a Bitcoin mixing service, which pays a user back with another user’s funds and thus, mixes money for a fee. The purpose of this service is to make it harder to trace a transaction to its original sender. HelixMixer was operating between April 2014 to December 2017, when it was shut down by the FinCEN for violating anti-money-laundering laws \[24, 60\].

WalletExplorer detected 35 clusters that belonged to HelixMixer. Each cluster was found by using the service and then using the common-input heuristic to find other
addresses in the same cluster. The receiving time series for 34 clusters are shown in color in Figure 5.12. The earliest detected cluster (HelixMixer-old) was active from 2014-12-05 to 2015-01-19; the second earliest cluster is “HelixMixer-old11”, which started on 2015-05-23. We chose the ”HelixMixer-old11” as our first address cluster to form the chain.

BitLINK linked 50 new clusters starting with ”HelixMixer-old11” shown in dark in Figure 5.12. The newly identified clusters filled the gaps in the existing clusters perfectly and expanded the timeline to December 2017 when HelixMixer ceased operation. We found one leaked address ”1C2pqxbQTPbx4xWNYCn7fome1T8wfCzCC3” in a Reddit post [11] where the owner of HelixMixer declared the closure of the service. This leaked address is included in the cluster at the end of the chain. Overall, BitLINK linked address clusters of HelixMixer for over three years between 2015 and 2018.

We found 200,845 new addresses related to HelixMixer using BitLINK. It is almost double the number of addresses (246,116) WalletExplorer reported. In order to find these additional addresses, we needed to manually check only the top 5 candidate clusters yielded by BitLINK for each link. In contrast, a simpler alternative (e.g., similarity search) method would require sifting through thousands of clusters for each link.

This almost complete chain of address clusters enables us to make a closer estimate (i.e., lower bound) of the total money transacted via HelixMixer. Our estimate is that over $115M US dollars were transacted between 2014 and 2018 via this service.

5.6.2 Primedice

Primedice is a betting site that was launched on May 18, 2013, and it is still operating. Primedice uses many cryptocurrencies including Bitcoin for transactions. WalletExplore reports 5 address clusters plotted in color in Figure 5.13. We linked an additional 10 clusters that complete 100% of the transaction history of this service until the most recent
Figure 5.13: The time series of daily counts of receiving transactions for 5 address clusters owned by Primedice identified by the WalletExplorer, which are shown in color. 10 additional address clusters linked by BitLINK shown in black. The first cluster had its first receiving on 18, May 2013 which is exactly when Primedice was launched. Primedice is still operating, hence, we could collect an address in the last block by simply being a customer of Primedice and sending Bitcoin to it. "3R1hAXKgaQESrtyzaHsYwPFjgZerm7fSyW" is that address. BitLINK has successfully linked this address in the last address cluster shown in Figure 5.13.

5.6.3 Bitcoin Fog

Figure 5.14: The left figure displays the time series data for Bitcoin Fog and its most frequent receiver. On the right, there are six chained clusters chain linked from the most frequent receiver. These linked clusters also demonstrate a notable correlation with Bitcoin Fog. The correlation between them is calculated using a normalized DTW distance with a 2-day warping band.

Bitcoin Fog is the longest-running Bitcoin money laundering service on the darknet. The address cluster for Bitcoin Fog found in WalletExplorer had never been flushed, and spans over ten years as shown by the cyan-colored time series in Figure 5.14. However, we extract the largest cluster of addresses that Bitcoin Fog sent money to, and link that cluster. This cluster has 59% of transactions, counting 115,334 originating at Bitcoin Fog and 56% of sender addresses belonging to the Bitcoin Fog. We could chain more than eight of the ten-year history of Bitcoin Fog. Moreover, the two clusters show a very
high correlation in their transaction counts, suggesting an exclusive trusted relationship between the two clusters.

5.6.4 Linking at Scale

BitLINK has enabled us to link clusters at a scale of hundreds of crypto-based services. We have successfully formed chains on 60 services tagged by WalletExplorer utilizing BitLINK, with a 1-fold increase in the number of linked clusters. We organize this information on http://www.bitlinkwallet.com/

5.7 Conclusion

This chapter introduces a method, BitLINK, to link address clusters in Bitcoin blockchain based on their temporal pattern in receiving money. BitLINK is trained in a self-supervised manner and can shrink the search space of possible clusters significantly. Thus, BitLINK enables scaling up linking address clusters for the entire life span of a service of interest.
Chapter 6

Conclusion

The surge in time series data, particularly from real-time sources challenges current processing capabilities, necessitating algorithms that deliver instant results no slower than the rate of data generation. In online settings, algorithms must not only be robust to pervasive noise but also maintain long-term stability. Moreover, online environments frequently introduce novel events or experience concept drift, where data patterns and relationships evolve unpredictably. Traditional offline methods often fail in such scenarios as they rely on static training sets and cannot adapt to new, unforeseen data patterns. Given these challenges, this thesis contributes to developing methodologies tailored for robust and adaptive online processing of time series data as follows.

- We introduce FilCorr, an innovative algorithm designed to calculate filtered and lagged correlations across streaming time series data. By integrating filtering and cross-correlation in a singular process, FilCorr efficiently computes lagged correlations, surpassing existing methods in speed and efficiency. It is characterized by its precision, lack of sensitivity to parameters, scalability, and capacity to handle noise and asynchronous data streams, providing an immediate, model-free solution for deployment.

- We have provided a suite of algorithms for computing distance profiles under Euclidean distance, including a novel Discrete Cosine Transform (DCT) based MASS algorithm. This enhancement retains the computational efficiency of its predecessors while improving robustness.

- We develop a novel online few-shot model, FewSig, for time series classification. Capable of learning from a minimal set of positive examples and adapting iteratively to new, unlabeled data, FewSig sets a new benchmark for efficiency and applicability in real-time monitoring systems.
Our development of the self-supervised learning model BitLink leveraging the inherent temporal patterns of data to construct supervised learning tasks from unlabeled datasets. By capturing the dynamics within the data, BitLink facilitates efficient time series assembly tasks, transforming raw, unlabeled data into actionable insights.

The practical application of our algorithms across various scenarios, from real-time monitoring and online classification to anomaly detection, demonstrates their potential to enhance accuracy and efficiency significantly. We have successfully deployed our algorithms on diverse time series datasets from various domains, such as motion capture systems, accelerometers, EEG, ECG, seismometers, solar energy measurements via pyranometers, and transaction data from the Bitcoin blockchain.

This thesis has started to impact the research community with tangible impacts through real-world applications:

- Our development of Seisviz, a real-time seismic event detection system powered by FilCorr, has proven its worth in monitoring seismic activity in Yellowstone, WY, USA, with real-time earthquake detections showcased on our website. [149].

- The application of FewSig to aftershock detection in the aftermath of significant earthquakes in Nepal (2015) and Mexico (2017) highlights its utility in reducing the manual monitoring workload by effectively filtering aftershock events.

- Through BitLink, we have advanced the analysis of Bitcoin transactions, linking address clusters across 60 services identified by WalletExplorer. This achievement is documented and made accessible on our dedicated website (http://www.bitlinkwallet.com).
Bibliography


[53] CBS Interactive Inc. Inside the fbi takedown of the mastermind behind website offering drugs, guns and murders for hire, 2024.

[54] IRIS. The facilities of iris data services, and specifically the iris data management center, were used for access to waveforms, related metadata, and/or derived products used in this study. iris data services are funded through the seismological facilities for the advancement of geoscience and earthscope (sage) proposal of the national science foundation under cooperative agreement ear-1261681, 2018. https://www.ir is.edu/hq/ (visited on 2022-01-03).


[132] Qingsong Wen, Liang Sun, Fan Yang, Xiaomin Song, Jingkun Gao, Xue Wang, and Huan Xu. Time series data augmentation for deep learning: A survey. In


