Mining Previously Unknown Patterns in Time Series Data

by

Zhuoer Gu

A thesis submitted to The University of Warwick

in partial fulfilment of the requirements

for admission to the degree of

Doctor of Philosophy

Department of Computer Science

The University of Warwick

September 2017

Abstract

The emerging importance of distributed computing systems raises the needs of gaining a better understanding of system performance. As a major indicator of system performance, analysing CPU host load helps evaluate system performance in many ways. Discovering similar patterns in CPU host load is very useful since many applications rely on the pattern mined from the CPU host load, such as pattern-based prediction, classification and relative rule mining of CPU host load.

Essentially, the problem of mining patterns in CPU host load is mining the time series data. Due to the complexity of the problem, many traditional mining techniques for time series data are not suitable anymore. Comparing to mining known patterns in time series, mining unknown patterns is a much more challenging task. In this thesis, we investigate the major difficulties of the problem and develop the techniques for mining unknown patterns by extending the traditional techniques of mining the known patterns.

In this thesis, we develop two different CPU host load discovery methods: the segment-based method and the reduction-based method to optimize the pattern discovery process. The segment-based method works by extracting segment features while the reduction-based method works by reducing the size of raw data. The segment-based pattern discovery method maps the CPU host load segments to a 5-dimension space, then applies the DBSCAN clustering method to discover similar segments. The reduction-based method reduces the dimensionality and numerosity of the CPU host load to reduce the search space. A cascade method is proposed to support accurate pattern mining while maintaining efficiency.

The investigations into the CPU host load data inspired us to further develop a pattern mining algorithm for general time series data. The method filters out the unlikely starting positions for reoccurring patterns at the early stage and then iteratively locates all best-matching patterns. The results obtained by our method do not contain any meaningless patterns, which has been a different problematic issue for a long time. Comparing to the state of art techniques, our method is more efficient and effective in most scenarios.

To my grandfather, Gu Youlai (1926 — 2017),

and to all who fought bravely for freedom in World War II.

Acknowledgements

Biggest thanks and gratitude go to my supervisor Dr. Ligang He, without whom I would not have had the change to pursuing my research. His guidance will always be my invaluable treasure.

Sincere thanks to my parents. It is their selfless care, support, and love that make me who I am today.

Thanks to my fantastic lab-mates, Bo Gao, Chao Chen, Huanzhou Zhu, Shenyuan Ren, Pengjiang, Junyu Li, Bo Wang, Mohammed Alghamdi and Nentawe Gurumdimma who I shared fantastic time with. It has been a great pleasure to work with them.

Special thanks to rainy Tocil Wood for teaching me a lesson that a tempting path sometimes leads to nowhere, and you better head back before your shoes stuck in mire.

Declarations

Parts of this thesis have been previously published by the author in the following:

- Z. Gu, C. Chang, L. He, and K. Li. Developing a pattern discovery model for host load data. In *Computational Science and Engineering (CSE), 2014 IEEE 17th International Conference on*, pages 265–271. IEEE, 2014
- Z. Gu, L. He, C. Chang, J. Sun, H. Chen, and C. Huang. An efficient method for motif discovery in cpu host load. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2015 12th International Conference on*, pages 1027–1034. IEEE, 2015
- Z. Gu, L. He, C. Chang, J. Sun, H. Chen, and C. Huang. Developing an efficient pattern discovery method for cpu utilizations of computers. *International Journal of Parallel Programming*, pages 1–26, 2016. ISSN 1573-7640. doi: 10.1007/s10766-016-0439-0. URL `http://dx.doi.org/10.1007/s10766-016-0439-0`
- S. Ren, L. He, H. Zhu, Z. Gu, W. Song, and J. Shang. Developing power-aware scheduling mechanisms for computing systems virtualized by xen. *Concurrency and Computation: Practice and Experience*, 29(3), 2017

In addition, the following works are either in progress:

- (Unsubmitted) Segmentation Clustering Based CPU Host Load Pattern Discovery

- (Unsubmitted) Exact Optimal Previous Unknown Time Series Pattern Discovery

Sponsorship and Grants

- China Scholarship Council

Abbreviations

CPU	Central Processing Unit
DTW	Dynamic Time Warping
DDTW	Derivative Dynamic Time Warping
PAA	Piecewise Aggregative Approximation
SAX	Symbolic Aggregative approXimation
ASPL	Approximate Similar Pattern Locating
AR	AutoregRessive model
MA	Moving Average model
ARMA	AutoregRessive Moving Average model
ARIMA	AutoRegressive Integrated Moving Average model
ARFIMA	AutoRegressive Fractionally Integrated Moving Average model
HMM	Hidden Markov Model
DFT	Discrete Fourier Transform
DWT	Discrete Wavelet Transform
LCSS	Longest Common SubSequence
ECG	Electrocardiograph
VM	Virtual Machine
HPC	High Performance Computing
PSR	Phase Space Reconstruction
SVD	Singular Value Decomposition
SAM	Spatial Access Method
RIP	Relative Important Point

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The last decade has been witnessing great changes in computing systems which are gaining variety, availability, and performance rapidly. Types of computing systems are now ranging from wearable and mobile devices like smartphones and smartwatches to publicly available distributed computing systems such as cloud platforms and clusters. Meanwhile, with the prevalence of mobile devices and more accessible internet services, applications now tend to offload their computations and even store data to servers. Cloud services are prevailing for its cross-platform and device independent features due to the accessibility from remote ends, despite the devices the clients are using.

Though the performance of modern computing systems is enhanced exponentially on every aspect, the demands on these systems are also increased greatly and the requirements to these systems are changed in many ways as well. With a larger number and wider range of people employing local or remote computing systems, the needs of exploiting potential from computing systems are gathering consistent attention.

We focus on mining unknown patterns in CPU host load of cluster computing system, which plays an important role in today's real-world applications. Much research has been conducted to enhance the performance of single machines, distributed systems or clusters. An important approach to enhance the performance is analysing the features in the host load. In [43], Peter A. Dinda defined the host load as the number of processes that are running or ready to run. Later in [42], CPU host is the ratio of total time a CPU is occupied to the measurement period. We adopt the latter definition as it matches with common CPU usage measure method and intuitively more comprehensible.

Time series data are ubiquitous in the real world. Unlike static data that mainly concern the values of the data, time series data are collected with time and concern both values and changes of the data. The presence of time series data ranges from stock market price chart to ECG, covers almost every aspect of human activity. Not only a large portion of data collected in human history is time series data, many static data mining problems can also be converted to mining time series data to improve their efficiency or accuracy [155]. Therefore, mining patterns in time series data is a more general and applicable task and therefore, it has more potential value.

CPU host load is essentially time series data. Mining unknown patterns in CPU host load closely relates to mining unknown patterns in time series data. On the contrary, investigating pattern discovery in time series data helps many time series pattern related applications including mining CPU host load. We extend our interest to discover unknown patterns in time series data in later part of this thesis. Though our work on mining CPU host load data can be transferred to mining time series data with almost no effort, it is believed that investigating time series pattern discovery has more significance.

The objectives of this thesis are two folds: we develop methods for mining unknown patterns in CPU host load and propose novel methods for discovering unknown patterns in time series data. In this chapter, we first introduce the problem of mining unknown patterns in CPU host load data, then we extend the discussion to time series data to reveal the necessity and difficulties of the problem.

1.1 CPU Host Load, CPU Host Load Analysing, and Time Series Unknown Pattern Discovery

As the major (if not the only) component that handles all the calculation, the performance and usage of CPUs determine the performance of a computing system to a great extent. In modern computing systems, performance evaluations which partly rely on evaluating and estimating performance and usage of CPUs in these systems are necessary for many applications. According to [8], performance modelling has at least 4 applications:

1. System design. Systems designs vary when dealing with different types of load, and distinguish these types often requires locating patterns in system load.

2. System tuning. System load often involves different patterns which include periodical subsequences, chaotic subsequences, and smooth subsequences. In many cases, the occurrences of these subsequences are related in some way. To analyse the relation to respond accordingly, pattern discovery should be done in the first place.

3. Application tuning. Different applications bring different effects on the system's load, and these effects are likely to be in several different categories. Analysing these patterns which are brought to CPU load can help gaining performance for these applications.

4. System procurement. Analysing patterns in CPU host load helps to understand the needs for a computing system which aids procuring systems accordingly.

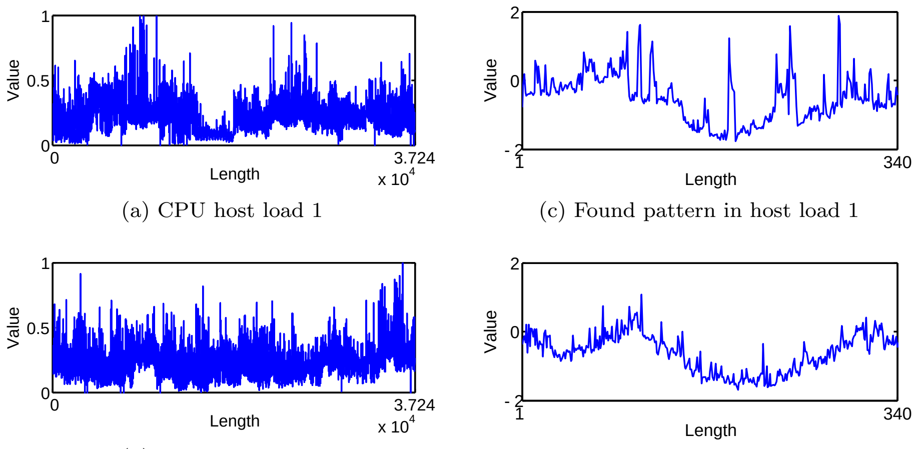
These applications all involve analysing and estimating CPU performance, which depends on analysing CPU host load of the system. Existing work on analysing CPU host load mainly concentrate on characterizing system workload and predicting CPU host load [41, 160]. The result can be useful for guiding scheduling strategies to achieve high application performance and efficient resource use [160].

However, locating patterns in CPU host load is less investigated. Discovery of unknown patterns in CPU host load is the task that finds the start and end positions of CPU host load subsequences that the two fond subsequences are similar. Figure 1.1 is an example of CPU host load pattern discovery. It is very useful to discover unknown repeated patterns in CPU host load of a computing system. Essentially, the CPU host load is time series data [60]. Many applications rely on pattern discovery in time series data [32], including:

1. The algorithms for mining association rules in time series data based on pattern discovery [38, 74].
2. Classification algorithms that are based on building typical prototypes of each class [70, 90].
3. Anomaly detection [39].
4. Finding periodic patterns [66].

Specifically, these applications of mining time series can easily find their counterparts in mining CPU host load. The use of mining patterns in CPU host load of a computing system includes but not limited to performance evaluation, system tuning, job and task scheduling, power usage optimization, fault detection and respond, CPU host load prediction and users' using habit mining etc.

Time series data, include CPU host load data, are ubiquitous. As mentioned previously, mining patterns in time series data is the subroutine of many real-world time series data mining applications. In bioinformatics, DNA can be regarded as time series data [9]. Patterns in DNA are often related to genes, which dominates organisms' characters. In astronomy, mined patterns from stars' light intensity changes are used to classify these stars. In medical science, locating abnormal ECG piece is based on the discovery of normal ECG subsequences in



(a) CPU host load 1

(c) Found pattern in host load 1

(b) CPU host load 2

(d) Found pattern in host load 2

Figure 1.1: Similar pattern discovery in two CPU host load

ECG data. By mining regularity of location information series generated by a smartphone user, a recommendation system is able to push accurate and useful information such as traffic information or weather information to the user.

Locating patterns in time series data has attracted a lot of attention. However, most of these existing work either require a predefined pattern and reduce the problem to the time series indexing problem, or computationally expensive. In reality, it is almost impossible for a domain expert to find all the patterns in a large dataset by experience. Moreover, there exist some patterns which are beyond the experts' knowledge. Therefore, an algorithm is needed to find unknown patterns automatically and efficiently. The algorithm should not require either the prior knowledge of the length or shape of patterns.

1.2 Challenges of Pattern Discovery in Time Series Data and CPU Host Load

Comparing to mining patterns in CPU host load data, mining patterns in time series data is a more general problem and, if there is such a method that can discover unknown patterns in time series data, we can modify and apply it to CPU host load. Due to the ubiquity of time series data, many real-life data mining tasks reduce to the tasks of mining time series data. These tasks often relate to locating known patterns in time series data or discovering common subsequences, or unknown patterns in time series data. Locating known patterns in time series data has drawn much attention in the last decades and the problem has been solved in many ways [2, 28, 57, 79]. However, the problem of discovering previously unknown patterns in time series database remains less focused.

As mentioned in [108] and [32], many applications rely on discovery of unknown patterns in time series data. In these applications, the task of locating unknown patterns in time series data reduces to the basic problem of discovering longest common subsequence pairs, also known as patterns, in two time series data. Figure 1.1 illustrates the problem intuitively, figure 1.1c and figure 1.1d are a pair of similar patterns found in time series 1 and time series 2 respectively.

Given the needs and importance of mining previously unknown patterns in time series data, an obvious method is using nested loops to compare all possible time series subsequences and find out those similar pairs. However, the time complexity of this brute force algorithm is as high as $O(n^6)$ [60, 62]. Though optimized by several methods [108], the approach is still unacceptable for mining patterns on large datasets. The reasons for the high complexity of a naive method should be owing to the fact that we do not have advance knowledge of the pattern we are to discover. Specifically, they are:

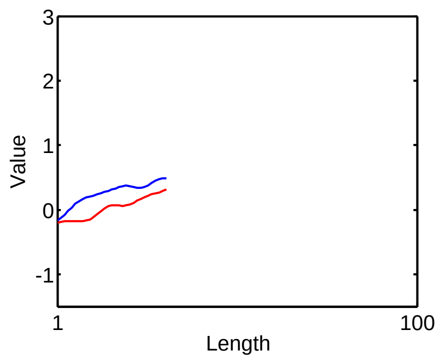
1. Position of the pattern. If we are able to acknowledge the positions or possible positions of patterns, the similarity measure can only be performed on these positions to find out whether they are similar to avoid unnecessary computation.

2. Length of the pattern. With the knowledge of pattern length, we can also increase mining efficiency greatly by removing those longer or shorter candidate subsequences from search space.

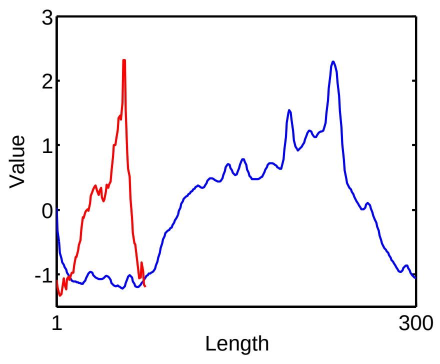
3. Shape of the pattern. If the shape of the pattern is known, the problem will reduce to search best match content by a given query problem, which

has been solved in many ways.

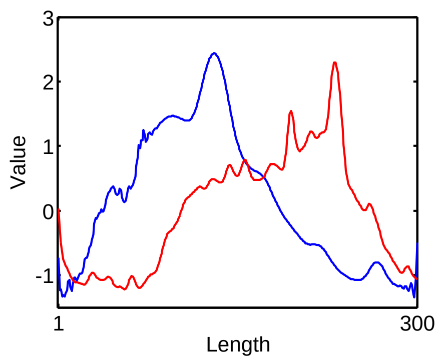
The brute force method also produces a large number of meaningless results. Figure 1.2 illustrates four typical meaningless results. Figure 1.2a,1.2b and 1.2c illustrate subsequences pairs that have relatively low distance value but are not intuitively similar, and figure 1.2d shows one subsequence (blue) is similar to two subsequences (red) from almost same positions of a time series. For the brute force method, picking out useful and accurate results from all results is even a more difficult task.



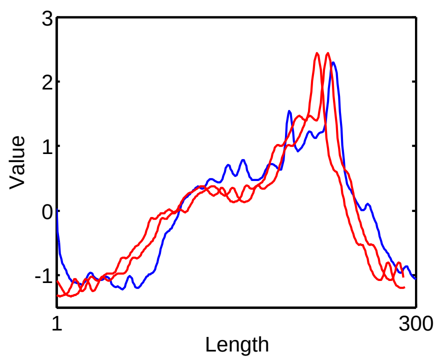
(a) Short similar subsequences



(b) Similar subsequences with extreme lengths differnence



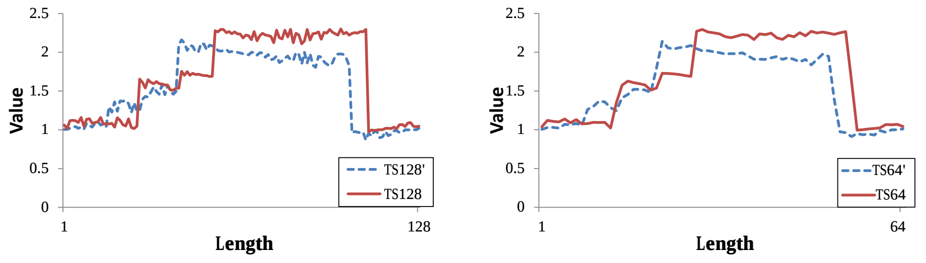
(c) Over-warped similar subsequences

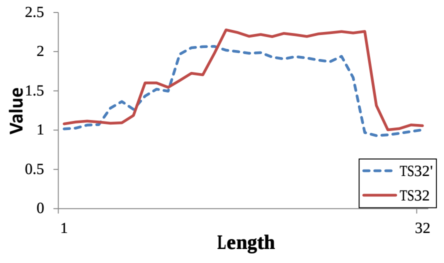


(d) Trivial matches

Figure 1.2: Four types of meaningless similar subsequences

Another problem of finding patterns with arbitrary lengths of is that most similarity measures are no longer practicable. For example, the two most commonly used distance measures, Euclidean distance and Dynamic Time Warping(DTW) distance tend to give higher distance value for the longer pair of time series being compared and lower distance value for the shorter pair of time series being compared. We use a simple experiment to show the effect of similarity bias as in figure 1.3.





Length	128	64	32
Euclidean	4.20	2.80	1.89
DTW	2.10	1.45	1.15

Figure 1.3: Example of similarity inconsistency. Top left: similar time series of length 128. Top right: similar time series of length 64. Bottom left: similar time series of length 32. Bottom right: Euclidean and DTW distance of the 3 time series.

In figure 1.3, we use a pair of similar time series as our example. The lengths of the data are originally 128 (top left), and we downsample the data to produce their shorter versions: top right with length 64 and bottom left with length 32. We measure the distance between the two time series in each pair with two mostly used distance measures, Euclidean distance and Dynamic Time Warping distance. Results show that though there is no obvious difference in similarity for each pair, their distances vary significantly. The properties of traditional distance measurements make mining patterns with different lengths almost impossible.

To summary, mining unknown patterns in time series data is a difficult problem due to its nature of high complexity caused by lack of prior knowledge and the exponential increase of problem scale.

Given the difficulty, some work related to pattern discovery has been done on time series data. However, mining CPU host load has its specific challenges.

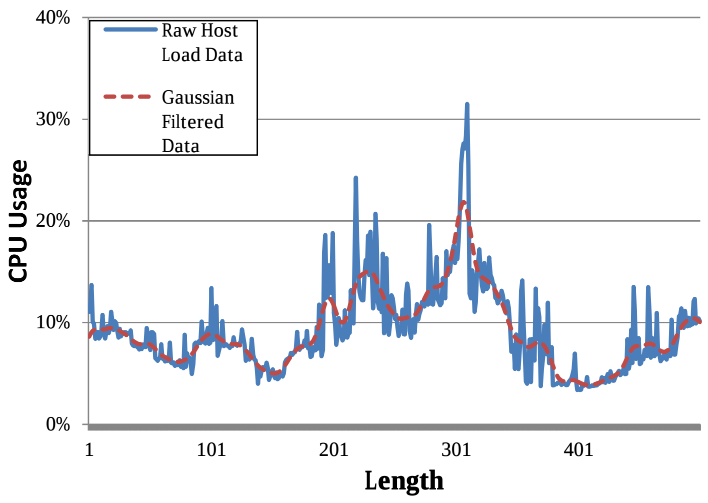


Figure 1.4: Illustration of Gaussian filter retaining the tread of raw data

CPU host load data are very noisy as shown in figure 1.4. The existence of noise in host load data could compromise the effectiveness of many data mining algorithms. Reduce or remove noise from data is necessary before any other data analysing task. As shown in figure 1.5, although one can easily identify the trend of a time series with noise, the presence of noise makes it very difficult for traditional similarity measure to produce consistent results as shown in figure 1.6. In [32], the authors illustrate that the pulse noise greatly affects similarity measure and casts difficulties in time series data mining. For similar reasons, the continuous noise will also reduce the effectiveness of classic similarity measure.

We conducted experiments and compared two time series data using two most classic similarity measures, Euclidean distance and Dynamic Time Warping (aka. DTW) distance. These two distance measures are widely used in many time series data mining techniques. The two selected time series data are three periods of sine waves and its slightly shifted version. They have similar trends and therefore should be considered as similar patterns. Figure 1.5 illustrates the experiment. Figure 1.5 a presents the two raw time series. Figure 1.5b is the noisy version of the time series in figure 1.5a. In figure 1.5c we can see that the original time series data are restored from the noisy data in figure 1.5b. The results of distance measurements are shown in figure 1.6. By adding Gaussian noise with the *variance* of σ^2, the effectiveness of both DTW distance and Euclidean distance measures are compromised severely. However, the Gaussian smoothed data have almost the same distance as the distance of raw data. These results indicate that when the noise is big enough, applying distance measure to untreated noisy data can no longer identify any similarity between two time series.

Another problem of discovering motifs for CPU host load is that the host load data do not obey the Gaussian distribution. This property of host load data weakens several effective indexing methods used for time series data. In [105], Lin et al. propose a promising symbolic indexing method for the time series data, which assumes that the time series data follow the Gaussian distribution. However, our observation shows that CPU host load do not obey Gaussian distribution. We compare the host load data with the Gaussian distribution in figure 5.2. It can be seen that more host load data points are at the lower end of the value range of the Gaussian distribution.

Given the challenges we are facing in mining time series patterns and CPU host load patterns, it is clear that CPU host load mining methods should solve these problems to present good results.

1.3 Pattern Discovery in CPU Host Load and Time Series: Our Contributions

CPU host load pattern discovery is a fundamental problem of multiple computing system applications. As stated previously, discovering patterns in CPU host load faces several difficulties. Though much research has addressed these problems, however, existing research either focuses on locating known patterns or requires

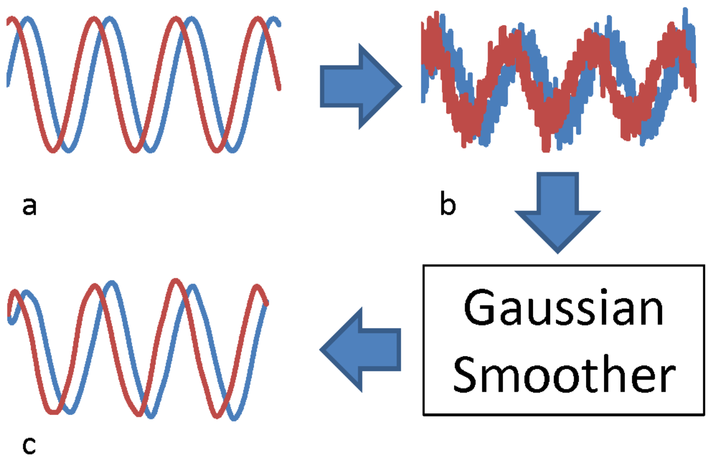


Figure 1.5: Illustration of noise impact: a) Raw data, b) Raw data with noise, c) Gaussian smoothed data

knowledge beyond one can provide in most scenarios.

In this thesis, we first extend the existing research and propose two different methods for mining unknown patterns in CPU host load, then present a novel method for mining unknown patterns in time series data.

The initial idea to discover patterns in CPU host load within a reasonable time is to apply divide-and-conquer method. Specifically, to avoid the high dimensionality problem of CPU host load data, a segmentation method is first applied to the data and each segment is then regarded as the minimum unit of CPU host load. By finding similarities of these segments using clustering method, it is possible to discover patterns from CPU host load.

Given the fact that segmentation method may damage the continuity of CPU host load and leads to the incapability of discovering all patterns, another effective method is to reduce the dimensionality of CPU host load by representing the data to a lower dimension. This concept motivates our dimensionality reduction based method for discovering patterns in CPU host load data. By applying several acceleration techniques we are able to find all patterns from CPU host load in reasonable time.

Though the above-mentioned work can also be applied to other time series

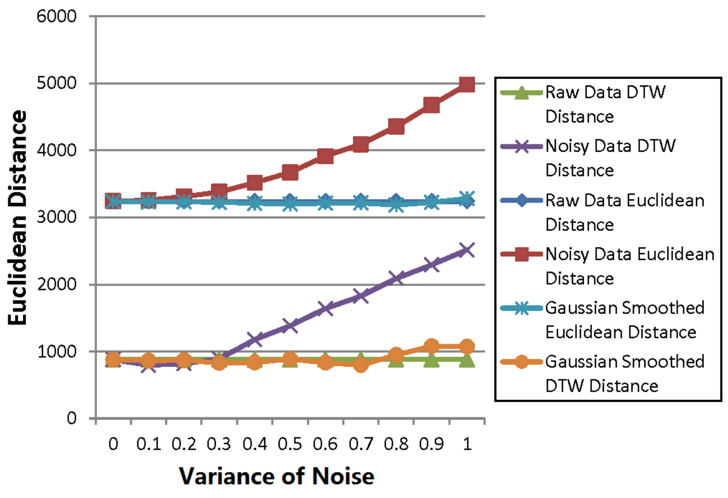


Figure 1.6: The effect of noise on distance measure and effectiveness of Gaussian filter

data, their applications are limited to mining fluctuating time series data. With the growing depth of our research, we find that it is necessary to propose a general time series pattern discovery method which, to the best of our knowledge, is neglected in the literature. Existing work in this field either requires prior-knowledge of pattern length or is computationally expensive. Our proposed method is able to discover unknown patterns of any lengths and shapes, and requires only two parameters as input. The method is proved to be efficient and robust to parameter settings.

More details of our work can be found in later chapters.

1.4 Thesis Organisation

This chapter provides a brief introduction to problems we are addressing as well as challenges we are facing. The next chapter, Chapter 2 reviews related work on distributed computing, data mining especially time series data mining and related machine learning techniques.

Background knowledge lies in Chapter 3. This chapter not only introduces some commonly used knowledge or notions in later chapters for the ease of further discussion, it also contains part of our work on solving general and basic problems of CPU host load pattern discovery and time series pattern discovery. Our contributions in this chapter are consistent similarity measure and CPU host load noise reduction, which are applied many times in later chapters and proved to be effective.

Chapter 4 proposed a CPU host load pattern discovery method based on segments clustering in feature space. Chapter 5 resolves the host load pattern discovery problem from a raw data reduction perspective. Lastly in Chapter 6 an exact and best-match unknown pattern mining method for time series data is proposed.

Chapter 7 concludes our work and suggests further research directions.

Chapter 2

Literature review

2.1 Introduction

In the past decades, analysing the performance of computing systems has drawn much attention. Specifically, performance modelling for a computing system can be used to gain a greater understanding of the performance phenomena involved and to project performance to other system/application combinations [8]. Mining and analysing CPU host load is one of the major components of evaluating the performance of a computing system. Much research has been conducted in this area.

Much work has been done on characterizing workload which includes modelling CPU host load [23, 33]. The existing work with respect to CPU host load mainly focuses on predicting CPU host load of a variety of computing system due to the needs (e.g. job scheduling, energy saving etc.) of knowing future CPU host load [13, 16, 45, 103]. Many methods have been proposed to predict CPU host load. The most used method is applying classic linear model like AR, MA, ARMA, ARIMA, and ARFIMA [21]. Other methods include neuro-fuzzy network, Bayesian model, Hidden Markov Model etc.

Though with the sheer volume of research addresses predicting host load, mining unknown similar patterns in CPU host load remains less focused. As a subcategory of time series data, techniques and concepts proposed in time series data mining can often be applied to mining CPU host load. Many techniques for predicting time series are transferred to predicting CPU host load, however, mining unknown similar patterns is barely investigated in the domain of time series data mining. Existing research on time series data mining contains the following aspects:

1. time series similarity measure [30]. Similarity measure compares two time series and gives a value that represents the similarity of the two time series. The similarity measure is of fundamental importance in time series data mining and it attracts much attention.

2. Time series segmentation [157]. Given a long time series, it is often necessary to segment the time series under different methods for further processing e.g. indexing, clustering etc.

3. Time series data representation [105, 157]. Represent the original time series data with another representation can reduce the storage needed, remove noise and remain the features of time series in the meantime.

4. Time series indexing [105, 106]. Indexing time series is to respond to a given time series query with the most similar time series or time series subsequence in a time series database.

5. Time series clustering [84]. Clustering analysis of time series locates different types of time series. Time series clustering methods involve the method of similarity measure, time series representation, and data clustering method itself.

6. Time series classification [90]. Time series classification methods investigate how to label unlabelled time series data fast and accurately when given a labelled time series dataset.

7. Similar pattern locating in time series [141]. Locating unknown subsequences from time series dataset is to find similar subsequences from

different time series.

For our CPU host load pattern discovery propose, we concern time series similarity measure, segmentation, representation, indexing, clustering and pattern discovery. The rest of this chapter reviews two categories of current literature: CPU host load analysis and time series data mining.

2.2 CPU Host Load Analysis

Performance estimation and optimization of computing systems has attracted many researcher's interest [64, 101, 102, 153]. Analysing and mining CPU host load of a computing system are very important for further optimization tasks. Much research has been conducted on CPU host load.

2.2.1 Statistical Features of CPU Host Load

In [43], Dinda analysed several statistical properties of CPU host load collected from a working cluster. The analysis found several statistical properties of CPU host load as follows:

1. The traces exhibit low means but very high standard deviations and maximums. This implies that these machines have plenty of cycles to spare to execute jobs, but the execution time of these jobs will vary drastically.

2. Absolute measures of variation are positively correlated with the mean while relative measures are negatively correlated. This suggests that it may not be unreasonable to map tasks to heavily loaded machines under some performance metrics.

3. The traces have complex, rough, and often multimodal distributions that are not well fitted by analytic distributions such as the normal or exponential distributions, which are particularly inept at capturing the tail of the distribution. This implies that modelling and simulation that as-

sumes convenient analytical load distributions may be flawed. Trace-driven simulation may be preferable.

4. Load is strongly correlated over time but has a broad, almost noise-like frequency spectrum. This implies that history-based load prediction schemes are feasible, but that linear methods may have difficulty. Realistic load models should capture this dependence, or trace-driven simulation should be used.

5. The traces are self-similar with relatively high Hurst parameters. This means that load smoothing will decrease variance much more slowly than expected. It may be preferable to migrate tasks in the face of adverse load conditions instead of waiting for the adversity to be ameliorated over the long term. Self-similarity also suggests certain modelling approaches such as fractional ARIMA models and non-linear models which can capture the self-similarity property.

6. The traces display epochal behaviour in that the local frequency content of the load signal remains quite stable for long periods of time and changes abruptly at the boundaries of such epochs. This suggests that the problem of predicting load may be able to be decomposed into a sequence of smaller subproblems.

In [41], Di et al. analysed some statistical properties of CPU host load data in a Google cluster [149] and compared it to another grid system. According to their research, Google hosts maximum CPU load is often close to the CPU capacity, and the maximum memory usage is about 80% of the memory capacity. The maximum load is actually controlled in the Google system for guaranteeing the service level of requests in case of unexpected load spikes. In contrast, Grid resources can be highly utilized without having a high risk of losing users or customers. Meanwhile, CPU and memory usage changes every 6 minutes, indicating again the volatility of load. CPUs are often idle, but memory usage is relatively high. CPU usage in Grids is higher and more stable. The noise of CPU load in the Google cluster is 20 times as high as that in Grids.

Zhang et al. [160] found that host load exhibits the extremely periodical patterns in short terms, while in a long period, the pattern of host load changes dramatically.

2.2.2 Mining and Prediction of CPU Host Load

Mining and predicting CPU host load can help improving the performance of a computing system in many ways and thus attracted much interest. Much research has been done on the characterization of workload. Various models have been proposed to characterise workload. Maria et al. [23] summarized the modelling as 5 steps: 1) formulation, 2) collection of the parameters, 3) statistical analyses of the measured data, and 4) representativeness.In [33], Cirne et al. proposed a comprehensive model for supercomputer workload. Different techniques of modelling workload to estimate performance of a computer system are proposed [7, 10, 36, 48].

Khan et al. [94] identified different types of CPU work load of virtual machines (VMs) of a cloud computing system. The authors applied a novel co-clustering analysis to group VMs with different host load changing patterns. Predictions of future system performance can be used to improve resource selection and scheduling [160]. Much work has been proposed on predicting CPU host load. Linear models are most used in predicting CPU host load. Specifically, these models are AR, MA, ARMA, ARIMA, and ARFIMA which are commonly used in predicting time series [21]. Peter A. Dinda tried to predict host load by using linear models [45], which reveals that host load is strongly autocorrelated. The author further evaluated all linear models for host load prediction in [44]. Their study strongly shows that host load on a real system is predictable to a very useful degree from pat behavior by using linear models. Zhang et al. [160, 161] applied a modified polynomial fitting to CPU host load data with consideration of sudden changes of CPU host load to reduce the error rate of prediction.

Beside linear models, other prediction techniques are also applied to CPU host load. Di et al. [42] proposed a Bayesian model-based method to predict host load. Khan et al. [94] used Hidden Markov Model (HMM) and achieved higher prediction accuracy. In [152], Yang et al. proposed a novel method for CPU host load prediction. The method uses phase space reconstruction (PSR) to extract features of CPU host load, and then uses the genetic algorithm to optimize parameters of the phase space reconstruction process. In a distributed computing system, the nature of CPU host load varies with different physical machines. Bey et al. [13, 15, 16] tried to distinguish different types of CPU host load using clustering method and then apply adaptive network based fuzzy inference system to each individual cluster of host load to predict CPU host load. Cao et al. [24] also addressed the problem of different host load types and applied dynamic ensemble model to adjust prediction parameters dynamically. A tendency based prediction method is proposed in [151] for one-step-ahead CPU host load prediction. Another tendency based long-term prediction method is proposed in [103].

2.3 Time Series Data Mining Techniques

In the last decades, many time series data mining techniques have been proposed. The most fundamental tool is similarity measure. Much research also segments time series for further processes to reduce the complexity of the problem. Similarly, indexing time series and dimensional reduction of time series data also help reduce the resource required of a time series data mining task. Clustering methods are commonly used in time series classification and similarity detection.

2.3.1 Similarity Measure

Similarity measure, also known as distance measure, is of fundamental importance for time series data mining. Similarity measure is a function $d = D(x, y)$ where d is a value that reflects how similar the time series x and y are. A large number of similarity measure methods have been proposed for general or specific time series similarity measure.

According to [49], an ideal similarity measure should provide the following properties:

1. It should provide a recognition of perceptually similar objects, even though they are not mathematically identical.
2. It should be consistent with human intuition.
3. It should emphasize the most salient features on both local and global scales.
4. A similarity measure should be universal in the sense that it allows to identify or distinguish arbitrary objects, that is, no restrictions on time series are assumed.
5. It should abstract from distortions and be invariant to a set of transformations.

The most well-known measure is Euclidean distance. Euclidean distance is a simple similarity measure which has good applicability in many real-life problems. Essentially, Euclidean distance is a L_p norm distance measure. Much research has pointed out the defects of L_p norms [46, 83, 156]. Though with many drawbacks, Euclidean distance is still widely adopted in many applications [32, 51, 87] and can still produce reasonably good results due to its simplicity and efficiency. Some techniques have been proposed to improve the robustness of Euclidean distance. Golding et al. [59] proposed a preprocessing step to handle noise and scaling problems under Euclidean distance.

Dynamic Time Warping (DTW) [14, 118, 148] are able to measure the distance between different subsequences with different lengths. The method elastically aligns peaks and valleys of time series despite their exact positions are slightly different.

DTW distance is robust and widely applicable, but its time and space complexity is much higher than Euclidean distance [85]. Keogh et al. [85] proposed a method to reduce its requirements to the computational resource. Several constraint methods have been proposed to optimize DTW. Early in the 1970s, Itakura et al. [76] and Sakoe et al. [133] proposed two constraint methods respectively which can be applied to cost matrix of DTW to improve its performance on speech recognition. Ratanamahatana et al. [129] proposed an adaptive constraint to DTW for increasing accuracy in DTW based time series classification.In [134], Salvador et al. proposed Fast DTW which first reduce dimensionality of time series and apply DTW to acquire a guess of warping path, then project the path to the original dimension of the two time series to be measured, with further process to refine the path to make it close to the real warping path. To prevent over-warping of DTW, DDTW [92] is proposed to find the best alignment rather than the lowest cost alignment. In [116], DTW is used for retrieve time series data.

Beside Euclidean distance and DTW distance which are shape based distance measure, there are other types of distance measure: edit based distance measure, feature based distance measure, structure based distance measure, model based distance measure and compression based distance measure.

Edit based distance measure compares the minimum operations required to transform one time series to another [49]. Longest Common SubSequence (LCSS) [20, 37] is a typical edit based distance measure. This method measures similarity by observing the extent of introducing outliers, scaling, translation, adding or removing some data points to transform one time series to another. Several improvements on LCSS were also proposed [114, 143, 144].

Feature based distance measure compares parameters which reflect some features of time series data. Measuring similarity by comparing coefficients of DFT [78, 139] or DWT transformed time series have been proposed. Zeng et al. [157] also proposed a similarity measure based on extracting features from segments of time series.

Several other similarity measure methods are also proposed in recent decades. Structure based similarity measure [104] aims to identify higher level similarity among time series which overcomes the incapability of traditional Euclidean distance and DTW distance. Model based similarity measure assumes time series follow some underlying models. Once the model has been used to achieve best fit to time series, measuring similarities are comparing the coefficients of that model. These models include HMM [57, 150], ARIMA [150]. Compression based similarity measure [40, 89] is rooted in the fact that the compressing ratio of compressing similar data is higher than that of dissimilar data.

2.3.2 Time Series Segmentation

The continuity nature makes it difficult to manipulate and analyse the time series data [55]. To find similar patterns in time series data, one feasible approach is to classify the segmented data pieces, also known as subsequences, by their similarities (i.e., distances). Major segmentation methods can be divided into three categories: sliding window, top-down and bottom-up [88].

Sliding window segmentation

The sliding window method is among the most used methods [35, 68, 125, 146]. Under sliding window method, a segment of time series grows until certain criteria are met citechen2005making. Though in [84] clustering width-fixed sliding window segments is shown to be meaningless, however, many other sliding widow segmentation methods are proved to be effective and efficient [34, 56].

[96] suggests that for some time series data, it is possible to speed up the sliding window process by increasing the step length to a certain value rather than 1. Another optimization of sliding window segmentation is proposed by [145]. Vullings et al. [145] suggest that the window width can be given by estimating the residual error of value at average segment length, then increase or decrease to a better window width until the optimum is reached. This method accelerates sliding window segmentation greatly.

In [121] Park et al. proposed a novel method which segments time series by their monotonicity. The method segments time series to subsequences which are monotonously increasing or decreasing. Obviously, this method is applicable only to smooth time series, as for noisy time series, the method will produce too many very short subsequences unless a noise reduction step is applied beforehand.

In [157], a time series segmentation method based on relative important point is proposed. Since relative important points are those local minimums and maximums, where are also noise has the most influences on, so the method cannot be applied to noisy data. Obviously, the method produces subsequences with different lengths. Also, one segmenting algorithm cannot fit all data and show satisfactory performance in all applications [83].

Several other sliding window segmentation algorithms are also popular in the medical data domain for its online nature [75, 96, 145].

Top-down segmentation

The top-down segmentation partitions time series until some criteria are met. In [47] Douglas et al. proposed a top-down segmentation algorithm which is well-known in cartography. Ramer et al. proposed a similar algorithm in the field of image processing to approximate curves with polygons [127].

[100] use top-down algorithm to give multiple abstractions of time series data for data mining. In [139] Shatlay et al. use top-down segmentation method to support time series indexing for large data set. In [119] Part et al. proposed a segmentation method which first locates all local maximum and minimum data points and then applies top-down segmentation for each segment.

Top-down segmentation can also be applied to text data [98] to discover the effect of news to the financial market.

Bottom-up segmentation

Unlike the top-down method, the bottom-up segmentation merges data points of a time series to form segments to met some stopping criteria.

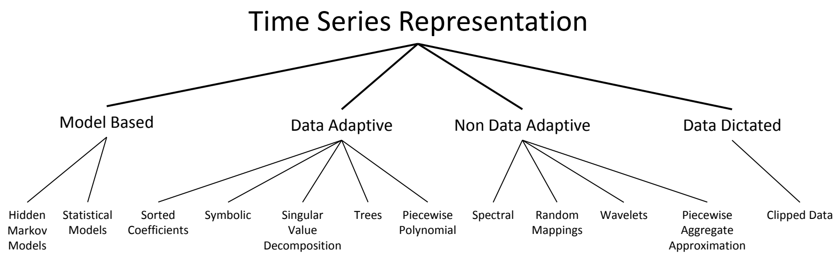


Figure 2.1: A hierarchy of time series representations in the literature

[90, 91] tried to index time series data with bottom-up segmentation and proposed a similarity measure based on the method to respond to the pathological output traditional Euclidean distance can give. In [93], Keogh et al. proposed a bottom-up linear segmentation method to probabilistically match patterns in time series data. In the field of medical science bottom-up segmentation method also used to represent medical data in higher level [73].

2.3.3 Time Series Data Representation

The problem of represent time series has attracted much attention in recent decades [4, 71, 131, 137]. According to [105], time series representation methods fall into several categories as shown in figure 2.1.

Essentially, representing time series data is transforming time series data to another dimensional space, typically a lower dimensional space, from its original space under a certain transform function. Represented time series can then be used for further tasks like indexing, clustering, and classification. Different time series representation methods are proposed for specific or general time series data types. As in figure2.1, most of them can be classified into three categories: data adaptive, non data adaptive and model based.

Non data adaptive representation remains the same parameters regardless the type and value of data. Faloutsos [51] proposed a Discrete Fourier Transform (DFT) based time series representation. Chan et al. proposed a Discrete Wavelet Transform (DWT) based time series representation in [28]. A number of wavelets have been applied to DWT methods. Chan et al. proposed a representation uses Haar wavelet [27] and [124] applied Daubechies wavelet. Coiflets wavelet also used to represent time series [138]. These methods transform time series from the time domain to frequency domain to utilize frequency information of time series.

In [128] Ratanamahatana et al. proposed a time series representation which uses bits to represent time series. The representation method can greatly reduce the space required for some time series data mining tasks, however, for most time series mining tasks, the representation is too coarse. Keogh et al. [87] proposed Piecewise Aggregate Approximation(PAA) which represents each time series subsequence of equal lengths with a constant value.

Data adaptive representation method changes its parameters adaptively with different data. Several piecewise linear representation of time series data is proposed in [58, 86, 139]. Adaptive Piecewise Constant Approximation method which is extended from PAA represents time series time series subsequences of varying lengths with constant values to gain accuracy. These methods have been popular for their simplicity and efficiency. SAX [105] also extends PAA to a data adaptive representation. The method uses symbols to represent time series rather than values, each symbol denotes a range of values. The value range of each symbol is adaptive to the change of time series data to guarantee the approximately same occurrence probability of these symbols. However, as suggested by [22], SAX representation does not guarantee the equiprobability of symbol occurrences in the original space of the time series data. This defect may further hamper the effectiveness of the representation method and any data mining algorithms that are based on it.

Singular Value Decomposition (SVD) is also used for time series representation [86]. This method is able to filter out noise while maintaining the trend of time series data.

Model based time series representation assumes that time series can be produced by a proper model. This representation tries to find the best fit model of a time series and represent the time series by parameters under that model. Kalpakis et al. [79] applied ARIMA model to time series. Markov Chains [136] and Hidden Markov Model [117] are also used to represent time series.

2.3.4 Time Series Data Indexing

Indexing time series data relates closely to represent time series data. Indexing time series is retrieving the most similar time series or time series subsequence from a time series database with a given time series query. Many time series indexing methods have been proposed in the literature [11, 12, 105].

According to [51] and [87], an indexing method should be much faster than sequential scanning, require little space overhead, able to handle queries of various lengths, allowing insertions and deletions without rebuilding the index, no false negative reports and able to adapt to different distance measures.

A time series with n data points can be regarded as a point in an n-dimensional space. This concept leads to indexing time series with Spatial Access Methods (SAM) such as B-trees [11] and R-tree [12]. These methods are not designed specifically for high dimensional and sequential data which are nature of time series data, thus their effectivities are greatly degraded when dealing with time series data [19, 26].

A suffix tree [65] based indexing method is proposed by Part et al. [120]. The method is based on the fact that distance computation compares the prefix firstly, then suffix. Though this method is efficient and effective in some way, it is difficult to index long time series or similar time series with little distance differences. The GEMINI [51] method is compatible with any dimensionality reduction method to speed up indexing process. Similarly [156] proposed an indexing method that can use arbitrary L_p norms distance measure.

[85] introduced the concept of lower bounding for time series indexing method. The lower-bounding feature of an indexing method can provide exact indexing. A symbolic aggregate approximation (SAX) representation proposed by Lin et al. [105]. The method reduces the time series dimensionality by applying

Piecewise Aggregate Approximation (PAA) [87], and then symbolises each PAA segment to obtain a discrete representation. The method provides quick indexing speed and requires little space. This method is lower-bounded which guarantees no false-positive matches.

Agrawal et al. [2] adopted Discrete Fourier Transformation to index time series data which indexes the frequency information of time series rather than time domain. Keogh et al. proposed adaptive piecewise constant approximation in [86] to index large time series database. The method approximates each time series by a set of constant value segments of varying lengths such that their individual reconstruction errors are minimal. In [87] a dimensionality reduction method, piecewise aggregate approximation is proposed. The method reduces time series dimensionality by resampling time series data. Keogh et al. [85] proposed an index method which adopts DTW distance while maintaining indexing speed. A fast indexing method which is capable to mine trillions of time series subsequences under DTW distance is proposed in [126]. In this work, many acceleration methods are developed to improve the speed of DTW measure and indexing.

In [90], the influence of noise on mining time series data is shown. Some noise resistant mining algorithms are proposed [106] to work with time series data with heavy noise.

2.3.5 Time Series Clustering

Clustering is the process of finding natural groups, called clusters, in a dataset [49]. The aim of clustering time series is to find time series that share the similar properties and are distinctive from other time series. Naturally, one can either cluster whole time series or time series subsequences in a time series dataset. Though as mentioned clustering sliding window segmented time series [29] has shown meaningless [84], clustering of whole time series or non-sliding window segments still produces meaningful and applicable results.

Clustering is an unsupervised machine learning technique. As a generic data mining technique, many clustering methods have been proposed [53, 54, 80, 99].

According to [67], clustering algorithms can be classified into five categories: partitioning methods, hierarchical methods, density-based methods, grid-based methods and model based methods. The brief explanation of each category is as follows.

Given a dataset with n data elements, a partition-based clustering method assign each data element into k groups where $k \leq n$ so that each group contains at least one element. Among all these partition-based clustering methods, hard clustering methods assign an object exactly to one group and soft clustering assign an object a label which records how much degree the object belongs to each group. k-means [110] is a typical partition-based algorithm. k cluster centres are given at the beginning of the algorithm, and each data element is then allocated to its nearest cluster centre. The position of each cluster centre is adjusted to the centre of data elements which are allocated to it. The process keeps looping until all cluster centre convergent to a certain position. Instead of using a centre position as the centre of a cluster, k-medoids [82] uses the most central data element as the representation of the cluster centre. Fuzzy c-means [17] and fuzzy c-medoids [97] algorithm are soft-clustering counterparts of k-means and k-medoids. These algorithms work well for discovering spherical clusters, however, their performance is compromised when the natural groups in a dataset are not spherical. Meanwhile, these methods require users to decide the number of clusters and their initial positions in the data space, which is critical to the final results of clustering. In [123], Pelleg et al. proposed a method which estimates the value of k to address the problem.

Hierarchical clustering methods group data elements into a tree of clusters. According to the different method of growing the tree, there are two types of hierarchical clustering, top-down (divisive) clustering method and bottom-up (agglomerative) clustering method. Top-down method regards the whole dataset as a cluster, and then divide the cluster into two most unrelated clusters. The process iteratively divides clusters until each cluster contains only one data element. Bottom-up method, on the contrary, regards each data element as a cluster, and merge the most related data elements until all data elements are in the same cluster. The defect of hierarchical clustering is that a cluster is not adjustable once merged or divided. The natural cluster can be mistakenly divided or merged in many cases. Chameleon [81] and CURE [63] tried to measure linkage of objects to improve the performance of hierarchical clustering, and BIRCH [158] optimize the method by relocating results iteratively. CLUBS [112] also addressed the defect by analysing the clustering results and reallocating clusters. In [132], Rodrigues et al. also proposed a hierarchical clustering method specifically for streaming data.

Density based clustering method groups data elements based on their distances to their nearby data elements, i.e. density. Apparently, this method is capable of finding clusters of different shapes and sizes. DBSCAN [50] and OPTICS [5] are two typical algorithms among density based clustering methods. Density based methods require the user to define a proper density value as their only parameter and do not group all data elements (data elements in sparse areas). These features make the method robust and simple.

Grid-based method [3, 135] embed a grid into data space to provide either multi-resolution clustering results or fast clustering. STING [147] is a typical grid clustering algorithm. The method partition data space with rectangles of different sizes based on the number of data elements they contain. The grid contains different levels of rectangular cells correspond to different resolutions, allowing multi-resolution queries.

Model based methods fit models to each cluster to achieve the lowest error. AutoClass [1] uses Bayesian model to estimate the number of clusters in a dataset. ART [25] and self-organizing map [55, 95] use neural network approach to cluster dataset.

2.3.6 Similar Pattern Locating in Time Series Data

Locating unknown similar subsequence in time series data has attracted less interest. However, some research has been conducted on the problem of locating similar subsequences in a single time series, also known as motif discovery.

Patel et al. [122] defined motif formally as typical nonoverlapping subsequences. Two motif discovering algorithms are proposed in [122]. The methods first reduce dimensionality and numerosity of time series and then apply motif discovery algorithms. In [108], Lin et al. developed a symbolic representation-based motif discovery algorithm to find the k^{th} most similar motif. A probabilistic method for motif discovery is proposed in [32] to accelerate the process. The method tries to match arbitrary subsequence of one time series to an arbitrary subsequence of another time series and analysis the statistical pattern of a result matrix to locate the position of patterns. This method requires the length of pattern to be known in advance. In [142], a motif discovery method for multi-dimensional data is proposed. These methods consider finding motifs from generic raw data. [115] presented a method of discovering exact motifs (the length and the shape of the motif are known in advance). The method gives a possible way to find exact motifs from short time series in a reasonable time. However, the time complexity of the method is still very high, which makes it not practical to work with long time series. Ferreira et al. [52] proposed a method that extracts approximate motifs. Liu et al. [107] formalize the motif discovery algorithm as a continuous top-k motif balls problem in an m-dimensional space. Heuristic approaches are applied to their motif discovery method to improve the quality of results. Bhaskar et al. [18] addressed the top-k motif problem under a sensitive-data scenario.

These methods above requires the length of the motif to be predefined, which is difficult to acquire for an unknown dataset. Tang et al. [140] addressed the problem by extending k-motif algorithm to discover motifs with random length and occurrence. Toyoda et al. [141] proposed a novel method to discover motifs in time series. The method creates two matrices, score matrix and position matrix respectively. The score matrix locates the end position of the found pattern, and the position matrix uses the information collected from the process of computing score matrix to calculate the start position of the pattern. This method is able to find patterns of any length and shape without producing abundant results and reduces required time and space complexity greatly in the meantime. Moreover, this method only needs 3 parameters and does not require prior knowledge of the patterns to be found. However, this method is still lack of scalability thanks to the exponential increase of matrix size with the accumulation of time series. The unconstrained application of DTW can also result in producing over-warped patterns.

Many contributions have been made to current motif mining algorithms. Yankov et al. [154] applied uniform scaling to discover motif of different lengths. Mohammad et al. [113] tried to introduce prior knowledge to mine motifs. Zhang et al. [159] applied motifs to time series classification to accelerate the classification process.

2.4 Summary

In this chapter, we carefully reviewed related work on CPU host load analysis and time series data mining. These work comprehensively addressed statistical features of CPU host load and characterization of the workload of a computing system, and further mining CPU host load mainly by transferring techniques in time series data mining. Researcher in time series data mining domain also proposed outstanding work in different time series data mining aspects which provide a rich source of knowledge to utilize to mining CPU host load.

Though much work has been done and many problems in these areas are carefully presented and solved in many ways, the defects of these work still exist. Moreover, the lack of attention in mining patterns in CPU host load makes it necessary to propose novel methods in the specific domain in contrast to the importance of the host load pattern mining problem. In the meantime, the fact that less research is conducted in mining unknown similar patterns in time series data also hampers applying time series data mining methods to CPU host load pattern discovery problem. This situation raises the necessity of investigating in a more general problem of locating unknown similar patterns in time series data.

Chapter 3

Problem Formalisation

In this chapter we present background knowledge including definitions, similarity measures and some of our fundamental work for the ease of further discussion.

Section 3.1 introduces some necessary notions and definitions relate to our study. Section 3.2 introduces similarity measures we adopted and later in Section 3.3 we points out the similarity inconsistency problem and gives our solution. Lastly in Section 3.4 we introduce the noise problem of CPU host load and our noise reduction method.

3.1 Definitions

In this section we introduce necessary definitions briefly.

Definition 1. *(Time Series) A time series $T = \{t_1, \cdots, t_m\}$ is an ordered set of m real-valued variables [108].*

Definition 2. *(CPU Host Load Data) A CPU host load data is a time series data, $T = \{t_1, \cdots, t_m\}$, where $0 \leq t_i \leq 1$. t_i is the ratio of clock cycles CPU where is occupied to the total clock cycles in period i.*

Definition 3. *(Subsequence) Given a time series of length m, $\{T = t_1, \cdots, t_m\}$, a subsequence of T is a time series in T, $S = \{t_i, \cdots, t_j\}$, where $1 \leq i < j \leq m$.*

Definition 4. *(Time Series Indexing) Given a time series dataset D that contains a set of time series data and a querying time series data Q, a time series indexing method I is a method that returns such a subset N of D that for every time series data N_i in N, $Distance(N_i, Q) \leq R$, where R is a predefined distance value.*

Definition 5. *(Match) Given two time series P and Q with length of m and of n respectively, if P and Q match then $Distance(P, Q) \leq R$, where $Disntance(X, Y)$ is the similarity function between two time series data and R is a predefined similarity value.*

Definition 6. *(Optimal Match) Suppose two subsequence $R = \{p_i, p_{i+1}, \cdots, p_j\}$ from time series $P = \{p_1, p_2, \cdots, p_m\}$ and $S = \{q_k, q_{k+1}, \cdots, q_l\}$ from time series $Q = \{q_1, q_2, \cdots, q_n\}$ where $1 \leq i \leq j \leq m$ and $1 \leq k \leq l \leq n$ are an optimal match if and only if*

1. *(Similarity condition) R and S match.*
2. *(Non-trivial condition) There does not exist an integer a make subsequences R' and S' where $R' = \{p_i + a, p_{i+a+1}, \cdots, p_{j+a}\}$ from P and $S' = \{q_k + a, q_{k+a+1}, \cdots, q_{l+a}\}$ from Q and $1 \leq i{+}a \leq j{+}a \leq m$, $1 \leq k{+}a \leq l{+}a \leq n$ satisfy all the conditions below:*
   - *(a) R' and S' match.*
   - *(b) $Distance(R', S') < Distance(R, S)$.*
3. *(Longest-possible condition) There do not exist non-negative integers a and b make subsequences R' and S' where $R' = \{p_i - a, p_{i-a+1}, \cdots, p_{j+b}\}$ from P and $S' = \{q_k - a, q_{k-a+1}, \cdots, q_{l+b}\}$ from Q and $1 \leq i - a \leq i \leq j \leq j + b \leq m$, $1 \leq k - a \leq k \leq l \leq l + b \leq n$ satisfy all the conditions below:*
   - *(a) R' and S' match.*
   - *(b) $Distance(R', S') < Distance(R, S)$.*

3.2 Similarity measure

In this section we introduce two similarity measures we use: Euclidean distance and DTW distance.

3.2.1 Euclidean Distance

Euclidean distance is widely used in many time series applications. Euclidean distance is defined as follows:

Definition 7. *(Euclidean Distance) For two time series* $T_1 = \{x_1, x_2 \cdots x_n\}$ *and* $T_2 = \{y_1, y_2 \cdots y_n\}$ *with length* n*, the Euclidean distance between* T_1 *and* T_2 *is defined as:*

$$d(T_1, T_2) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{3.1}$$

Intuitively, the distance measure maps two time series of length n to two points in n-dimensional space and calculate their distance in the space. The one-to-one alignment strategy of Euclidean distance raise the problem that it is unable to handle distorted or noisy time series, as well as time series with different length. However, Euclidean distance measure is simple and effective for many applications as long as being used properly. Meanwhile, L_p-norm distance measure like Euclidean distance considers two time series as two points in corresponding dimensional space, which also helps understanding the similarity inconsistency phenomenon. This will be described later in this chapter.

3.2.2 Dynamic Time Warping and Warping Path Constraint

Dynamic time warping distance is also used in part of our work. DTW is selected for several reasons. First, DTW is an elastic distance measure which allows measuring time series of different lengths. This feature will be especially useful in our proposed method, which will be illustrated in the next section. The elastic nature of DTW also makes it a robust distance measure. Though we suggested in Chapter 1 that comparing subsequences with very different lengths is mostly meaningless, however, constraint can be applied to DTW to allow slight differences in lengths. It is necessary to review DTW briefly as an important background method.

Definition 8. *(Dynamic time warping) Giving two time series,* $X = \{x_1, x_2, \cdots, x_m\}$ *of length* m *and* $Y = \{y_1, y_2, \cdots, y_n\}$ *of length* n*. To calculate DTW distance, an* $(m+1) \times (n+1)$ *cost matrix* D *is created as follow:*

$$\begin{aligned}
&d(0,0) = 0 \\
&d(i,0) = d(0,j) = \infty \\
&d(i,j) = \| x_i - y_j \| + min \begin{cases} d(x_{i-1}, y_j) \\ d(x_i, y_{j-1}) \\ d(x_{i-1}, y_{j-1}) \end{cases} \\
&D(X,Y) = d(m,n)
\end{aligned} \tag{3.2}$$

Where $i = 1, 2, \cdots, m$ *and* $j = 1, 2, \cdots, n$ *and* $D(X,Y)$ *is the DTW distance between* X *and* Y*.*

$\| x_i - y_j \|$ *is the distance between* x_i *and* y_j*, which can be* $(x_i - y_j)^2$ *or simply* $| x_i - y_j |$*, or other possible measures.*

DTW distance does not require one-to-one alignment. This distance measure allows time series to be "warped" to produce minimum distance value. Obviously, there exist many possible alignments for two given time series. We call each of these alignment a warping path:

Definition 9. *(Warping path) Giving two time series,* $X = \{x_1, x_2, \cdots, x_m\}$ *of length* m *and* $Y = \{y_1, y_2, \cdots, y_n\}$ *of length* n*, and* D *is their DTW cost matrix. The warping path is a sequence of cells* $w = a_1, \cdots, a_i, \cdots, a_l$ *in* D*,where* $a_i = (x, y)$ *denotes cell* (x, y) *in* D *satisfying the following conditions:*

1. *Boundary condition:* $a_1 = (1, 1)$ *and* $a_l = (m, n)$

2. *Monotonicity and continuity condition: if*$a_i = (x, y)$ *and* $i \neq 1$*, then* a_{i+1} *must be one of* $(x + 1, y), (x, y + 1)$ *or* $(x + 1, y + 1)$

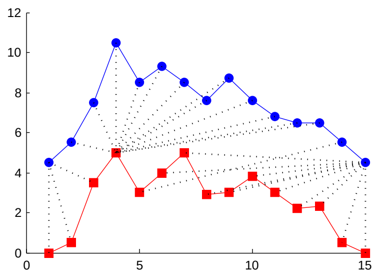
DTW algorithm [14] dynamically finds the optimal warping path that has minimal $\sum_{i=1}^{l} D_{a_i}$.

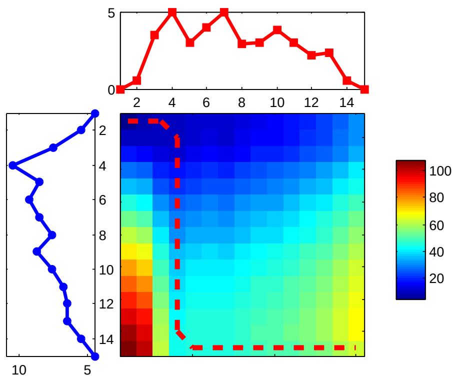
We can visualize this algorithm by plotting matrix D. As shown in figure 3.1a, DTW dynamically finds a path which can align two time series at the lowest cost. Plotting the cost matrix of DTW (figure 3.1b) can help understanding the algorithm. As shown in figure 3.1b, the optimal warping path (red dash line) goes from top left corner of the matrix to the bottom right of the matrix to aligned the start and end point of the time series, while the rest of the optimal warping path curves in a way to make sure the sum of cells it passes by is minimized.

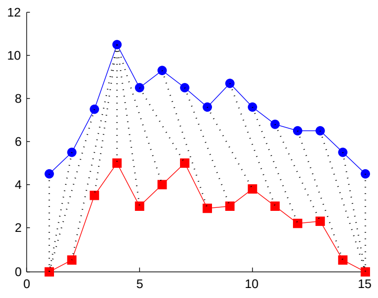
An obvious observation of figure 6.11 is that too many data points of blue (round mark) time series are aligned to only several points of red (square mark) time series. As a result, the optimal warping path in figure 3.1b warps too much. We can apply Sakoe-Chiba band [133] to limit the maximum warp as shown in figure 3.1c and 3.1d. Constraint not only helps to prevent over-warping, it also improves the performance of DTW since calculation is only performed on constrained part of the cost matrix.

3.3 The Similarity Inconsistency Problem

As in Chapter 1 the problem of similarity inconsistency is introduced. Similarity inconsistency results in a difficult situation that for time series subsequences with different lengths, it is impossible to decide whether any two subsequences are similar or not with a given threshold. In the case we show in figure 1.3, if we set the threshold of similarity to be 3 under Euclidean distance or 2 under DTW distance, traditional method will not be able to find longer patterns such as the 128 data points pair. On contrary, the threshold could be too large for shorter time series such as the 32 data points ones. In this situation, long similar


(a)


(b)


(c)

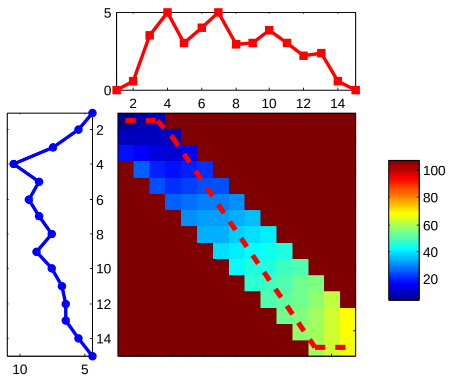
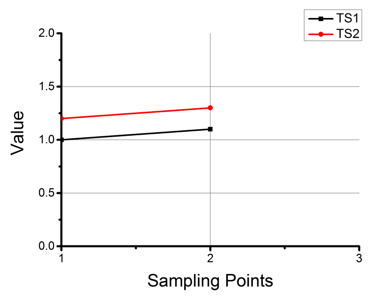

(d)

Figure 3.1: Illustration of DTW distance. Figure a shows how data points in one time series are measured with diffent datapoints of another time series under DTW distance. Figure b shows cost matrix and warping path of DTW distance. Figure c and d illustrate the situation when Sakoe-Chiba constraint with $width = 2$ (coloured area in figure d) is applied.

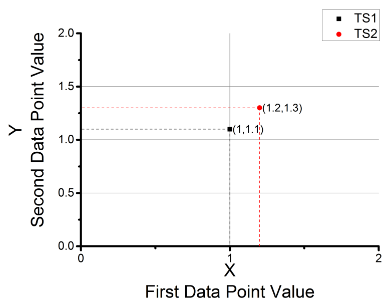
patterns are ignored and short dissimilar time series are picked up as patterns. We take Euclidean distance to explain the problem.

3.3.1 Similarity Inconsistency in Original Space

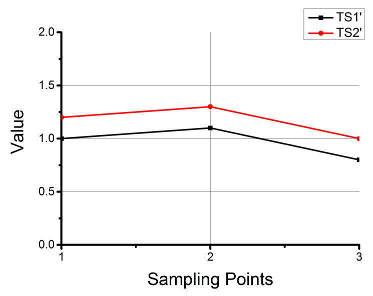
Let us begin with the simplest example. Assume we have all the subsequences with 2 data points as $TS1$ and $TS2$ in figure 3.2a. We plot these time series in points in a 2 dimension rectangular coordinate system in figure 3.2b as they all have only two data points. As the definition of Euclidean distance, Euclidean distance between time series $TS1$ and $TS2$ in figure 3.2a is the length between the two points $TS1$ and $TS2$ in figure 3.2b. Then we extend time series $TS1$ and $TS2$ with one more data point as in figure 3.2c. Similarly, the Euclidean distance between $TS1'$ and $TS2'$ is the length between points $TS1'$ and $TS2'$ in figure 3.2d. Note that the projection of points $TS1'$ and $TS2'$ in figure 3.2d on $X - Y$ plane is $TS1$ and $TS2$ in original space as in figure 3.2b. Thus, as we add a dimension for $TS1$ and $TS2$, the length from $TS1'$ to $TS2'$ in figure 3.2d must be no less than the length from $TS1$ to$TS2$ in figure 3.2b, since in figure 3.2d line segment $TS1 - TS2$ constructs the cathetus and line segment $TS1' - TS2'$ constructs the hypotenuse. Though for both time series pairs in figure 3.2a and figure3.2c, we can lift $TS1$ and $TS1'$ by 0.2 to make them the same time series as $TS2$ and $TS2'$ respectively, the similarity under Euclidean distance measure of the two pairs changes when the length of time series grow.



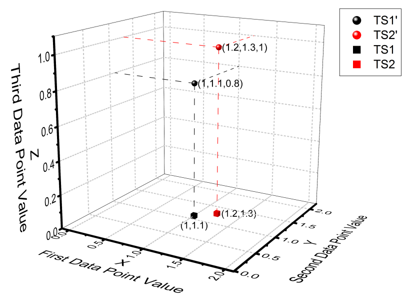
(a) 2 Data Points Time Series



(b) 2 Data Points Time Series in Original Space



(c) 3 Data Points Time Series



(d) 3 Data Points Time Series in Original Space

Figure 3.2: Demonstration of similarity inconsistency

We can develop the simplest case to longer time series pairs in a higher dimensional space and derive the same result, that is for any two pairs of time series with different time series in each pair, if they have the same similarity visually, the longer pair will always have larger Euclidean distance. Further more, we can confidently deduce that for any distance measure, if the measurement is carried out on spaces with different dimensions for different time series, the distance measure can not guarantee similarity consistency. This conclusion reveals that for discovering arbitrary length of similar patterns in time series data, it is necessary to make the distance measure being applied on a fixed dimensional space.

3.3.2 Consistent Similarity Measure

As illustrated above, the increase of dimensions is the reason of similarity inconsistency. An intuitive method to produce consistent similarity is to interpolate time series to a dimensional space with fixed number of dimensions. Chapter 4 adopted this method. In that chapter, we interpolate time series subsequences to a 5-dimensional space to achieve consistent similarity measure. Details can be found in Chapter 4.

Another simple but effective method is to average result distance values to eliminate the effect of dimension increase. This method maintains features of original distance measure and is wide applicable. We present the method as follows:

$$d_{consistent}(T_1, T_2) = \sqrt{\frac{\sum_{i=1}^{n}(x_i - y_i)^2}{n}} \tag{3.3}$$

In the above equation, the terms are averaged to eliminate the effect of distance accumulation. Similarly for the DTW distance, the number of terms is the length of warping path. The similarity consistent DTW distance can be calculated by:

$$DTW_{consistent}(T_1, T_2) = \sqrt{\frac{\sum_{k=1}^{n} w_k}{K}} \tag{3.4}$$

Where K is the length of warping path.

Table 3.1: Distance of figure 1.3 case under our proposed consistent distance measure

Series Length	128	64	32
Consistent Euclidean Distance	0.37	0.35	0.33
Consistent DTW	0.14	0.14	0.18

Both methods average original distance value with the number of alignments. For Euclidean distance the number of alignments is the length of time series being measured, and for DTW distance the number is the length of warping path. The effectiveness of the two similarity consistent distance measurements can be validated using the case study shown in figure 1.3. Table 3.1 shows the distance values of the 3 pairs of motifs shown in figure 1.3 under our proposed distance measure. It is clear that our similarity consistent distance measure is length-invariant.

3.4 Noise Reduction of CPU host load

As shown in figure 1.6 and also in [32], noise is a non-trivial problem in time series data mining. In [160], Zhang et al. found that CPU host load data present a frequent periodical peak pattern in short terms. Although the property casts a light on short-term prediction, noise can dominate the distance measure between two similar host loads for the long-term pattern discovery, which will cause them to be deemed different based on the distance measure. As the process goes on, the noise may often be amplified and deteriorate the situation.

Fortunately the high frequency noise can be smoothed by applying several low-pass filters. Gaussian smoother is one of these filters. Gaussian filter has been used in image processing to remove high frequency noise of an image. As in [72], Gaussian filter has a direct relation with neurophysiological findings in animals and psychophysics in human, which supports our assumption that time series smoothed by Gaussian filter can maintain the major trend similar to human intuition. This finding provides the ideas of designing a method that can maintain intuitive pattern of time series data without losing important information.

For a single CPU, its host load can be considered as a function $f(t)$, where t is time. Therefore, we can apply 1-dimension Gaussian filter defined as follows.

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \tag{3.5}$$

The effect of a Gaussian filter is a convolution of Gaussian function $g(x)$ and signal $f(t)$

$$(f * g)(\tau) = \int f(\tau) g(x - \tau) d\tau \tag{3.6}$$

The discrete implementation of Gaussian filter can be found in various literature. Theoretically a Gaussian smoother has infinite length. However, one can easily calculate that the elements within 3σ accounts for 99.73% of all elements, where σ is the standard deviation of the Gaussian function. For applicability and without sacrificing much accuracy, we only examine the elements within 3σ.

Figure 1.6 shows the effectiveness of a Gaussian filter. The original distance of two time series data is restored excellently from the highly noisy versions of them. Figure 1.4 shows how the Gaussian filter removes pulse noise and high frequency noise intuitively. It is clear in the figure that the filter retains the intuitive trend of host load data.

Chapter 4

Clustering Based CPU Host Load Similar Pattern Discovery

As mentioned in previous chapters, the most difficult problem lies in front of mining unknown similar patterns is the scale of time series data, which makes brute force algorithm impossible for its high time complexity. To reduce the time complexity required, we present a method that compares CPU host load data segment-to-segment rather than point-to-point.

In this chapter we investigate segmentation based similar pattern discovery in CPU host load data. In our proposed method, CPU host load data is first segmented and then each segment is represented as a 5-dimension vector. Thus the data are embedded into a 5-dimension space. To find similar patterns, a clustering method is applied to the data. Specifically, our method has five steps: noise reduction, segmentation, feature extraction and clustering.

As suggested in Chapter 1, it is necessary to reduce the noise of CPU host load. We use Gaussian filter presented in Chapter 3 to reduce the noise of CPU host load. Treated raw data are then segmented using the Relative Important

Point segmentation method [157]. To further reduce the dimensionality of each segment, feature extraction is then applied to represent each segment to five parameters. Similar patterns are finally captured by clustering method which identifies similar segments. This method greatly reduces time complexity of mining patterns in CPU host load data.

4.1 Segmentation Based Data Representation

Because of the continuity of time series data, it is difficult to split the data without loss of information. In this section we propose a representation for highly fluctuating time series data. We first segment the series into subsequences and then describe subsequences with feature vectors. This method efficiently represents the time series data and greatly reduces the data volume, which can save a lot of storage resource and computing time in later processes.

4.1.1 Noise reduction

As shown in figure 1.4, CPU host load data is very noisy, which could severely affect the decision of segmenting position host load. A noise reduction step is first applied to CPU host load data. The major term of CPU host load data is high-frequency noise which can be eliminated by using low-pass filters. We use the Gaussian filter to reduce the noise of CPU host load data as presented in Section 3.4.

4.1.2 Segmentation

Our segmenting algorithm is based on the fact that the host load data fluctuate considerably. The method provides different segmentation scales which controls what extent of change should be considered significant and thus should be segmented. For the task of mining patterns in CPU host load data, we concentrate on the trend of the data. The segmentation algorithm proposed in [157] segments time series data by observing their trend changes.

The segmentation method proposed in this work introduces the notion of *important point*. The *important point* is defined below:

Definition 10 (Important Minimum Point)**.** *Given a constant $R(R > 1)$ and time series $\{< x_1 = (v_1, t_1), \cdots, x_n = (v_n, t_n) >\}$, if a data point $x_m (1 \leq m \leq n)$ is called an Important Minimum Point, it must satisfy one of the following conditions:*

1) if $1 < m < n$, there are subscripts $i, j (1 \leq i < m < j \leq n)$ which makes (i) v_m the minimum among $v_m, \cdots, v_j$ and (ii) $v_i - v_m \geq R$ and $v_j - v_m \geq R$;

2) if $m = 1$, then x_m is the first data point of the time series. There is a subscript $j(m < j \leq n)$, which makes (i) v_m the minimum among $v_m, \cdots, v_j$, and $v_j/v_m \geq R$;

3) if $m = n$, then x_m is the last data point of the time series. There is a subscript $i(1 \leq i < m)$, which makes (i) v_m the minimum among $v_i, ..., v_m$, (ii) $v_i/v_m \geq R$.

On the definition of Important Maximum Point. Intuitively, the method searches local maximum and minimum points alternately with a parameter R input by users. R decides the significance of value change to be considered as important. According to the definition, the algorithm for searching the important points in the data series is proposed below.

Algorithm 1: Find Important Points

```
input  : Time series data TS with length $L(TS)$, R
output : Important Points Series(PS)
find first important point in TS;
if first important point is maximum then
    find next minimum points;
while Not at the last point of TS do
    find next important maximum point;
    if Not at the last point of TS then
        find next important maximum point;
```

This method has a time complexity of $O(n)$. By applying this method, the host load is efficiently divided into short pieces, each represents a major increase or decrease in host load.

4.1.3 Feature Extraction

Segmentation has greatly reduced the complexity of the pattern discovery problem. The presented segmentation method also unifies the result segments to simple increase or decrease pieces. This property provides us with the possibility of further dimension reduction without much loss of original information contained in raw data. Given the simplicity of result segments and for the ease of further processing, we propose a uniform representation method for each segment, called feature vector, which allows fast clustering of the data series that contain the major features of subsequences.

Definition 11 (Feature Vector)**.** *Given a time series data subsequence* $\{< x_1 = (v_1, t_1), \cdots, x_l = (v_l, t_l) >\}$*, the feature vector of a subsequence is defined as a five-tuple as follows, where* l *is the length of the subsequence;* v_1 *is the first value of the subsequence;* v_d *is the difference between the first and last value of the subsequence;* σ *is the maximum deviation of the raw data from the straight line between the first and last point in the sequence and* $\bar{\sigma}$ *is the mean deviation.*

$$(l, v_1, v_d, \sigma, \bar{\sigma})$$

v_d, σ and $\bar{\sigma}$ can be calculated by the following equations, where $1 \leq i \leq l$, $i \in N$ and *Extremum* is the maximum or minimum value the sequence deviates from the line between first and last value.

$$v_d = v_l - v_1 \tag{4.1}$$

$$\sigma = Extremum(v_i - (\frac{(t_i - t_1)(v_l - v_1)}{t_l - t_1} + v_1)) \tag{4.2}$$

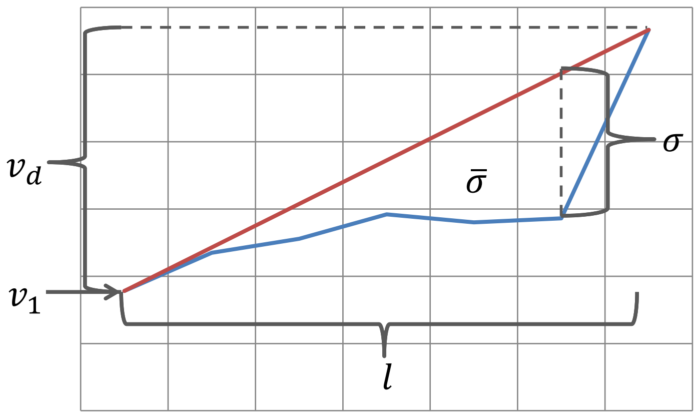


Figure 4.1: Physical meaning of feature vector

$$\bar{\sigma} = \sum_{i=1}^{l}(v_i - (\frac{(t_i - t_1)(v_l - v_1)}{l(t_l - t_1)} + \frac{v_1}{l})) \tag{4.3}$$

Figure 4.1 shows the physical meaning of each feature value. It can be observed that the feature values preserve the main character of a subsequence regardless of the noise. The first and last value, v_1, and the deviation from the first value, v_d, depict the initial value of a sequence and how much the sequence changes. Then $\bar{\sigma}$ presents the average fluctuation of a subsequence. To maintain the trend of a sequence, σ shows how the data in subsequence vary, that is, its concavity and convexity. Also, we concern the length of a time series l, which is another important attribute.

It is important to note that this representation method works well not only for short sequences, but also for long time series. When we want to study longer patterns, we change the observation scale by giving a greater segmenting parameter R introduced in Section 4. Then longer subsequences will be produced. Given a greater observation scale, the representation method will also represent the subsequence in a coarser way. Therefore, the good scalability of the proposed segmentation method guarantees the good scalability and the effectiveness of the proposed representation method.

4.2 Pattern Discovery by Clustering

Similar subsequences in time series data comprise a pattern. We use the clustering analysis to discover the similar subsequences in a subsequence set that are segmented from a host load dataset. We first give our consistent distance measure for clustering. then we describe how DBSCAN is applied for the pattern discovery of host load.

Consistent Similarity Measure in Interpolated Space

As mentioned in Section 4, for all host load subsequence we extract their features as a 5 dimension vector. Thus all time series are interpolated into a 5 dimension space. If similarity inconsistency also exists in interpolated space, we will not be able to find patterns with arbitrary length with the proposed feature vector.

In section 4 we defined the feature vector as below:

$$(l, v_1, v_d, \sigma, \bar{\sigma})$$

To eliminate the effect of similarity inconsistency mentioned in Section 3.3, the represented vector should be length-independent when distance measure is applied. According to the definition of each element in feature vector, v_1, v_d and σ is length irrelevant. v_1 and v_d are independent of length themselves, and any variable or constant used to calculate σ is independent of length. For $\bar{\sigma}$, the effect of length is eliminated by average l terms.

The element l is the length itself and will change its value when we have different length of time series. However, when calculating similarity, the effect of length to similarity is eliminated by the term $\sqrt{(l_1 - l_2)^2 \cdots}$ under Euclidean distance in interpolated space. So we can confidently say only the difference of length between two time series affects their similarity, and that is what we need because a long time series is less likely to be similar with a short one.

In the proposed method, the patterns in the host load are discovered by DBSCAN clustering. In Section 4, the host load data are segmented and represented as a set of 5-tuple vectors. In order to apply the clustering method, the distance measurement should be defined first. The distance between two vectors can be easily defined on a vector set. As discussed in Section 5.3, applying traditional distance measure on raw time series with different lengths will result in the problem of similarity inconsistency. Only in the interpolated space with fixed dimensions we can apply traditional distance measure.

As the original segments have been interpolated to a 5-dimension space, the distance of feature vectors can be simply defined with Euclidean distance:

$$EuclideanDistance = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{4.4}$$

In our case, the distance of two feature vectors $(l_1, v_{11}, v_{d1}, \sigma_1, \bar{\sigma}_1)$ and $(l_2, v_{12}, v_{d2}, \sigma_2, \bar{\sigma}_2)$ can be calculated by:

$$Distance = \sqrt{(l_2 - l_1)^2 + \cdots + (\bar{\sigma}_2 - \bar{\sigma}_1)^2} \tag{4.5}$$

Since the distance measurement is defined, we can apply the clustering algorithm to the data set.

4.2.1 Clustering

DBSCAN is a density-based clustering algorithm proposed in [50]. The algorithm does not require the users to determine the quantity of the resulting clusters beforehand, which is very difficult to be determined in many cases and has a crucial impact on the clustering quality. Meanwhile, considering the existence of outlying data, those clustering algorithms which require all data to be clustered are not satisfactory. Moreover, the algorithm has the benefit of discovering dense clusters of any shapes and effectively removing noisy points.

The algorithm finds dense clusters by searching for density connected points. As in [50], the fundamental concepts and steps of DBSCAN are shown below.

Definition 12 (Eps-neighborhood of a point)**.** *The Eps-neighborhood of a point* *p, denoted by* $N_{Eps}(P)$*, is defined by* $N_{Eps}(p) = \{q \in D | dist(p,q) \leq Eps\}$.

Definition 13 (directly density-reachable)**.** *A point p is directly density-reachable from a point q with radius Eps and the minimum inner points* $MinPts$*, if*

1) $p \in N_{Eps}(q)$

2) $|N_{Eps}(q)| > MinPts$ *(core point condition).*

Definition 14 (density-reachable)**.** *A point p is density reachable from a point q with Eps and* $MinPts$*, if there is a chain of points* $p_1 \cdots p_n, p_1 = q, p_n = p$ *such that* p_{i+1} *is directly density-reachable from* p_i*.*

With the chain of points that are directly density-reachable in succession, we have a chain of relatively close points. If we find all the chains, we have a point set with high density. The point set forms a cluster. The two parameters *Eps* and $MinPts$ control the clustering range and accuracy, which is very important and will be discussed in Section 4.3.

Considering most datasets have very clear density difference between natural clusters and noise, one can roughly estimate the values of *Eps* and $MinPts$ by repeatedly running the algorithm on a small test data set. In addition, the two parameters can also control the scale of clusters.

4.3 Experimental Evaluation

We applied our model on host load data trace produced by Google [149]. Google traced the data of a Google cluster with about 11,000 machines. We transformed the resource usage of 7 randomly selected machines into the average CPU load spanning 18,605 minutes. In this section, we will evaluate the effectiveness of the pattern discovery framework proposed in this paper and the choice of parameters based on the randomly selected datasets.

4.3.1 Effectiveness of Pattern Discovery

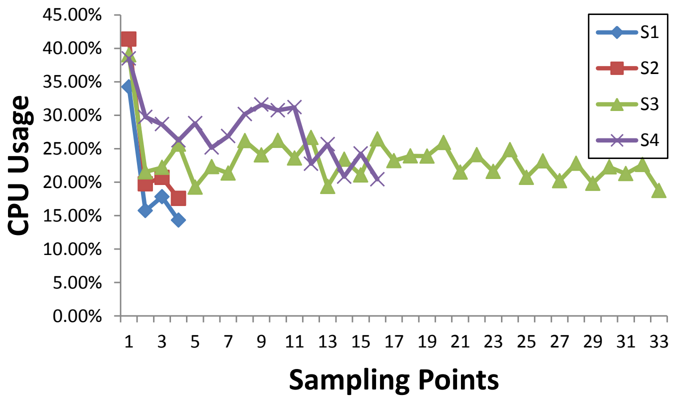
Pattern discovery aims to find similar patterns among the data sequences. We apply the DBSCAN clustering algorithm to find patterns since DBSCAN is able to deal with the outlying points and achieve higher intra-similarity in a cluster, which is what other clustering methods fail or have difficulty to reach. In order to reveal the benefit of DBSCAN, we compared our method with two of the most popular and classic clustering methods, k-means [69] and top-down hierarchical clustering [132]. In the evaluation, the three clustering algorithms are run on the datasets. After obtaining the clustering results, we randomly pick 4 subsequences from one cluster and another 4 subsequences from different clusters for each clustering algorithm. We compare the intra-cluster subsequences as well as the inter-cluster subsequences. A good clustering result should produce highly similar intra-cluster subsequences and very different inter-cluster subsequences. The results obtained by k-means, top-down hierarchical and DBSCAN are shown in figure 4.2, 4.3 and 4.4, respectively.

As Shown in figure 4.2a, intra-cluster subsequences produced by k-means algorithm have less similarity, compared with the results by other two algorithms. Although subsequence S_1 is very similar to S_2, S_3 and S_4 are totally different from each other. In figure 4.2b, Sequence S_1 and S_2 are similar (note S_i in 4.2b is not S_i in figure 4.2a), although they are picked from different clusters.

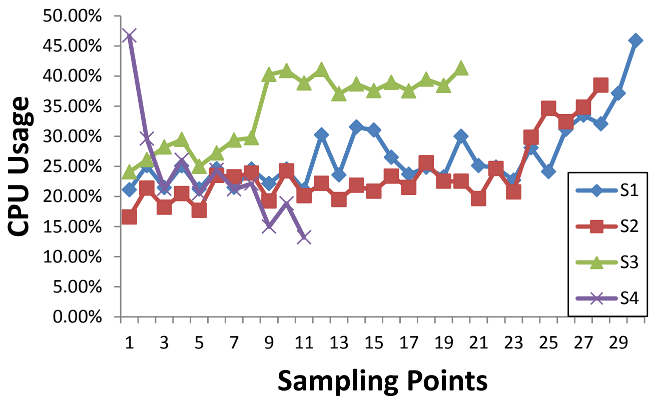
In figure 4.3a, all 4 subsequences have relatively high similarity, while figure 4.3b shows that the subsequences from different clusters also resemble each other.

As can be observed in figure 4.4, the intra-cluster similarity obtained by DBSCAN is very high and the inter-cluster subsequences have much less similarity. From the results obtained by DBSCAN, we can conclude that we have discovered a pattern from this cluster shown in figure 4.4a, and that we have discovered four different patterns shown in figure 4.4b.

The reasons for these above results can be explained as follows. k-means tries to cluster every data points into specific clusters, which may result in less intra-similarity because there may be many outlying points in the data sets which



(a) Intra-cluster data



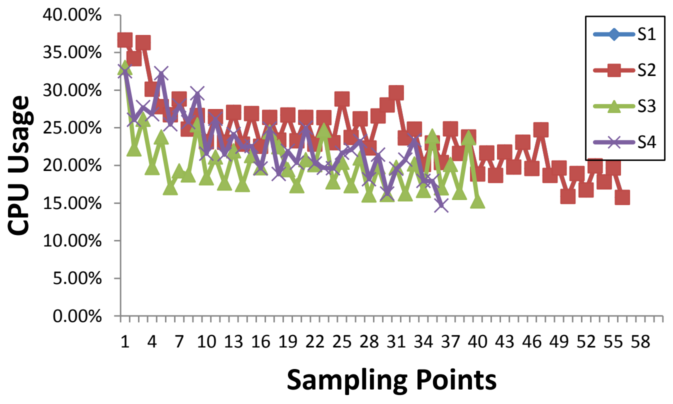
(b) Inter-cluster data

Figure 4.2: Typical cluster results of k-means clustering

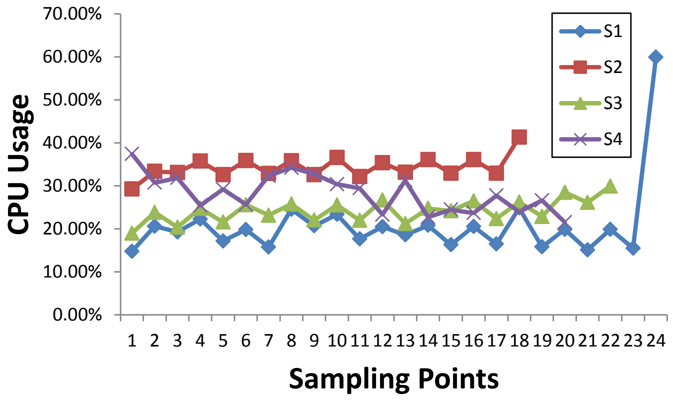
do not bear similarity with other subsequences. Furthermore, those points which are far from all cluster centres could be relatively close to each other. However, they may be assigned to different clusters. Also, without prior knowledge, it is difficult to find the right k and initial points.

As for the top-down hierarchical clustering algorithm in figure 4.3, the intra-cluster similarity seems not bad, but the inter-cluster similarity is not satisfactory. The culprit is the dividing process in the hierarchical clustering. Although this method can effectively pick out the noise points which are far away from any clusters, these outlying points could result in the division of natural cluster.

However, as shown in figure 4.4, since DBSCAN requires a particular density



(a) Intra-cluster data



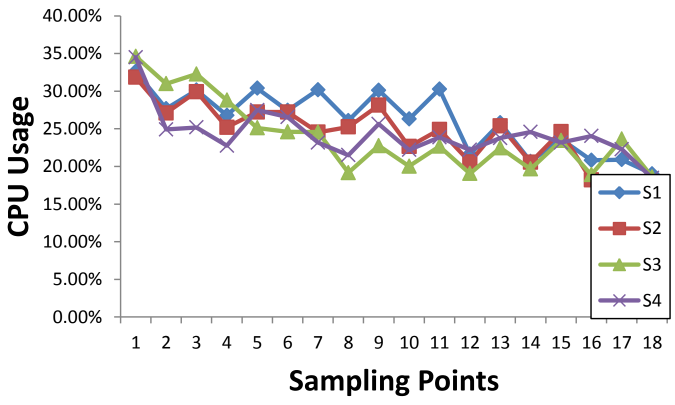
(b) Inter-cluster data

Figure 4.3: Typical cluster results of top-down hierarchical clustering

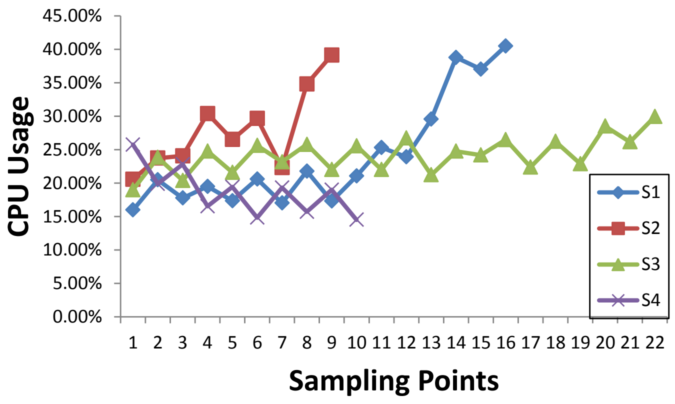
for data in clusters, there is a considerably high intra-cluster similarity. With the minimum requirements of $MinPts$, the method can exclude any outlying points from the clusters. Owing to the special nature of DBSCAN, the better results are produced in discovering the patterns for host load.

4.3.2 The Choice of Parameters

In Section 4.2.1 we have presented the meaning of the segmentation parameter R. Generally, the greater value R is, the longer subsequences will be produced and longer patterns will be found from them. By contrast, a smaller value of



(a) Intra-cluster data



(b) Inter-cluster data

Figure 4.4: Typical cluster results of DBSCAN clustering

R produces shorter subsequences and it is more likely to find short patterns. Depending on the scale of the patterns we want to discover, there is typically no "optimal" R. However, a too large or too small R often leads to a segmentation fault. That is, either the entire set of data is regarded as a segment, or every single line between two neighbouring nodes is segmented. To tackle this problem, a simple but effective way is to try the R value on a small test data. The selections of $MinPts$ and Eps have been discussed in [50]. However, the pattern discovery task requires higher inner-cluster similarity. If the elements in a cluster vary too much, it will be much more difficult to conclude a uniform pattern featuring the cluster.

Table 4.1: Average Inner-cluster DTW Distance

MinPts/Eps	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.1
2	0.004	0.047	0.159	0.3	1.236	4.385	11.444	13.116	25.36	22.377
4	-	0.114	0.562	0.794	2.351	12.469	20.435	62.738	111.841	117.486
8	-	0.183	0.331	1.764	1.675	3.783	28.844	70.841	58.124	107.083
16	-	-	-	-	1.664	3.724	5.318	34.122	52.063	81.605
32	-	-	-	-	-	-	2.087	4.106	7.779	35.512

Table 4.2: Average Inter-cluster DTW Distance

MinPts/Eps	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.1
2	1.389	5.566	14.418	24.543	33.27	25.455	25.441	32.675	23.808	22
4	-	0.533	0.844	3.198	6.796	4.353	5.642	1.858	1.42	1.47
8	-	-	-	0.376	0.421	3.227	1.395	1.058	2.128	1.346
16	-	-	-	-	-	0.35	1.045	0.837	0.974	1.109
32	-	-	-	-	-	-	-	0.398	1.147	0.774

As in [77], one of the main tasks in clustering is to minimize local error and maximize global error. We measured average inner-cluster distance and inter-cluster distance for different values of $MinPts$ and Eps, with $R = 0.1$. The results for inner-cluster distance and inter-cluster distance are shown in table 4.1 and 4.2 respectively.

The results in table 4.1 and 4.2 show that small changes in the two parameters can cause great differences in the clustering results. Note that in the DBSCAN algorithm, although high Eps and small $MinPts$ will produce very low inner-cluster average distance, they will also produce much fewer clusters and similar patterns, causing many meaningful patterns excluded. Conversely, small Eps and large $MinPts$ are likely to produce more clusters and more patterns in a cluster. However, this will also bring in dissimilar patterns.

We evaluated the combinations of the parameters in table 4.1 and 4.2. When $Eps = 0.04$ and $MinPts = 2$, we obtained a preferable parameter. However, this is just a simple analysis method for the determination of the clustering parameters. The optimum varies with the choice of segmentation parameter and features of different CPU host load.

4.3.3 Effectiveness of Distance Measure

We show the problem of similarity inconsistency in Section 3.3 pointing out that any distance measure which applies directly to time series data with arbitrary lengths will inevitably lead to similarity inconsistency. Here we compare our proposed distance measure to DTW distance measure to show the effectiveness of our proposed distance measure.

We use the bottom-up hierarchical clustering algorithm to classify 7 randomly selected CPU host load subsequences obtained from the Google cluster trace [149]. The dendrogram that the clustering method builds can help us visualise the clustering process and the level of similarity between any pair of subsequences [90]. For any clustering algorithm, distance measure is the key to determine whether two data should be in a cluster or not. We can utilise this property of clustering algorithms to have a glimpse of how the distance measure work.

In figure 4.5, we can see two dendrograms built by bottom-up clustering. Figure 4.5 left is the clustering results of our proposed distance measure. We use the Euclidean distance measure in 5-dimension interpolated space. The increasing subsequences are very well separated from those decreasing ones on the root node, and the most similar pieces of data are perfectly joined in groups. It is also notable that the subsequence which is most dissimilar to all others is far from all other data.

As a comparison, we also evaluated the effectiveness of DTW distance on the same dataset as in figure 4.5 right. We can clearly see how DTW based clustering influenced by lengths of subsequences. Under the situation of arbitrary length of time series, the length of time series dominates the distance value of DTW. In figure 4.5 right, the clustering result is more likely to be length based rather than similarity based. Though the top two time series and the bottom two have the same increasing trend, they are separated to the two ends of the dendrogram because the length of the two pairs is different.

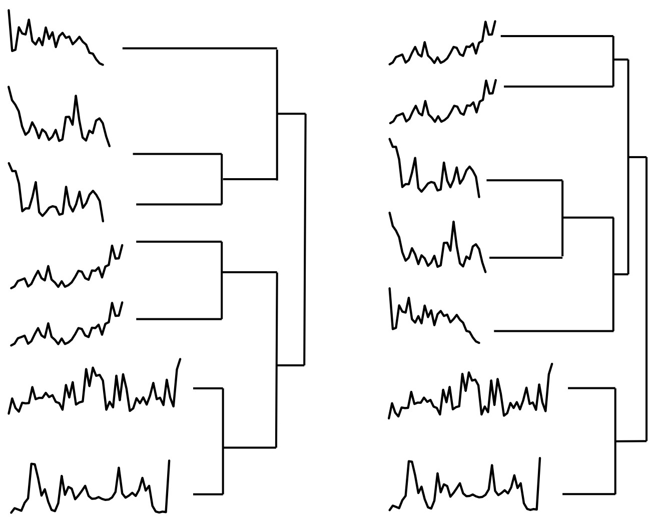


Figure 4.5: Left: bottom-up hierarchical clustering result using our proposed distance measure. Right: bottom-up hierarchical clustering result using dynamic time warping.

4.4 Summary

In this chapter, we proposed a novel method to discover unknown similar patterns in CPU host load data. The method features mining patterns on segmented CPU host load data rather than on raw data, which is more intuitive and efficient.

An important point based segmentation method is chosen to segment CPU host load for its particular compatibility with fluctuating time series which are unable to be properly segmented by other linear segmentation methods. We utilize the segments' uniform shapes the segmentation method provide, segments are further reduced to 5-tuple vectors which have experimentally shown to be representative, robust and distinguishing by a bottom-up clustering test.

In order to locate similar patterns, a clustering method, DBSCAN is deliberately chosen for its independence of prior knowledge of cluster quantity, ability to discover clusters of any shapes and capability of eliminating outlying data. Experiments compared the method and several other clustering methods, proved the method to be strikingly effective in both finding similar host load subsequences and distinguishing different types of host load subsequences.

In summary, this method solves the time complexity problem faced by mining unknown patterns in CPU host load by both divide-and-conquer method and dimensionality reduction methods at a small cost of accuracy, and is widely applicable to other types of time series data which fluctuation and change of trends are the major concerned features.

Chapter 5

Reduction Based CPU Host Load Pattern Discovery

In some cases, we need to discover exact patterns from CPU host load data, which produces all similar patterns without any loss. In this case, the segments clustering method are not capable of producing no false-negative errors. In the previous chapter, we have proposed a segments clustering based method for pattern discovery in CPU host load.

To discover patterns in CPU host load within a reasonable time and to achieve exact discovery, a proper CPU host load representation method should be applied. The method should satisfy following properties:

1. The representation is lower-bounded.
2. The representation reduces dimensionality or numerosity of the original data.

The lower-bounding property guarantees the distance of the same subsequences in represented space is greater than in original space, which ensures all possible patterns can be found in represented space without false-negative errors. Since the high time complexity of locating patterns in raw data, it is also necessary to have original data reduced.

In this chapter, we develop a method which discovers exact patterns in CPU host load. The method first reduces dimensionality and numerosity of CPU host load by using PAA and SAX representation, then apply a cascade search algorithm to find patterns efficiently.

5.1 CPU Host Load Representation

Represent time series data to a different dimensional space is an efficient way to reduce search space and accelerate pattern discovery. In this section we briefly overview the SAX indexing method proposed in [105] and then present our improved method.

5.1.1 Overview of PAA

Piecewise Aggregate Approximation (PAA) [86] is a non data-adaptive dimension reduction method for time series data. In [105], Lin et al. proposed symbolic aggregate approximation (SAX) representation based on PAA which transforms time series into symbol series. We first introduce the PAA method.

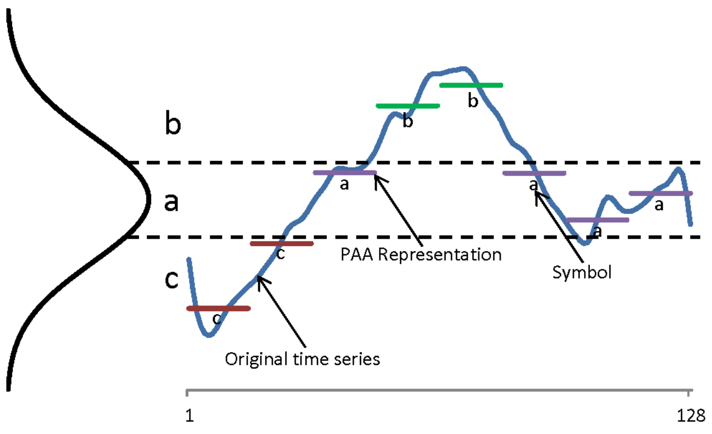


Figure 5.1: A time series data reduced dimensionally by PAA and then symbolised by SAX

A time series C of length n can be represented in a w-dimension space by a vector $\bar{C} = \bar{c_1}, \cdots, \bar{c_w}$. The i^{th} element of $\bar{C}$ is

$$\bar{c_i} = \frac{w}{n} \sum_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}i} c_j \tag{5.1}$$

That is, the PAA method segments time series into $\frac{n}{w}$ parts of equal lengths and represents each part with a new value which averages all old values in that part. Hence, the presented time series is reduced to $\frac{w}{n}$ dimension. Now we can have the definition of dimensional reduction rate:

Definition 15. *(Dimensional Reduction Rate) Given a time series T with length n, if T is represented with length w in a indexing method I, then the dimensional reduction rate of I is $\frac{n}{w}$.*

Figure 5.1 shows how the PAA representation works. In this case, a time series with the length of 128 data points is reduced to a PAA representation with 8 data points. In other words, PAA averages over every 16 consecutive points to generate a new value.

The benefits of PAA is that the method provides effective dimensional reduction with time complexity $O(n)$. Meanwhile, PAA provides lower bounding which ensures that for two time series data P and Q, the distance between them in the representation space will be no more than that in the original space. Namely, the following equation holds.

$$Distance_{RepresentationSpace}(P, Q) \leq Distance(P, Q) \tag{5.2}$$

Lower bounding ensures the real distance between two time series will not be underestimated. The benefits of this property are that for a given query time series data Q, one can find all the possible results in the representation space. More specifically for the pattern discovery task, lower bounding enables one to find all possible patterns and reduce the time complexity by applying the techniques such as early abandon.

5.1.2 Overview of SAX

After the dimension of a time series data has been reduced using PAA, we can then discretize the time series data symbolically. The basic idea of symbolizing time series data is proposed in [105]. We briefly review SAX representation.

Symbolic Aggregate Approximation, also known as SAX, divides the value range of a time series dataset into a given number of areas with equal sizes under the Gaussian distribution. If we use m symbols to discretize these values, it is preferable to assign these symbols in a equal probability way to achieve best encoding efficiency, which means these symbols occur similar number of times in the discretized time series [6, 109]. Assume normalized time series data values obey Gaussian distribution $N(0,1)$, to achieve this, the range of values (S_{min}, S_{max}) of any two symbols S^1 and S^2 represent should satisfy following equation [111]:

$$\int_{S^1_{min}}^{S^1_{max}} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} = \int_{S^2_{min}}^{S^2_{max}} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} \tag{5.3}$$

In the case shown in figure 5.1, the value range is divided into three area, i.e. the area above the top dash line, the area between the top and bottom dash line and the area below the bottom dash line. We assign a symbol to each area and then all the values which fall into an area in a time series data are represented by the symbol assigned to that area. Such a symbol assignment method ensures that all symbols are assigned in an equiprobable way. It is easy to know that given a limited number of symbols, the even symbol assignment across data points maximizes the representation effectiveness, since each symbol carries the same volume of information to avoid unnecessary information loss.

5.1.3 The Refined Symbolic Aggregate Approximation Indexing

The original work which invented SAX assumes time series data values are governed by the Gaussian distribution. This is a correct assumption as to

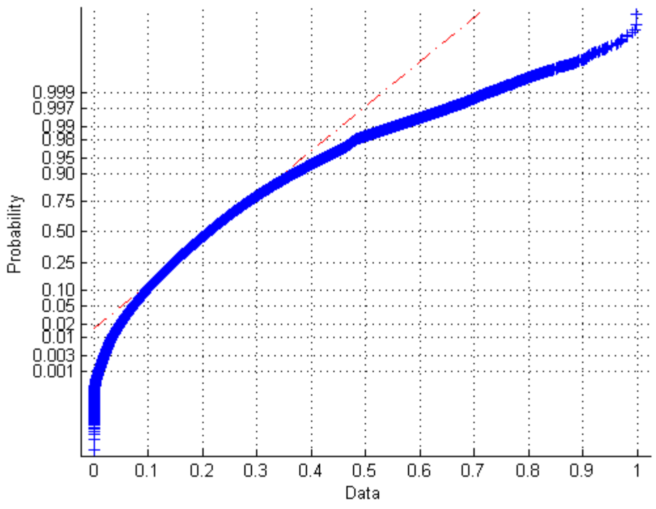


Figure 5.2: Normal probability plot of host load data

most real-world time series, however, CPU host load data disobey Gaussian distribution. We use a normal probability plot shown in figure 5.2 to illustrate this. In the plot, the vertical axis is the probability of a data point to be less than the value shown in the horizontal axis. Note that the values of the vertical axis are plotted under the cumulative distribution function of values in horizontal axis rather than linearly. In this way, a dataset which obeys Gaussian distribution should be plotted along a straight line rather than a curve. In figure 5.2, data are plotted as a curve which far deviates from the red reference line. This indicates that the data do not follow the Gaussian distribution. Therefore, assuming that host load fluctuate following the Gaussian distribution will greatly hamper the efficiency of the SAX indexing method. With the Gaussian distribution, the number of symbols assigned to greater values is approximately the same as those to lower values. However, the data with lower values actually occur more frequently.

Moreover, as pointed out by [22], the PAA representation also compromises equiprobability. Since PAA uses the average over a certain number of consecutive raw data points, the raw data represented by PAA may embody a different probability distribution. The inequality in assigning symbols may cause overfitting by assigning more than enough symbols to those data points that are rarely observed, while the parts of a time series that occur more frequently and therefore potentially contain the fundamental patterns are not represented with an adequate number of symbols.

Our strategy is simple but effective. We measure the actual distribution of a data set before the SAX indexing. A real-world dataset can be very large, especially under the "Big Data" context. However, we can always take a reasonable amount of data as our sampling data. The sampling methods have been used for a long time in many different fields and its validity has been theoretically and practically verified.

Let us assume that m symbols are assigned to a time series at the dimension reduction rate n. We arbitrarily take a number of time series to form a sample set $S = \{T_1, \cdots, T_n\}$. The number of values in the set, denoted by Q, is

$$Q = \sum_{i=0}^{n} Lengthof(T_i) \tag{5.4}$$

The raw data are first represented by PAA. These PAA values are then sorted in the ascending order. Because of the limited size of the sample set, this process can be completed quickly. As we are assigning m symbols, we create a bucket with the volume of $V = \frac{Q}{m}$. Next, we sequentially put the sorted values into the bucket. When the bucket is full, we record the last value that was put, which represents the upper bound of the values that are put in this round. After this, the bucket is cleared and then refilled by the remaining data values. After all values are put, m values are recorded, each of which corresponds to the maximum value in each bucket. Compared to the sample dataset, the entire dataset always has a wider value range. To make sure every value in the whole dataset can be assigned with a symbol, we do not use the maximum value of the last bucket but use the previous $m - 1$ values as the dividing points of m data intervals, which are used to determine which interval a raw data is located and therefore determine which symbol should be assigned to the data. This method may cause the first and last intervals to have slightly more values than other intervals. However, this will not compromise the statistical equality of each value interval because of the strong representativeness of the sample dataset. This way we transform the dimensionally reduced time series data into a string of symbols, based on which we can further apply discretization.

5.2 Similarity Measure

Since our data are represented by PAA and refined SAX successively, similarity measures based on original data space are no longer feasible. We introduce Euclidean and DTW similarity measure under PAA and SAX respectively.

PAA simply reduce dimensionality of time series by average a fixed length of data series, it is easy to define the two similarity measures under PAA. Given two time series Q of length m and C of length n, window width w, their PAA represented version $\bar{Q}$ and $\bar{C}$, their Euclidean distance is as follow:

$$DR_{euc}(\bar{Q},\bar{C}) = \sqrt{w * \sum_{i=1}^{\frac{n}{w}} (\bar{q_i} - \bar{c_i})^2} \tag{5.5}$$

Iff. $n = m$.

Their DTW distance is as follow:

$$\begin{aligned} &d(0,0) = 0 \\ &d(i,0) = d(0,j) = \infty \\ &d(i,j) = (\bar{q_i} - \bar{c_j})^2 + min \begin{cases} d(\bar{q}_{i-1}, \bar{c}_j) \\ d(\bar{q}_i, \bar{c}_{j-1}) \\ d(\bar{q}_{i-1}, \bar{c}_{j-1}) \end{cases} \\ &DR_{dtw}(\bar{Q},\bar{C}) = \sqrt{w * d(\frac{m}{w}, \frac{n}{w})} \end{aligned} \tag{5.6}$$

Where $i = 1, 2, \cdots, \frac{m}{w}$ and $j = 1, 2, \cdots, \frac{n}{w}$ and, $\bar{q_i}$, $\bar{c_i}$ are values of each data point under PAA representation.

We have introduced the similarity inconsistency problem in Section 3.3, we need to introduce consistent similarity measures under PAA as well:

$$DRC_{euc}(\bar{Q},\bar{C}) = \frac{DReuc(\bar{Q},\bar{C})}{n} \tag{5.7}$$

And consist DTW distance:

$$DRC_{dtw}(\bar{Q},\bar{C}) = \frac{DR_{dtw}(\bar{Q},\bar{C})}{w * \bar{K}} \tag{5.8}$$

Where $\bar{K}$ is the length of warping path of $\bar{Q}$ and $\bar{C}$.

Measuring similarity under SAX representation is slightly more complex than that under PAA because SAX represent data by symbols rather than values. In SAX, each symbol S^i represent a range of values:

$$S^i_{min} \leq S^i < S^i_{max} \tag{5.9}$$

To provide lower bounding, the SAX represented data should be considered representing the loosest case of the original data, which means we need to take the lowest value to represent the value of a symbol for positive values and highest value for negative values. Hence, the distance between each symbol forms a distance table:

Table 5.1: Example of symbol distance look up table

		S^1	S^2	S^3	S^4
		a	b	c	d
S^1	a	0	0	0.67	1.34
S^2	b	0	0	0	0.67
S^3	c	0.67	0	0	0
S^4	d	1.34	0.67	0	0

Value of each cell (S^r, S^c) can be calculated by the following equation:

$$S^r - S^c = S^c - S^r = \begin{cases} 0, if \mid r - c \mid \leq 1 \\ max(S^r_{max}, S^c_{max}) - min(S^r_{min}, S^c_{min}), otherwise \end{cases} \tag{5.10}$$

Distance of any two symbols can be aquired by looking up the table. Now we can define Euclidean distance measures under SAX:

$$MINDIST_{euc}(\hat{Q}, \hat{C}) = \sqrt{w * \sum_{i=1}^{\frac{n}{w}} (\hat{q}_i - \hat{c}_i)^2} \tag{5.11}$$

Iff. $n = m$. There DTW distance can also be defined:

$$\begin{aligned} &d(0,0) = 0 \\ &d(i,0) = d(0,j) = \infty \\ &d(i,j) = (\hat{q}_i - \hat{c}_j)^2 + min \begin{cases} d(\hat{q}_{i-1}, \hat{c}_j) \\ d(\hat{q}_i, \hat{c}_{j-1}) \\ d(\hat{q}_{i-1}, \hat{c}_{j-1}) \end{cases} \\ &MINDIST_{dtw}(\hat{Q}, \hat{C}) = \sqrt{w * d(\frac{m}{w}, \frac{n}{w})} \end{aligned} \tag{5.12}$$

Where $i = 1, 2, \cdots, \frac{m}{w}$ and $j = 1, 2, \cdots, \frac{n}{w}$. Consider the similarity inconsistency problem, we have their consistent version:

$$MINDISTC_{euc}(\hat{Q}, \hat{C}) = \frac{MINDIST_{euc}(\hat{Q}, \hat{C})}{n} \tag{5.13}$$

And consist DTW distance:

$$MINDISTC_{dtw}(\hat{Q}, \hat{C}) = \frac{MINDIST_{dtw}(\hat{Q}, \hat{C})}{w * \hat{K}} \tag{5.14}$$

Where $\hat{K}$ is the length of warping path of $\hat{Q}$ and $\hat{C}$. Both Euclidean and DTW distance measures and their consistent versions can be calculated incrementally by storing previous similarity value for Euclidean distance and by storing DTW matrix for DTW distance.

5.3 Efficient Pattern Discovery

By applying SAX representation, we can reduce dimensionality and numerosity of a given series of CPU host load which reduces search space greatly. In this section, we start from brute force algorithm, then present its improved algorithms as our pattern discovery algorithm.

5.3.1 Brute Force Pattern Discovery

One of the pattern discovery methods in the SAX representation is the brute force algorithm. For two CPU host load data P and Q, a brute force algorithm finds all possible subsequences in P and try to match each subsequence to Q. The pseudo code of brute force algorithm is as Algorithm 2.

Algorithm 2: Brute force pattern discovery algorithm

```
input  : CPU host load: P, Q;
         Maximum allowed distance: R;
output : Set of patterns
foreach
  $HostLoadSubsequenceP$ in $P$
foreach
  $HostLoadSubsequenceQ$ in $Q$
if $ConsisDistance(subsequenceP, subsequenceQ) < R$ then
  $patternSet.add(subsequenceP, subsequenceQ)$;
```

In this algorithm, finding all similar subsequences in a CPU host load is a very computational expensive task. The time complexity of each brute force subsequence is $O(n!)$, which is unbearable in reality. Moreover, the algorithm finds all patterns without considering the length of them. In other words, most of the patterns found by this algorithm are very short (e.g., containing two data points), which are meaningless compared to longer ones. In this work, an improved algorithm is proposed to find potentially long patterns.

5.3.2 Improved Pattern Discovery Algorithm

Our first improvement technique modifies the brute force to find longest possible patterns and does not require search all subsequences. The algorithm is outlined in algorithm 3. Once a pattern is located, the algorithm keeps calculating distance for following data points until the distance value grows larger than the threshold. This algorithm reduces the number of iterations greatly. However, in the worst case, the algorithm still has the time complexity of $O((nm)^2)$.

Algorithm 3: Longest possible pattern discovery algorithm

```
input  : CPU host load: P, Q;
         Maximum allowed distance: R;
output : Set of patterns
for $i = 0; i < P.length; i++$ do
    for $j = 0; j < Q.length; j++$ do
        if $|P[i] - Q[j]| < R$ then
            $disntance = |P[i] - Q[j]|$;
            $cnt = 1$ while $D_{consistent}(P[i : i + cnt]), Q[j : j + cnt] < R$ do
                $cnt++$;
            $patternSet.add(P[i : i + cnt - 1], Q[j : j + cnt - 1])$;
```

However, though the algorithm eliminated short repeated patterns, it still produces long trivial matches. We now define trivial match formally as definition 16.

As in definition 16 and figure 1.2d, trivial matches are essentially same patterns which are slightly different. Lin et al. suggest that trivial matches occur more often in smooth time series [84]. Since the noise reduction method smoothed our host load data, many trivial matches can be found which slow down the patterns discovery process. A simple method is recording the increment of distance when searching a time series data. Typically when the distance between a pattern series and a time series data reaches a local minimum, the current match is possibly the best match. Certainly, this method will take more time. To balance the performance and matching quality, this work develops a simple but efficient algorithm, which is outlined in algorithm 4.

Definition 16. *(Trivial Match) In a time series data, given a subsequence C beginning at position p, a matching subsequence M beginning at q and a distance R, we claim that M is a trivial match to C of order R, if either $p = q$ or there does not exist a subsequence M' beginning at q' such that $D(C, M') > R$, and either $q < q' < p$ or $p < q' < q$.*

Algorithm 4: Trivial match skip algorithm

```
input  : CPU host load: P, Q;
         Maximum allowed distance: R;
output : Set of patterns
for $i = 0; i < P.length; i++$ do
    for $j = 0; j < Q.length; j++$ do
        if $P[i] \notin patternSet \;||\; Q[j] \notin patternSet$ then
            if $|P[i] - Q[j]| < R$ then
                $disntance = |P[i] - Q[j]|$;
                $cnt = 1$ while $D_{consistent}(P[i : i + cnt]), Q[j : j + cnt] < R$
                do
                    $cnt++$;
                $patternSet.add(P[i : i + cnt - 1], Q[j : j + cnt - 1])$;
```

Algorithm 4 eliminates trivial results by not comparing subsequences that start from found patterns. Since trivial matches are abandoned before calculation, this algorithm speeds up the search process even further.

5.3.3 Cascade Pattern Discovery

To further accelerate pattern discovery process, it is necessary to represent CPU host load using representation methods introduced in Section 5.1. The lower bounding property of representation methods we use allow us to skip most of the dissimilar patterns which account for a very high proportion of search space as early as possible. Considering the rarity of true patterns, this cascade method can greatly reduce distance measurements in original space.

For a pattern pair which considered to be similar, their distance under both raw data space and indexed space are sure to be at least lower than the threshold given, and we only know the pair are similar after we measured their distance under raw data space. But another case is, a pair of patterns are similar under indexed space, but not actually similar under raw data space. In this case, we waste time on measuring the true distance between a pair of dissimilar patterns. The latter case is what we stress on, that is, to skip as much as dissimilar pairs as possible before they have to be calculated under raw data space.

In the field of mining known patterns in time series data, Rakthanmanon et al. [126] applied a cascading method to accelerate mining process. Similarly, we can transplant the idea to mining unknown patters.

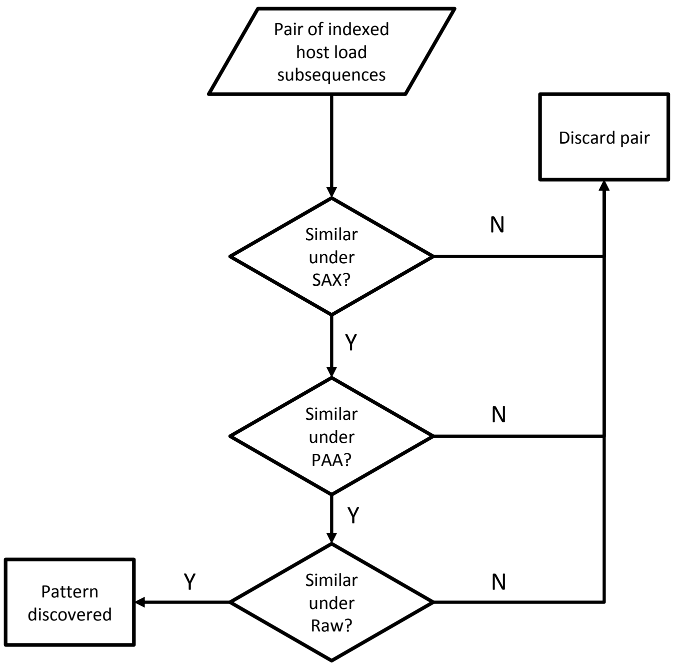


Figure 5.3: Cascade of different indexing technique for pattern discovery

Figure 5.3 shows how cascade indexed pattern discovery improves the efficiency of the algorithm. Initially, two series of CPU host load data are indexed with SAX, under the same length of dimensional reduction rate. Meanwhile, we also have PAA indexed version of the two host load, as well as raw data. We apply our trivial match skip algorithm to find possible patterns. Since SAX indexing guarantees lower-bounding, we can confidently say all unqualified pattern pair candidates are removed and the remaining host load subsequence pairs contains all possible patterns. Then, we retrieve PAA indexed version of qualified candidates under SAX representation and apply trivial match skip algorithm to the data. Again, PAA also lower bounds Euclidean distance, and all candidate pattern pairs measured greater distance than the given threshold are disposed. Since the tightness of PAA lower bounding is higher than that of SAX, now we have only a small part of remaining candidates compared to qualified candidates under SAX representation. For this small part of candidates, we will have to compare their real distance by retrieving raw data correspond to them. However, we have eliminated most of the unqualified candidates by applying pattern discovery algorithm on reduced search space and avoided unnecessary calculation to the best we can.

5.4 Experimental Evaluation

We performed an experimental evaluation of the effectiveness of our proposed method. The experiments were conducted on an Intel Core i5 4-core 3.2-GHz machine with 16GB memory. Our experiments aim to answer the following questions:

1 The effectiveness of our pattern discovery algorithm.

2 The efficiency of pattern discovery.

3 How PAA and SAX representation effects pattern discovery efficiency.

4 How much efficiency cascade method can perform than mining raw data.

The experiments we conduct are based on Google cluster trace, which records the resource usage of 12,000 machines of a real-world cluster, spanning 30 days. We reorganised the dataset to gather CPU host load of each machine. All the data are z-normalized to eliminate the negative effect of scaling and vertical shift.

5.4.1 Mining patterns in Google Cluster Trace

To illustrate the effectiveness of our proposed pattern discovery method, we start with showing some discovered patterns in the whole dataset.

As discussed earlier, the aim of our algorithm is to locate similar subsequences in different CPU host load series. Figure 5.4 shows patterns found in the Google cluster trace dataset. The 5 host load series, a, b, c, d and e are collected from 5 different machines in a Google cluster. To show the effectiveness of our method, we discover subsequences in host load b, c, d and e that are similar to subsequences in host load a. Once a pattern is found, we cover the pattern with a coloured rectangle to show its position in both host load a, the query host load, and the content load where the pattern is located in.

As shown in figure 5.4, each of host load b, c, d and e has one subsequence that is similar with part of host load a. In host load b, the subsequence covered with the blue rectangle is similar to that covered with the blue rectangle in host load a. Similarly, in host load a and c the red rectangle covered subsequences are similar. In d, the green rectangle covered subsequence is similar to that in a and purple rectangle covered subsequence in e is similar to that in a as well. This intuitive experiment shows that our method finds patterns effectively. It is clear that host load b, c and d have similar periodicity as host load a, giving the fact that in this dataset periodical CPU host load are relatively rare. This result strongly suggests that host load which share same similar patterns are more likely to be the same type, which indicates the possibility of using discovered patterns to achieve better classification for CPU host load.

5.4.2 Efficiency of Indexing

Although the efficiency of indexing methods have been compared in various papers [105, 108], we still want to investigate whether the performance results follow the similar trend when they are applied for the pattern discovery in the host load. As it has been shown in [83], different types of datasets can

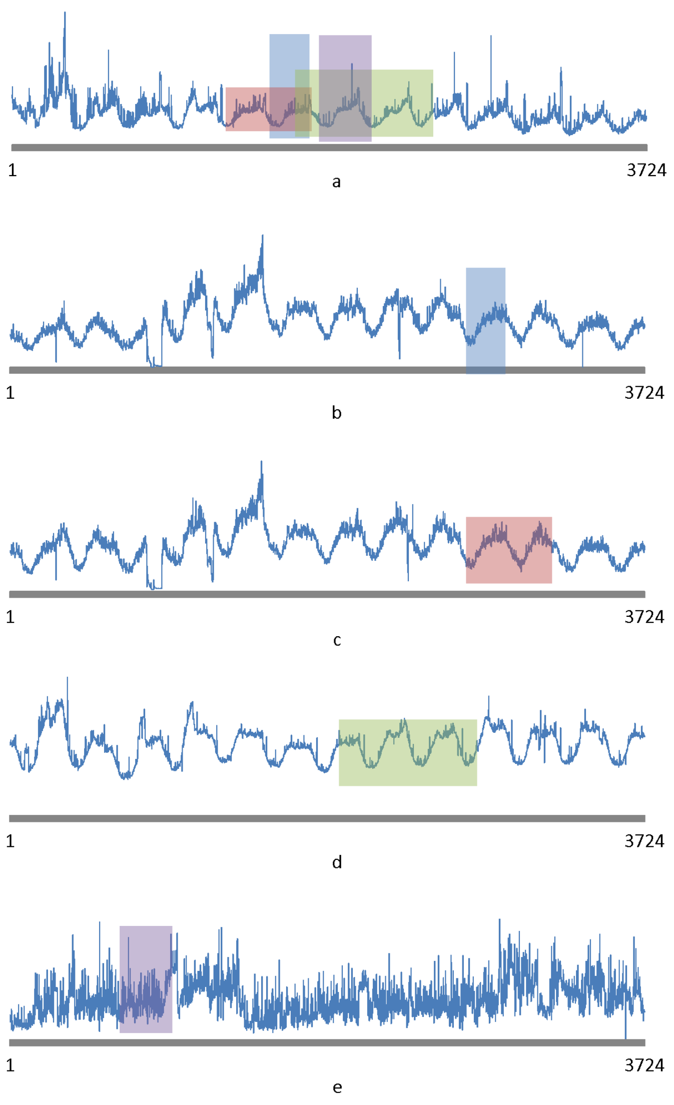


Figure 5.4: Patterns discovered using host load a as reference

dramatically affect the efficiency of the indexing methods.

The efficiency of an indexing method can be defined as follows.

Definition 17. *(Indexing Efficiency) Given an indexing method I with the dimensional reduction rate R, the indexing efficiency of I is the time spent in indexing with the reduction rate R.*

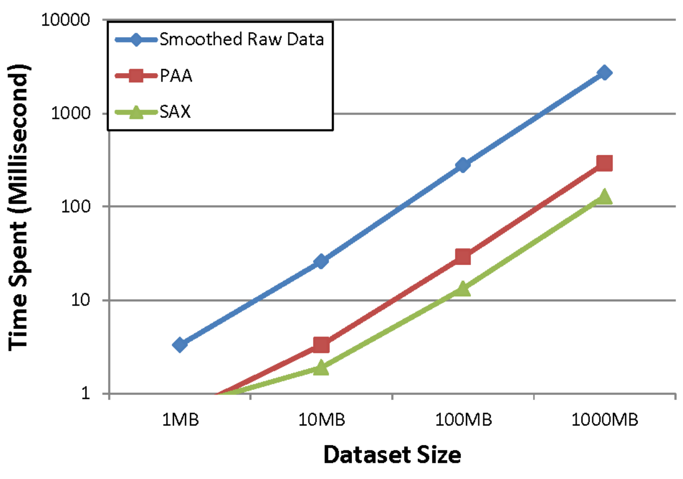


Figure 5.5: Indexing efficiency of three indexing methods with dimensional reduce rate of 10

We conducted the experiments to compare the indexing efficiency among raw data indexing, PAA indexing and SAX indexing methods. We take 1MB, 10MB, 100MB and 1000MB of test dataset from the whole dataset. In each dataset, we take a host load series as our query. By linearly searching the test dataset, all host load within R from the query should be found according to the definition.

It can be seen from figure 5.5 that PAA and SAX reduce the time consumption greatly, and the SAX indexing is even faster than the PAA indexing. It is notable that the PAA indexing is approximately 10 times faster than the raw data indexing, which is expected as we use 1 PAA data point to represent 10 raw data points, namely the reduction rate is 10. As for the SAX indexing, since its distance measure comes from looking up a distance table, it is slightly faster than PAA.

As mentioned above, the reduction rate is the fundamental factor that affects the efficiency of indexing. Meanwhile, it also increases the probability of incorrect indexing.

Definition 18. *(False-positive indexing result) Given an indexing method $S = I(Q)$, where S is the resulting data of a query data Q. a false-positive indexing result is a time series T_{wrong} in S where the distance between T_{wrong} and Q under I is less than R, but the distance under raw data indexing is greater than R.*

The solution of the false-positive indexing problem is simple. We can deploy a cascade indexing method, in which the lower level of the indexing method returns a dataset of possible results, while the higher level identifies and throws away the false-positive indexing results. The requirement for a cascade indexing method is that the lower level indexing should be fast and provide the lower bound of the real distance, while the higher level of indexing should return the exact distance. If the lower level indexing is unable to determine the lower bound of the distance measure, it is possible that the indexing method misses a qualified result. Our carefully selected indexing method, namely SAX and PAA, provide lower bounding.

In [105], the authors give the notion of the tightness of lower bounding to indicate how accurately the distance in the representation space can represent the real distance. Theoretically, the more dimensional reduction in the representation space, the less tightness the lower bounding is. Although higher dimensional reduction rate can accelerate the searching process, the later exact search may slow the whole process down. To show the numerical relation between indexing efficiency and dimensional reduction rate, we conducted the experiments and plot the results in figure 5.6.

In the experiments, we used the three indexing methods on a 1000MB dataset with different dimensional reduction rate and recorded the average time of returning the result from a query. As shown in figure 5.6, the ratio of efficiency

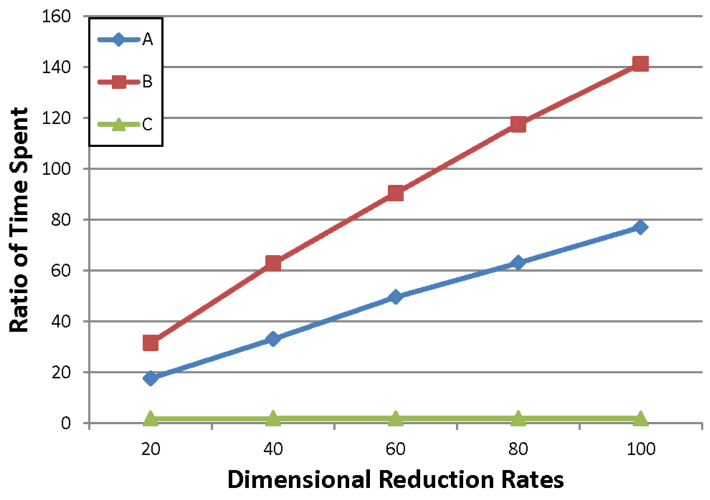


Figure 5.6: The proportional relation of efficiency between three indexing methods with different dimensional reduction rates. $A = \frac{t(SmoothedRawData)}{t(SAX)}$, $B = \frac{t(SmoothedRawData)}{t(PAA)}$, $C = \frac{t(PAA)}{t(SAX)}$, where $t(X)$ is the time spent with the indexing method X

between raw data and SAX increase linearly with the dimensional reduction rate. The same trend is also observed for the PAA indexing. Therefore, we can conclude that the efficiency of each indexing method can be deduced from equation 5.15, where I is the PAA or SAX indexing method, Raw_data is raw data indexing and a is a constant coefficient.

$$Efficiency(I) = \frac{Efficiency(Raw\_data)}{Dimensinal\_reduction\_rate * a} \tag{5.15}$$

According to the experimental results in figure 5.6, we can determine the coefficient a is 0.82 and 1.5 for the PAA indexing and the SAX indexing, respectively. Therefore, the PAA indexing is $0.82 * dimensional\_reduction\_rate$ times faster than the raw data indexing, while the SAX indexing is $1.5 * dimensional\_reduction\_rate$ times faster. We can also deduce that the SAX indexing is 1.8 times faster than the PAA indexing. This result strongly supports the hierarchy structure of our cascade discovery method, where the most efficient indexing method designed to be on the top and the most inefficient indexing method on the bottom.

5.4.3 Efficiency of Pattern Discovery

In Section 5.3, we presented three pattern discovery methods and a cascade discovery framework to further accelerate the process. Theoretically, the brute force method is expected to be the slowest, while the trivial match skip method is the fastest and can produce the best result. We start with comparing how much trivial match skip algorithm improved efficiency of the other two algorithms, and then we evaluate the performance of cascade accelerating method.

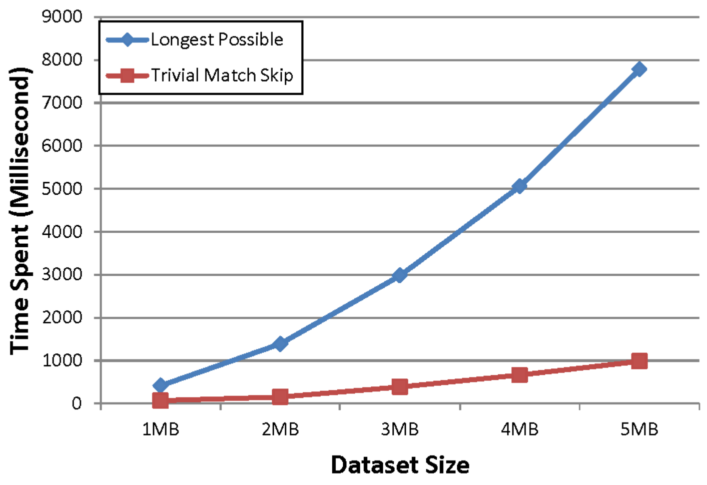


Figure 5.7: The time spent by the two algorithms with different data sizes; the dimensional reduction rate is 20

As shown in figure 5.7, as the dataset size increases, the time spent by the second algorithm (i.e., the method to discover the longest possible pattern) increases very fast. By contrasting our improved algorithm which skips the trivial matches can still finish in a reasonable time. The brute force algorithm is unable to produce any result in an acceptable time.

To evaluate the efficiency of pattern discovery under cascade framework under different settings of parameters, we mine patterns hide in an arbitrarily selected host load in a 1.20GB dataset contains more than 12,000 CPU host load come from different machines in a cluster of Google. The experiment is repeated for 30 times to acquire average value. For each selected host load, we mine patterns by setting a variety of PAA dimensional reduction rate ranging from 5 to 50, and the number of symbols used in SAX representation, ranging from 30 to 110. The result is shown in figure 5.8.

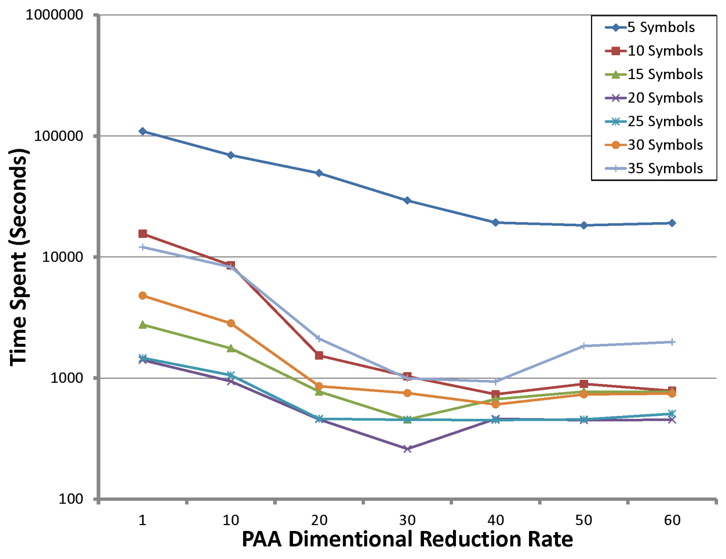


Figure 5.8: Time spent of cascade mining method under different patameter set

As in figure 5.8, It is clear that the number of symbols assigned to SAX representation decides how fast the algorithm process. When more symbols are assigned to a host load, distance measure under SAX representation bounds more tightly to the true distance of the raw data, thus unqualified pattern pair candidates can be easier eliminated before proceed to retrieve PAA indexing. However, when the number of symbols assigned increase, the benefits the algorithm takes is lessened. Assigning more symbols than enough, in this case, 30, will not result in notable improvements in efficiency.

Another factor influences time consumption of cascade pattern discovery method is dimensional reduction rate of PAA. When the rate is 1, PAA indexing simply degrades to raw data, which result in the absence of the middle level of our cascade method, and all unqualified candidates which are unable to be picked out by SAX indexing will be handed over directly to raw data pattern discovery, which is much less efficient. However, when the dimension reduction rate is greater than a certain value, the efficiency of the cascade framework will also be reduced. The reason is given by [22]. When dimensional reduction rate increase, the PAA indexed time series tend to have less average value and standard deviation, result in poor tightness of lower bounding. In this case, the ability to identify unqualified pattern pair candidates of the middle level of our cascade model is weakened.

5.5 Summary

In this chapter we present a host load reduction based pattern discovery method. This method applied two levels of reduction steps, PAA representation and refined SAX representation. To increase efficiency while maintaining accuracy, a cascade discovery method is proposed.

To find exact patterns, it is important to reduce dimensionality and numerosity of CPU host load. This chapter discussed the effectiveness of SAX representation and suggested that the original work does not guarantee equal probability of symbol assigning when representing CPU host load. The defect is fixed by measuring the actual distribution of CPU host load data.

We developed DTW distance on both PAA and SAX representation, as well as their similarity consistent version. This contribution makes unknown similar pattern discovery under the two representation possible and can be extended to other time series data mining applications.

Though brute force pattern discovery algorithm has shown to be infeasible, we achieved efficient discovering by improving brute force method. The major problems of brute force method are high time complexity and producing meaningless results as shown in Chapter 1. By applying reduction methods, the raw data are reduced to a reasonable amount. Our method also eliminated overlapped patterns and trivial match patterns by finding longest possible patterns and trivial match detection. The two techniques also increase the efficiency of the discovering process by avoiding repeated calculation.

To find patterns accurately and efficiently, a cascade discovering method is proposed in this chapter. The lower bounding property of both PAA and SAX ensure all patterns can be found in CPU host load. Experimental results indicate the method is efficient and effective.

Chapter 6

Iterative Similar Pattern Discovery in Time Series Data

CPU host load is essentially time series data. Previously presented pattern mining methods for CPU host load can also be applied to some time series data. On the other hand, a time series pattern discovery method can be immediately applied to mining patterns in CPU host load.

The two methods proposed for mining patterns in CPU host load in Chapter 4 and 5 discover patterns in CPU host load either efficiently or less efficient but accurately. However, the two methods have their limitations when being applied to other time series data. The segments clustering method proposed in Chapter 4 requires time series that have smooth and undulant trends. Though the method proposed in Chapter 5 has fewer requirements to the input data, it still prefers smooth data that have long-term trends. These requirements reduced the applicability of the two methods. A universal time series pattern mining method that covers most time series data types not only addresses the subject of mining time series patterns which has been less focused in the literature, but also helps discover CPU host load patterns in a more fundamental point of view.

In this chapter, we investigate mining patterns in time series data from a higher and more essential level to develop a general method for discovering previously unknown patterns in time series data. The method has two steps: approximate pattern position locating and exact pattern discovery. Section 6.1 presents how we transfer expectations for patterns to prior knowledge of patterns to help to propose an efficient pattern discovery method. In Section 6.2 we propose a method to find approximate pattern positions. Then in Section 6.3 we utilise the found possible locations to refine the result iteratively. Finally in Section 6.4 experiments are conducted to illustrate the effectiveness and efficiency of the method.

6.1 Creating Prior Knowledge for Patterns

As illustrated in Section 1.2, the difficulties we are facing is that we do not have prior knowledge of the patterns we are mining. Though little knowledge about the pattern can be extracted from the data, we still have our expectation on the patterns to be discovered. A pattern mining algorithm is expected to find patterns that have following characteristics:

1. The first and most obvious feature of a pattern is that it contains two similar time series subsequences. In other words, the distance of the two time series subsequences should be lower than a given similarity threshold.
2. Patterns should have moderate lengths. One of the reasons why the brute force method is not feasible is that the method produces many results which have only a few data points for each of them. These very short patterns contain little information and are often not understandable due to their limited lengths. Moreover, short time series data subsequences are more likely to have low distance value because the higher probability for fewer data points to fall into the same range of values. Imagine the task is to find whether two pieces of music contain similar melody, it is not

possible to support the decision by two 1 second clips, though the pitch of the sound clips is the same.

3. Subsequences of a pattern should have similar lengths. The difference in lengths of subsequences indicates their difference in time spans, which often generated by different types of events. A specific case of whether two similar subsequences should have similar lengths is often argued against for the fact that time series data can be collected by sensors with different sampling rates [141]. However, the sampling rate problem can be solved simply by resample the shorter time series to the length of the longer ones before mining process rather than allowing subsequences with huge length difference to be comparable, which could also bring in false positive results.

4. A pattern should not be a part of another pattern. The motive of discovering the longest possible pattern is to avoid the pattern containing problem. If we have two matching time series, the subsequences of corresponding time slot will match as well, and will be considered to be a pattern in brute force algorithm. However, this shorter pattern is a redundant result because information of the shorter pattern is already contained in the longer one.

5. The subsequences of a pattern should not come from trivial matches. The concept of trivial match has been proposed in [84]. The original work points out that given a similarity threshold, the subsequences around the best match subsequence of a given query also similar to the query. We can generalize the idea to pattern discovery as shown in figure 1.2d. Giving a similarity threshold, the similarity of subsequences pairs around the best match subsequence pair are tend to be less than the threshold as well. Essentially, trivial matches are repeats of the best match subsequence and are also redundant results.

We can utilize the requirements of patterns as prior knowledge to design an algorithm which is able to perform fast and accurate similar pattern discovery.

6.2 Approximate Similar Pattern Position Locating

In this section, we illustrate the first phase of our proposed method, Approximate Similar Pattern Locating algorithm (ASPL). This phase locates the approximate positions of possible patterns. Provided with the position information, we can then apply accurate pattern search algorithm in phase two of our proposed method. We start introducing ASPL by introducing a naive method, and then introduce ASPL by improving the naive algorithm.

6.2.1 Naive Pattern Position Locating

As presented above, one of the difficulties of time series unknown pattern discovery is that we have little knowledge of the pattern we are to find. However, from definition 5 we can deduce an obvious theorem:

Theorem 1. *Giving two time series subsequences* $R = \{p_i, p_{i+1}, \cdots, p_x, \cdots, p_j\}$ *and* $S = \{q_k, q_{k+1}, \cdots, q_y, \cdots, q_l\}$ *and distance threshold* τ*, if* R *and* S *match, then there exist* p_x *and* q_y *where* $i \leq x \leq j$ *and* $k \leq y \leq l$ *make* $|| p_x - q_y || \leq \tau$ *.*

We can utilize this property to estimate the starting indices $[i, j]$ of a similar pattern starting form i^{th} and j^{th} data points of the two time series respectively. The naive method uses nested loops to compare each possible pair of data points in each time series. We present the algorithm briefly:

Algorithm 5: Naive pattern starting position locating algorithm

```
input  : Time Series $P = \{p_1, \cdots, p_n\}$, $Q = \{q_1, \cdots, q_m\}$
output : List of positions
for $i \leftarrow 1$ to $n$ do
    for $j \leftarrow 1$ to $m$ do
        if $|| p_i - q_j || \leq \tau$ then
            Add $[i, j]$ to position list;
```

This algorithm finds all positions where the distance of corresponding time series data points less than τ. Physically, the algorithm generates cells of a

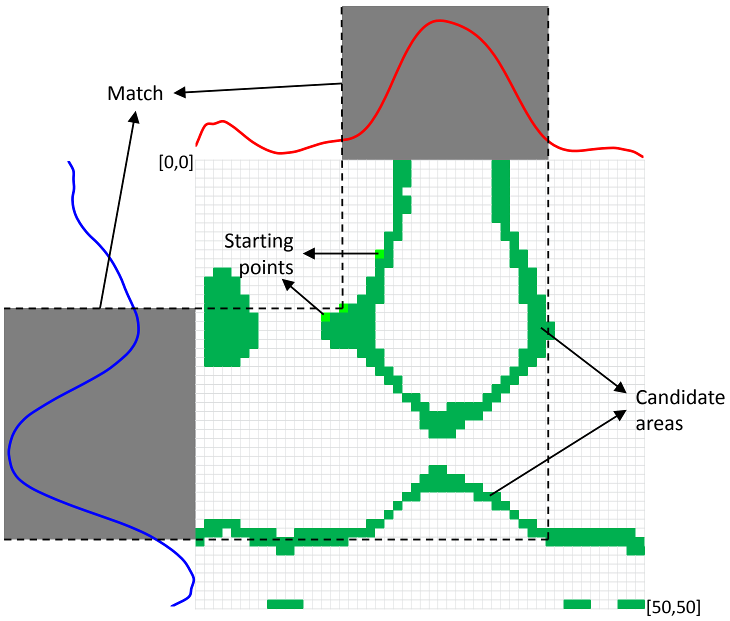


Figure 6.1: A visualization of algorithm 5 and 6. The central matrix is the imagined matrix produced by algorithm 5. Green cells are "candidate areas" where $|| p_i - q_j || \leq \tau$ found by algorithm 5. The light green cells are part of possible starting positions found by algorithm 6.

m by n matrix (note that this matrix only exists in imagination for the ease of discussion), each cell represents a distance value of the corresponding data points of each time series. Then the algorithm cut off cells where their values are greater than τ. The remained cells are typically very close or neighbouring to each other because of the existence of trivial matches. We call a set of neighbouring cells which satisfy $|| p_i - q_j || \leq \tau$ a "candidate area" of all possible starting positions. The concept is illustrated in figure 6.1. The two gray-shaded subsequences in figure 6.1 are a match under $\tau = 0.11$ in this example. Knowing the two subsequences are a match, the drawbacks of this naive algorithm are obvious. First, this algorithm produces too many abundant positions. Each candidate area indicates a possible pattern across it. Similarity measurement of two subsequences starts either from their first data points or their last data points, therefore, only one cell for each candidate area is required to locate a match. However, with all cells in candidate areas to be considered, we have to repeat the

Table 6.1: Example of time series grouping compare

Time	1	2	3	4	5	6	7	8	9	10
A	1	2	3	6	2	4	5	1	2	6
B	2	3	3	1	5	3	2	5	5	1

(a) Time series A and B

Group	A_1				A_2		A_3			
Time	1	2	5	8	9	3	6	4	7	10
A	1	2	2	1	2	3	4	6	5	6

(b) Groups of time series A,$\tau = 2$

Group	B_1				B_2			B_3		
Time	1	4	7	10	2	3	6	5	8	9
B	2	1	2	1	3	3	3	5	5	5

(c) Groups of time series B,$\tau = 2$

measurement for each of these positions, which will be a time-consuming process. Second, the time complexity of this algorithm is $O(mn)$, which is too high for a scalable method. Actually, state of the art [141] can already find all optimal matches under the same time complexity. We improve this naive method in two ways: search space reduction and invalid results reduction.

6.2.2 Search Space Reduction

The first improvement of the naive algorithm is reducing the search space required to accelerate the algorithm.

From figure 6.1 we can observe that the white area occupies most space of the matrix. In fact, notable matches are relatively sparse for most time series data. For two time series, we can group their data points by their values so that only data points in neighbouring groups are possible to be similar.

For conciseness of discussion, we assume that all data points in each time series are non-negative in the discussion of this subsection. It is easy to extent the method and conclusion to real number time series.

We use a simple example to illustrate this method. Suppose two time series $A = \{1, 2, 3, 6, 2, 4, 5, 1, 2, 6\}$ and $B = \{2, 3, 3, 1, 5, 3, 2, 5, 5, 1\}$ as in table 6.2a. We set the similarity threshold to be $\tau = 2$. Each data point x_i is assigned to the w^{th} group where w is calculated by following equation:

$$w = \begin{cases} \lceil \frac{x_i}{\tau} \rceil, & x_i \neq 0 \\ 1, & x_i = 0 \end{cases} \tag{6.1}$$

In our example shown in table 6.2b, data points of time series A are assigned to 3 groups: A_1, A_2 and A_3. Similarly, in table 6.2c, data points of time series B are assigned to B_1,B_2 and B_3. Because this method rearranges the original temporal order of time series, it is necessary to record time stamp of each data point (second row of table 6.2b and 6.2c) when manipulating. In this example, we can see that data points in group A_1 and B_1, A_2 and B_2, A_3 and B_3 are similar. Data points in A_2 only need to be compared with data points in B_1 and B_3, and data points in A_1 only need to be compared with data points in B_2.

We can draw a conclusion that giving two time series A and B, data points in w^{th} group of time series A must be similar with the w^{th} of group B, data points of the w^{th} group of time series A are only possible to be similar with data points of the $(w-1)^{th}$ and $(w+1)^{th}$ group of time series B. Therefore, for each group A_w in time series A, we only need to compare its data points to data points in B_{w-1} and B_{w+1}. The total comparison c conforms the following equation:

$$c = \sum_{w=1}^{i} \mid A_w \mid \times (\mid B_{w-1} \mid + \mid B_{w+1} \mid) \tag{6.2}$$

where $\mid X \mid$ is the number of data points in group X and i is the number of total groups which can be calculated by the equation below:

$$i = \lceil \frac{max(A \cup B) - min(A \cup B)}{\tau} \rceil \tag{6.3}$$

According to equation 6.2, the total comparison needed for example shown in table 6.2a is 38, by contrast this value is 100 in naive algorithm. In practical, τ is typically a very small value, so the total comparisons needed can be considerably reduced by several orders of magnitude of that required by the naive algorithm.

6.2.3 Invalid Results Reduction

To decide which cell is the best cell to start searching optimal match is as difficult as locating the exact position of the optimal match itself. However, as shown in Section 6.3 our proposed search algorithm is capable of locating the exact position of nearest optimal match of any given starting points. The closer the given point is to the actual start point of an optimal match, the faster our algorithm will find the actual position. By knowing this, we should now concern two questions: how to ensure we can find all optimal matches and how to find cells near the actual optimal match.

The answer to the first question is that we need at least one cell for each candidate area. candidate areas are the cells where $|| \ p_i - q_j \ || \leq \tau$ and for an optimal match, its optimal warping path must satisfy $|| \ \sum_{i=1}^{l} D_{a_i} \ || \leq \tau$. Therefore, the warping path of any optimal match will cross at least one candidate area. Searching all candidate areas will guarantee all optimal matches are found.

Definition 6 indicates that optimal match is the longest possible match with its similarity less than τ. Typically, a candidate area is a rectangle-like polygon and covers up to one pattern. An usual case is that the warping path of a pattern starts from the near corner (closer to $[0, 0]$ of of the cost matrix) of a candidate area , and ends at the far corner (closer to $[m, n]$ of the cost matrix) of another candidate area. Without knowing the location of the optimal warping path, it is natural that we start searching from the near corner cell of a candidate area. However, determining which cell of a polygon candidate area is closer to $[0, 0]$ requires knowing the exact shape of the candidate area, which is another time-consuming task. Instead of doing this, we introduce the concept of "qualified cell" as below:

Definition 19 (Qualified cell)**.** *Giving two time series* $X = \{x_1, x_2, \cdots, x_i, \cdots, x_m\}$ *of length* m *and* $Y = \{y_1, y_2, \cdots, y_j \cdots, y_n\}$ *of length* n *and similarity threshold* τ, (x_i, y_j) *is a qualified cell if and only if:*

1. $|| \ x_i - y_j \ || \leq \tau$

2. *if at least one of* (x_{i+1}, y_j),(x_i, y_{j+1}) *and* (x_{i+1}, y_{j+1}) *exist, they must satisfy one of* $\| x_{i+1} - y_j \| \leq \tau$, $\| x_i - y_{j+1} \| \leq \tau$ *and* $\| x_{i+1} - y_{j+1} \| \leq \tau$

3. *If at least one of* (x_{i-1}, y_j),(x_i, y_{j-1}) *and* (x_{i-1}, y_{j-1}) *exist, they must satisfy all of* $\| x_{i-1} - y_j \| > \tau$, $\| x_i - y_{j-1} \| > \tau$ *and* $\| x_{i+1} - y_{j+1} \| > \tau$

Intuitively, we simply record any cell which has no candidate cells on its left, top and top-left neighbours, but has candidate cells on its right, bottom or bottom-right neighbour. The benefit of doing so is that we do not need to search all the candidate area to ascertain the most promising starting points. The stand-alone cells can also be eliminated where $D[i, j] \leq \tau$ but has no neighbouring candidate cells. In this way, one candidate area may have several cells recorded rather than only one.

Having our improvements introduced, the fast pattern locating algorithm is described as algorithm 6.

Algorithm 6: Fast pattern locating algorithm

```
input  : Time Series: $P = \{p_1, \cdots, p_n\}$
         $Q = \{q_1, \cdots, q_m\}$
         Similarity threshold: $\tau$
output : List of positions
for $i \leftarrow 1$ to $n$ do
    if $p_i == 0$ then
        Add $p_i$ to groupP[1];
    else
        Add $p_i$ to groupP[$\lceil \frac{p_i}{\tau} \rceil$];
for $j \leftarrow 1$ to $m$ do
    if $q_j == 0$ then
        Add $q_j$ to groupQ[1];
    else
        Add $q_j$ to groupQ[$\lceil \frac{q_j}{\tau} \rceil$];
for $w \leftarrow 1$ to groupCount do
    foreach $p_a$ in groupP[$w$]
        foreach $q_b$ in groupQ[$w \pm 1$]
            if Qualified($p_a$,$p_b$) then
                Add $[a, b]$ to position list;
```

6.3 Similar Pattern Discovery Based on Possible Starting Points

In section 6.2 we have found a set of approximate potential positions as initial starting positions of locating exact similar patterns, but yet we know the length of the similar patterns. In other words, we also need to ascertain the ending positions similar patterns. In this section we introduce our exact optimal match locating algorithm. We begin with a naive match locating algorithm to present the distance measure we used as well as the necessity of iterative locating, then we introduce our proposed exact similar pattern discovery algorithm.

6.3.1 Naive Similar Pattern Discovery

Giving two time series $P = \{p_1, \cdots p_i, \cdots, p_n\}$ and $Q = \{q_1, \cdots, q_j, \cdots, q_m\}$ and a n-by-m matrix D is the DTW cost matrix of P and Q. According to definition 8, $D[i, j]$ is the distance value of subsequences $\{p_1, \cdots, p_i\}$ and $\{q_1, \cdots, q_j\}$. In other words, the cost matrix contains all distance value of any two subsequences start at p_1 and q_1 respectively. It is natural to start measuring DTW distance starts at starting positions found by method shown in section 6.2 to locate the proper end position of a match. Since the similarity measure we use (equation 3.4) requires the length of warping path for each cell in D, we propose the dynamic warping path length K calculate method as below:

$$\begin{aligned}
&K(0,0) = 0\\
&K(i,j) = 1 + K(a,b)\\
&where\\
&D(a,b) = min \begin{cases} d(x_{i-1}, y_j) \\ d(x_i, y_{j-1}) \\ d(x_{i-1}, y_{j-1}) \end{cases}
\end{aligned} \tag{6.4}$$

For each cell of cost matrix we calculate $DTW_{consis}(P_i, Q_j) = \frac{D(i,j)}{K(i,j)}$ as shown in equation 3.4. If $DTW_{consis}(P_i, Q_j)$ is greater than similarity threshold τ, we return the corresponding position as the end position of minimum value of last row calculated $Min(D[i-1, *])$. The algorithm is shown in 7.

Algorithm 7: Naive pattern searching algorithm

```
input  : Time Series: $P = \{p_1, \cdots, p_n\}$
         $Q = \{q_1, \cdots, q_m\}$
         Similarity threshold: $\tau$
         DTW window width: $w$
         Starting position: $C = (s, t)$
output : Ending position: $E = (x, y)$
$prevMin = (1, 1)$;
// Initialize previous minimum distance cell position
while row $i \leftarrow 1$ to end do
    if $\frac{Min(D(i,*))}{K(index\_of(Min(D(i,*))))} > \tau$ then
        return $(prevMin)$;
    else
        $prevMin = index\_of(Min(D(i,*)))$;
```

As shown in figure 6.2, the algorithm searches for similar subsequences from each starting point. The algorithm measures DTW starting from a starting point, and a DTW cost matrix is then built and a Sakoe-Chiba constraint is applied to the matrix to save computational resource and avoid over-warping. Once the minimum distance value in a row of cost matrix is greater than distance threshold τ, the algorithm returns the cell of the last row with minimum distance value as the ending position of the match.

Apparently, algorithm 7 does not find optimal matches, since the starting points given by algorithm 6 is not guaranteed to be optimal starting points. Also, the algorithm is not capable of avoiding repeated match. In figure 6.2, several other starting points (green cells) can be observed around the one we selected. By comparing this figure to figure 6.1, it is obvious that matches starting from these starting points will eventually turn out to be trivial matches of the one we show in figure 6.2 as they share the same parts of candidate areas. The existence of trivial matches is not only a waste of computational resources but also cause

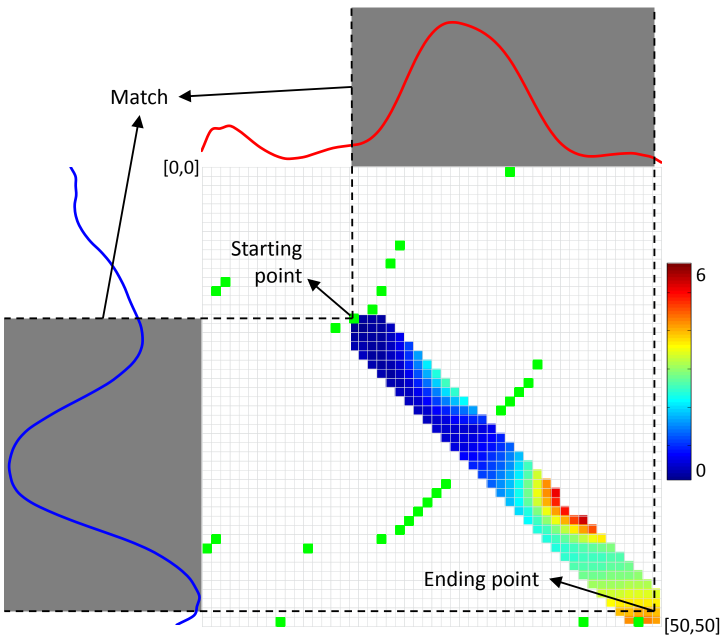


Figure 6.2: A visualization of algorithm 7. The search starts from a staring point and continuously builds DTW cost matrix until the similarity is greater than τ. Note that a constraint window is applied to DTW.

an unnecessary process of picking out the most representative match from a bunch of similar trivial match results.

6.3.2 Iterative Exact Similar Pattern Discovery

Our proposed exact similar pattern discovery algorithm is based on the fact that the optimal warping path of an optimal match always extends along the "valley" area (where values are lower) of the DTW cost matrix built by the optimal match to achieve lowest total cost as shown in figure 6.3a. It is natural thinking of detecting these "valley" areas on cost matrix to locate similar subsequences, but the dynamic programming natural of DTW makes it impossible because cost matrices are different for each starting point $[i, j]$. Different from Euclidean distance which guarantees for time series $X = \{x_1, \cdots, x_i, \cdots, x_m\}$ and $Y = \{y_1, \cdots, y_j, \cdots, y_n\}$ there will always be $D_{Euc}(X, Y) = D_{Euc}(X_{1:i}, B_{1:j}) + D_{Euc}(X_{i:m}, Y_{j:n})$, elastic distance measure, like DTW, does not possess this prop-

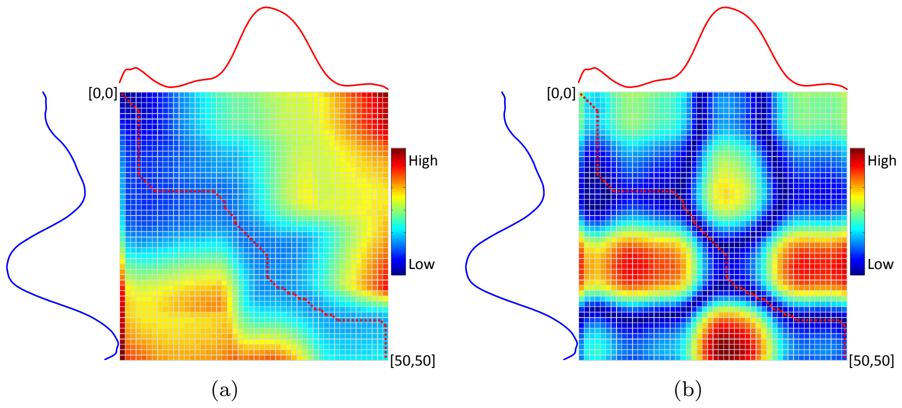
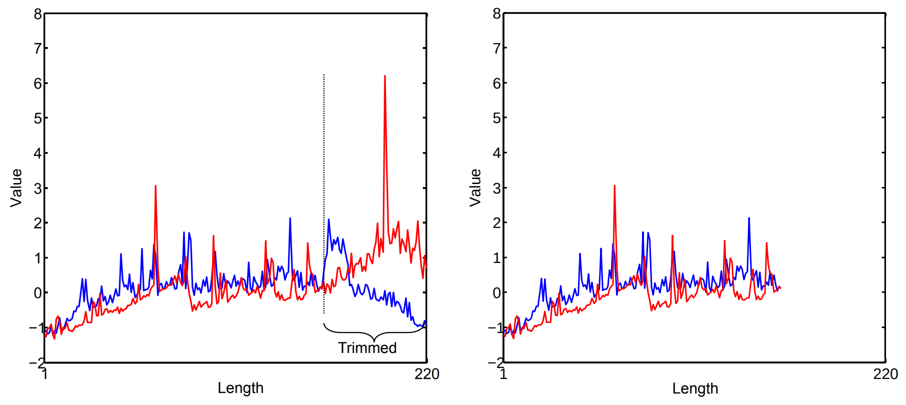


Figure 6.3: Cost matrix (left) and local cost matrix (right) of two example time series. The optimal warping path is plotted in red dash line.

erty, causing difficulties in extracting distance values of arbitrary subsequences pair from single cost matrix.

We can plot a local cost matrix where cell $[i, j]$ denotes $|| \ x_i - y_j \ ||$ of corresponding time series $X = \{x_1, \cdots, x_m\}$ and $Y = Y\{y_1, \cdots, y_n\}$, as shown in figure 6.3b.The warping path of two given subsequences respectively from X and Y can be regarded as a path which starts from a starting position and sums up all the values of cells it goes through then ends at an ending position. Particularly, the optimal warping path is one of all warping paths that has the minimum sum of these values. The optimal warping path tends to go through cells where values are low. Unlike on cost matrix where optimal warping paths always span across "valley", to achieve minimum total cost, optimal warping paths may go through cells with higher values on local cost matrix.

Our similar pattern discovery method works in an iterative way. First, the algorithm is given an arbitrary starting position $[a, b]$. Open-end consistent DTW starts measuring DTW distance for time series $\{x_a, x_{a+1}, \cdots, x_m\}$ and $\{y_b, y_{b+1}, \cdots, y_n\}$ until the distance value is greater than threshold τ as naive search algorithm does. The cell where measuring stops is marked as current ending position, say, $[c, d]$. Obviously in many cases, subsequences $\{X_{a:b}\}$, and $\{Y_{b:d}\}$ starts to be less similar before position $[c, d]$ as shown in figure 6.4 because



(a) Two match subsequences under $\tau = 0.37$. Data on right side of the dash line are obviously not similar.

(b) Trimmed subsequences using algorithm 8.

Figure 6.4: Illustration of algorithm 8

when two subsequences are similar, their succeeding data points are the cause of major increase of similarity value. The algorithm then traces back along warping path of current end position to trim dissimilar parts and updates current end position as shown in algorithm 8. This process makes sure the ending position of warping path closer to the warping path of the optimal match and hence results in a faster convergence. Then the algorithm takes this end position as its new starting position, and measures DTW distance in the same procedures but reversely from latter data points in time series to former data points in time series values. The two steps repeat until the starting position or end position converge. A step limit σ is given to remove short results. Since a Sakoe-Chiba band is applied to the measurement, the warping path of each measurement is restricted in the constraint window. The concept of this algorithm is shown in figure 6.5.

The iterative pattern locating algorithm is able to produce local optimal matches near given starting positions. As discussed in the former section, once a starting position is given for each candidate area, all optimal matches will be guaranteed to be found. But the original iterative discovery algorithm still suffers from producing repeated results. Because for each candidate area, algorithm

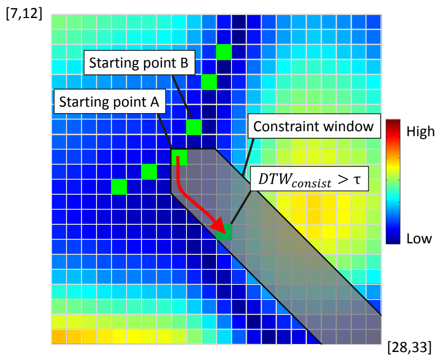
Algorithm 8: Dissimilar section trimming algorithm

```
input  : Cost Matrix: $D$
         Starting position $[x, y]$
         Current ending position $[a, b]$
         Similarity threshold: $\tau$
output : Updated current ending position: $[a', b']$
$[a', b'] = [a, b]$;
// Initialization
while $[a, b]! = [x, y]$
    if $ConsisDistance([a, b], [a', b']) \leq \tau$ then
        $[a, b] = GetPreviousPoint(D, [a', b'])$;
    else
        $[a, b] = GetPreviousPoint(D, [a', b'])$;
        $[a', b'] = [a, b]$;
return $[a', b']$;
```

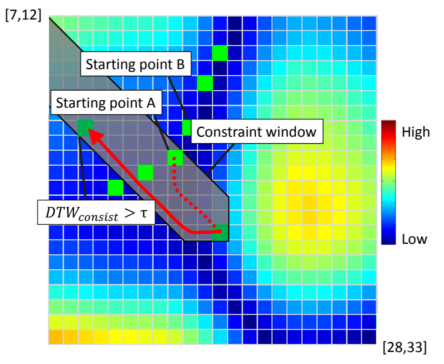
6 may produce more then one starting position, and even if we have only one starting position for each candidate area, an optimal match may probably span cross several candidate areas, making starting positions of these candidate areas reduce to the same optimal match. This phenomenon pollutes results with repeated matches and wastes computational resources. When iteratively searching for an optimal match from a starting position, we mark cells which warping paths of the iterative algorithm go by as a visited cell. If warping paths of another iterative search start from a different starting position overlap with a went-by cell, the search stops and the algorithm moves on to the next starting position on the list. If the two searches' warping paths share same cells, they will result in finding same optimal matches because the warping path has been proved to be global optimal by the first search. The algorithm is shown in algorithm 9.

6.4 Experimental Evaluation

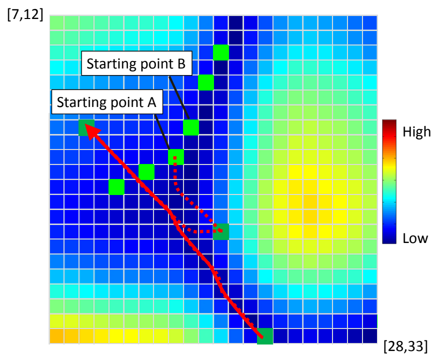
Experiments are performed to evaluate the effectiveness and efficiency of our proposed method. The experiments are conducted on a 3.2-GHz Intel Core i5 machine with 16 GB of memory running Windows 7. The experiments are



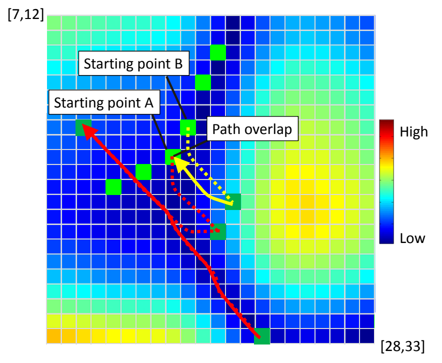
(a) Measure of DTW distance starts at starting point A and ends at the cell where $DTW_{consist} > \tau$. Trim result and report new ending position.



(b) Measure DTW distance reversely from previous ending cell and produce a new starting point.



(c) Process (a) and (b) repeat iteratively until warping path converges.



(d) Search starts from starting point B (yellow) overlaps with warping path (red) of search starts from starting point A, search stopped.

Figure 6.5: A visualization of algorithm 9

designed to respond these concern:

1. The effectiveness of the method.

2. The efficiency of the method.

3. Scalability of the method.

6.4.1 Locating Similar Patterns on Different Datasets

We use two synthetic datasets and four real-world datasets to validate the effectiveness of our proposed method. For each test dataset, we plot their time

Algorithm 9: Iterative exact pattern searching algorithm

```
input : Time Series: $P = \{p_1, \cdots, p_n\}$
        $Q = \{q_1, \cdots, q_m\}$
        Similarity threshold: $\tau$
        DTW window width: $w$
        Starting position list: $C = \{(s_i, t_i)\}$
        Step limit $\sigma$
output: Optimal match list: $E = \{[sx_j, sy_j], [ex_j, ey_j]\}$
foreach $(s_i, t_i)$ in Starting position list
    while true
        startPoint = $(s_i, t_i)$;
        $endPoint = FwardDTW(startPoint, w, \tau)$;
        // open end forward DTW without constrain of end point
        if $TraceBack(endPoint) == overlap$ then
            break;
        // If warping path overlap then break
        $newStartPoint = RevDTW(endPoint, w, \tau)$;
        // open end reverse DTW without constrain of end point
           and update start point
        if $TraceBack(newStartPoint) == overlap$ then
            break;
        if $startPoint == newStartPoint$ & $steps \geq \sigma$ then
            add $(newStartPoint, endPoint)$ to MatchList;
            break;
        else
            $startPoint = newStartPoint$;
            continue;
```

series in blue lines in each figure respectively. To show the patterns' spans and positions, we plot the cost matrix for each pair of time series and highlight the start position, end position and warping path of each pattern with red lines. The details of parameter settings for each dataset is shown as below.

Inserted pattern

This dataset is a synthesis of CPU usage data from three different machines of a cluster, each data point indicates the average usage of CPU in 5 minutes. two parts of a periodical CPU usage series of a single machine (figure 6.6c and 6.6d) are inserted into two relatively chaotic CPU usage of two different machines (figure 6.6a and 6.6b) on time tick 2000 and 1000 respectively. The data are

Table 6.2: Parameter settings of each dataset

Datasets	Sequence length 1	Sequence length 2	Similarity	Constraint	Step limit	Z-normalized?
Inserted pattern	4625	4625	0.2	50	300	Yes
RandomSines	25000	25000	0.5	100	1000	Yes
Web	32000	32000	0.3	500	1000	Yes
Temperature	28000	24000	0.5	500	3000	No
CPU Hostload	37240	37240	0.2	30	300	Yes
ECG5000	630000	70000	0.07	5	140	Yes

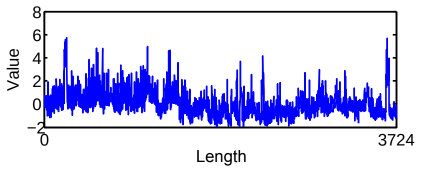
normalized individually to the same mean value and standard error. As shown in figure 6.6, there is no statistical difference between inserted patterns and the being-inserted time series. In figure 6.6g, the red line in the centre denotes the major pattern that starts from time tick 2000 in *inserted host load* 1 and 1000 in *inserted host load* 2 and ends at time tick 2901 and 1901 respectively, corresponds to series 0 to 901 in figure 6.6c and series 0 to 901 in figure 6.6d. The two shorter red lines beside the centre line indicate two minor patterns. One of them corresponds to series 300 to 901 in figure 6.6c and 0 to 600 in figure 6.6d, the other corresponds to series 0 to 600 in figure 6.6c and 300 to 901 in figure 6.6d. The result shows our method finds inserted pattern precisely.

RandomSines

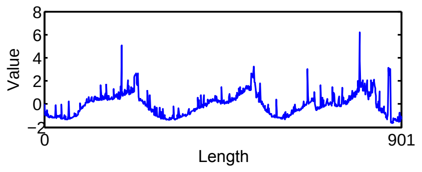
This dataset contains sine waves with different period and white noise(see figs. 6.7a and 6.7b). As we set the constraint to 100, the warping paths are limited, patterns are located within given constraint. In figure 6.7c, it is clear that subsequences pair with large period differences is not recognized as patterns, but subsequences with similar periods remain.

Web

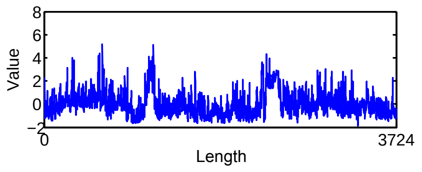
The web dataset includes access amount of blog sites and mail sites (see figs. 6.8a and 6.8b). The data are z-normalized to remove their statistical characteristics. This operation is proved to be effective in reducing computational expense and maintaining the accuracy of pattern locating in later part of this section. As in figure 6.8, the two time series have similar trends but with different scales on



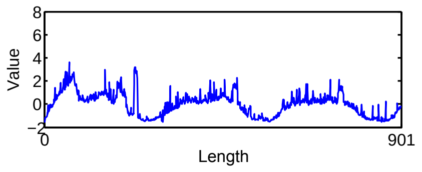
(a) CPU hostload 1



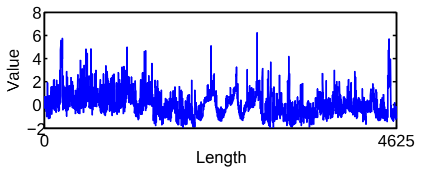
(c) CPU hostload pattern 1



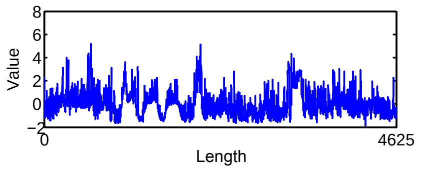
(b) CPU hostload 2



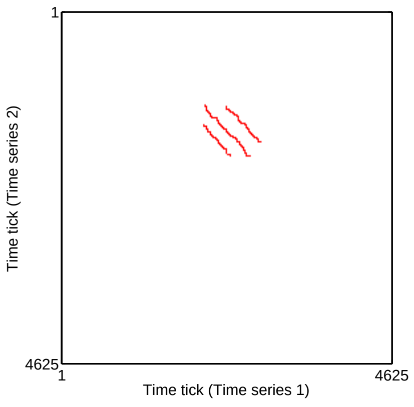
(d) CPU hostload pattern 2



(e) Hostload 1, pattern 1 inserted



(f) Hostload 2, pattern 2 inserted



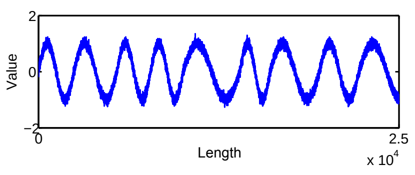
(g) Patterns found

Figure 6.6: Pattern discovery on inserted pattern dataset

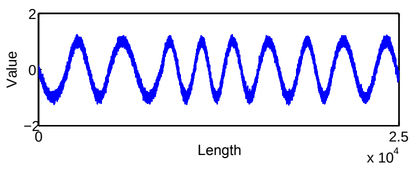
their values. When the data are z-normalized, their scale on values are unified, and we can locate patterns with smaller constraint to increase efficiency and locate patterns effectively.

Temperature

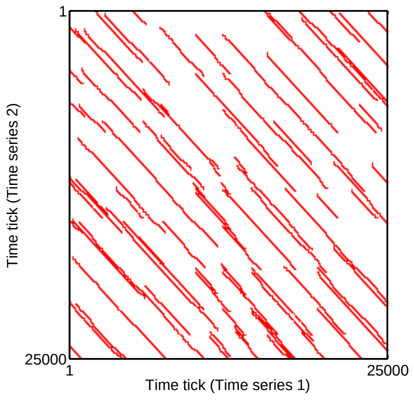
The temperature dataset records temperature of rooms from small sensors to identify whether the rooms are occupied (see figs. 6.9a and 6.9b). This dataset contains missing measurements. As in figure 6.9, our proposed method also able to locate patterns with the presence of missing values and variable periods.



(a) Random sine wave 1

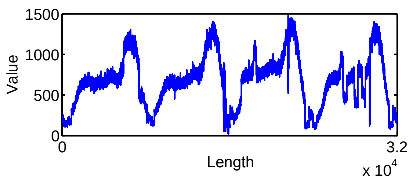


(b) Random sine wave 2

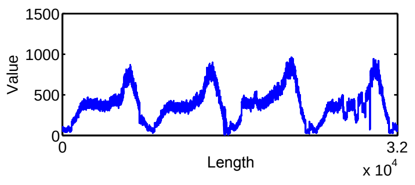


(c) Patterns found

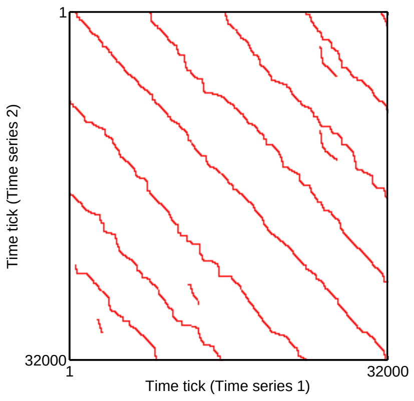
Figure 6.7: Pattern discovery on RandomSines dataset



(a) Blog site visit



(b) Mail site visit

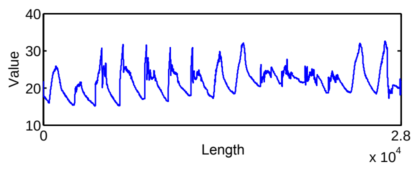


(c) Patterns found

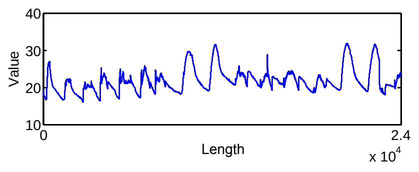
Figure 6.8: Pattern discovery on Web dataset

Cluster CPU host load

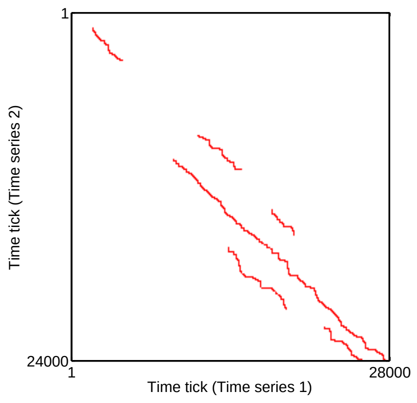
This dataset consists of CPU host load collected from a cluster of Google [149]. To find similar host load subsequences of 10 machines in host load of another 10 machines, CPU usage of the 20 machines a separated into 2 groups with 10 machines in each group, and their host load are z-normalized individually and concatenated (see figs. 6.10a and 6.10b). the result shows that our proposed method is capable to locate short patterns in much longer time series.



(a) Temerature 1

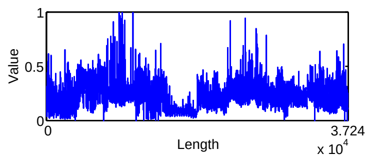


(b) Temerature 2

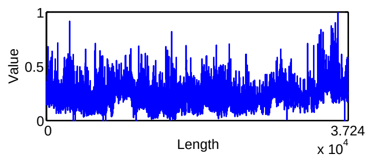


(c) Patterns found

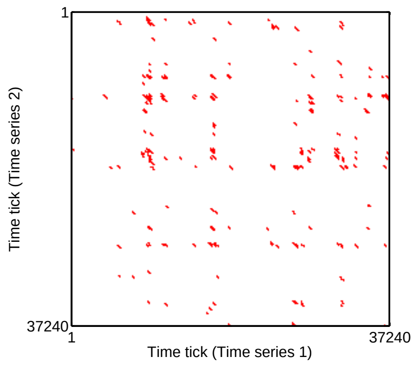
Figure 6.9: Pattern discovery on temperature dataset



(a) Concatenated CPU hostload 1



(b) Concatenated CPU hostload 2

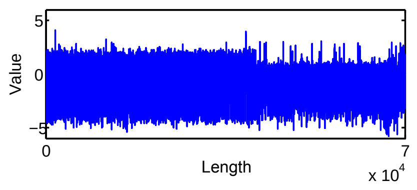


(c) Patterns found

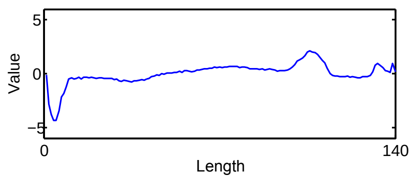
Figure 6.10: Pattern discovery on CPU host load dataset

ECG

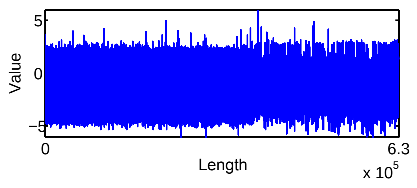
The ECG5000 dataset is available on [31], containing electrocardiograms of 5,000 subjects. Each individual data element contains a piece of ECG with 140 data points. Figure 6.11c shows an example of one piece of ECG. The dataset has a training set and a test set with 500 subjects and 4,500 subjects respectively. The dataset is collected for machine learning, so data elements have labels ranging from 1 to 5, denoting different classifications. To use the data for our pattern locating purpose, we connected data elements in training set and test set respectively, trying to identify similar patterns between test dataset and training dataset(see figs. 6.11a and 6.11b). When the data are connected, the training set becomes a time series with 70,000 data points and test set becomes a time series with 630,000 data points. The result (figure 6.11d) shows that our proposed method divide ECG subsequences into two different parts. The two parts correspond exactly to class 1 ECG pieces and non class 1 ECG pieces. This example illustrates the accuracy of our method and suggests a great number of potential applications.



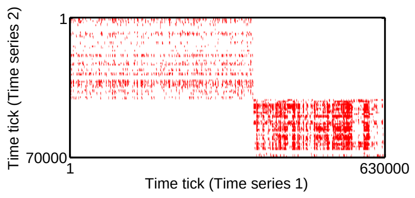
(a) Concatenated training set



(c) Example of 1 piece of ECG



(b) Concatenated test set



(d) Patterns found

Figure 6.11: Pattern discovery on ECG dataset

6.4.2 Performance

We compared computation time and memory usage of our method to a state of art method, CrossMatch, for each test dataset. The source code of CrossMatch algorithm is not provided, we implement the algorithm carefully according to pseudo code in [141]. As our proposed method discovers similar subsequences regardless of their positions, for the sake of fairness, we do not specify a global constraint to CrossMatch. In this case, its time complexity is $O(mn)$ and space complexity is $O(n)$.

Algorithm time complexity

As shown in figure 6.12, our proposed method outperforms CrossMatch on all tested dataset. Our method is notably faster than CrossMatch on the Inserted

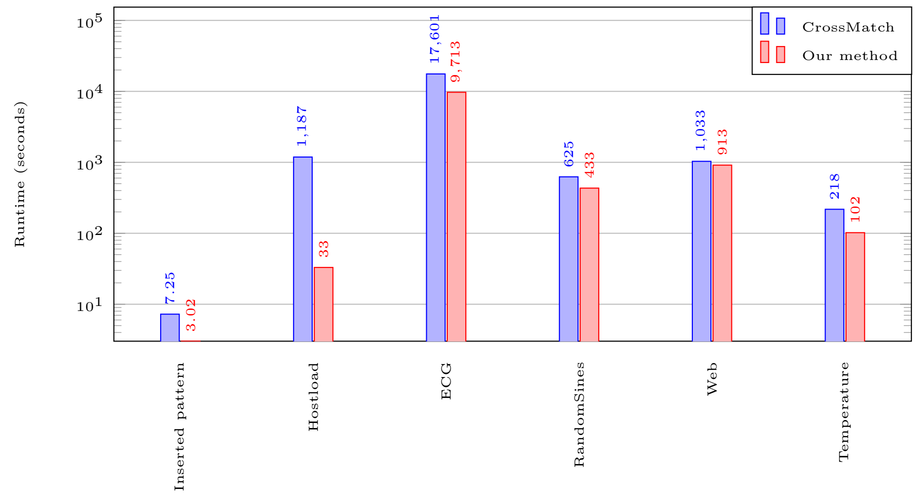


Figure 6.12: Runtime of CrossMatch and our method on different datasets

pattern, Host load, ECG dataset. When patterns are rare in time series, our proposed method can reduce search space greatly. By contrast, CrossMatch searches the whole dataset regardless of the rarity of patterns.

The time complexity of our proposed method subject to the rarity and length of patterns, which varies for different data and parameter settings. Imagine a worst case scenario. In this scenario, the candidate starting positions are maximized and patterns are very dense in search space. As for each candidate area, we only keep one point as starting position, so the worst case starting points must not be neighbouring. A typical and extreme case is that any two starting positions are only separated by one non-starting position. For two time series with n and m data points respectively, we have $\lceil \frac{m}{2} \rceil \lceil \frac{n}{2} \rceil$ starting positions. The length of patterns is set to 1 to achieve the extreme density of patterns in the two time series. In this case, the time complexity of our method is $O(mn)$.

Another worst case scenario is when the candidate starting positions are sparse in two time series, but for each starting position, it requires many iterations to make the result pattern locations converge. Assume the whole sequences of the two time series are similar. In this case, the warping path of the pattern goes diagonally from the $[0, 0]$ to $[m, n]$ of the cost matrix. The starting positions

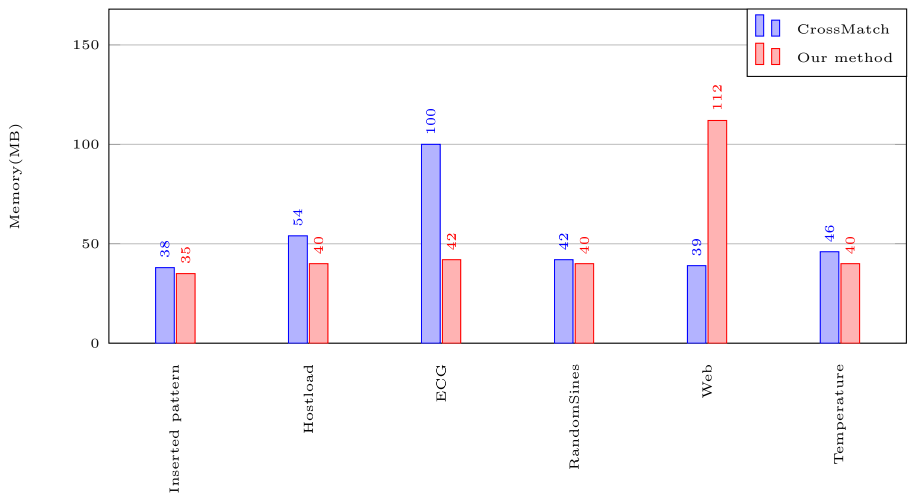


Figure 6.13: Memory consumption of CrossMatch and our method on different datasets

are far from the warping path of the pattern and the constraint are set to 1, the minimum possible value. There are two farthest possible starting positions, which locate at $[0, n]$ and $[m, 0]$ respectively. These two positions take $\frac{n^2+n}{2}$ or $\frac{m^2+m}{2}$ calculations. The warping path of all other starting positions will collide immediately with the warping path of the farthest two starting positions. To sum up, the time complexity of this case is $O(n^2)$ or $O(m^2)$.

Algorithm space complexity

Figure 6.13 shows the memory usage for each dataset of CrossMatch and our proposed method. There is no major difference in memory usage between the two methods. CrossMatch stores candidate locations which consumes most space of its memory usage, and our method also stores paths for each search process. In extreme cases, both the storage of candidate locations and the storage of warping paths can have space complexity up to $O(mn)$. However, the minimum space complexity for CrossMatch is $O(n)$ where n is the length of time series and our proposed method has space complexity of $O(s)$ where s is the max length of patterns which is smaller than n and typically a relatively small value.

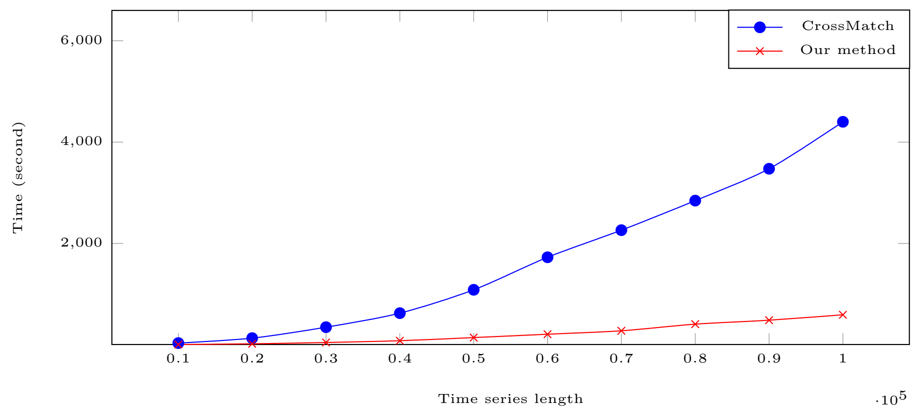


Figure 6.14: Runtime comparison between CrossMatch and our method as a function of dataset size

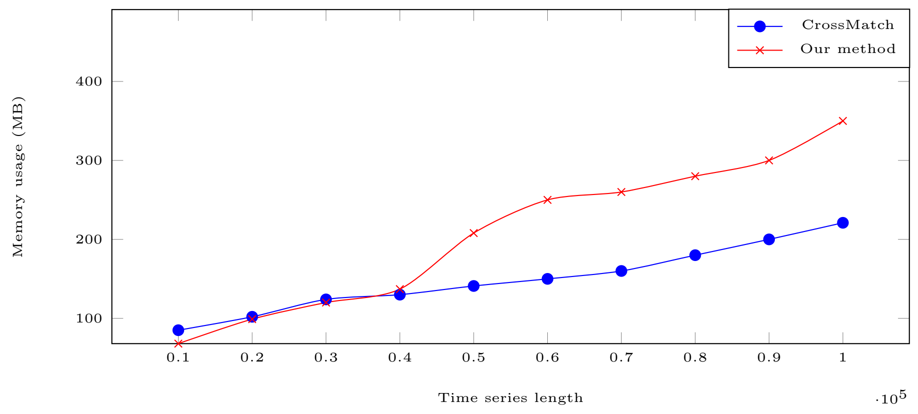


Figure 6.15: Memory usage comparison between CrossMatch and our method as a function of dataset size

Scalability

Figure 6.14 and 6.15 show the runtime and memory usage of our method and CrossMatch. Given the same dataset, our method returns results much faster then CrossMatch, and the runtime increase of our method is also much slower the CrossMatch. The memory usage, however, for our method is greater than that of CrossMatch when the dataset size increase. Considering the difference is not significantly high in memory usage, and the fact that memory resource is typically more sufficient and cheap for most computer systems and application scenarios, our method does have great advantages for its efficiency.

6.5 Summary

Locating previously unknown similar patterns in time series data not only helps discover patterns CPU host load data as discussed in earlier chapters, but also has great importance for any time series data mining task that depends on mined patterns, such as time series clustering which uses different patterns to measure their similarities, time series classification that uses pattern as the label of each class, association rule discovery of mined patterns etc.

In this chapter we proposed a novel method to discover unknown patterns in time series data. We start by creating prior knowledge for patterns from the characteristics that patterns should have. With these prior knowledge, the theoretical search space of patterns is greatly reduced. Then several techniques are used to further reduce the search space.

Firstly the method efficiently locates possible locations of patterns to avoid searching in a large volume of impossible positions. When searching in these positions where patterns are highly likely starting from, an iterative search method is proposed so that the exact positions of patterns are quickly converged to from their possible positions. The search paths are then ”locked” to block other search processes which are searching for the same pattern. To further increase the accuracy of the proposed method to locate optimal matches, the abundant segment of each pattern is trimmed off.

This proposed method is accurate and efficient. The result patterns are guaranteed to be optimal matches, which can be immediately used to any pattern-based time series data mining tasks. Its efficiency also enables a pattern-based time series data mining programme to apply it as a subroutine.

Experiment results have illustrated the method is more efficient in most cases. The method is also tested on multiple datasets to show its effectiveness. These test datasets of different types and sources also suggest a great potential of the method’s applicability.

Chapter 7

Conclusions and further work

This thesis has addressed discovering previously unknown patterns in CPU host load data and time series data. The recent decade has seen significant growth in the number of applications of distributed computing as well as the emerging interests of improving the performance of computing systems on both efficiency and robustness. As one of the most important and fundamental performance indicator, CPU host load contains a substantial amount of knowledge ranges from users' behaviour to early warning of a system fault. This knowledge can help improve the performance of the computing system in many ways. Repeated patterns in CPU host load are obviously notable for the fact that these similar patterns are often produced by the same underlying cause. Given the importance of mining similar patterns in CPU host load and the fact that many methods have applied patterns in CPU host load to improve or analyse computing systems, little work has been done on mining patterns in CPU host load. this thesis proposed two different methods to discover previously unknown patterns in CPU host load.

CPU host load data are time series data. The investigation on mining patterns in CPU host load data raises our interest of proposing a more general method to discover patterns in time series data. Several works have been proposed in the literature on this specific field, however, existing methods are either inaccurate or computationally too expensive. We formally defined the problem of mining previously unknown patterns in time series data to provide a solid base for further study. We proposed a recursive method which utilises the feature of DTW to accurately and efficiently locate similar patterns in time series data.

Major contributions for each technical chapters are summarised in the first four sections of this chapter, the discussion of further work is in section 7.4.

7.1 Clustering Based CPU Host Load Pattern Discovery

The complexity of pattern discovery in time series data is very high. The reason underlying the high time complexity is an algorithm needs to compare every possible subsequence of two time series. By contrast, comparing segments of time series data greatly reduce the search space.

In Chapter 4, a RIP based data-adaptive segmentation method is introduced. This segmentation method is specifically suitable for discovering peak-valley-peak patterns which abound in CPU host load. The segmentation method produces segments between locally notable peaks and valleys, each segment is then either uptrend or downtrend, providing enough abundant information for further reduction.

Depending on the parameter set, segments can have very different lengths. To make them comparable efficiently, each segment is reduced to a 5-dimensional feature vector. Owe to the high uniformity of segments, the feature vector which maintains initial value, trend, crookedness and fluctuation can properly distinguish different segments. In this way, the original data is embedded into a 5-dimension feature space. Experimental results have shown distance measure on feature space is more effective than DTW distance on original space.

A deliberately chosen clustering algorithm, DBSCAN is used to locate similar segments in reduced space. The clustering algorithm finds clusters of any shape and can effectively eliminate outlying points with less dependency on choose of parameters. Due to the embedded space has fixed dimensions, distance measure in the reduced space is fast and consistent for different lengths of segments. We experimentally examined the effectiveness of DBSCAN with two most used clustering algorithm, k-means and hierarchical clustering, showing DBSCAN outperforms other methods remarkably.

7.2 Reduction Based CPU Host Load Pattern Discovery

In Chapter 5 a CPU host load pattern discovery method is developed from a time series reduction perspective. The core of an efficient time series pattern discovery method is the reduction of search space. Other than in Chapter 4 which applies segmentation and feature extraction methods, we propose a time series reduction method in this work for discovering patterns in CPU host load.

The reduction based method applied PAA representation and refined SAX representation by assigning symbols according to the true distribution of CPU host load. To find exact patterns while maintaining efficiency, a cascade discovering method is proposed to filter out unqualified subsequences at minimum cost.

The proposed pattern mining algorithm is able to find longest possible patterns without trivial matches. The effectiveness of the algorithm has been sown experimentally.

The parameters of representation methods have critical effects on the efficiency of pattern mining. We conducted experiments with different parameter sets to show the optimal parameters for mining CPU host load data.

7.3 Iterative Pattern Discovery of Time Series Data

By investigating CPU host load data, we find that there are emerging needs on developing a pattern discovery method for time series data which has not been addressed very much in the literature. Our proposed CPU host load pattern discovery algorithms are dependent more or less on the specific features of CPU host load, which limited its generalization.

Our proposed time series pattern mining method is based on the fact that any similar pattern must contain shorter patterns. By utilizing the property, we presented an approximate similar pattern position locating algorithm. The algorithm filters out a large number of subsequences efficiently.

To find optimal matches, we need to refine approximate pattern positions to exact pattern positions. An iterative home-coming algorithm is proposed to achieve this goal. The algorithm utilizes open-end DTW distance measure to iteratively approach the optimal match.

The iterative pattern discovery method can discover all patterns exist in two time series. The great advantage of our method is it produces only optimal matches, which means the result patterns are of reasonable lengths and are the best and longest possible matches among all possible matches. This feature allows the method to be applied to any application that requires found patterns in time series as a preprocessing step.

Experiments have been conducted on multiple datasets to illustrate the effectiveness and efficiency. The results show that in most cases, our proposed method can outperform the state of the art method greatly in efficiency while maintaining similar space complexity.

7.4 Further Work

The high time complexity nature of mining patterns in CPU host load or general time series data draws the attention of the efficiency of pattern mining methods. The direction of speed up the mining process is either the further reduction of search space or developing parallel mining algorithm. Based on our current work, there is great potential for further development.

The iterative mining method proposed in Chapter 6 can be parallelized since the iterative mining for each approximate pattern position is independent. The ASPL algorithm can also be parallelized easily and thus make it a fully parallel method.

In Chapter 5 we introduced two time series representation methods that can lower bound Euclidean distance. However, currently there is no time series representation that can lower bound DTW distance. With a DTW lower bounded representation, it is possible to reduce search space for iterative mining method of Chapter 6 without losing its advantageous exact optimal match discovery feature.

With the popularity of mobile computing and the explosion of time series data sources, multiple and co-evolving time series data have attracted more attention than ever. Though we can apply two time series mining methods to this field, the patterns of co-evolving time series data cannot be defined by traditional similarity measure in many cases. Pattern mining on multiple time series or co-evolving time series is also an emerging research topic that worth investigating.

Finally, The field of time series unknown pattern mining has been looking forward to a high-level formalization for a long time. Current research on this particular problem stays at applying traditional methods or their modified versions, causing bottlenecks of both the performance and the ability of generalization. Investigating on formalizing the problem of mining unknown patterns in time series data mathematically will guide researchers in the relevant field to a clear and traceable direction. The study on mining unknown patterns in time series will also be put forward greatly.

Bibliography

[1] R. ACS. Bayesian classification (autoclass): Theory and results. 1996.

[2] R. Agrawal, C. Faloutsos, and A. Swami. *Efficient similarity search in sequence databases*. Springer, 1993.

[3] R. Agrawal, J. E. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications, Dec. 14 1999. US Patent 6,003,029.

[4] H. André-Jönsson and D. Z. Badal. Using signature files for querying time-series data. In *European Symposium on Principles of Data Mining and Knowledge Discovery*, pages 211–220. Springer, 1997.

[5] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. Optics: ordering points to identify the clustering structure. In *ACM Sigmod record*, volume 28, pages 49–60. ACM, 1999.

[6] A. Apostolico, M. E. Bock, and S. Lonardi. Monotony of surprise and large-scale quest for unusual words. *Journal of Computational Biology*, 10 (3-4):283–311, 2003.

[7] M. F. Arlitt and C. L. Williamson. Web server workload characterization: The search for invariants. *ACM SIGMETRICS Performance Evaluation Review*, 24(1):126–137, 1996.

[8] D. H. Bailey and A. Snavely. Performance modeling: Understanding

the past and predicting the future. In *European Conference on Parallel Processing*, pages 185–195. Springer, 2005.

[9] Z. Bar-Joseph, G. Gerber, D. K. Gifford, T. S. Jaakkola, and I. Simon. A new approach to analyzing gene expression time series data. In *Proceedings of the sixth annual international conference on Computational biology*, pages 39–48. ACM, 2002.

[10] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *ACM SIGMETRICS Performance Evaluation Review*, volume 26, pages 151–160. ACM, 1998.

[11] R. Bayer and E. McCreight. Organization and maintenance of large ordered indexes. In *Software pioneers*, pages 245–262. Springer, 2002.

[12] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: an efficient and robust access method for points and rectangles. In *Acm Sigmod Record*, volume 19, pages 322–331. ACM, 1990.

[13] F. Benhammadi, Z. Gessoum, A. Mokhtari, et al. Cpu load prediction using neuro-fuzzy and bayesian inferences. *Neurocomputing*, 74(10):1606–1616, 2011.

[14] D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, AAAIWS'94, pages 359–370. AAAI Press, 1994. URL `http://dl.acm.org/citation.cfm?id=3000850.3000887`.

[15] K. B. Bey, F. Benhammadi, A. Mokhtari, and Z. Guessoum. Cpu load prediction model for distributed computing. In *Parallel and Distributed Computing, 2009. ISPDC'09. Eighth International Symposium on*, pages 39–45. IEEE, 2009.

[16] K. B. Bey, F. Benhammadi, A. Mokhtari, and Z. Gessoum. Mixture of anfis

systems for cpu load prediction in metacomputing environment. *Future Generation Computer Systems*, 26(7):1003–1011, 2010.

[17] J. C. Bezdek. *Pattern recognition with fuzzy objective function algorithms.* Springer Science & Business Media, 2013.

[18] R. Bhaskar, S. Laxman, A. Smith, and A. Thakurta. Discovering frequent patterns in sensitive data. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 503–512. ACM, 2010.

[19] C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys (CSUR)*, 33(3):322–373, 2001.

[20] B. Bollobás, G. Das, D. Gunopulos, and H. Mannila. Time-series similarity problems and well-separated geometric sets. In *Proceedings of the thirteenth annual symposium on Computational geometry*, pages 454–456. ACM, 1997.

[21] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung. *Time series analysis: forecasting and control.* John Wiley & Sons, 2015.

[22] M. Butler and D. Kazakov. Sax discretization does not guarantee equiprobable symbols. *Knowledge and Data Engineering, IEEE Transactions on*, 27(4):1162–1166, April 2015. ISSN 1041-4347. doi: 10.1109/TKDE.2014.2382882.

[23] M. Calzarossa and G. Serazzi. Workload characterization: A survey. *Proceedings of the IEEE*, 81(8):1136–1150, 1993.

[24] J. Cao, J. Fu, M. Li, and J. Chen. Cpu load prediction for cloud environment based on a dynamic ensemble model. *Software: Practice and Experience*, 44(7):793–804, 2014.

[25] G. A. Carpenter and S. Grossberg. A massively parallel architecture for

a self-organizing neural pattern recognition machine. *Computer vision, graphics, and image processing*, 37(1):54–115, 1987.

[26] K. Chakrabarti and S. Mehrotra. The hybrid tree: An index structure for high dimensional feature spaces. In *Data Engineering, 1999. Proceedings., 15th International Conference on*, pages 440–447. IEEE, 1999.

[27] F.-P. Chan, A.-C. Fu, and C. Yu. Haar wavelets for efficient similarity search of time-series: with and without time warping. *IEEE Transactions on knowledge and data engineering*, 15(3):686–705, 2003.

[28] K.-P. Chan and A.-C. Fu. Efficient time series matching by wavelets. In *Data Engineering, 1999. Proceedings., 15th International Conference on*, pages 126–133. IEEE, 1999.

[29] J. R. Chen. Making subsequence time series clustering meaningful. In *Data mining, fifth IEEE international conference on*, pages 8–pp. IEEE, 2005.

[30] Y. Chen, B. Hu, E. Keogh, and G. E. Batista. Dtw-d: Time series semi-supervised learning from a single example. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Knowledge Discovery and Data Mining, pages 383–391, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2174-7. doi: 10.1145/2487575.2487633. URL `http://doi.acm.org/10.1145/2487575.2487633`.

[31] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista. The ucr time series classification archive, July 2015. URL `www.cs.ucr.edu/~eamonn/time_series_data/`.

[32] B. Chiu, E. Keogh, and S. Lonardi. Probabilistic discovery of time series motifs. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 493–498. ACM, 2003.

[33] W. Cirne and F. Berman. A comprehensive model of the supercomputer workload. In *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*, pages 140–148, Dec 2001. doi: 10.1109/WWC.2001.990753.

[34] P. Cotofrei and K. Stoffel. *Classification Rules + Time = Temporal Rules*, pages 572–581. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002. ISBN 978-3-540-46043-5. doi: 10.1007/3-540-46043-8_58. URL `https://doi.org/10.1007/3-540-46043-8_58`.

[35] P. Cotofrei and K. Stoffel. Rule extraction from time series databases using classification trees. In *APPLIED INFORMATICS-PROCEEDINGS-*, number 2, pages 327–332. UNKNOWN, 2002.

[36] M. E. Crovella. Performance evaluation with heavy tailed distributions. *Lecture Notes in Computer Science*, 1786:1–9, 2000.

[37] G. Das, D. Gunopulos, and H. Mannila. Finding similar time series. In *European Symposium on Principles of Data Mining and Knowledge Discovery*, pages 88–100. Springer, 1997.

[38] G. Das, K. ip Lin, H. Mannila, G. Renganathan, and P. Smyth. Rule discovery from time series. In *In Proceedings of the 1997 ACM SIGKDD International Conference, ACM SIGKDD*, 1997.

[39] D. Dasgupta and S. Forrest. Novelty detection in time series data using ideas from immunology. In *Proceedings of the international conference on intelligent systems*, pages 82–87, 1996.

[40] M. Degli Esposti, C. Farinelli, and G. Menconi. Sequence distance via parsing complexity: Heartbeat signals. *Chaos, Solitons & Fractals*, 39(3): 991–999, 2009.

[41] S. Di, D. Kondo, and W. Cirne. Characterization and comparison of cloud

versus grid workloads. In *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*, pages 230–238. IEEE, 2012.

[42] S. Di, D. Kondo, and W. Cirne. Host load prediction in a google compute cloud with a bayesian model. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 21. IEEE Computer Society Press, 2012.

[43] P. A. Dinda. The statistical properties of host load. *Scientific Programming*, 7(3):211–229, 1999.

[44] P. A. Dinda and D. R. O'Hallaron. An evaluation of linear models for host load prediction. In *High Performance Distributed Computing, 1999. Proceedings. The Eighth International Symposium on*, pages 87–96. IEEE, 1999.

[45] P. A. Dinda and D. R. O'hallaron. Host load prediction using linear models. *Cluster Computing*, 3(4):265–280, 2000.

[46] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. Querying and mining of time series data: Experimental comparison of representations and distance measures. *Proc. Very Large Data Bases Endow.*, 1(2):1542–1552, Aug. 2008. ISSN 2150-8097. doi: 10.14778/1454159.1454226. URL `http://dx.doi.org/10.14778/1454159.1454226`.

[47] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[48] A. B. Downey and D. G. Feitelson. The elusive goal of workload characterization. *ACM SIGMETRICS Performance Evaluation Review*, 26(4): 14–29, 1999.

[49] P. Esling and C. Agon. Time-series data mining. *ACM Comput. Surv.*, 45 (1):12:1–12:34, Dec. 2012. ISSN 0360-0300. doi: 10.1145/2379776.2379788. URL `http://doi.acm.org/10.1145/2379776.2379788`.

[50] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[51] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. *Fast subsequence matching in time-series databases*, volume 23. ACM, 1994.

[52] P. G. Ferreira, P. J. Azevedo, C. G. Silva, and R. M. Brito. Mining approximate motifs in time series. In *Discovery Science*, volume 4265, pages 89–101. Springer, 2006.

[53] A. L. Fred and A. K. Jain. Data clustering using evidence accumulation. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 4, pages 276–280. IEEE, 2002.

[54] A. L. Fred and A. K. Jain. Combining multiple clusterings using evidence accumulation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(6):835–850, 2005.

[55] T.-c. Fu, F.-l. Chung, V. Ng, and R. Luk. Pattern discovery from stock time series using self-organizing maps. In *Workshop Notes of KDD2001 Workshop on Temporal Data Mining*, pages 26–29, 2001.

[56] M. Gavrilov, D. Anguelov, P. Indyk, and R. Motwani. Mining the stock market (extended abstract): which measure is best? In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 487–496. ACM, 2000.

[57] X. Ge and P. Smyth. Deformable markov model templates for time-series pattern matching. In *Proceedings of the sixth ACM SIGKDD international*

conference on Knowledge discovery and data mining, pages 81–90. ACM, 2000.

[58] P. Geurts. Pattern extraction for time series classification. In *PKDD*, volume 1, pages 115–127. Springer, 2001.

[59] D. Q. Goldin and P. C. Kanellakis. On similarity queries for time-series data: constraint specification and implementation. In *International Conference on Principles and Practice of Constraint Programming*, pages 137–153. Springer, 1995.

[60] Z. Gu, C. Chang, L. He, and K. Li. Developing a pattern discovery model for host load data. In *Computational Science and Engineering (CSE), 2014 IEEE 17th International Conference on*, pages 265–271. IEEE, 2014.

[61] Z. Gu, L. He, C. Chang, J. Sun, H. Chen, and C. Huang. An efficient method for motif discovery in cpu host load. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2015 12th International Conference on*, pages 1027–1034. IEEE, 2015.

[62] Z. Gu, L. He, C. Chang, J. Sun, H. Chen, and C. Huang. Developing an efficient pattern discovery method for cpu utilizations of computers. *International Journal of Parallel Programming*, pages 1–26, 2016. ISSN 1573-7640. doi: 10.1007/s10766-016-0439-0. URL `http://dx.doi.org/10.1007/s10766-016-0439-0`.

[63] S. Guha, R. Rastogi, and K. Shim. Cure: an efficient clustering algorithm for large databases. In *ACM Sigmod Record*, volume 27, pages 73–84. ACM, 1998.

[64] P. Guo, L. Wang, and P. Chen. A performance modeling and optimization analysis tool for sparse matrix-vector multiplication on gpus. *IEEE Transactions on Parallel and Distributed Systems*, 25(5):1112–1123, May 2014. ISSN 1045-9219. doi: 10.1109/TPDS.2013.123.

[65] D. Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology.* Cambridge university press, 1997.

[66] J. Han, G. Dong, and Y. Yin. Efficient mining of partial periodic patterns in time series database. In *Data Engineering, 1999. Proceedings., 15th International Conference on*, pages 106–115. IEEE, 1999.

[67] J. Han, J. Pei, and M. Kamber. *Data mining: concepts and techniques.* Elsevier, 2011.

[68] S. Harms, J. Deogun, and T. Tadesse. Discovering sequential association rules with constraints and time lags in multiple sequences. *Foundations of Intelligent Systems*, pages 373–376, 2002.

[69] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Applied statistics*, pages 100–108, 1979.

[70] M. Hegland, W. Clarke, and M. Kahn. Mining the macho dataset. *Computer Physics Communications*, 142(1):22–28, 2001.

[71] Y.-W. Huang and P. S. Yu. Adaptive query processing for time-series data. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 282–286. ACM, 1999.

[72] D. H. Hubel. *Eye, brain, and vision*, volume 22. Scientific American Library New York, 1988.

[73] J. Hunter and N. McIntosh. Knowledge-based event detection in complex time series data. In *Joint European Conference on Artificial Intelligence in Medicine and Medical Decision Making*, pages 271–280. Springer, 1999.

[74] F. Hppner. Discovery of temporal patterns. In L. De Raedt and A. Siebes, editors, *Principles of Data Mining and Knowledge Discovery*, volume 2168 of *Lecture Notes in Computer Science*, pages 192–203. Springer Berlin Heidelberg, 2001. ISBN 978-3-540-42534-2. doi: 10.1007/3-540-44794-6_16. URL `http://dx.doi.org/10.1007/3-540-44794-6_16`.

[75] M. Ishijima, S.-B. Shin, G. H. Hostetter, and J. Sklansky. Scan-along polygonal approximation for data compression of electrocardiograms. *IEEE Transactions on Biomedical Engineering*, (11):723–729, 1983.

[76] F. Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23(1):67–72, 1975.

[77] A. K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, 2010.

[78] G. J. Janacek, A. J. Bagnall, and M. Powell. A likelihood ratio distance measure for the similarity between the fourier transform of time series. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 737–743. Springer, 2005.

[79] K. Kalpakis, D. Gada, and V. Puttagunta. Distance measures for effective clustering of arima time-series. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 273–280. IEEE, 2001.

[80] D. Karaboga and C. Ozturk. A novel clustering approach: Artificial bee colony (abc) algorithm. *Applied Soft Computing*, 11(1):652–657, 2011.

[81] G. Karypis, E.-H. Han, and V. Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *Computer*, 32(8):68–75, 1999.

[82] L. Kaufman and P. J. Rousseeuw. *Finding groups in data: an introduction to cluster analysis*, volume 344. John Wiley & Sons, 2009.

[83] E. Keogh and S. Kasetty. On the need for time series data mining benchmarks: a survey and empirical demonstration. *Data Mining and knowledge discovery*, 7(4):349–371, 2003.

[84] E. Keogh and J. Lin. Clustering of time-series subsequences is meaningless: implications for previous and future research. *Knowledge and information systems*, 8(2):154–177, 2005.

[85] E. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and information systems*, 7(3):358–386, 2005.

[86] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Locally adaptive dimensionality reduction for indexing large time series databases. *SIGMOD Rec.*, 30(2):151–162, May 2001. ISSN 0163-5808. doi: 10.1145/376284.375680. URL `http://doi.acm.org/10.1145/376284.375680`.

[87] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and information Systems*, 3(3):263–286, 2001.

[88] E. Keogh, S. Chu, D. Hart, and M. Pazzani. An online algorithm for segmenting time series. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 289–296, 2001. doi: 10.1109/ICDM.2001.989531.

[89] E. Keogh, S. Lonardi, and C. A. Ratanamahatana. Towards parameter-free data mining. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 206–215. ACM, 2004.

[90] E. J. Keogh and M. J. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *Knowledge Discovery and Data Mining*, volume 98, pages 239–243, 1998.

[91] E. J. Keogh and M. J. Pazzani. Relevance feedback retrieval of time series data. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 183–190. ACM, 1999.

[92] E. J. Keogh and M. J. Pazzani. Derivative dynamic time warping. In *Proceedings of the 2001 SIAM International Conference on Data Mining*, pages 1–11. SIAM, 2001.

[93] E. J. Keogh and P. Smyth. A probabilistic approach to fast pattern matching in time series databases. In *Knowledge Discovery and Data Mining*, volume 1997, pages 24–30, 1997.

[94] A. Khan, X. Yan, S. Tao, and N. Anerousis. Workload characterization and prediction in the cloud: A multiple time series approach. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 1287–1294. IEEE, 2012.

[95] T. Kohonen. The self-organizing map. *Neurocomputing*, 21(1):1–6, 1998.

[96] A. Koski, M. Juhola, and M. Meriste. Syntactic recognition of ecg signals by attributed finite automata. *Pattern Recognition*, 28(12):1927–1940, 1995.

[97] R. Krishnapuram, A. Joshi, O. Nasraoui, and L. Yi. Low-complexity fuzzy relational clustering algorithms for web mining. *IEEE transactions on Fuzzy Systems*, 9(4):595–607, 2001.

[98] V. Lavrenko, M. Schmill, D. Lawrie, P. Ogilvie, D. Jensen, and J. Allan. Mining of concurrent text and time series. In *KDD-2000 Workshop on Text Mining*, volume 2000, pages 37–44, 2000.

[99] Y. Leung, J.-S. Zhang, and Z.-B. Xu. Clustering by scale-space filtering. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(12): 1396–1410, 2000.

[100] C.-S. Li, P. S. Yu, and V. Castelli. Malm: A framework for mining sequence database at multiple abstraction levels. In *Proceedings of the seventh international conference on Information and knowledge management*, pages 267–272. ACM, 1998.

[101] K. Li, X. Tang, B. Veeravalli, and K. Li. Scheduling precedence constrained stochastic tasks on heterogeneous cluster systems. *Computers, IEEE Transactions on*, 64(1):191–204, 2015.

[102] K. Li, W. Yang, and K. Li. Performance analysis and optimization for spmv on gpu using probabilistic modeling. *Parallel and Distributed Systems, IEEE Transactions on*, 26(1):196–205, 2015.

[103] J. Liang, J. Cao, J. Wang, and Y. Xu. Long-term cpu load prediction. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, pages 23–26. IEEE, 2011.

[104] J. Lin and Y. Li. Finding structural similarity in time series data using bag-of-patterns representation. In *Scientific and statistical database management*, pages 461–477. Springer, 2009.

[105] J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 2–11. ACM, 2003.

[106] R. A. K.-l. Lin and H. S. S. K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *Proceeding of the 21th International Conference on Very Large Data Bases*, pages 490–501, 1995.

[107] Z. Liu, J. X. Yu, X. Lin, H. Lu, and W. Wang. Locating motifs in time-series data. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 343–353. Springer, 2005.

[108] J. L. E. K. S. Lonardi and P. Patel. Finding motifs in time series. In *Proc. of the 2nd Workshop on Temporal Data Mining*, pages 53–68, 2002.

[109] S. Lonardi and A. Apostolico. Global detectors of unusual words: design, implementation, and applications to pattern discovery in biosequences. *Department of Computer Sciences, Purdue University*, 2001.

[110] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on*

mathematical statistics and probability, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[111] M. L. Marx and R. J. Larsen. *Introduction to mathematical statistics and its applications*. Pearson/Prentice Hall, 2006.

[112] E. Masciari, G. M. Mazzeo, and C. Zaniolo. A new, fast and accurate algorithm for hierarchical clustering on euclidean distances. In *Advances in Knowledge Discovery and Data Mining*, pages 111–122. Springer, 2013.

[113] Y. Mohammad and T. Nishida. Constrained motif discovery in time series. *New Generation Computing*, 27(4):319, 2009.

[114] M. D. Morse and J. M. Patel. An efficient and accurate method for evaluating time series similarity. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 569–580. ACM, 2007.

[115] A. Mueen, E. J. Keogh, Q. Zhu, S. Cash, and M. B. Westover. Exact discovery of time series motifs. In *SDM*, pages 473–484. SIAM, 2009.

[116] M. Müller. Dynamic time warping. *Information retrieval for music and motion*, pages 69–84, 2007.

[117] A. Panuccio, M. Bicego, and V. Murino. A hidden markov model-based approach to sequential data clustering. *Structural, Syntactic, and Statistical Pattern Recognition*, pages 734–743, 2002.

[118] P. Papapetrou, V. Athitsos, M. Potamias, G. Kollios, and D. Gunopulos. Embedding-based subsequence matching in time-series databases. *ACM Transactions on Database Systems (TODS)*, 36(3):17, 2011.

[119] S. Park, D. Lee, and W. W. Chu. Fast retrieval of similar subsequences in long sequence databases. In *Knowledge and Data Engineering Exchange, 1999.(KDEX'99) Proceedings. 1999 Workshop on*, pages 60–67. IEEE, 1999.

[120] S. Park, W. W. Chu, J. Yoon, and C. Hsu. Efficient searches for similar subsequences of different lengths in sequence databases. In *Data Engineering, 2000. Proceedings. 16th International Conference on*, pages 23–32. IEEE, 2000.

[121] S. Park, S.-W. Kim, and W. W. Chu. Segment-based approach for subsequence searches in sequence databases. In *Proceedings of the 2001 ACM symposium on Applied computing*, pages 248–252. ACM, 2001.

[122] P. Patel, E. Keogh, J. Lin, and S. Lonardi. Mining motifs in massive time series databases. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 370–377. IEEE, 2002.

[123] D. Pelleg, A. W. Moore, et al. X-means: Extending k-means with efficient estimation of the number of clusters. In *International Conference on Machine Learning*, pages 727–734, 2000.

[124] I. Popivanov and R. J. Miller. Similarity search over time-series data using wavelets. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 212–221. IEEE, 2002.

[125] Y. Qu, C. Wang, and X. S. Wang. Supporting fast search in time series for movement patterns in multiple scales. In *Proceedings of the Seventh International Conference on Information and Knowledge Management*, Conference on Information and Knowledge Management '98, pages 251–258, New York, NY, USA, 1998. ACM. ISBN 1-58113-061-9. doi: 10.1145/288627.288664. URL `http://doi.acm.org/10.1145/288627.288664`.

[126] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 262–270. ACM, 2012.

[127] U. Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer graphics and image processing*, 1(3):244–256, 1972.

[128] C. Ratanamahatana, E. Keogh, A. J. Bagnall, and S. Lonardi. A novel bit level time series representation with implication of similarity search and clustering. In *Advances in knowledge discovery and data mining*, pages 771–777. Springer, 2005.

[129] C. A. Ratanamahatana and E. Keogh. Making time-series classification more accurate using learned constraints. In *Proceedings of the 2004 SIAM International Conference on Data Mining*, pages 11–22. SIAM, 2004.

[130] S. Ren, L. He, H. Zhu, Z. Gu, W. Song, and J. Shang. Developing power-aware scheduling mechanisms for computing systems virtualized by xen. *Concurrency and Computation: Practice and Experience*, 29(3), 2017.

[131] J. F. Roddick, K. Hornsby, and M. Spiliopoulou. An updated bibliography of temporal, spatial, and spatio-temporal data mining research. In *Temporal, Spatial, and Spatio-Temporal Data Mining*, pages 147–163. Springer, 2001.

[132] P. P. Rodrigues, J. Gama, and J. P. Pedroso. Hierarchical clustering of time-series data streams. *Knowledge and Data Engineering, IEEE Transactions on*, 20(5):615–627, 2008.

[133] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing*, 26(1):43–49, 1978.

[134] S. Salvador and P. Chan. Toward accurate dynamic time warping in linear time and space. *Intell. Data Anal.*, 11(5):561–580, Oct. 2007. ISSN 1088-467X. URL `http://dl.acm.org/citation.cfm?id=1367985.1367993`.

[135] E. Schikuta. Grid-clustering: An efficient hierarchical clustering method

for very large data sets. In *Pattern Recognition, 1996., Proceedings of the 13th International Conference on*, volume 2, pages 101–105. IEEE, 1996.

[136] P. Sebastiani, M. Ramoni, P. Cohen, J. Warwick, and J. Davis. Discovering dynamics using bayesian clustering. *Advances in intelligent data analysis*, pages 199–209, 1999.

[137] C. Shahabi, X. Tian, and W. Zhao. Tsa-tree: A wavelet-based approach to improve the efficiency of multi-level surprise and trend queries on time-series data. In *Scientific and Statistical Database Management, 2000. Proceedings. 12th International Conference on*, pages 55–68. IEEE, 2000.

[138] D. Shasha and Y. Zhu. High performance discovery in time series: Techniques and case studies (monographs in computer science). 2004.

[139] H. Shatkay and S. B. Zdonik. Approximate queries and representations for large data sequences. In *Data Engineering, 1996. Proceedings of the Twelfth International Conference on*, pages 536–545. IEEE, 1996.

[140] H. Tang and S. S. Liao. Discovering original motifs with different lengths from time series. *Knowledge-Based Systems*, 21(7):666–671, 2008.

[141] M. Toyoda, Y. Sakurai, and Y. Ishikawa. Pattern discovery in data streams under the time warping distance. *The Very Large Data Bases Journal*, 22 (3):295–318, 2013.

[142] A. Vahdatpour, N. Amini, and M. Sarrafzadeh. Toward unsupervised activity discovery using multi-dimensional motif detection in time series. In *IJCAI*, volume 9, pages 1261–1266, 2009.

[143] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multi-dimensional trajectories. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 673–684. IEEE, 2002.

[144] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh. Indexing

multidimensional time-series. *The VLDB JournalThe International Journal on Very Large Data Bases*, 15(1):1–20, 2006.

[145] H. Vullings, M. Verhaegen, and H. B. Verbruggen. Ecg segmentation using time-warping. In *Advances in Intelligent Data Analysis. Reasoning about Data: Second International Symposium, IDA-97, London, UK, August 1997. Proceedings*, page 275. Springer, 1997.

[146] C. Wang and X. S. Wang. Supporting content-based searches on time series via approximation. In *Proceedings. 12th International Conference on Scientific and Statistica Database Management*, pages 69–81, 2000. doi: 10.1109/SSDM.2000.869779.

[147] W. Wang, J. Yang, R. Muntz, et al. Sting: A statistical information grid approach to spatial data mining. In *Very Large Data Bases*, volume 97, pages 186–195, 1997.

[148] L. Wei, E. Keogh, H. Van Herle, A. Mafra-Neto, and R. J. Abbott. Efficient query filtering for streaming time series with applications to semisupervised learning of time series classifiers. *Knowledge and Information Systems*, 11 (3):313–344, 2007. ISSN 0219-3116. doi: 10.1007/s10115-006-0033-7. URL `http://dx.doi.org/10.1007/s10115-006-0033-7`.

[149] J. Wilkes. More Google cluster data. Google research blog, Nov. 2011. Posted athttp://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html,.

[150] Y. Xiong and D.-Y. Yeung. Time series clustering with arma mixtures. *Pattern Recognition*, 37(8):1675–1689, 2004.

[151] L. Yang, I. Foster, and J. M. Schopf. Homeostatic and tendency-based cpu load predictions. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, pages 9–pp. IEEE, 2003.

[152] Q. Yang, C. Peng, H. Zhao, Y. Yu, Y. Zhou, Z. Wang, and S. Du. A new method based on psr and ea-gmdh for host load prediction in cloud computing system. *The Journal of Supercomputing*, 68(3):1402–1417, 2014.

[153] W. Yang, K. Li, Z. Mo, and K. Li. Performance optimization using partitioned spmv on gpus and multicore cpus. *Computers, IEEE Transactions on*, 64(9):2623–2636, 2015.

[154] D. Yankov, E. Keogh, J. Medina, B. Chiu, and V. Zordan. Detecting time series motifs under uniform scaling. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 844–853. ACM, 2007.

[155] L. Ye and E. Keogh. Time series shapelets: a novel technique that allows accurate, interpretable and fast classification. *Data mining and knowledge discovery*, 22(1):149–182, 2011.

[156] B.-K. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary lp norms. Very Large Data Bases, 2000.

[157] H. Zeng, Z. Shen, and Y. Hu. Mining sequence pattern from time series based on inter-relevant successive trees model. In G. Wang, Q. Liu, Y. Yao, and A. Skowron, editors, *Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing*, volume 2639 of *Lecture Notes in Computer Science*, pages 734–737. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-14040-5. doi: 10.1007/3-540-39205-X_127. URL `http://dx.doi.org/10.1007/3-540-39205-X_127`.

[158] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: an efficient data clustering method for very large databases. In *ACM Sigmod Record*, volume 25, pages 103–114. ACM, 1996.

[159] X. Zhang, J. Wu, X. Yang, H. Ou, and T. Lv. A novel pattern extraction method for time series classification. *Optimization and Engineering*, 10(2): 253–271, 2009.

[160] Y. Zhang, S. Wei, and Y. Inoguchi. Cpu load predictions on the computational grid. *IEICE TRANSACTIONS on Information and Systems*, 90(1): 40–47, 2007.

[161] Y. Zhang, W. Sun, and Y. Inoguchi. Predict task running time in grid environments based on cpu load predictions. *Future Generation Computer Systems*, 24(6):489–497, 2008.