Online Data Mining for Co-Evolving Time Sequences

B.-K. Yi[1] *N.D. Sidiropoulos*[2] *T. Johnson*[3]

H.V. Jagadish[4] *C. Faloutsos*[5] *A. Biliris*[6]

October 1999

CMU-CS-99-171

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

To appear in ICDE 2000

[1]Dept. of CS, Univ. of Maryland, `kee@cs.umd.edu`

[2]Dept. of EE, Univ. of Virginia (`nds5j@virginia.edu`)

[3]AT&T Labs (`johnsont@research.att.com`)

[4]Dept. of EECS, Univ. of Michigan (`jag@eecs.umich.edu`). On leave from AT&T Labs.

[5]School of Computer Science, CMU (`christos@cs.cmu.edu`)

[6]AT&T Labs, (`biliris@research.att.com`)

This material is based upon work supported by the National Science Foundation under Grants No. IRI-9625428, DMS-9873442, IIS-9817496, and IIS-9910606, and by the Defense Advanced Research Projects Agency under Contract No. N66001-97-C-8517. Additional funding was provided by donations from NEC and Intel. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, DARPA, or other funding parties.

Keywords: Data bases, Data mining, time sequences

Abstract

In many applications, the data of interest comprises multiple sequences that evolve over time. Examples include currency exchange rates, network traffic data, and demographic data on multiple variables. We develop a fast method to analyze such co-evolving time sequences *jointly* to allow (a) estimation/forecasting of missing/delayed/future values, (b) quantitative data mining, discovering correlations (with or without lag) among the given sequences, and (c) outlier detection.

Our method, MUSCLES, adapts to changing correlations among time sequences. It can handle indefinitely long sequences efficiently using an incremental algorithm and requires only small amount of storage so that it works well with limited main memory size and does not cause excessive I/O operations. To scale for a large number of sequences, we present a variation, the *Selective* MUSCLES method and propose an efficient algorithm to reduce the problem size.

Experiments on real datasets show that MUSCLES outperforms popular competitors in prediction accuracy up to 10 times, and discovers interesting correlations. Moreover, *Selective* MUSCLES scales up very well for large numbers of sequences, reducing response time up to 110 times over MUSCLES, and sometimes even improves the prediction quality.

1 Introduction

In many applications, the data of interest comprises multiple sequences that each evolve over time. Examples include currency exchange rates, network traffic data from different network elements, demographic data from multiple jurisdictions, patient data varying over time, and so on.

These sequences are not independent: in fact, they frequently exhibit high correlations. Therefore, much useful information is lost if each sequence is analyzed individually. What we desire is to study the entire set of sequences as a whole, where the number of sequences in the set can be very large. For example, if each sequence represents data recorded from a network element in some large network, then the number of sequences could easily be in the several thousands, and even millions.

To make our task even more challenging, it is typically the case that the results of analysis are most useful if they are available immediately, based upon the portion of each sequence seen so far, without waiting for "completion" of data streams. In fact, these sequences can be indefinitely long, and may have no predictable termination in the future. What we require is the capability to "repeat" our analysis over and over as the next element (or batch of elements) in each data sequence is revealed, so that accurate estimations of delayed/missing elements and/or up-to-date correlations are available quickly. And we have to do this on potentially very long sequences, indicating a need for analytical techniques that have low *incremental* computational complexity.

Table 1 illustrates a typical setting: Suppose that we have k time sequences, and that we obtain the value of each at every time-tick (say, every minute). Suppose that one of the time sequences, say, s_1, is delayed or missing. Our goal is to do our best prediction for the last "current" value of this sequence, given all the past information about this sequence, and all the past and current information for the other sequences. We wish to be able to do this at every point of time, given all the information up to that time.

More generally, given a delayed or missing value in some sequence, we would like to estimate it as best as we can, using all other information available to us from this and other related sequences. Using the same machinery, we can also find "unexpected values" when the actual observation differs greatly from its estimate computed as above. Such an "outlier" may be indicative of an interesting event in the specific time series affected.

Another problem to solve is the derivation of (quantitative) correlations; *e.g.*, "the number

sequence / time	s_1 packets-sent	s_2 packets-lost	s_3 packets-corrupted	s_4 packets-repeated
1	50	20	10	3
2	55	20	10	10
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$N-1$	73	25	18	12
N	??	25	18	18

Table 1: Snapshot of a set of co-evolving time sequences. Goal: predict the delayed value of s_1.

of packets-lost is perfectly correlated with the number of packets corrupted", or "the number of packets-repeated lags the number of packets-corrupted by several time-ticks". This type of information can be used for various purposes such as tracking of the source of cascaded network fault or overload, or discovering unknown relationships between bank/credit accounts to spot suspected criminal behaviors.

In light of the preceding discussion, our goal is to seek a technique which satisfies the following requirements.

- It should provide all the machinery to solve both problems we introduced earlier.
- It should be on-line and scalable, operating in time that is independent of the number, N, of past time-ticks.
- It should scale up well with the number of time sequences, k.
- It should also be capable of adapting quickly to the change of trends.

This is a challenging task because direct application of the standard mathematical/statistical methods such as the linear regression model may fail to meet the above requirements. Hence our focus lies on the development of an elaborated technique such that it satisfies all of our requirement and verifies its effectiveness through extensive experimental evaluations.

Applications: We embarked upon this work motivated by a network management application similar to the one described below. However, we soon discovered that our concepts applied equally to *any* collection of co-evolving time sequences. Sample applications include the following:

- Network management: Time sequences are measurements (for example, number of packets lost, sent, and received for a collection of nodes). Then, we want to (a) fill in missing/delayed values;(b) spot outliers; (c) group "alarming" situations together;(d) possibly, suggest the earliest of the alarms as the cause of the trouble.
- Sales data: Given, say, AT&T customers and their calling patterns over time, spot outliers; these may be indicators of fraud or a change in customer behavior. Also, correlations between geographic regions may be of interest.
- Web and intra-net management: For each site, consider the time sequence of the number of hits per minute; try to find correlations between access patterns, to help forecast future requests (prefetching and caching); try to detect outliers, to spot intruders/malicious users.
- Law enforcement: A large collection of bank accounts owned by criminal suspects and their associates can be continuously monitored so that money laundering or other illegal activities can be uncovered as soon as they occur.

Related Work: Time series forecasting has been a major focus for research in other fields. In particular, valuable tools for forecasting and time series processing appear in statistics and signal processing. The traditional, highly successful methodology for forecasting is the so-called Box-Jenkins methodology, or Auto-Regressive Integrated Moving Average (ARIMA for short) [6, 8]. Variations of it have been used for voice compression, under the name of Linear Predictive Coding (LPC) [20]. ARIMA falls under the class of linear time-series forecasting, because it postulates a linear dependency of the future value on the past values. More recent, non-linear forecasting methods, constitute an open research area [7, 23]. DeCoste [10] proposed a technique based on linear regression and neural network for multivariate time sequences. It is, however, limited to outlier detection and does not scale well for large set of dynamically growing time sequences.

The closest related database work concerns similarity searching in time sequences: When the distance is the Euclidean metric, we have proposed an indexing method using the first few Discrete Fourier Transform (DFT) coefficients, for matching full sequences [1], as well as for sub-pattern matching [13]. This technique has been extended by Goldin and Kanellakis [14] for matching time sequences, so that it allows for shifts and scalings. In a previous paper [16], we developed a general framework for posing queries based on similarity. The framework enables a formal definition of the notion of similarity for an application domain of choice, and then its use in queries to perform similarity-based search. Both we [11] and Agrawal *et al* [5] developed indexing methods to search for sequences that are similar, despite gaps, translation and scaling. In [24], we developed efficient indexing techniques for similar time sequences under time warping distance. Das *et al* [9] considered a problem of finding rules relating patterns in a time sequence to patterns in other sequences as well as itself. They used clustering of similar subsequences within sliding windows.

Data mining in large databases is also related: Agrawal *et al* [2] proposed an interval classifier method for large databases; in [4, 3] they proposed fast algorithms to search for association rules in binary matrices, by detecting large itemsets. To the best of our knowledge, however, there were no database work that attempted to address the types of data mining problems we try to solve in this paper.

Organization of the paper In the rest of the paper, we describe proposed methods in detail and report experimental results with real datasets in Section 2 and 3. Section 4 concludes this paper and presents future research direction. Appendices provide mathematical details of the proposed methods.

2 MUSCLES

Here we describe the first version of the proposed method, MUSCLES(MUlti-SequenCe LEast Squares). Table 2 gives a list of acronyms and symbols used in the rest of this paper.

The first problem we want to investigate is concerned with delayed sequences. We formulate it as follows:

Problem 1 (Delayed sequence) *Consider k time sequences $s_1, \ldots, s_k$, being updated at every time-tick. Let one of them, say, the first one s_1, be consistently late (*e.g.*, due to a time-zone difference, or due to a slower communication link). Make the best guess for $\widehat{s_1}[t]$, given all the information available.*

Our proposed solution is to set up the problem as a *multi-variate linear regression*,[1] by using *two* sources of information: (1) the past of the given time sequence s_1, *i.e.*, $s_1[t-1], s_1[t-2], \ldots$;

Symbol	Definition
MUSCLES	Multi-Sequence Least Squares
v	number of independent variables in multi-variate regression
k	number of co-evolving sequences
y	the *dependent* variable, that we try to estimate
$\widehat{y}$	estimate of the dependent variable y
$\mathbf{y}$	the column vector with all samples of the dependent variable x
$y[j]$	the j-th sample of the dependent variable y
x_i	the i-the independent variable
$x_i[j]$	the j-th sample of the variable x_i
$\mathbf{x}_i$	the column vector with all the samples of the variable x_i
$\mathbf{x}[j]$	the row vector with j-th samples of all variables x_i
w	span of tracking window
b	count of 'best' ind. variables, used for Sel. MUSCLES
λ	forgetting factor (1, when we don't forget the past)

Table 2: List of symbols and acronyms

(2) the past and present of the other time sequences $s_2, s_3, \ldots, s_v$. Next, we describe the way to achieve this set up: For the given stream s_1, we try to estimate its value as a linear combination of the values of the same and the other time sequences within a window of size w. We refer to w as the *tracking window*. Mathematically, we have the following equation:

$$\begin{aligned} \widehat{s_1}[t] = & & a_{1,1}s_1[t-1] + \ldots + a_{1,w}s_1[t-w] + \\ & a_{2,0}s_2[t]+ & a_{2,1}s_2[t-1] + \ldots + a_{2,w}s_2[t-w] + \\ & & \ldots \\ & a_{k,0}s_k[t]+ & a_{k,1}s_k[t-1] + \ldots + a_{k,w}s_k[t-w], \end{aligned} \tag{1}$$

for all $t = w{+}1, \ldots, N$.

We define the delay operator $D^d(.)$ as follows.

[1]The details of the multi-variate linear regression model can be found elsewhere, such as [19].

Definition 1 *For a sample $s[t]$ from a time sequence $s = (s[1], \ldots, s[N])$, the delay operator $D^d(.)$ delays it by d time steps, that is,*

$$D^d(s[t]) \equiv s[t-d], \quad d+1 \leq t \leq N. \tag{2}$$

Then, Eq. 1 is a collection of linear equations for $t = w+1, \ldots, N$, with $s_1[t]$ being the dependent variable ("y"), and

$$D^1(s_1[t]), \ldots, D^w(s_1[t]), s_2[t], D^1(s_2[t]), \ldots, D^w(s_2[t]), \ldots, s_k[t], D^1(s_k[t]), \ldots, D^w(s_k[t])$$

the independent variables. The least square solution of this system—$a_{i,j}$'s which minimize the sum of $(s_1[t] - \widehat{s_1}[t])^2$ is given by the multi-variate regression. Each $a_{i,j}$ is called a *regression coefficient.* Notice that the number of independent variables is $v = k * (w+1) - 1$.

With this set up, the optimal regression coefficients are given by

$$\mathbf{a} = (\mathbf{X}^T \times \mathbf{X})^{-1} \times (\mathbf{X}^T \times \mathbf{y}) \tag{3}$$

where each column of the matrix $\mathbf{X}$ consists of sample values of the corresponding independent variable in Eq. 1, and each row is observations made at time t. $\mathbf{y}$ is a vector of desired values($s_1[t]$).

Efficiency Although Eq. 3 gives the best regression coefficients, it is very inefficient in terms of both storage requirement and computation time. First, we need $O(N \times v)$ storage for the matrix $\mathbf{X}$. Since, in our setting, the number of samples N is not fixed and can grow indefinitely, we may have to store $\mathbf{X}$ in secondary storage such as disks. The number of disk blocks required is $\lceil \frac{N \times v \times d}{B} \rceil$, where B is the capacity of a disk block and d is the size of floating number representation. With limited main memory, the computation of $(\mathbf{X}^T \times \mathbf{X})$ may require quadratic disk I/O operations very much like a Cartesian product in relational databases. A brute-force solution to this problem could be reduce the size of the matrix, but it creates other problems such as follows:

- How often do we discard the matrix?
- How large a portion of it do we discard?

Even with enough main memory to keep the matrix $\mathbf{X}$, the computational cost for Eq. 3 is $O(v^2 \times (v+N))$ and we have to repeat it as new data sample is available.

We propose to avoid all these problems, by taking advantage of a useful mathematical result called *matrix inversion lemma* [17]. Thanks to its special form, the lemma holds for the matrix

$\mathbf{X}$. Let $\mathbf{X}_n$ denote $\mathbf{X}$ with the first $N = n$ samples and define $\mathbf{G}_n$ as $(\mathbf{X}_n^T \times \mathbf{X}_n)^{-1}$. Then, $\mathbf{G}_n$ can be calculated using $\mathbf{G}_{n-1}$ as follows (see Appendix A for the details):

$$\mathbf{G}_n = \mathbf{G}_{n-1} - (1 + \mathbf{x}[n] \times \mathbf{G}_{n-1} \times \mathbf{x}[n]^T)^{-1} \times (\mathbf{G}_{n-1} \times \mathbf{x}[n]^T) \times (\mathbf{x}[n] \times \mathbf{G}_{n-1}), \quad n > 1 \tag{4}$$

where $\mathbf{x}[n]$ is a row vector of the n-th sample values.

The above equation has some desirable properties. First, the inside of the inversion is just a scalar and, hence, no matrix inversion is required in the above equation. Second, $\mathbf{G}_n$ is much smaller than $\mathbf{X}_n$ because $N = n \gg v$. Its size is fixed and independent of the number of samples. Also, we don't need to keep the original matrix $\mathbf{X}_n$ explicitly. Therefore, it is more likely that we can keep $\mathbf{G}_n$ in main memory. Using Eq. 4, the computational cost for updating regression coefficients $\mathbf{a}$ is only $O(v^2)$ for each new sample. Even when it is not possible to keep $\mathbf{G}_n$ in main memory, we only need $\lceil \frac{v^2 \times d}{B} \rceil$ disk blocks to store it. It is sufficient to scan the blocks at most twice, reducing I/O cost significantly.

As a reference point, for a modest dataset of 100 sequences with 10000 samples each, Eq. 3 takes almost 84 hours to do the estimation on a Sun UltraSparc-1 workstation. On the other hand, for a larger dataset of 100 sequences with 100000 samples each ($\approx$ 80MB), Eq. 4 takes only about 1 hour on the same workstation. Note that the dataset is 10 times larger, but the computation is 80 times faster!

Adaptiveness In addition to its fast execution time, MUSCLES is able to adapt to changes over time: Consider the case where there is a change in the underlying processes that cause multiple sequences to be correlated (such as a trade treaty between two countries, affecting their currency exchange rates). When this happens, formulae derived based on old observed values will no longer be correct. Even worse, they can affect future estimations indefinitely. What we would like to do is to adapt the prediction coefficients so that they reflect the new rather than historical reality.

It turns out that our MUSCLES can be slightly modified to "forget" older samples gracefully. We call the method *Exponentially Forgetting MUSCLES*. That is, let $0 < \lambda \leq 1$ be the forgetting factor, which determines how fast the effect of older samples fades away. Then we try to find the optimal regression coefficient vector $\mathbf{a}$ to minimize

$$\min_{\mathbf{a}} \sum_{i=1}^{N} \lambda^{(N-i)} (y[i] - \hat{y}[i])^2 \tag{5}$$

For $\lambda < 1$, errors for old values are downplayed by a geometric factor, and hence it permits the estimate to adapt as sequence characteristics change. The formulae for exponentially forgetting MUSCLES are given in Appendix A.

It is clear that we can apply this set up for *any* delayed sequence, as opposed to the first only. That is, we can solve the following problem:

Problem 2 (Any Missing Value) *Consider k time sequences $s_1, \ldots, s_v$, being updated at every time-tick. Let one value, $s_i[t]$, be missing. Make the best guess for $\widehat{s}_i[t]$, given all the information available.*

The solution for this is no different than for Problem 1 – we simply have to keep the recursive least squares going for each choice of i. Then, at time t, one is immediately able to reconstruct the missing or delayed value, irrespective of which sequence i it belongs to.

2.1 Making the most out of MUSCLES

We have described how to estimate a missing value in a time sequence, using MUSCLES. Here we justify our initial claim that the solution to this problem (Problem 2) can help us meet all the data mining goals listed in the introduction. The trick is to pretend as if all the sequences were delayed and apply MUSCLES to each of the sequences. Specifically:

- **Correlation detection**: A high absolute value for a regression coefficient means that the corresponding variable is highly correlated to the dependent variable (or current status of a sequence) as well as it is valuable for the estimation of the missing value. Note that the regression coefficients should be normalized *w.r.t.* the mean and the variance of the sequence. Practically, it can be done by keeping track of them within a sliding window. The appropriate window size is $1/(1-\lambda)$, which is approximately the length of memory imposed by the forgetting factor.
- **On-line outlier detection**: Informally, an outlier is a value that is very different from what we expected. In our case, if we assume that the estimation error follows a Gaussian distribution with standard deviation σ, then we label as "outlier" every sample of s_1 that is $\geq 2\sigma$ away from its estimated value. The reason is that, in a Gaussian distribution, 95% of the probability mass is within $\pm 2\sigma$ from the mean.

- **Corrupted data and back-casting**: If a value is corrupted or suspected in our time sequences, we can treat it as "delayed", and forecast it. We can even estimate past (say, deleted) values of the time sequences, by doing back-casting: in this case, we express the past value as a function of the future values, and set up a multi-sequence regression model.

2.2 Experimental Set-up

We performed experiments on several real datasets

- **CURRENCY**: exchange rates of k=6 currencies Hong-Kong Dollar (HKD), Japanese Yen (JPY), US Dollar (USD), German Mark (DEM), French Franc (FRF), and British Pound (GBP) *w.r.t.* Canadian Dollar (CAD). There are N=2561 daily observations for each currency.

- **MODEM**: modem traffic data from a pool of k=14 modems. N=1500 time-ticks, reporting the total packet traffic for each modem, per 5-minute intervals.

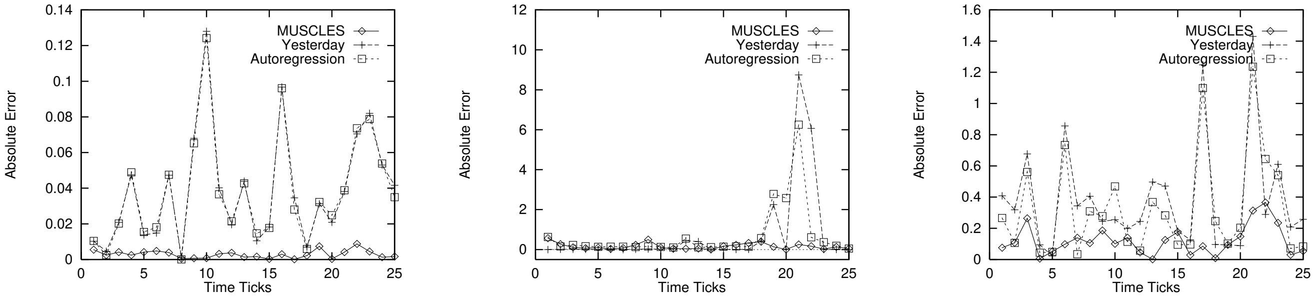
- **INTERNET**: internet usage data for several states. We have four data streams per site, measuring different aspects of the usage (*e.g.*, connect time, traffic and error in packets *etc.*) For each of the data streams, N=980 observations were made.

The experiments were designed to address the following questions:

- **Prediction accuracy:** how well can we fill in the missing values, compared with straightforward heuristics? Following the tradition in forecasting, we use the RMS (root mean square) error.
- **Correlation detection:** can MUSCLES detect hidden yet interesting correlation patterns among sequences?
- **Adaptability to changes:** how well can MUSCLES adapt to changes in correlation pattern over time?

2.3 Accuracy

We used a window of width w=6 unless specified otherwise. As mentioned, the choice of the window is outside the scope of this paper; textbook recommendations include AIC, BIC, MDL, *etc.*[6, 21].



(a) US Dollar (**CURRENCY**) (b) 10-th modem (**MODEM**) (c) 10-th stream (**INTERNET**)

Figure 1: Absolute estimation error as time evolves for the selected sequences.

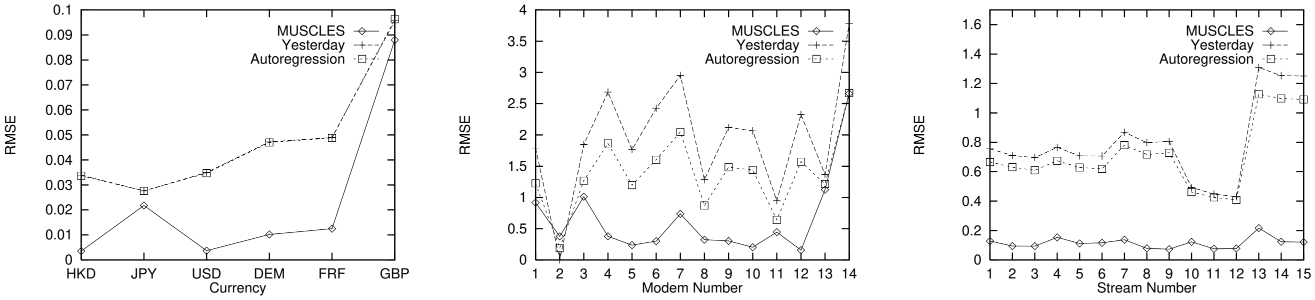
We also used two popular, successful prediction methods:

- "yesterday": $\hat{s}[t] = s[t-1]$, that is, choose the latest value as the estimate for the missing value. It is the typical straw-man for financial time sequences, and actually matches or outperforms much more complicated heuristics in such settings as shown in [18].
- Single-sequence AR (auto-regressive) analysis. This is a special case of the traditional, very successful, Box-Jenkins ARIMA methodology, which tries to express the $s[t]$ value as a linear combination of its past w values.[2]

Figure 1 shows the absolute estimation error of MUSCLES and its competitors, for three sequences, one from each dataset, for the last 25 time-ticks. In all cases, MUSCLES outperformed the competitors. It is interesting to notice that, for the US Dollar, the "yesterday" heuristic and the AR methodology gave very similar results: This is understandable, because the "yesterday" heuristic is a special case of the "AR" method, and, for currency exchange rates, "yesterday" is extremely good. However, our MUSCLES method does even better, because it exploits information not only from the past of the US Dollar, but also from the past and present of other currencies.

Figure 2 shows the RMS error for some sequences of the three real datasets, **CURRENCY**, **MODEM** and **INTERNET**. For each of the datasets, the horizontal axis lists the source, that is, the "delayed" sequence, s_1. We can observe several things. First, MUSCLES outperformed all alternatives, in all cases, except for just one case, the 2nd modem. The explanations is that in

[2]We have chosen AR over ARIMA, because ARIMA requires that an external input source (moving-average term) be specifically designated beforehand and it is impossible in our setting since we are oblivious on specific relationship among sequences.



(a) **CURRENCY** (b) **MODEM** (c) **INTERNET**

Figure 2: RMS error comparisons of several alternatives.

the 2nd modem, the traffic for the last 100 time-ticks was almost zero; and in that extreme case, the "yesterday" heuristic is the best method. For **CURRENCY**, the "yesterday" and the AR methods gave practically identical errors, confirming the strength of the "yesterday" heuristic for financial time sequences. In general, if the MUSCLES method shows large savings for a time sequence, the implication is that this time sequence is strongly correlated with some other of the given sequences. The "yesterday" and AR methods are oblivious to the existence of other sequences, and thus fail to exploit correlations across sequences.

2.4 Correlation detection - Visualization

As we mentioned earlier, a high absolute value for a regression coefficient means that the corresponding variable is highly correlated to the dependent variable (or current status of a sequence) as well as it is valuable for the estimation of the missing value. As we will show in Theorem 1, the correlation coefficient picks the single best predictor for a given sequence. The correlation coefficient ranges from -1 to 1, where high absolute values show strong correlations.

We can turn it into a dis-similarity function, and apply FastMap [12] to obtain a low dimensionality scatter plot of our sequences. Figure 3 does that for the currencies. We took 100 samples back from the last 6 time-ticks ($t, t-1, \ldots, t-5$) for each currency and calculated the dis-similarity based on mutual correlation coefficients. Closely located sequences mean they are highly correlated.

We can see that HKD and USD are very close at every time-tick and so are DEM and FRF. GBP is the most remote from the others and evolves toward the opposite direction. JPY is also

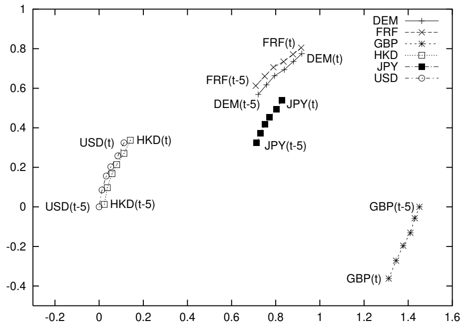


Figure 3: FastMap-based visualization: **CURRENCY**

relatively independent of others. By applying MUSCLES to USD, we found that

$$\widehat{\mathrm{USD}}[t] = 0.9837 * \mathrm{HKD}[t] + 0.6085 * \mathrm{USD}[t-1] - 0.5664 * \mathrm{HKD}[t-1] \tag{6}$$

after ignoring regression coefficients less than 0.3. The result confirms that the USD and the HKD are closely correlated and perfectly agrees with Figure 3 as well as Figure 2 (a).

2.5 Adapting to Change

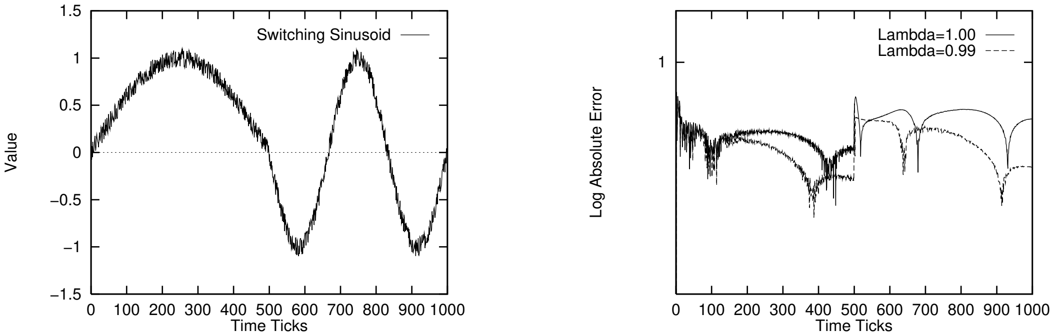
To demonstrate the adaptability of MUSCLES, we created a synthetic dataset in the following way:

- **SWITCH**("switching sinusoid") 3 sinusoids s_1, s_2, s_3 with N=1,000 time-ticks each;

$$\begin{aligned} s_1[t] &= s_2[t] + 0.1 * n[t] \qquad t \le 500 \\ &= s_3[t] + 0.1 * n'[t] \qquad t > 500 \\ s_2[t] &= \sin(2\pi t/N) \\ s_3[t] &= \sin(2\pi 3t/N) \end{aligned}$$

  where $n[t], n'[t]$ are white noise that is, Gaussian, with zero mean and unit standard deviation. Thus, s_1 switches at $t = 500$, and tracks s_3, as opposed to s_2. This switch could happen, *e.g.*, in currency exchange rates, due to the signing of an international treaty between the involved nations.

We tested the effect of the forgetting factor (λ) on the synthetic **SWITCH** dataset. Recall that s_1 tracks s_2 for the first half of the time, and then suddenly switches and tracks s_3. Figure 4 shows the absolute error versus time-ticks, with $\lambda = 1$ and $\lambda = 0.99$.



(a) Switching sinusoid (b) Prediction errors with $\lambda = 1$ and $\lambda = 0.99$

Figure 4: The effect of different forgetting factor (λ) for the **SWITCH** dataset.

Notice that the MUSCLES method without "forgetting" does not adapt so quickly to the change: there is a big surge at $t = 500$, as expected, but MUSCLES with λ=0.99 recovers faster from the shock. The regression equations after $t = 1000$ when $w = 0$ are,

$$\widehat{s_1}[t] = 0.499 * s_2[t] + 0.499 * s_3[t] \qquad (\lambda = 1) \tag{7}$$

for the "non-forgetting" version and

$$\widehat{s_1}[t] = 0.0065 * s_2[t] + 0.993 * s_3[t] \qquad (\lambda = 0.99) \tag{8}$$

for the "forgetting" one. That is, the "forgetting" version of MUSCLES has effectively ignored the first 500 time-ticks, and has identified the fact that s_1 has been tracking s_3 closely. In contrast, the non-forgetting version gives equal weight (≈ 0.5) to s_2 and s_3 alike, as expected.

3 Scaling-up: Selective MUSCLES

In case we have too many time sequences (*e.g.*, k=100,000 nodes in a network, producing information about their load every minute), even the incremental version of MUSCLES will suffer. The solution we propose is based on the conjecture that we do not really need information from every sequence to make a good estimation of a missing value – much of the benefit of using multiple sequences may be captured by using only a small number of carefully selected other sequences. Thus, we propose to do some preprocessing of a training set, to find a promising subset of sequences, and to apply MUSCLES only to those promising ones (hence the name Selective MUSCLES).

Following the running assumption, sequence s_1 is the one notoriously delayed, which needs to be estimated. For a given tracking window span w, among the v independent variables, we have to choose the ones that are most useful in estimating the delayed value of s_1.

Problem 3 (Subset selection) *Given v independent variables $x_1, x_2, \ldots, x_v$ and a dependent variable y with N samples each, find the best $b(< v)$ independent variables to minimize the mean-square error for $\widehat{y}$ for the given samples.*

We need a measure of goodness to decide which subset of b variables is the best we can choose. Ideally, we should choose the best subset that yields the smallest estimation error in the future. Since, however, we don't have future samples, we can only infer the *expected estimation error* (EEE for short) from the available samples as follows:

$$\mathrm{EEE}(S) = \sum_{i=1}^{N} (y[i] - \widehat{y_S}[i])^2$$

where S is the selected subset of variables and $\widehat{y_S}[i]$ is the estimation based on S for the i-th sample. Note that, thanks to Eq. 4, $\mathrm{EEE}(S)$ can be computed in $O(N \times |S|^2)$ time.

Let's say that we are allowed to keep only $b = 1$ independent variable - which one should we choose? Intuitively, we could try the one that has the highest (in absolute value) correlation coefficient with y. It turns out that this is indeed optimal: (to satisfy the unit variance assumption, we will normalize samples by the sample variance within the window.)

Theorem 1 *Given a dependent variable y, and v independent variables with unit variance,the best single variable to keep to minimize EEE(S) is the one with the highest absolute correlation coefficient with y.*

Proof: See Appendix B. **QED**

The question is how we should handle the case when $b > 1$. Normally, we should consider all the possible groups of b independent variables, and try to pick the best. This approach explodes combinatorially; thus we propose to use a greedy algorithm (see Algorithm 1). At each step s, we select the independent variable x_s that minimizes the EEE for the dependent variable y, in light of the $s - 1$ independent variables that we have already chosen in the previous steps.

Bottleneck of the algorithm is clearly the computation of EEE. Since it computes EEE approximately $O(v \times b)$ times and each computation of EEE requires $O(N \times b^2)$ in average, the overall complexity mounts to $O(N \times v \times b^3)$. To reduce the overhead, we observe that intermediate results produced for $\mathrm{EEE}(S)$ can be re-used for $\mathrm{EEE}(S \cup \{x\})$.

```
algorithm Selection
   S := {};                 /* Set of selected variables */
   R := {x_1, ..., x_v};    /* Set of remaining variables */
   while ( S contains less than b variables )
      foreach x in R
         Compute EEE for S ∪ {x};
      pick x with minimum EEE;
      remove x from R and add to S;
   end while
   report variables in S;
end algorithm
```

Algorithm 1: Algorithm to select b variables

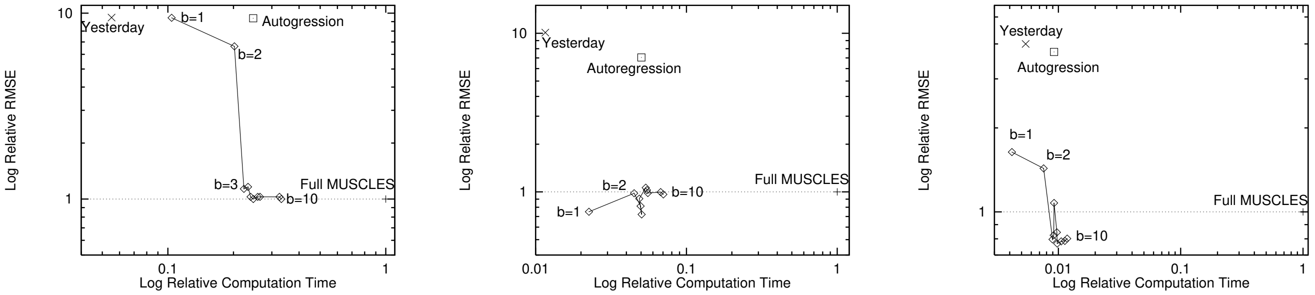
Theorem 2 *The complexity of Algorithm 1 is $O(N \times v \times b^2)$.*

Proof: Let S^+ be $S \cup \{x\}$. The core in computing EEE($S \cup x$) is the inverse of $\mathbf{D}_{S^+} = (\mathbf{X}_{S^+}^T \times \mathbf{X}_{S^+})$. Thanks to *block matrix inversion* formula [17, p. 656] and the availability of $\mathbf{D}_S^{-1}$ from the previous iteration step, it can be computed in $O(N \times |S| + |S|^2)$. Hence, summing it up over $v - |S|$ remaining variables for each b iteration, we have $O(N \times v \times b^2 + v \times b^3)$ complexity. Since $N \gg b$, it reduces to $O(N \times v \times b^2)$. See Appendix B for more detail. **QED**

We envision that the subset-selection will be done infrequently and off-line, say every $N = W$ time-ticks. The optimal choice of the reorganization window W is beyond the scope of this paper. Potential solutions include (a) doing reorganization during off-peak hours, (b) triggering a reorganization whenever the estimation error for $\hat{y}$ increases above an application-dependent threshold *etc.* Also, by normalizing the training set, the unit-variance assumption in Theorem 1 can be easily satisfied.

3.1 Experiments

One obvious question that arises is how much faster the Selective MUSCLES method is than MUSCLES, and at what cost in accuracy. We ran experiments with the datasets described in Section 2.2.



(a) US Dollar (**CURRENCY**) (b) 10-th modem (**MODEM**) (c) 10-th stream (**INTERNET**)

Figure 5: The relative RMS error vs. relative computation time, for several values of b 'best-picked' independent variables. proposed($\diamond$) with varying b, Full MUSCLES($+$), yesterday ($\times$), and auto-regression($\Box$).

Figure 5 shows the speed-accuracy trade-off for Selective MUSCLES. It plots the RMS error versus the computation time with varying number of independent variables ($b = 1, \ldots, 10$), in double logarithmic scale. The computation time adds the time to forecast the delayed value, plus the time to update the regression coefficients. The reference point is the MUSCLES method on all v (referred to as the Full MUSCLES in this subsection). For ease of comparison across several datasets, we normalize both measures (the RMS error as well as the computation time), by dividing by the respective measure for the Full MUSCLES. For each set-up, we vary the number b of independent variables picked. The Figure shows the error-time plot for the same three sequences (the US Dollar from **CURRENCY**, the 10-th modem from **MODEM**, and the 10-th stream from **INTERNET**).

For every case, we have close to an order of magnitude (and usually much more) reduction in computation time, if we are willing to tolerate up to a 15% increase in RMS error. We also observe that in most of the cases b=3-5 best-picked variables suffice for accurate estimation. The Figure also shows our Selective MUSCLES is very effective, achieving up to 2 orders of magnitude speed-up (**INTERNET**, 10-th stream), with small deterioration in the error, and often with gains.

4 Conclusions and Future Research

We have presented fast methods to build analytical models for co-evolving time sequences, like currency exchange rates and network traffic data to name a few. The proposed methods (MUS-

CLES and Selective MUSCLES) have the following advantages: (1) they are useful for data mining and discovering correlations (with or without lag); (2) they can be used for forecasting of missing/delayed values; (3) they can be made to adapt to changing correlations among time sequences; and (4) they scale up well for a large number of sequences which can grow indefinitely long.

We showed that the proposed methods are mathematically sound as well as computationally efficient. They require much less storage overhead so that even with limited main memory, they do not cause excessive I/O operations as a naive method does. We suggested how the proposed methods could be used for various data mining tasks in co-evolving time sequences. Experiments on real datasets show that our methods outperform some popular successful competitors in estimation accuracy up to 10 times, and they discover interesting correlations (*e.g.*, USD and HKD). The Selective MUSCLES scales up very well for a large number of sequences, in a variety of real-world settings (currency exchange rates, network traffic data, and internet usage data), reducing response time up to 110 times over MUSCLES, and sometimes even improves estimation accuracy.

For future research, the regression method called *Least Median of Squares* [22] is promising. It is more robust than the Least Squares regression that is the basis of MUSCLES, but also requires much more computational cost. The research challenge is to make it scale up for a massive database environment. Another interesting research issue in time sequence databases is an efficient method for forecasting of non-linear time sequences such as chaotic signals [23].

A Appendix: Incremental Computation

Given N samples, $(x_1[i], x_2[i], \ldots, x_v[i], y[i]), i = 1, \ldots, N$, our goal is to find the values $a_1, \ldots, a_v$ that give the best estimations for y in the sense of least squares error. That is, we look for the $a_1, \ldots, a_v$ that minimize

$$\min_{a_1,\ldots,a_v} \sum_{i=1}^{N} (y[i] - a_1 x_1[i] - \ldots - a_v x_v[i])^2 \tag{9}$$

Using matrix notation, the solution to Eq. 9 is given compactly by [19, pp. 671–674]:

$$\mathbf{a} = (\mathbf{X}^T \times \mathbf{X})^{-1} \times (\mathbf{X}^T \times \mathbf{y}) \tag{10}$$

where the super-scripts T and -1 denote the transpose and the inverse of a matrix, respectively; $\times$ denotes matrix multiplication; $\mathbf{y}$ is the column vector with the samples of the dependent variable; $\mathbf{a}$ is the column vector with the regression coefficients. The matrix $\mathbf{X}$ is the $N \times v$ matrix with the N samples of the v independent variables. That is:

$$\mathbf{X} = \begin{bmatrix} x_1[1] & x_2[1] & \dots & x_v[1] \\ x_1[2] & x_2[2] & \dots & x_v[2] \\ \vdots & \vdots & \dots & \vdots \\ x_1[N] & x_2[N] & \dots & x_v[N] \end{bmatrix} \tag{11}$$

Recall that $x_j[i]$ denotes the i-th sample of the j-th independent variable.

Let $\mathbf{X}_n$ be the matrix with all the independent variables, but with only the first n samples. Thus, its dimensions are $n \times v$. Let $\mathbf{D}_n = {\mathbf{X}_n}^T \times \mathbf{X}_n$, where D stands for "data". The goal is to invert the matrix $\mathbf{D} = \mathbf{D}_n$. Notice that its dimensions are $v \times v$, and its inversion would normally take $O(v^3)$ time. Since the construction of $\mathbf{D}$ takes $O(n \times v^2)$, the total computation time is $O(n \times v^2 + v^3)$. From a database point of view, this is acceptable only when the number of data samples is fixed and small. However, it is not suitable for applications where there are large number of data samples and new samples are added dynamically, because the new matrix $\mathbf{D}$ and its inverse should be computed whenever a new set of samples arrive. Thanks to its special form and thanks to the so-called *matrix inversion lemma* [15, 17], $(\mathbf{D}_n)^{-1}$ can be incrementally computed using previous value $(\mathbf{D}_{n-1})^{-1}$. This method is called *Recursive Least Square* (RLS) and its computation cost is reduced to $O(v^2)$.

Next we present the final formulas for the solution; the proofs are in *e.g.*, [15]. Following the notation in the statistics literature, the inverse $\mathbf{G}_n = (\mathbf{D}_n)^{-1}$ is called the *gain matrix*. Let $\mathbf{x}[i]$ be a row vector, denoting the i-th sample (row) of $\mathbf{X}$. That is $\mathbf{x}[i] = (x_1[i], x_2[i], \dots, x_v[i])$ Also, let $y[i]$ be the i-th sample of $\mathbf{y}$. Then, we can compute $\mathbf{G}_n$ $(n = 1, \dots)$ *recursively*, as follows:

$$\mathbf{G}_n = \mathbf{G}_{n-1} - (1 + \mathbf{x}[n] \times \mathbf{G}_{n-1} \times \mathbf{x}[n]^T)^{-1} \times (\mathbf{G}_{n-1} \times \mathbf{x}[n]^T) \times (\mathbf{x}[n] \times \mathbf{G}_{n-1}), \quad n > 1 \tag{12}$$

with $\mathbf{G}_0 = \delta^{-1} \times \mathbf{I}$, where δ is a small positive number (*e.g.*, 0.004), and $\mathbf{I}$ is the identity matrix. The coefficient vector $\mathbf{a}_n$ after the n-th sample has arrived, can also be updated incrementally

$$\mathbf{a}_n = \mathbf{a}_{n-1} - \mathbf{G}_n \times \mathbf{x}[n]^T \times (\mathbf{x}[n] \times \mathbf{a}_{n-1} - y[n]) \qquad n > 1 \tag{13}$$

and $\mathbf{a}_0 = \mathbf{0}$, where $\mathbf{a}_n$ is the vector of regression coefficients when we consider the first n samples only, and $\mathbf{0}$ is a column vector of v zeros. Notice that Eq. 12 needs only matrix multiplications with complexity $O(v^2)$, a function of v only. If we repeat this for N sample arrivals, the total computation cost becomes $O(N \times v^2)$.

In addition to its lower complexity, it also allows for graceful "forgetting" of the older samples. More specifically, we wish to have the effect of each samples diminished by a factor of λ ($0 < \lambda \leq 1$) at each time-tick, thus allow *exponential forgetting*. In this setting, $\mathbf{G}_n$ now can be computed by the following equation:

$$\mathbf{G}_n = \lambda^{-1}\mathbf{G}_{n-1} - \lambda^{-1}(\lambda + \mathbf{x}[n] \times \mathbf{G}_{n-1} \times \mathbf{x}[n]^T)^{-1} \times (\mathbf{G}_{n-1} \times \mathbf{x}[n]^T) \times (\mathbf{x}[n] \times \mathbf{G}_{n-1}), \quad n > 1. \quad (14)$$

Of course, it agrees with Eq. 12 when λ=1 (*i.e.*, no "forgetting"). The $\mathbf{a}_n$ is still given by Eq. 13.

B Appendix: Subset Selection

Here we present the formulas for the solution of Problem 3, that is, we describe the procedure to select the b best independent variables to estimate y, given the past N time-ticks. As a basis, we must choose the first variable to regress on. For each independent variable x_i, let a scalar a be the least-squares solution of Eq. 9. Then, EEE($\{x_i\}$) can be expressed by matrix notation as follows:

$$\begin{aligned}
\mathrm{EEE}(\{x_i\}) &= \|\mathbf{y} - a\mathbf{x}_i\|^2 \\
&= (\mathbf{y} - a\mathbf{x}_i)^T \times (\mathbf{y} - a\mathbf{x}_i) \\
&= \|\mathbf{y}\|^2 - 2a(\mathbf{y}^T \times \mathbf{x}_i) + a^2\|\mathbf{x}_i\|^2
\end{aligned}$$

Let d and p denote $\|\mathbf{x}_i\|^2$ and $(\mathbf{x}^T \times \mathbf{y})$, respectively. Since $a = d^{-1}p$,

$$\begin{aligned}
\mathrm{EEE}(\{x_i\}) &= \|\mathbf{y}\|^2 - 2p^2d^{-1} + p^2d^{-1} \\
&= \|\mathbf{y}\|^2 - p^2d^{-1}
\end{aligned}$$

To minimize the error, we must choose $\mathbf{x}_i$ which maximize p^2 and minimize d. Assuming unit-variance ($d = 1$), such $\mathbf{x}_i$ is the one with the biggest correlation coefficient to $\mathbf{y}$. This proves Theorem 1.

Now suppose we have chosen a subset of variables, say S, and try to select one more variable. Let $\mathbf{X}_S$ denote a matrix of column vectors $\mathbf{x}_i$ which correspond to variables x_i in S. We define $\mathbf{D}_S$ as $(\mathbf{X}_S^T \times \mathbf{X}_S)$ and $\mathbf{P}_S$ as $(\mathbf{X}_S^T \times \mathbf{y})$. We assume that $(\mathbf{D}_S)^{-1}$ is available from the previous selection step. We consider one of the remaining variables in R, say x_j. If we denote $S \cup \{x_j\}$

by S^+, then,

$$\begin{aligned}
\mathrm{EEE}(S^+) &= \|\mathbf{y} - \widehat{\mathbf{y}}_{S^+}\|^2 \\
&= \|\mathbf{y} - \mathbf{X}_{S^+} \times \mathbf{a}_{S^+}\|^2 \\
&= \|\mathbf{y}\|^2 - 2(\mathbf{P}_{S^+}^T \times \mathbf{a}_{S^+}) + (\mathbf{a}_{S^+}^T \times \mathbf{D}_{S^+} \times \mathbf{a}_{S^+})
\end{aligned}$$

where $\mathbf{a}_{S^+}$ is the optimal regression coefficient vector for variables in S^+ *w.r.t.* Eq. 9, and given by,

$$\mathbf{a}_{S^+} = (\mathbf{D}_{S^+})^{-1} \times \mathbf{P}_{S^+}.$$

Thus, the expected estimation error becomes,

$$\begin{aligned}
\mathrm{EEE}(S^+) &= \|\mathbf{y}\|^2 - 2(\mathbf{P}_{S^+}^T \times \mathbf{D}_{S^+}^{-1} \times \mathbf{P}_{S^+}) + (\mathbf{D}_{S^+}^{-1} \times \mathbf{P}_{S^+})^T \times \mathbf{D}_{S^+} \times (\mathbf{D}_{S^+}^{-1} \times \mathbf{P}_{S^+}) \\
&= \|\mathbf{y}\|^2 - (\mathbf{P}_{S^+}^T \times \mathbf{D}_{S^+}^{-1} \times \mathbf{P}_{S^+}).
\end{aligned}$$

Now we show how to compute $(\mathbf{D}_{S^+})^{-1}$ efficiently without explicit matrix inversion. Thanks to *block matrix inversion* formula [17, p. 656], we can avoid explicit inversion. The general form of this formula is as follows:

$$\begin{bmatrix} \mathbf{A} & \mathbf{D} \\ \mathbf{C} & \mathbf{B} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{E} \times \mathbf{\Delta}^{-1} \times \mathbf{F} & -\mathbf{E} \times \mathbf{\Delta}^{-1} \\ -\mathbf{\Delta}^{-1} \times \mathbf{F} & \mathbf{\Delta}^{-1} \end{bmatrix}$$

where $\mathbf{\Delta} = \mathbf{B} - \mathbf{C} \times \mathbf{A}^{-1} \times \mathbf{D}$, $\mathbf{E} = \mathbf{A}^{-1} \times \mathbf{D}$, and $\mathbf{F} = \mathbf{C} \times \mathbf{A}^{-1}$. Since

$$\mathbf{D}_{S^+} = \begin{bmatrix} \mathbf{X}_S^T \times \mathbf{X}_S & \mathbf{X}_S^T \times \mathbf{x}_j \\ \mathbf{x}_j^T \times \mathbf{X}_S & \mathbf{x}_j^T \times \mathbf{x}_j \end{bmatrix},$$

we substitute $\mathbf{A}$, $\mathbf{\Delta}$, $\mathbf{E}$, and $\mathbf{F}$ as follows:

$$\begin{aligned}
\mathbf{A} &= \mathbf{X}_S^T \times \mathbf{X}_S = \mathbf{D}_S \\
\mathbf{\Delta} &= \|\mathbf{x}_j\|^2 - \mathbf{x}_j^T \times \mathbf{X}_S \times \mathbf{D}_S^{-1} \times \mathbf{X}_S^T \times \mathbf{x}_j \\
\mathbf{E} &= \mathbf{D}_S^{-1} \times \mathbf{X}_S^T \times \mathbf{x}_j \\
\mathbf{F} &= \mathbf{x}_j^T \times \mathbf{X}_S \times \mathbf{D}_S^{-1}
\end{aligned}$$

Note that $\mathbf{\Delta}$ is essentially a scalar and $\mathbf{D}_S^{-1}$ is available from the previous step. The complexity of $\mathbf{D}_{S^+}^{-1}$ computation is $O(N \times |S| + |S|^2)$. We compute $\mathrm{EEE}(S \cup x_j)$ for each remaining variable x_j and select the one with the minimum value. We repeat these steps until we select all b variables. Given N, v, and b, the total computational complexity is $O(N \times v \times b^2 + v \times b^3)$. Since $N \gg b$, we finally have $O(N \times v \times b^2)$. This proves Theorem 2.

References

[1] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *Fourth Int. Conf. on Foundations of Data Organization and Algorithms (FODO)*, pages 69–84, Evanston, IL, Oct 1993.

[2] R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer, and A. Swami. An interval classifier for database mining applications. *VLDB Conf. Proc.*, pages 560–573, Aug 1992.

[3] R. Agrawal, T. Imielinski, and A. Swami. Database mining: a performance perspective. *IEEE Trans. on Knowledge and Data Engineering*, 5(6):914–925, 1993.

[4] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. *Proc. ACM SIGMOD*, pages 207–216, May 1993.

[5] R. Agrawal, K.-I. Lin, H. S. Sawney, and K. Shim. Fast similarity search in the presence of noise, scaling and translation in time-series databases. *Proc. of VLDB*, pages 490–501, September 1995.

[6] G. Box, G. Jenkins, and G. Reinsel. *Time Series Analysis: Forecasting and Control.* Prentice Hall, Englewood Cliffs, NJ, 1994. 3rd Edition.

[7] M. Castagli and S. Eubank. *Nonlinear Modeling and Forecasting.* Addison Wesley, 1992. Proc. Vol. XII.

[8] C. Chatfield. *The Analysis of Time Series: an Introduction.* Chapman and Hall, London & New York, 1984.

[9] G. Das, K.-I. Lin, H. Mannila, G. Renganathan, and P. Smyth. Rule Discovery from Time Series. In *Proc. of KDD'98*, Aug 1998.

[10] Dennis DeCoste. Minig Multivariate Time-Series Sensor Data to Discover Behavior Envelopes. In *Proc. of KDD'97*, Aug 1997.

[11] C. Faloutsos, H.V. Jagadish, A. Mendelzon, and T. Milo. A signature technique for similarity-based queries. In *Proceedings of SEQUENCES97*, Salerno, Italy, Jun 1997. IEEE Press.

[12] C. Faloutsos and K.-I. Lin. Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. *Proc. of ACM-SIGMOD*, pages 163–174, May 1995.

[13] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. *Proc. ACM SIGMOD*, pages 419–429, May 1994.

[14] D. Goldin and P. Kanellakis. On similarity queries for time-series data: Constraint specification and implementation. *Int. Conf. on Principles and Practice of Constraint Programming (CP95)*, Sep 1995.

[15] S. Haykin. *Adaptive Filter Theory.* Prentice Hall, Englewood Cliffs, New Jersey, 1996.

[16] H.V. Jagadish, A. Mendelzon, and T. Milo. Similarity-based queries. *Proc. ACM SIGACT-SIGMOD-SIGART PODS*, pages 36–45, May 1995.

[17] T. Kailath. *Linear Systems.* Prentice Hall, Englewood Cliffs, New Jersey, 1980.

[18] B. LeBaron. Nonlinear forecasts for the s\&p stock index. In M. Castagli and S. Eubank, editors, *Nonlinear Modeling and Forecasting*, pages 381–393. Addison Wesley, 1992. Proc. Vol. XII.

[19] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C.* Cambridge University Press, 1992. 2nd Edition.

[20] L. Rabiner and B.-H. Juang. *Fundamentals of Speech Recognition.* Prentice Hall, 1993.

[21] J. Rissanen. Minimum description length principle. In S. Kotz and N. L. Johnson, editors, *Encyclopedia of Statistical Sciences*, volume V, pages 523–527. John Wiley and Sons, New York, 1985.

[22] P. Rousseeuw and A. Leroy. *Robust Regression and Outlier Detection.* John Wiley, New York, 1987.

[23] A. Weigend and N. Gerschenfeld. *Time Series Prediction: Forecasting the Future and Understanding the Past.* Addison Wesley, 1994.

[24] B.-K. Yi, H.V. Jagadish, and C. Faloutsos. Efficient Retrieval of Similar Time Sequences under Time Warping. In *IEEE Proc. of ICDE*, Feb 1998.

Contents