Mining for weak periodic signals in time series databases

Article *in* Intelligent Data Analysis · March 2005

DOI: 10.3233/IDA-2005-9103

CITATION
1

READS
376

2 authors:

Christos Berberidis
International Hellenic University
22 PUBLICATIONS **365** CITATIONS
SEE PROFILE

I. Vlahavas
Aristotle University of Thessaloniki
327 PUBLICATIONS **6,781** CITATIONS
SEE PROFILE

Intelligent Data Analysis 6 (2005) 1–14 1
IOS Press

Mining for weak periodic signals in time series databases

Christos Berberidis* and Ioannis Vlahavas
Department of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece

Received 6 October 2003
Revised 13 December 2004
Accepted 24 May 2004

Abstract. Periodicity is a particularly interesting feature, which is often inherent in real world time series data sets. In this article we propose a data mining technique for detecting multiple partial and approximate periodicities. Our approach is exploratory and follows a filter/refine paradigm. In the filter phase we introduce an autocorrelation-based algorithm that produces a set of candidate partial periodicities. The algorithm is extended to capture approximate periodicities. In the refine phase we effectively prune invalid periodicities. We conducted a series of experiments with various real-world data sets to test the performance and verify the quality of the results.

Keywords: Time series mining, periodicity detection

1. Introduction

Data mining has emerged from the need to exploit the large amounts of various data collected in electronic format. It is the analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner. Data mining typically deals with data that have already been collected for some purpose other than the data mining analysis [1]. As a result, the data usually contain significant amount of noise and randomness, further enhancing the challenge of the knowledge discovery process. Having its roots in artificial intelligence, machine learning and statistics, data mining involves a set of tools for extracting valuable knowledge to provide the decision maker with valuable insight. Depending on the nature of the data and on the kind of information one is interested in, different techniques can be applied. Temporal data are a special category and periodicity is a particularly interesting feature to mine for.

Periodicity is a key feature that could be used for understanding time series data and predicting future trends. In real world data, rarely a pattern is perfectly periodic (according to the strict mathematical definition of periodicity) and therefore an almost periodic pattern can be considered as periodic with some confidence measure. Partial periodic patterns are patterns that are not periodic over the entire time series but over a fraction of it. The notion of partial periodicities has been discussed in [12]. An interesting extension of the problem of capturing all kinds of periodicities that might occur in real world

*Corresponding author: Christos Berberidis, Department of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece. Tel.: +30 231 099 8418; Fax: +30 231 099 8362, E-mail: berber@csd.auth.gr.

time series data is the discovery of approximate periodicities. That is, periodicities when a number of occurrences are not 100% punctual. Nevertheless, little attention has been paid on the study of the periodic behavior of a temporal attribute. Approaches regarding time series mining so far focus on the extraction of periodic patterns, presuming the period is known in advance.

In this paper we attempt to detect weak periodic signals in large, real world time in exploratory fashion. Usually, noise and uncertainty are inherent in those data, which makes knowledge discovery a really challenging task. The type of noise and uncertainty we face here can be caused from various reasons that all have the same result; valid periods feature low score and become losers and "ghost" periods become winners, making it hard to discover the right winners and to eliminate the false periods, especially when the dataset features more than one periods.

The method is applied on various real world data sets. By "weak periodic signals" we mean partial and approximate periodicities. Our work extends and complements the algorithm introduced in [18], for discovering multiple and partial periodicities, without any previous knowledge of the nature of the data. We add a refine phase for removing a large number of invalid periods, plus more experiments using some extra real world data. At preprocessing we use discretization to reduce the cardinality of the continuous data. Our method follows a filter/refine paradigm and it is complete. In the filter phase we utilize autocorrelation function to produce a set of candidate periodicities, while in the refine phase we effectively filter out invalid periodicities from the previous phase.

The remainder of the paper is organized as follows. In the next section we present representative papers in the area of sequence and time series mining along with papers in the specific area of periodicity mining. Then we present our partial periodicity detection algorithm as well as the experiments we performed and we discuss its performance. In Section 4 we describe our approximate periodicity detection algorithm and finally, in Section 5 we end with our conclusions and some directions for further research.

2. Related work

Early work in time-series data mining addresses the pattern-matching problem. Agrawal et al. in the early 90's developed algorithms for pattern matching and similarity search in time series databases [2–4]. Their work is extended by Garofalakis et al. in [5], introducing a set of algorithms with the use of Regular Expressions as a constraint specification tool. In [6] Agrawal et al. define a shape definition language for retrieving user-specified shapes (episodes) in histories (time series). Mannila et al. [7] introduce an efficient solution to the discovery of frequent patterns in a sequence database, using a moving time window. Chan et al. [8] study the use of wavelets in time series matching and Faloutsos et al. in [9] and Keogh et al. in [10] propose indexing methods for fast sequence matching using R* trees, the Discrete Fourier Transform and the Discrete Wavelet Transform. Keogh et al. in [10] also target the problem of dimensionality reduction in time series mining, through a multidimensional index structure. Bettini et al. in [15] propose a method for finding frequent patterns in time series with the use of event structures, i.e. sets of temporal constraints between events in terms of temporal granularities.

Several approaches for time series mining have also been proposed, with respect to periodicity. Toroslu et al. in [11] address the problem of mining cyclically repeated patterns. Han et al. [12] introduce the concept of partial periodic patterns and propose a data structure called the Max Subpattern Tree for finding partial periodic patterns in a time series. Yang et al. [22] propose a method for mining partial asynchronous (shifted) periodic patterns. In [13] Wang et al. extend the problem posed in [12], by defining meta-patterns for capturing high-level periodicities. Finally, Yang et al. propose a method for capturing all possible partial and asynchronous (approximate) periods and extracting the longest valid

periodic subsequence. Their algorithm uses a moving window to perform a distance-based pruning of candidate patterns.

2.1. Our approach

Most of the algorithms proposed in the above articles, discover periodic patterns for a user-defined period length. If the period length is not known in advance, then these algorithms are not directly applicable. One would have to exhaustively apply them for each possible period length, which is impractical. In other words, it is assumed that the period is known in advance thus making the process essentially ad-hoc, since unsuspected periodicities will be missed. This paper extends the approach described in [18] where they propose an algorithm for extracting candidate periods when searching for multiple and partial periodic patterns in large time series.

Based on the Apriori property, we propose the Partial Periodicity Detection (PPD) algorithm, an algorithm that generates a set of candidate periods for the symbols of a time series. Additionally, we propose APPD, an extension of PPD for capturing approximate periodicities. Our approach follows a filter/refine paradigm, a technique that has been used in several contexts, e.g., in spatial query processing [16]. The filter phase reduces the search space by eliminating a large number of objects that are unlikely to contribute to the final solution. The refine phase, which is CPU-intensive, involves testing the candidate set produced at the filter step in order to verify which objects fulfill the query condition. The filter/refine paradigm can be applied in various search problems such as the search for periodicity in a time series.

3. The Partial Periodicity Detection (PPD) algorithm

The algorithm we propose involves two phases. In the filter phase, PPD utilizes the Fast Fourier Transform to compute an autocorrelation function that provides us with a conservative set of candidate period lengths for every letter in the alphabet of our time series. We use discretization to reduce the cardinality of the data, transforming the time series into a sequence of characters. In the refine phase we apply a simple random verification technique, which experiments show that effectively reduces the candidate set, removing a large number of false cases. The complexity of our algorithm is O($ANlogN$), where A is the size of the alphabet and N the size of the time series. The performance of the algorithm scales up linearly both to the number of time points and the size of the alphabet.

3.1. Definitions

A **pattern** is a string $s = s_1 \dots s_p$ over an **alphabet** $L \cup \{*\}$, where the letter * stands for *any single symbol from* L. A pattern $s' = s'_1 \dots s'_p$ is a *subpattern* of another pattern s if for each position $i, s'_i = s_i$ or $s'_i = *$. For example, ab^*d is a subpattern of $abcd$. Assume that a pattern is periodic in a time series S of length N with a **period** of length p. Then, S can be divided into $\lfloor N/p \rfloor$ segments of size p. These segments are called **periodic segments**. The **frequency count** of a pattern is the number of the periodic segments of the time series that match this pattern. The **confidence** of a pattern is defined as the division of its frequency count by the number of period segments in the time series ($\lfloor N/p \rfloor$). For example, in the series *abcdabddabfcccba*, the pattern ab^{**} is periodic with a period length of 4, a frequency count of 3, and a confidence of 3/4.

According to the **Apriori property on periodicity** discussed in [12] "each subpattern of a frequent pattern of period p is itself a frequent pattern of period p". For example, assume that ab^{**} is a periodic

pattern with a period of 4, then a^{***} and $^*b^{**}$ are also periodic with the same period. Conversely, knowing that a^{***} and $^*b^{**}$ are periodic with period 4 does not necessarily imply that ab^{**} is periodic with period 4.

3.2. Outline of the algorithm

Time series is defined as a set of observations ordered in time. We will consider only discrete time series, with observations x_t at times $t = 1, 2, \ldots, N$, where N is the length of the time series, which may consist of categorical or continuous values (e.g. hourly temperature measurement, daily price of a stock etc.). In order to reduce the cardinality of the continuous data, we perform a standard discretization procedure, where the assistance and guidance of the domain expert is particularly desired. The range of continuous values of the time series is divided into a number of segments and a symbol is assigned to each one of them. This way the time series is transformed into a character sequence. The set of the different symbols used in this process is the alphabet of the (transformed) series. Let M be the number of those symbols, that is the size of the alphabet. The major steps of the algorithm are outlined below.

1. Create M bit vectors of size N, one for every symbol in the alphabet. This step requires one scan over the time series.
2. Calculate the autocorrelation function of every bit vector, producing M vectors of size N. This step requires *NlogN* computations.
3. Based on the user-specified minimum confidence threshold, filter out the valid periodicities from the autocorrelation vectors. This step requires one scan over the autocorrelation vectors.
4. Refine the results, producing a conservative set of candidate periods for every symbol.
5. Based on the Apriori property on periodicity, use the results of the previous step as a seed to a pattern extraction algorithm.

Steps 1–3 correspond to the filter phase while steps 4 and 5 correspond to the refine phase. Various such algorithms have been proposed for step 5, such as the Max-subpattern Hit Set Algorithm that mines for partial periodic patterns in a time series database. It utilizes a tree structure, the Max-Subpattern tree, whose nodes represent a candidate frequent pattern for the time series. Each node has a count value that reflects the number of occurrences of the pattern represented by this node in the entire time series. We refer the reader to [12] for further details.

3.3. The filter phase

First step of the filter phase is the creation of M bit vectors of size N, one vector for every symbol in the alphabet. The vectors are created as follows: For every symbol in the alphabet, we create a vector where an ace will be present for every occurrence of the corresponding letter and a zero for every other letter.

The next step is to calculate the circular autocorrelation function for every bit vector. The term autocorrelation [15] means self-correlation, i.e., discovering correlations among the elements of the same vector. It refers to the correlation of a time series with its own past and future values. One of the purposes autocorrelation is used for is to detect non-randomness in time series data. We use autocorrelation as a tool to discover score estimates for every possible period length.

The computation of autocorrelation function is the sum of N dot products between the original signal and itself shifted every time by a lag k. In *circular* autocorrelation, one point, at the end of the series,

Table 1
Circular autocorrelation example

1 0 0 1 0 0	Lag = 0
1 0 0 1 0 0	Autocorrelation = 2
1 0 0 1 0 0	Lag = 1
0 1 0 0 1 0	Autocorrelation = 0
1 0 0 1 0 0	Lag = 2
0 0 1 0 0 1	Autocorrelation = 0
1 0 0 1 0 0	Lag = 3
1 0 0 1 0 0	Autocorrelation = 2

is shifted out of the product in every step and is moved to the beginning of the shifting vector. Hence in every step we compute the following product, for all N points:

$$R_{xx}(k) = \frac{1}{N-k} \sum_{n=1}^{N-k} x(n)x((n+k) \bmod N) \tag{1}$$

The value of the autocorrelation function at lag 0 is the power of $x(n)$:

$$R_{xx}(0) = \frac{1}{N} \sum_{i=1}^{N} x(n)^2 \tag{2}$$

This convolution-like formula calculates the discrete 1D circular autocorrelation function for a lag k. For our purposes we need to calculate the value of this function for all lags, that is, for N lags. Therefore, Equation (1) is computed for all $k = 1, \ldots, N$. The complexity of this operation is $O(N^2)$, which is quite expensive. Utilizing the Fast Fourier Transform (FFT) effectively reduces the cost down to *O*(*NlogN*), as follows:

$$f(x) = \stackrel{FFT}{\longrightarrow} F(x) \rightarrow R(F(x)) = \frac{1}{N} F(x)^* \bar{F}(x) \stackrel{IFFT}{\longrightarrow} r(f(x)) \tag{3}$$

In the above formula is the dot product of $F(x)^* \bar{F}(x)$ with its complex conjugate. The mathematical proof can be found in the bibliography. The following table shows how circular autocorrelation works. When the periodic aces are aligned, they are added, resulting in a large autocorrelation value.

Example 1. Consider the series ***a****bcd****a****be****b****a****d****f****c****a****c****d****c****f****c****a****a*** of length 20, where a is periodic with a period of 4 and a confidence of 3/4. We create the bit vector 10001000100010000011. The autocorrelation of this vector is given in Fig. 1.

The first value of the autocorrelation vector is the dot product of the bit vector with itself, since the shifting lag is 0 and therefore the two vectors align perfectly. Thus, the resulting value is the total number of aces, which is the total number of occurrences of the letter a. The peak identified in the above chart at position 4 implies that there is probably a period of length 4 and the value of 3 at this position is an estimate of the frequency count for this period. According to this observation, we can extract those peaks, hence acquiring a set of candidate periods. Notice that a period of length 4 also results in peaks at positions 4, 8, 12 and 16.

The user can specify a minimum confidence threshold c and the algorithm will simply extract those autocorrelation values that are greater than or equal to cN/p, where p is the current position where a period could exist.

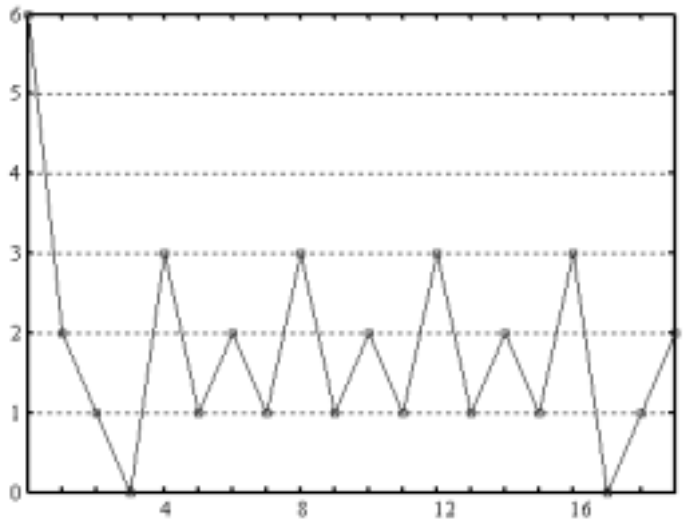


Fig. 1. Circular autocorrelation function when the length is a multiple of the period.

3.4. Experimental results

We tested our algorithm over a number of data sets. The most interesting data sets we used were supermarket and power consumption data. The former contain sanitized data of timed sales transactions for some Wal-Mart stores over a period of 15 months. The latter contain power consumption rates of some customers over a period of one year and were made available through a funded project. Synthetic control data taken from the Machine Learning Repository [17] were also used. Different runs over different portions of the data sets showed that the execution time is almost linearly proportional to the size of the time series as well as the size of the alphabet. Figure 2(a) shows the behavior of the algorithm against the number of the time points in the time series.

Figure 2(b) shows that the algorithm speeds up linearly to alphabets of different size. The size of the alphabet implies the number FFT computations of size N required. The times shown on the chart below correspond to a synthetic control data set of $N = 524288$ time points.

Experiments have confirmed our expectation regarding the completeness of PPD. In three datasets containing the number of customers per hour in three Wal-Mart stores, the algorithm returned the period that is most likely to be correct. Alternatively, instead of searching for a single candidate period, we could mine for a larger set of candidates. Table 2(a) summarizes the results. The "ACF" column is the Autocorrelation estimate produced for the periodic occurrences of a letter, while the "Freq." column is the number of occurrences of each letter. Notice that for most letters in all three datasets the suggested period is 24 or a multiple of it (e.g. 168, 336). Table 2(b) contains the patterns produced by Max Subpattern Hit algorithm [12] for a period length of 24.

3.5. Performance discussion

One of the most important issues one has to overcome when dealing with real world data is the inevitable presence of noise. The computation of the autocorrelation function over bit vectors eliminates

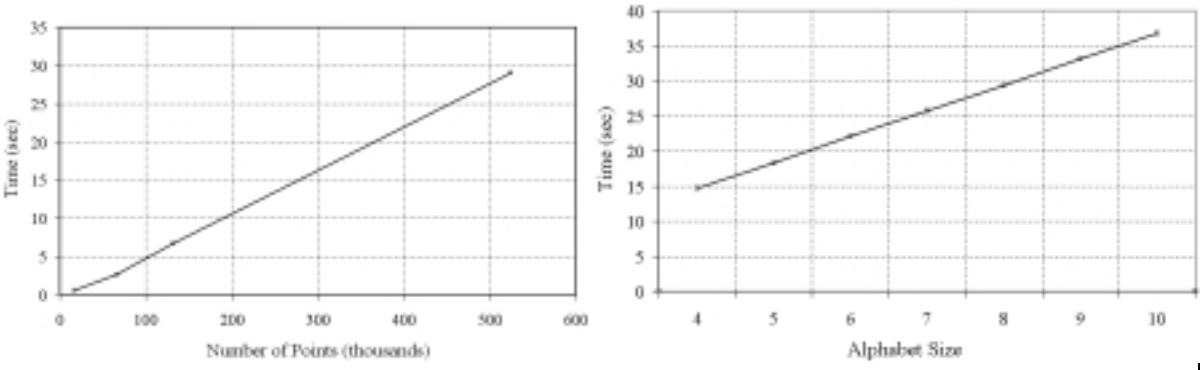
Table 2
(a) Results for the Wal-Mart stores. (b) Verification with Han's algorithm

(a)

Data	Symbols	Period	ACF	Freq.
Store 1	A	24	228	3532
	B	168	1140	2272
	C	24	94	1774
	D	336	648	874
	E	504	2782	2492
	F	4105	81	48
Store 2	A	24	252	3760
	B	168	1750	2872
	C	168	936	2199
	D	168	851	2093
	E	1176	90	140
Store 3	A	168	2034	3920
	B	168	1436	2331
	C	168	950	2305
	D	336	434	655
	E	24	99	1830
	F	–	–	23

(b)

Pattern	Conf.
AAAAAABBBB**********B*A	62.4
AAAAAA**BB**********AA	72.6
AAAAAA***BC**********AA	60.9
AAAAAA***B**********AA	75.7
AAAAAA*BB**********BAA	63.3
AAAAAA*BBB**********AA	60.9
AAAAAABBB**********BAA	61.3
AAAAAABBB**********B*A	69.6
AAAAAABBB**********AA	65.7

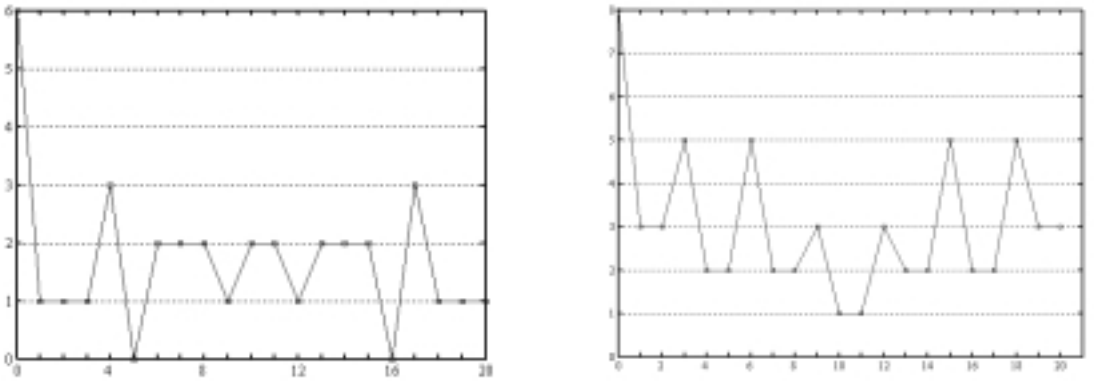


(a) (b)

Fig. 2. Run time against data sets of different size.

a large number of non-periodic aces due to their multiplication with zeroes, and hence leaving the periodic aces basically to contribute to the resulting value. Otherwise, using autocorrelation over the original signal, would cause all the non-periodic instances to contribute into a totally unreliable score estimate. Consequently, this value could be an acceptable estimate of the frequency count of a period. Note that the value of the estimate can never be smaller than the real one. Therefore, all the valid periodicities will be included in the candidate set together with a number of false ones that are the effect of the accumulation of random, non-periodic occurrences along with the periodic ones.

One major weakness of circular autocorrelation is that when the length of the series is not an integer multiple of the period, the circularly shifting mechanism results in vectors with a higher occurrence of unexpected values. This is usually increased by the randomness of real world data and the presence of noise. In our example the length of the series is $N = 20$, which is an integer multiple of the period $p =$



(a) (b)

Fig. 3. (a) Circular Autocorrelation Function when the length is not a multiple of the period. (b) Circular Autocorrelation Function when successive occurrences of a letter are repeated periodically.

4. When the length of the series is 21 (e.g., by adding a zero at the end of the bit vector), this results in the circular autocorrelation given in Fig. 3(a).

Another problem could arise when a number of successive occurrences of a letter are repeated periodically. For example the periodic repetition of aa^* would result in an unusually high autocorrelation value. Consider the series *aabaacaadacdbdbdabcdb*, where aa^* is repeated in 3 out of 7 periodic segments, while a^{**} is repeated in 4 periodic segments. *The* circular autocorrelation chart for the symbol a is given in Fig. 3(b). A peak (a frequency value greater than the minimum user-specified threshold) at position 4 can be seen, implying the existence of a period of 3. The frequency estimate according to the autocorrelation function is 6, which happens to be two times the actual frequency count, which is 3. This effect has been discussed in [23] by Hoeppner et al. where the authors suggest another way for the calculation of the confidence.

Repeating the algorithm described so far, for every symbol in the alphabet of our time series will result in a set of possible periods for each one of them. Note that a letter can have more than one period. For every candidate period, there will be an estimate of its confidence, according to their autocorrelation value. Utilizing the Apriori property on periodicity discussed earlier in this article, we can create periodicity groups, that is, groups of letters that have the same period. Han's algorithm [12] can be applied to verify the valid periods and extract the periodic patterns.

Lemma: Consider a time series with N points. Also let a letter x of that time series feature periodicity with a period p_1 with a confidence c_1. We can prove that x is also periodic with a period of p_2 and confidence $c_2 \geqslant c_1$, when p_2 is a multiple of p_1.

For example, if a is periodic with a period length of 4 and a confidence of 75% then it is also periodic with a period of 8, 12, 16 etc. and the corresponding confidence measures are equal to or greater than 0.75. Assume that b is periodic with a period of 8. Based on the previous lemma we know that a is also periodic with a period of 8 and therefore, we can create a periodicity group consisting of those two letters and apply Han's algorithm to check whether there is a periodic pattern with a period of 8 or any of its multiples.

Our algorithm requires 1 scan over the database in order for the bit vectors to be created. Then it runs in *O*(*NlogN*) time for every symbol in the alphabet of the series. Consequently, the total run time

is *MxO*(*NlogN*) and it depends on the size of the alphabet M. Generally speaking, we can say that this number is usually relatively small since it is a number of *user specified classes* in order to divide a range of continuous values. Despite the fact that some non-periodic peaks might occur, the method we propose is complete since all valid periods are extracted.

3.6. The refine phase

The major drawback of the filter phase of our method is the large number of candidate periods produced. Due to the reasons explained in the previous sections, a large number of "ghost" periods may feature high frequency estimates, often making it hard for the user to identify the valid ones. We provide a simple but effective solution to this problem, through a random iterative process. Randomly we select a small segment of the time series, e.g. of size *N/10*. Then we apply the algorithm described in the filter phase, extracting the periods for this segment only. We repeat the process a few more times for other segments of the same size. The periods that "survive" in most of the segments of this process are the winners of the refine phase.

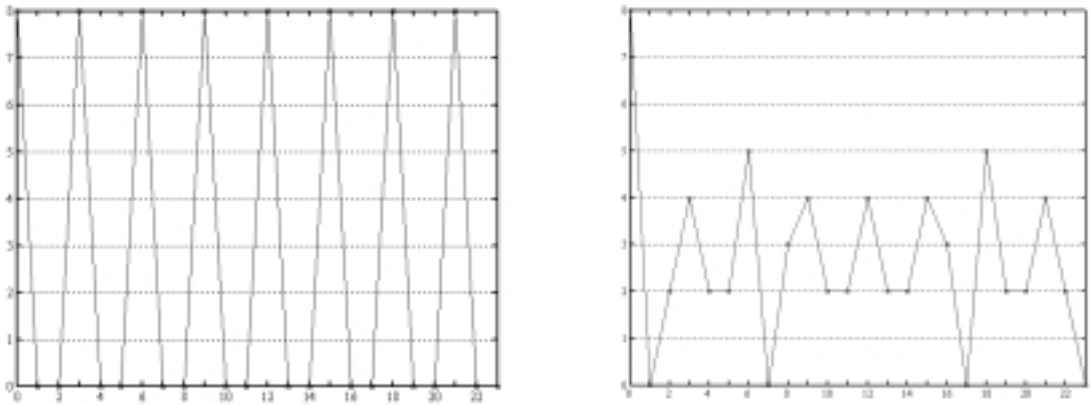
The idea behind it is that "ghost" periods are not likely to occur regularly and frequently throughout the entire series like the valid ones. Moreover, this way the vast majority of non-periodic occurrences of a symbol are excluded, critically increasing the reliability of the result produced for a single segment. The periods that do not regularly appear in most test segments aren't likely to be valid. The multiple execution of the algorithm over different segments is a simple but efficient way to dramatically eliminate the "ghost" periods effect produced by the non-periodic occurrences of a symbol. The number of executions depends on the strength of the periodicities we are mining for. If the user-specified confidence threshold is low, i.e. then it is possible that a weak period (with confidence close to the threshold) may not appear in some test segments. In this case we need to test more segments to make sure we will not miss a possible winner. Apparently, the procedure is semi-automatic, which means that it requires the user's intervention and guidance.

4. Approximate periodicities

We define approximate periodicity as a periodicity, some periodic instances of which, might be shifted a user-limited number of time points before or after their expected periodic occurrence. Normally, these instances would be considered missing and therefore this would be a loser kind of periodicity. Capturing those instances is a particularly interesting task that provides us with useful information regarding the strength of a periodicity. We try to capture those "shifted" occurrences in terms of frequency estimate. In other words, we use the autocorrelation function over the vectors of the occurrences of a letter, as a means in order to acquire a rough estimate of the strength of a periodicity. We call our algorithm APPD, which stands for Approximate Periodicity Detection.

4.1. The Approximate Periodicity Detection (*APPD*) *algorithm*

Our approach on approximate periodicities is an extension to PPD. At the preprocessing stage, we assume that all the occurrences of a letter could be part of a periodicity and that they might be shifted. Every such occurrence is represented in a bit vector by an ace. By replacing zeroes around every ace with values in the range between 0 and 1, we attempt to capture all these possible shiftings. Consider the following example:



(a) (b)

Fig. 4. Autocorrelation of vectors u and v.

Example 3. Given the following bit vector of the occurrences of a letter in a time series: $u =$ [1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0], consisting of 24 points and featuring a perfect periodicity with period length 3, we shift the 3 last aces by 1 position before or after (arbitrarily), thus taking the following vector: $v =$ [1,0,0,1,0,0,1,0,0,1,0,0,1,0,1,0,0,0,1,0,0,0,1, 0]. The autocorrelation function of vectors u and v are shown in the following figures.

In the charts above we can clearly see the change in the autocorrelation values. In the second vector period 3 is a loser with a score of 4 while the winner period is 6 with a score of 5. However, both periods imply that the dominant period in this series is 3 and some instances of it are just not perfectly accurate. In real world data, where randomness and noise is always present, such effects are usually expected, while perfectly distributed periodic instances are not very likely to occur. Changing the two zeroes, before and after every ace, to 0.5 we make them contribute to the accuracy of the estimate of the periodicity, implying thus that there is a 50% probability that every ace's regular occurrence might be one time point before or one after.

The above chart shows that the autocorrelation value at position 3 is now 10, denoting that the implied periodicity might actually be stronger than the one implied be the autocorrelation of v. Additionally, we can insert values other than 0.5 before and after the aces, depending whether one wants to increase the probability, and therefore the contribution, of the possibly shifted aces. It is totally up to the user or the domain expert to alter this according to his knowledge about the nature of the data. We would like to clarify that we do not assume that the probability distribution of the data is known. APPD can easily be used to assist periodicity mining with PPD when one suspects that there might be some shifted periodicities. Furthermore one can also increase the area around every ace to be covered with values between 0 and 1. Replacing zeroes around an ace like [0.2, 0.6, 1, 0.6, 0.2] would be similar to using a triangular membership function in a fuzzification process. The main advantage is that the computational cost of our approach is much smaller than the one of a fuzzy algorithm.

Finally, it should be stressed that the estimate provided by APPD is a reliable indication of the strength of a periodicity, and not a frequency estimate, like the one produced by PPD. The non-zero entries inserted here result in a more blurred output, which is not evidence but a serious hint that could provide the user with useful insight about the data. One should combine the two methods in order to mine for

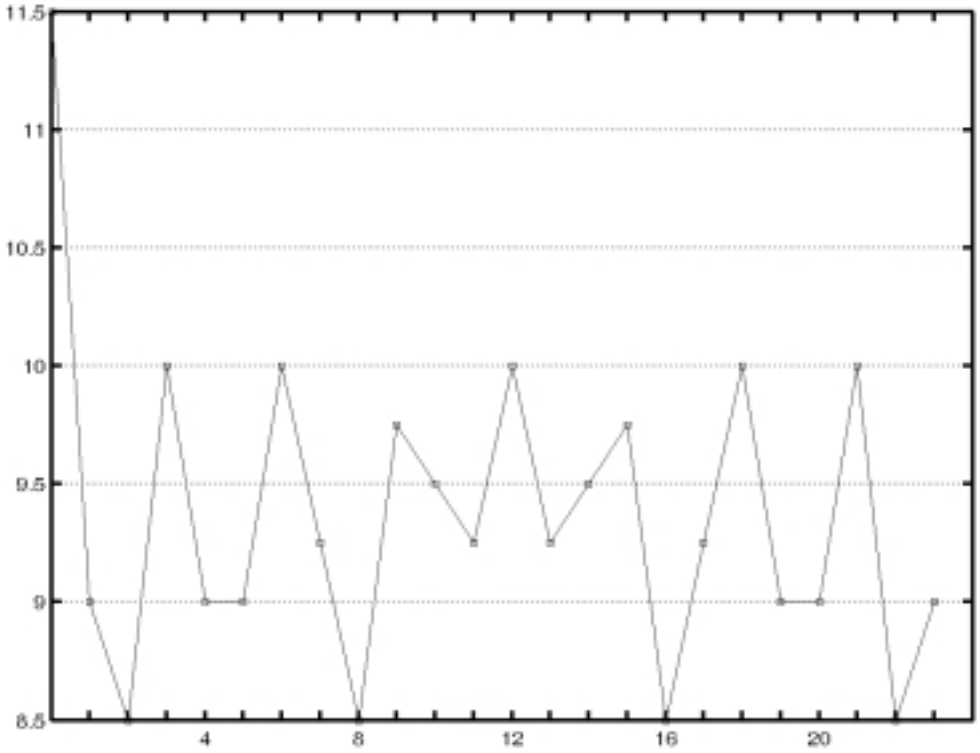


Fig. 5. Autocorrelation of vector w.

weak periodicities in a time series. If the increase of the autocorrelation value is significant then it is highly possible that its actual confidence is greater than the one produced by the first method.

APPD's computational complexity is exactly the same as PPD's. It engages at the preprocessing stage, during the first scan of the data, when the bit vectors are created. One can create both sets of vectors during the same scan and then run the autocorrelation step twice, avoiding thus another scan over the data on the disk.

4.2. Experimental results

We applied our method on 3 power demand datasets. The first two datasets (Customer 1 and Customer 2) are taken from the CIMEG project[1] and contain hourly measurements of power consumption of American households in the greater area of Chicago. The source of the third dataset is the UCR Time Series Data Mining Archive [21]. The file contains the 15 minutes averaged values of power demand for a research center in Eindhoven, Netherlands, for the full year 1997. The first value is at 00.15 am, January 1st, 1997. The datasets were transformed into character sequences through a standard discretization

[1] CIMEG: Consortium for the Intelligent Management of the Electric Power Grid. hlios.ecn.purdue.edu/_cimeg.

Table 3
Experimental results for APPD

		Customer 1			Customer 2			Power Data		
	Symbol	Freq.	Period	ACF	Freq.	Period	ACF	Freq.	Period	ACF
(PPD)	A	2485	25	1629.28	3760	336 (1w)	11714	672	72 (18h)	1013.77
	B	5302	3	4060.78	2872	48	7370	10880	4 (1h)	30275.5
	C	787	23	256.79	2199	48	5125	2126	1344 (2w)	4443.90
	D	141	334 (1w)	23.60	2093	48	5106	3370	4032 (6w)	9467.17
	E	39	–	–	140	336 (1w)	143.79	472	1344 (2w)	722.87
	F	7	–	–						
(APPD)	A	2485	24	905.21	3760	336 (1w)	23494	672	80 (20h)	2416.60
	B	5302	120 (5d)	1939.87	2872	24	9218.1	10880	4 (1h)	66428.9
	C	787	48	170.77	2199	336 (1w)	11496	2126	1344 (2w)	10054.2
	D	141	458 (19d)	15.57	2093	48	10990	3370	5376 (2m)	22741.8
	E	39	–	–	140	336 (1w)	365.5	472	1344 (2w)	1679.62
	F	7	–	–						

process; every character corresponds to a consumption range (e.g. $a =$ high, $b =$ medium, $c =$ low). In all tests, for uniformity, the confidence threshold was set to 0.7 (70%). The results are displayed in the tables below and are indicative of a series of tests we conducted. Note that for each symbol, only the periodicity with the highest score is displayed.

Experiments have confirmed our expectation regarding the completeness of our approach. Below, Table 3 summarizes the results. The "ACF" column is the autocorrelation function estimate produced for the periodic occurrences of a letter, while the "Freq." column is the number of occurrences of each letter. Both algorithms successfully detected the expected daily (24-hour) periodicity and the weekly (7-day) periodicity of all three datasets. Note that in most cases APPD verified the periods found by PPD although in some cases there were some "corrections", that didn't actually change the daily or weekly cycle of the corresponding symbol. Also note that for the "Customer 2" and "Power Data" datasets, there is not an F symbol, while for the "Customer 1" data set, symbols E and F feature no period at all.

5. Conclusions and further work

In this paper we present a method for efficiently discovering weak (partial and approximate) periodicities in large time series. We propose an algorithm based on the use of the circular autocorrelation function over bit vectors. It consists of two phases: a filter and a refine phase. In the filter phase our algorithm discovers a set of candidate periods for every symbol in the time series, according to a user specified confidence threshold, without any previous knowledge of the data along with an acceptable estimate of the strength of a candidate periodicity.

It is useful when dealing with data whose period is not known or when mining for unexpected periodicities. Algorithms such [12] can be used to extract the periodic patterns. We tried our method against various data sets and it proved to scale up linearly against different alphabets and different numbers of time points. We also verified its expected completeness using the Max Subpattern Hit algorithm.

We also proposed a method for capturing approximate periodicities in a time series. Our algorithm (APPD) is an extension to the partial periodicity detection algorithm (PPD), at the preprocessing stage. We provide the user with a reliable strength indication for approximate periodicities. Its usefulness lies on the fact that in real world data several instances of a periodic pattern or symbol might not be accurately

distributed over the time series. It adds no computational overhead to the previous algorithm, since it can be integrated into the first scan of the data, at the preprocessing stage.

Finally at the refine phase, the candidate period set produced by the filter phase algorithm is dramatically reduced. We apply the same period extraction algorithm on random segments of the time series. As a result, the vast majority, if not all, of the ghost periods are eliminated.

We implemented and tested our algorithm using a main memory FFT algorithm, which is more than enough for time series of up to a billion points, however, a disk-based FFT algorithm [20,21] would be more appropriate for handling larger time series that do not fit in the main memory. Interesting extension of our work would be the development of an algorithm to perform over other kinds of temporal data such as distributed.

Acknowledgements

The authors would like to thank the anonymous referees for their valuable comments and suggestions on this work.

References

[1] D. Hand and H. Mannila, *Padhraic Smyth*, Principles of Data Mining, The MIT Press, 2001.
[2] R. Agrawal, C. Faloutsos and A. Swami, *Efficient Similarity Search in Sequence Databases*, In Proc. of the 4th Int. Conf. on Foundations of Data Organization and Algorithms, Chicago, Illinois, October, 1993.
[3] R. Agrawal, K. Lin, H.S. Sawhney and K. Shim, *Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases*, In Proc. of the 21st Int. Conf. on Very Large Databases, Zurich, Switzerland, September, 1995.
[4] R. Agrawal and R. Srikant, *Mining Sequential Patterns*, In Proc. of 1995 Int. Conf. on Data Engineering, Taipei, Taiwan, March, 1995.
[5] M. Garofalakis, R. Rastogi and K. Shim, Mining Sequential Patterns with Regular Expression Constraints, *IEEE Transaction on Knowledge and Data Engineering* (*TKDE*) (2002), 530–552.
[6] R. Agrawal, G. Psaila, E.L. Wimmers and M. Zaït, *Querying Shapes of Histories*, Proc. 21st Int'l Conf. Very Large Data Bases (VLDB95), Sept., 1995.
[7] H. Mannila, H Toivonen and A.I. Verkamo, *Discovering Frequent Episodes in Sequences*, In Proc. of the 1st Int. Conf. on Knowledge Discovery and Data Mining, Montreal, Canada, August, 1995.
[8] K. Chan and A. Fu, *Efficient Time-Series Matching by Wavelets*, In Proc. of 1999 Int. Conf. on Data Engineering, Sydney, Australia, March, 1999.
[9] C. Faloutsos, M. Ranganathan and Y. Manolopoulos, *Fast Subsequence Matching in Time-Series Databases*, In Proc. of the 1994 ACM SIGMOD Int. Conf. on Management of Data, Minneapolis, Minnesota, May, 1994.
[10] E. Keogh, K. Chakrabarti, M. Pazzani and S. Mehrotra, Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases, *Springer-Verlag, Knowledge and Information Systems* (2001), 263–286.
[11] H. Toroslu and M. Kantarcioglu, Mining Cyclically Repeated Patterns, *Springer Lecture Notes in Computer Science* **2114** (2001), 83.
[12] J. Han, G. Dong and Y. Yin, *Efficient Mining of Partial Periodic Patterns in Time Series Databases*, In Proc. of 1999 Int. Conf. on Data Engineering, Sydney, Australia, March, 1999.
[13] W. Wang, J. Yang and P. Yu, *Meta-patterns: revealing hidden periodical patterns*, Proceedings of the 1st IEEE International Conference on Data Mining (ICDM), 2001, pp. 550–557.
[14] B. George and J. Gwilym, *Time series analysis: forecasting and control*, San Francisco (etc.): Holden-Day, 1976.
[15] C. Bettini, S. Wang, S. Jajodia and J. Lin, Discovering frequent event patterns with multiple granularities in time sequences, *IEEE Transactions on Knowledge and Data Engineering* **10**(2) (March/April 1998), 222–237.
[16] J.A. Orenstein, *Redundancy in Spatial Databases*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Portland, USA, 1989, pp. 294–305.
[17] C.L. Blake and C.J. Merz, *UCI Repository of Machine Learning Databases, University of California*, Department of Information and Computer Science, 1998.
[18] C. Berberidis, I. Vlahavas, W. Aref, M. Atallah and A. Elmagarmid, *On the Discovery of Weak Periodicities in Large Time Series*, Proc. 6th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD '02), Springer-Verlag, LNAI 2431, 2002, pp. 51–61.

[19] Numerical Recipes in C: The Art of Scientific Computing. External Storage or Memory-Local FFTs, Copyright 1988–1992 by Cambridge University Press, pp. 532–536.
[20] J.S. Vitter, External memory algorithms and data structures: Dealing with massive data, *ACM Computing Surveys* **33**(2) (June, 2001).
[21] E. Keogh and T. Folias, The UCR Time Series Data Mining Archive [www.cs.ucr.edu/∼eamonn/TSDMA/index.html]. Riverside CA. University of California – Computer Science & Engineering Department, 2002.
[22] J. Yang, W. Wang and P. Yu, *Mining asynchronous periodic patterns in time series data*, Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2000, pp. 275–279.
[23] F. Höppner and F. Klawonn, *Finding Informative Rules in Interval Sequences*, Advances in Intelligent Data Analysis, Proc. of the 4th International Symposium, Lecture Notes in Computer Sciences 2189, Springer. Lissabon, Portugal, Sept. 2001, pp. 123–132.