On the Discovery of Weak Periodicities in Large Time Series

Christos Berberidis[1], Ioannis Vlahavas[1], Walid G. Aref [2], Mikhail Atallah[2,*], and Ahmed K. Elmagarmid[2]

[1]Department of Informatics, Aristotle University of Thessaloniki
54006 Thessaloniki Greece
{berber,vlahavas}@csd.auth.gr
[2]Dept. of Computer Sciences, Purdue University
{aref,mja,ake}@cs.purdue.edu

Abstract. The search for weak periodic signals in time series data is an active topic of research. Given the fact that rarely a real world dataset is perfectly periodic, this paper approaches this problem in terms of data mining, trying to discover weak periodic signals in time series databases, when no period length is known in advance. In existing time series mining algorithms, the period length is user-specified. We propose an algorithm for finding approximate periodicities in large time series data, utilizing autocorrelation function and FFT. This algorithm is an extension to the partial periodicity detection algorithm presented in a previous paper of ours. We provide some mathematical background as well as experimental results.

1 Introduction

Periodicity is a particularly interesting feature that could be used for understanding time series data and predicting future trends. However, little attention has been paid on the study of the periodic behavior of a temporal attribute. 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 periodic over some but not all the points in it. An interesting extension of the problem of capturing all kinds of periodicities that might occur in real world time series data is the discovery of approximate periodicities. That is, periodicities where a small number of occurrences are not 100% punctual.

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 simi-

* Portions of this work were supported by Grant EIA-9903545 from the National Science Foundation, Contract N00014-02-1-0364 from the Office of Naval Research, and by sponsors of the Center for Education and Research in Information Assurance and Security.

T. Elomaa et al. (Eds.): PKDD, LNAI 2431, pp. 51-61, 2002.

larity search in time series databases [1, 2, 3]. Mannila et al. [4] introduce an efficient solution to the discovery of frequent patterns in a sequence database. Chan et al. [5] study the use of wavelets in time series matching and Faloutsos et al. in [6] and Keogh et al. in [7] propose indexing methods for fast sequence matching using R* trees, the Discrete Fourier Transform and the Discrete Wavelet Transform. Toroslu et al. [8] introduce the problem of mining cyclically repeated patterns. Han et al. [9] 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. Aref et al. in [10] extend this work by introducing algorithms for incremental, on-line and merge mining of partial periodic patterns.

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. Berberidis et al. [13] propose an algorithm for detecting the period when searching for multiple and partial periodic patterns in large time series.

In this paper we attempt to detect weak periodic signals in large, real world time series. By "weak periodic signals" we mean partial and approximate periodicities. We introduce the notion of approximate periodicities, which is the case when some periodic instances of a symbol might be appearing a number of time points before or after their expected periodic occurrence. Our work extends the algorithm introduced in [13], for discovering multiple and partial periodicities, without any previous knowledge of the nature of the data. We use discretization to reduce the cardinality of our data. The time series is divided into a number of intervals and a letter is assigned to each interval. Thus, the original time series is transformed into a character sequence. The algorithm follows a filter-refine paradigm. In the filter step, the algorithm utilizes the *Fast Fourier Transform* to compute a *Circular Autocorrelation Function* that provides us with a conservative set of candidate period lengths for every letter in the alphabet of our time series. In the refine step, we apply Han's algorithm [9] for each candidate period length. The complexity of our algorithm is $O(AN\log N)$, where A is the size of the alphabet and N the size of the time series. The algorithm speeds up linearly both to the number of time points and the size of the alphabet.

The rest of this paper proceeds as follows: the next section contains notation and definitions for the problem. In section 3 we outline the steps of the algorithm we propose for discovering partial periodicities and we explain how it works in detail. We provide some theoretical background and we discuss the computational complexity of the algorithm. We test our algorithm with various data sets, produce some experimental results and verify them using Han's algorithm. In section 4 we discuss an extension to the partial periodicity algorithm of section 3, for finding approximate periodicities. In the last section we conclude this paper and suggest some directions for further research.

2 Notation

A **pattern** is a string $s = s_1 \ldots s_p$ over an **alphabet** $L \cup \{*\}$, where the letter * stands for *any single symbol from L*. A pattern $s' = s_1' \ldots s_p'$ 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 [9] "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 Discovering Partial Periodicities – The PPD Algorithm

Based on the Apriori property described in the previous section, we present the algorithm we proposed in [13], that generates a set of candidate periods for the symbols of a time series. We call this algorithm PPD, which stands for Partial Periodicity Detection.

The *filter/refine paradigm* is a technique that has been used in several contexts, e.g., in spatial query processing [11]. The filter phase reduces the search space by eliminating those 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. We use the *circular autocorrelation function* as a tool to filter out those periods that are definitely not valid.

We outline the major steps performed by our algorithm. The explanation of the steps is given further down in this section.

- Scan the time series once and create a binary vector of size N for every symbol in the alphabet of the time series.
- For each symbol of the alphabet, compute the circular autocorrelation function vector over the corresponding binary vector. This operation results in an output autocorrelation vector that contains frequency counts.
- Scan only half the autocorrelation vector (maximum possible period is N/2) and filter out those values that do not satisfy the minimum confidence threshold and keep the rest as candidate periods.

- Apply Han's algorithm to discover periodic patterns for the candidate periods produced in the previous step.

Steps 1—3 correspond to the filter phase while Step 4 corresponds to the refine phase, which uses Han's Max-subpattern Hit Set Algorithm that mines for partial periodic patterns in a time series database. It builds a tree, called 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. For brevity, we refer the reader to [9] for further details.

3.1 The Filter Step

The first step of our method is the creation of a number of binary vectors. Assume we have a time series of size *N*. We create a binary vector of size *N* for every letter in our alphabet. 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 binary vector. The term autocorrelation means self-correlation, i.e., discovering correlations among the elements of the same vector. We use Autocorrelation as a tool to discover 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, 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 dot product, for all *N* points:

$$r(k) = \frac{1}{N}\sum_{x=1}^{N} f(x) f(x+k) \tag{1}$$

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 every lag, that is for *N* lags. Therefore, (1) is computed for all *k=1 ... N*. The complexity of this operation is $O(N^2)$, which is quite expensive, especially when dealing with very large time series. Utilizing the Fast Fourier Transform (FFT) effectively reduces the cost down to $O(N\log N)$, as follows:

$$f(x) \xrightarrow{FFT} F(x) \longrightarrow R(F(x)) = \frac{1}{N} F(x) * \overline{F}(x) \xrightarrow{IFFT} r(f(x)) \tag{2}$$

In the above formula $F(x) * \overline{F}(x)$ is the dot product of *F(x)* with its complex conjugate. The mathematical proof can be found in the bibliography.

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

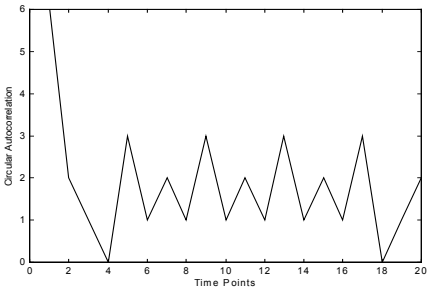

Fig. 1. Circular Autocorrelation Function when the length is a multiple of the period

The first value of the autocorrelation vector is the dot product of the binary 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 5 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 of 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 5, 9, 13 etc.

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.

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 binary vectors eliminates 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 with the periodic ones.

One major weakness of the 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=4*. When the length of the series is 21 (e.g., by adding a zero at the end of the binary vector), this results in the circular autocorrelation given in Figure 2.

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 *aabaacaadacdbdbdabc*, where *aa** is repeated in 3 out of 6 periodic segments, while *a*** is repeated in 4 periodic segments. *The* circular autocorrelation chart for the symbol *a* is given in

Figure 2b. A clear peak 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.

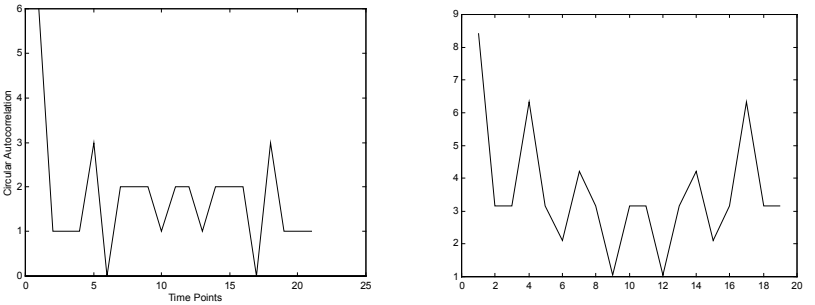

Fig. 2. (a) Circular Autocorrelation Function when the length is not a multiple of the period. (b) Circular Autocorrelation Function when successive occurrences of a letter are repeated periodically

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 [9] can be applied to verify the valid periods and extract the periodic patterns.

Theorem: Consider a time series with N points. Also let a letter of that time series feature periodicity with a period p_1 with a confidence c_1. We can prove that this letter is also periodic with a period of p_2 and confidence $c_2 \geq c_1$, 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 theorem 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.

3.2 Analysis

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

3.3 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 [12] were also used. Different runs over different portions of the data sets showed that the execution time is linearly proportional to the size of the time series as well as the size of the alphabet. Figure 3a shows the behavior of the algorithm against the number of the time points in the time series.

Figure 3b 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.

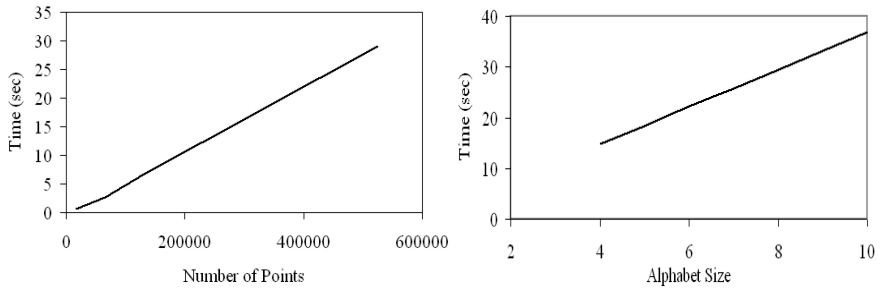

Fig. 3. Run time against data sets of different size

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 1a 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 1b contains the patterns produced by Han's algorithm for a period length of 24.

4 Capturing Approximate Periodicities – The APPD Algorithm

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 periodicity would be considered partial. 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 binary vectors of the occurrences of a letter, as a means in order to acquire a reliable measure of the strength of a periodicity. We call our algorithm APPD, which stands for Approximate Periodicity Detection.

Table 1. (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

Our approach 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 binary 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:

Example 3. Given the following binary vector of the occurrences of a letter in a time series: u = [1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0], consisting of 44 points and featuring a perfect periodicity with period length 4, we shift the 3 last aces by 1 position before or after (arbitrarily), thus taking the following vector: v = [1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0]. The autocorrelation function of vectors u and v are shown in the following figures.

The autocorrelation value of 11 at position 5 of the first vector implies a periodicity of length 4. Shifting 3 aces by 1 position, results in an autocorrelation value of 8 at position 5. Thus, those 3 aces were not considered at all. In real world data, where randomness and noise is always present, such effects are usually expected, while perfectly distributed periodic instances are quite unlikely 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 natural position might be the one before or the one after.

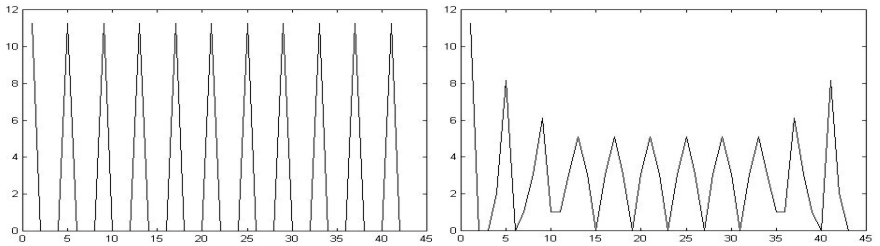

Fig. 4. Autocorrelation of vectors *u* and *v*

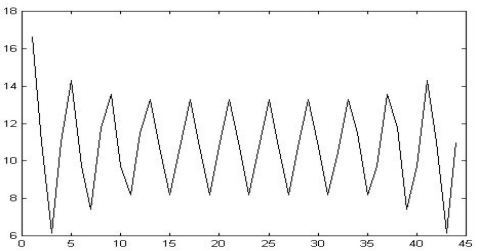

Fig. 5. Autocorrelation of vector w

The above chart shows that the autocorrelation value at position 5 is now 14.3, 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. 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, we should make clear 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. It 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 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 binary 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.

5 Conclusions and Further Work

In this paper we presented a method for efficiently discovering a set of candidate periods in a large time series. Our algorithm can be used as a filter to discover the candidate periods without any previous knowledge of the data along with an acceptable estimate of the confidence of a candidate periodicity. It is useful when dealing with data whose period is not known or when mining for unexpected periodicities. Algorithms such as Han's described in [9] can be used to extract the periodic patterns. We tried our method against various data sets and it proved to speed up linearly against different alphabets and different numbers of time points. We also verified its expected completeness using Han's algorithm.

We also proposed a method for capturing approximate periodicities in a time series. Our method is an extension to the partial periodicity detection algorithm, at the preprocessing stage. We provide the user with a reliable strength measure 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.

We implemented and tested our algorithm using a main memory FFT algorithm, however, a disk-based FFT algorithm [14, 15] 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.

6 References

1. 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.
2. 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.
3. R. Agrawal and R. Srikant. Mining Sequential Patterns. In *Proc. of 1995 Int. Conf. on Data Engineering*, Taipei, Taiwan, March 1995.
4. 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.
5. K. Chan and A. Fu. Efficient Time-Series Matching by Wavelets. In *Proc. of 1999 Int. Conf. on Data Engineering*, Sydney, Australia, March 1999.
6. 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.

7. 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) p. 263–286.
8. H. Toroslu and M. Kantarcioglu. Mining Cyclically Repeated Patterns. *Springer Lecture Notes in Computer Science* 2114, p. 83 ff., 2001.
9. 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.
10. W. G. Aref, M. G. Elfeky and A. K. Elmagarmid. Incremental, Online and Merge Mining of Partial Periodic Patterns in Time-Series Databases. Submitted for journal publication. Purdue Technical Report, 2001.
11. Orenstein J. A. Redundancy in Spatial Databases*, Proc. ACM SIGMOD Int. Conf. on Management of Data*, Portland, USA, 1989, pp. 294-305.
12. Blake, C. L. & Merz, C. J. (1998) UCI Repository of Machine Learning Databases, University of California, Department of Information and Computer Science.
13. Christos Berberidis, Aref G. Walid, Mikhail Atallah, Ioannis Vlahavas, Ahmed K. Elmagarmid, "Multiple and Partial Periodicity Mining in Time Series Databases", F. van Harmelen (ed.): ECAI 2002. Proceedings of the 15th European Conference on Artificial Intelligence, IOS Press, Amsterdam, 2002.
14. Numerical Recipes in C: The Art of Scientific Computing. External Storage or Memory-Local FFTs. pp 532-536. Copyright 1988-1992 by Cambridge University Press.
15. J. S. Vitter. External Memory Algorithms and Data Structures: Dealing with Massive Data. *ACM Computing Surveys, Vol. 33, No. 2, June 2001.*