CERIAS Tech Report 2002-53
Multiple and Partial Periodicity Mining in Time Series Databases
by Mikhail J. Atallah
Center for Education and Research
Information Assurance and Security
Purdue University, West Lafayette, IN 47907-2086

Multiple and Partial Periodicity Mining in Time Series Databases

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

Abstract. Periodicity search in time series is a problem that has been investigated by mathematicians in various areas, such as statistics, economics, and digital signal processing. For large databases of time series data, scalability becomes an issue that traditional techniques fail to address. In existing time series mining algorithms for detecting periodic patterns, the period length is user-specified. This is a drawback especially for datasets where no period length is known in advance. We propose an algorithm that extracts a set of candidate periods featured in a time series that satisfy a minimum confidence threshold, by utilizing the autocorrelation function and FFT as a filter. 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.

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 [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 Hit Set 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 partial 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.

Our contribution in this paper is a new algorithm for detecting all the periodicities in a time series without any previous knowledge of the nature of the data. The time series is considered as 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, the algorithm applies Han's algorithm [9] for each candidate period length to find partial periodic patterns, if any, within this 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 we contains notation and definitions for the problem. In section 3 we outline the steps of the algorithm we propose and we explain how it works in detail. We provide some theoretical background and we discuss the computational complexity of the algorithm. In section 4 we test our algorithm with various data sets, produce some experimental results and verify them using Han's algorithm. 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.

[1] Department of Informatics, Aristotle University of Thessaloniki, Thessaloniki 54006, Greece, email: {berber, vlahavas}@csd.auth.gr
[2] Department of Computation, Purdue University, West Lafayette, IN 47907, USA, email: {aref, mja, ake}@cs.purdue.edu

3 OUR APPROACH

Based on the Apriori property described in the previous section, we present a new algorithm that generates a set of candidate periods for the symbols of a time series. 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 fulfil 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.

1. Scan the time series once and create a binary vector of size N for every symbol in the alphabet of the time series.
2. 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.
3. 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.
4. 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 Phase

The first phase 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, the 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:

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

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, which means for *N* lags. Therefore, the overall formula for *r(k)* 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*)*. The overall procedure is depicted as follows:

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

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****beb****a****dfc****a****cdcfc****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 circular autocorrelation of this vector is given in Figure 1.

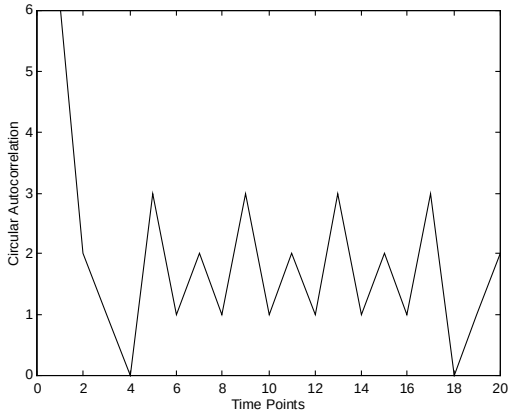

Figure 1. Circular Autocorrelation Function when the length is a multiple of the period

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 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, such a 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.

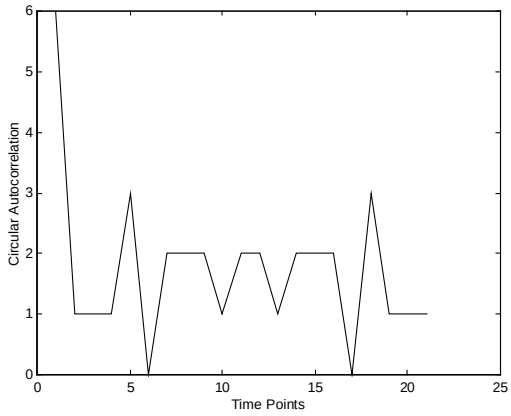

Figure 2. Circular Autocorrelation Function when the length is not a multiple of the period

Notice that although the chart is different, there is still a peak at position 5 implying the existence of a period of 4 with a frequency count of 3. Since the maximum period is theoretically equal to *N/2*, the peak at position 18 is ignored.

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 might 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.

A problem could arise when a number of successive occurrences of a letter are repeated periodically.

Example 2: 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 3.

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.

The presence of noise does not affect the completeness of the algorithm, since it discovers all the periodicities that exist. However, it might also produce some non-existing periodicities, which are pruned in the refine phase.

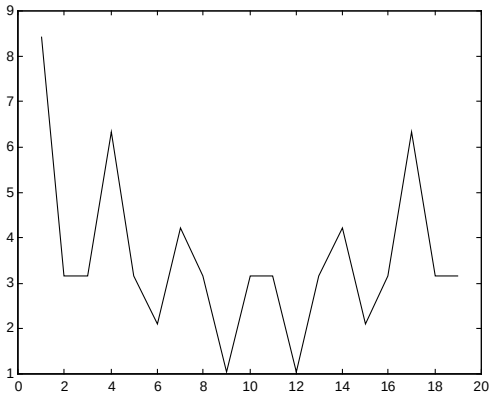

Figure 3. Circular Autocorrelation Function when successive periodic occurrences of the letter

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.

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 the CIMEG[1] project. Synthetic control data taken from the Machine Learning Repository [12] were also used. Different runs over

[1] CIMEG: Consortium for the Intelligent Management of the Electric Power Grid. helios.ecn.purdue.edu/~cimeg/Index.html.

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 4 shows the behavior of the algorithm against the number of the time points in the time series.

Figure 5 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.

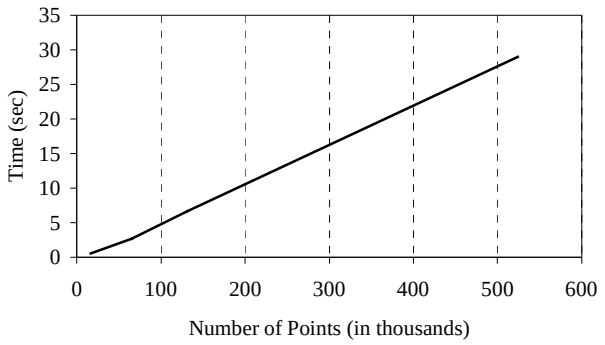

Figure 4. Run time against data sets of different size

.

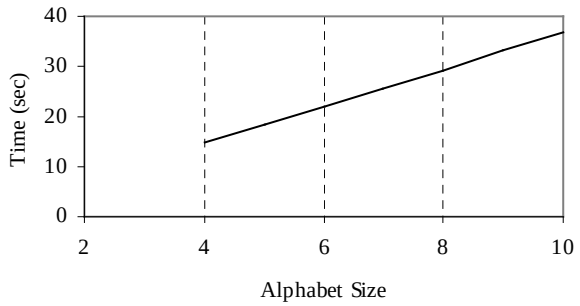

Figure 5. Run time against different alphabets

As far as accuracy is concerned, our algorithm was proved to be complete. We tried three datasets containing the number of customers per hour in three Wal-Mart stores. Letter A stands for nearly zero transactions per hour, while F stands for high number of transactions per hour. The algorithm returned the one 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 1 summarizes the results. The "ACF" column is the Autocorrelation estimate produced for the periodic occurrences of a letter, while the "Frequency" column is the number of occurrences of each letter in the Time Series. Notice that for most letters in all three datasets the suggested period is 24 or a multiple of it (e.g. 168, 336, 504).

Given the output of the filter step described in the previous experiment we tried to verify it using Han's algorithm for the extraction of partially periodic patterns. For reasons of brevity we provide periodic patterns produced by Han's algorithm only for the third data set of Table 1. We mined for patterns of period length 24 and we set the minimum confidence threshold to 60%. Alternatively, we could have mined for patterns with a period length of 168, however we chose to verify the daily cyclic behavior of these data. The results are displayed in Table 2, showing that 24 is a valid period length.

Table 1. Results for the 3 Wal-Mart stores

Data	Symbols	Period	ACF	Frequency
Store 1	A	24	227.88	3532
	B	168	1140.5	2272
	C	24	93.76	1774
	D	336	648.17	874
	E	504	2782.02	2492
	F	4105	81.61	48
Store 2	A	24	252.43	3760
	B	168	1750.37	2872
	C	168	936.36	2199
	D	168	851.93	2093
	E	1176	90	140
Store 3	A	168	2034.53	3920
	B	168	1436.71	2331
	C	168	950.68	2305
	D	336	434.05	655
	E	24	99.85	1830
	F	-	-	23

Table 2. Verification with Han's algorithm

Pattern	Confidence (%)
AAAAAABBBB**********B*A	62.47288
AAAAAA**BB**********AA	72.66811
AAAAAA***BC**********AA	60.95444
AAAAAA***B**********AA	75.70499
AAAAAA*BB**********BAA	63.34056
AAAAAA*BBB**********AA	60.95444
AAAAAABBB**********BAA	61.38828
AAAAAABBB**********B*A	69.63123
AAAAAABBB**********AA	65.72668
AAAAAABBBB**********B*A	62.47288

5 CONCLUSIONS AND FURTHER WORK

In this paper we proposed 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 looking for unexpected periodicities. Algorithms such as Han's described in [9] can be used to extract the 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 implemented and tested our algorithm using a main memory FFT algorithm, however, a disk-based FFT algorithm [13, 14] 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 and fuzzy. Finally, we intend to investigate the application of an algorithm or a function, other than the circular autocorrelation, that would require a smaller number of FFT computations.

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. www.ics.uci.edu/~mlearn/MLRepository.html. Irvine, CA: University of California, Department of Information and Computer Science.

[13] 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.

[14] J. S. Vitter. External Memory Algorithms and Data Structures: Dealing with Massive Data. *ACM Computing Surveys, Vol. 33, No. 2, June 2001.*