

# Efficient Evaluation of Composite Correlations for Streaming Time Series

Min Wang<sup>1</sup> and X. Sean Wang<sup>2</sup>

<sup>1</sup> Data Management Dept., IBM T. J. Watson Research Ctr., Hawthorne, NY 10532  
min@us.ibm.com

<sup>2</sup> ISE Department, George Mason University, Fairfax, Virginia 22030  
xywang@gmu.edu

**Abstract.** In applications ranging from stock trading to space mission operations, it is important to monitor the correlations among multiple streaming time series efficiently in order to make timely decisions. The challenge is that both the number of streaming time series and the number of interested correlations can be large. The straightforward way of performing the evaluation by computing the correlation value for each relevant stream pair at each time position is not efficient enough in many situations.

In this paper, we introduce an efficient method for the case where we need to monitor composite correlations, i.e., conjunctions of high correlations among multiple pairs of streaming time series. We use a simple mechanism to predict the correlation values of relevant stream pairs at the next time position and rank the stream pairs carefully so that the pairs that are likely to have low correlation values are evaluated first. We show, through experiments, that the method significantly reduces the total number of pairs for which we need to compute the correlation values due to the conjunctive nature of the composites.

## 1 Introduction

Many applications have a need to deal with multiple streaming time series. Examples of such applications include financial applications, network monitoring, security, telecommunications data management, and sensor network. In these applications, we are usually interested in monitoring the occurrences of strong correlations among pairs of streams efficiently in order to make timely decisions. The challenge is that both the number of streaming time series and the number of interested correlations can be large. The straightforward way of performing the evaluation by computing the correlation value for each relevant stream pair at each time position is not efficient enough in many situations.

In [16], Zhu and Shasha propose efficient methods for monitoring high correlations among all pairs of streams. Their methods are based on Discrete Fourier Transform (DFT) and a three-level time interval hierarchy and can monitor the pairwise correlation of thousands of data streams in real time. For example, one can use their methods for the following application:

**Example 1** There are 5,000 stocks trading in a stock market. Report which pairs of stocks were correlated with a correlation value of over 0.95 for the last hour.  $\square$

In this paper, we target at a different application scenario. Instead of monitoring high correlations among *all* pairs of streams, we assume we have knowledge of the interesting correlation patterns and need to monitor the occurrences of those interested patterns only. Such knowledge may be obtained from data mining tools on historical data. We consider the general case where the interested patterns are in the form of composite correlations, i.e., conjunctions of high correlations among multiple pairs of streaming time series. For example, our method can be used for the following application:

**Example 2** There are 5,000 stocks, denoted as  $\{s^1, s^2, \dots, s^{5000}\}$ , trading in a stock market. Give an alert, together with the corresponding pattern identifier, when any of the given patterns occurs. Two example patterns are as follows.

- $p_1$  For the last hour, stocks  $s^1$  and  $s^2$  were correlated with a correlation value above 0.85 *and* stocks  $s^1$  and  $s^3$  were correlated with a correlation value above 0.75.
- $p_2$  For the last hour, stocks  $s^4$  and  $s^5$  were correlated with a correlation value above 0.8 *and* stocks  $s^6$  and  $s^7$  were correlated with a correlation value above 0.7 *and* stocks  $s^8$  and  $s^9$  were correlated with a correlation value above 0.85.

$\square$

In the above example, we want to monitor the occurrence of any pre-defined interested pattern. In practice, the number of interested patterns are usually very large while the likelihood of a pattern to be false is usually much higher than that of a pattern to be true. That is, at any time position, the number of alerts is usually very small comparing to the total number of interested patterns. In other words, we are usually monitoring *rare events*.

In this paper, we use a simple mechanism to predict the correlation values of relevant stream pairs at the next time position and rank the stream pairs carefully so that the pairs that are likely to have low correlation values are evaluated first. We show, through experiments, that the method significantly reduces the total number of pairs for which we need to compute the correlation values due to the conjunctive nature of the composites.

Correlation on time series is related to similarity-based queries on time series and streaming time series. These have attracted attention recently. For example, [4] uses an FFT (Fast Fourier Transform) and prediction-based approach to optimize queries involving streaming time series. The same authors also considered a pre-fetching algorithm when the data volume is huge [5]. However, they only discuss the operation of finding the nearest neighbor of a streaming series in isolation. Also, streaming data in general has attracted a lot of attention recently. For example, [14, 2, 11, 1, 9, 3]. In terms of continuous queries, this paper deals with a special case where correlation patterns need to be monitored.

The rest of the paper is organized as follows. In the next section, we introduce some basic notation and formulate the problem. Section 3 describes our predication-based method of evaluating composite correlations for streaming time series. We present our experimental results in Section 4 and draw conclusions in Section 5.

## 2 Problem Formulation

We first define time series and streaming time series used in applications. A *time series* is a finite sequence of real numbers. A *streaming time series* is an infinite sequence of real numbers. At each time position  $t$ , however, the streaming time series takes the form of a finite sequence, assuming the last real number is the one that arrived at the time position  $t$ .

We adopt the notations in [16]. Data entries in a streaming time series are in the triple form of  $(streamID, timePosition, value)$ . Each stream consists of all those triples having the same *streamID*. (In the above examples, a *streamID* corresponds to a stock.) The streams are synchronized.

Each stream has a new value available at every unit time interval, e.g., every second. We call the interval value index the *timePosition*. For example, if the unit time interval is second and the current time position is  $i$ , after one second, all the streams will have a new value with *timePosition*  $i + 1$ .

Let  $s_i$  or  $s[i]$  denote the value of stream  $s$  at *timePosition*  $i$  and  $s[i \dots j]$  denotes the subsequence of stream  $s$  from *timePosition*  $i$  through  $j$  inclusive. Let  $s^i$  denote the stream with *streamID*  $i$ . We use  $t$  to denote the latest *timePosition*, i.e., now.

**Definition** Consider two streams,  $s$  and  $r$ . The *correlation value* of them are defined over a *sliding window*. Given  $w$ , the length of a sliding window, and  $t$ , the current *timePosition*, the current correlation value of  $s$  and  $r$  is computed as follow:

$$corr(s, r) = \frac{\sum_{i=1}^w s_i r_i - w \bar{s} \bar{r}}{\sqrt{\sum_{i=1}^w s_i^2 - w \bar{s}^2} \sqrt{\sum_{i=1}^w r_i^2 - w \bar{r}^2}},$$

where  $\bar{s}$  ( $\bar{r}$ ) is the average value of stream  $s$  ( $r$ ) over the sliding window.

**Definition** A correlation condition is defined on a pair of streams,  $s$  and  $r$ , and a given threshold value range  $c_r = [c_l, c_h]$  ( $-1 \leq c_l \leq c_h \leq 1$ ). We say  $s$  and  $r$  are correlated with respect to  $c_r$  if and only if  $c_l \leq corr(s, r) \leq c_h$ . We call  $c_l \leq corr(s, r) \leq c_h$  a *correlation term*.

The correlation value<sup>3</sup> of any steam pair is always in the range of  $[-1, 1]$ . In most applications, we are interested in correlation condition with a high threshold value range, i.e., we are interested in the occurrence of high correlation value. By high correlation value, we mean the magnitude (absolute value) of the correlation value is large. High correlation values thus correspond to values that are close to -1 (for high negative correlations) and 1 (for high positive correlations).

<sup>3</sup> We use the term “correlation value” and “correlation” interchangeably.

**Definition** A composite correlation pattern is in the form of  $t_1 \wedge t_2 \wedge \dots \wedge t_n$ , where  $t_i$  ( $1 \leq i \leq n$ ) is a correlation term.

A composite correlation pattern can be evaluated at any *timePosition* and is evaluated to be either *true* or *false* at any given *timePosition*. Different applications may have different requirements on how often a correlation pattern needs to be evaluated. Similar to [16], we subdivide the sliding windows equally into shorter windows, which we call *basic window*. The size of basic window models the time granularity of the evaluation.

Using the above definitions, the problem we address in this paper can be stated formally as follows. Given a set of streams  $S = \{s^1, s^2, \dots, s^m\}$  and a set of composite correlation patterns  $P = \{p_1, p_2, \dots, p_k\}$  defined over the streams in  $S$ , at each *timePosition* that corresponds to the last *timePosition* of a basic window, we want to report all the patterns in  $P$  that are evaluated to be true.

### 3 Prediction-Based Evaluation of Composite Correlations

To evaluate the composite correlation patterns, a straightforward algorithm, which we call the *naive* algorithm, is to evaluate all the correlation terms for each correlation pattern at the end of each basic window. If a correlation term turns out to be false, then the pattern itself is false and we can skip evaluating the rest of the correlation terms in this pattern.

From the above naive algorithm, we see that the order of evaluation of the correlation terms in a pattern may be important. Indeed, if the a pattern is eventually evaluated false, at least one of the terms is false and an obvious strategy is to evaluate that particular term before other terms. As mentioned in the introduction, the likelihood of a pattern to be false is usually high and this strategy should help a lot in reducing computation cost.

Another observation is that when correlation terms in multiple patterns involve the same pair of streams, the computation of the correlation value on the pair of streams can be shared. Furthermore, since our intention is to look for high (positive or negative) correlations, if a pair of streams have low correlation (both positive and negative), then it is most likely that all the patterns involving this pair of streams will be false due to the false value of the term with this stream pair.

Our optimized algorithm is based on the above observations. The question remains as how we decide the order of evaluation of correlation terms. In this paper, we use a simple prediction model. Specifically, we postulate that *if a pair of streams has low correlation at the end of one basic window, it likely remains so at the end of the next basic window*. We use this simple prediction model to decide the order of the correlation evaluation of pairs of streams and quickly “falsify” patterns. The expected overall effect is reduced number of correlation calculations. We call this algorithm the *optimized algorithm*, and Figure 1 shows an outline of it.

A number of issues need consideration in the algorithm of Figure 1. The first is *how exactly the pairs are ranked* in Step 1. If there are prediction models for

<b>Input:</b>	<ol style="list-style-type: none"> <li>1. A set of streaming time series.</li> <li>2. A set of correlation patterns.</li> <li>3. <math>w</math>, the length of the sliding window.</li> <li>4. <math>b</math>, the length of the basic window.</li> </ol>
<b>Output:</b>	At the end of each basic window, report all the correlation patterns that are evaluated true.
<b>Method:</b>	<p>Let <math>\mathcal{SP}</math> be the set of stream pairs that appear in the given correlation patterns. For each stream pair in <math>\mathcal{SP}</math>, associate it with the number of the correlation patterns it appears in. For each pair <math>(s, r)</math>, denote this as <math>\#P(s, r)</math>.</p> <p>At the end of each basic window, restore <math>\#P(s, r)</math> to their original value for all pairs <math>(s, r)</math>. Then do</p> <p><b>Step 1.</b> Rank all the pairs of streams in <math>\mathcal{SP}</math>.</p> <p><b>Step 2.</b> Evaluate the correlation of each pair <math>(s, r)</math> of streams in the order given by Step 1. Only need to evaluate the pairs <math>(s, r)</math> such that <math>\#P(s, r) &gt; 0</math>.</p> <ol style="list-style-type: none"> <li>1. For each correlation pattern that involves the evaluated pair, decide if it is falsified.</li> <li>2. For each falsified pattern, go over each of stream pair <math>(s, r)</math> in it, and reduce <math>\#P(s, r)</math> by 1.</li> </ol>

**Fig. 1.** Optimized algorithm.

time series in the application domain we are dealing with, we may use them. As mentioned earlier, we use a simple prediction model by looking at how many patterns a pair of streams falsified (Step 2.1), denoted  $rej$ , and in which basic window, denoted  $win$ , this happened (since the correlations of pairs are not calculated at every basic window). We want to “reward” the pairs with greater  $rej$  value while at the same time reduce the “reward” if it happened too far in the past. Therefore, if  $win$  is taken as the last *timePosition* of the basic window (hence the bigger the value, the more recent the basic window is), we use the formula

$$(win)^2 * rej$$

to give the weight to a pair. We rank all the pairs in the decreasing order of their weights. If the correlation of a pair has never been calculated, we set  $win$  and  $rej$  to be 0.

Another issue is the *prediction frequency*. Indeed, although in our algorithm we use a simple method, it still costs time to rank and sort. There is a tradeoff between the overhead of this cost and the gain in terms of the reduction of calculation. When done too frequently, prediction step will out-weight the gain of saving, while too infrequently, the saving in the prediction cost may not justify the loss of savings obtained from prediction. Obviously, where is the point of “best return” depends on the situation we are dealing with. In this paper, we use experiments to determine this point of best return.

## 4 Experimental Results

In this section, we describe the experiments used to evaluate the performance of our method and compare it with that of the naive method.

Through our experiments, we answer four important questions about the effectiveness of our predication-based optimization method:

1. Does the optimized algorithm significantly reduce the total number of pairs for which we need to evaluate the correlation values compared to the naive algorithm?
2. Does the optimized algorithm significant reduce the CPU time per basic window compared to the naive method?
3. Is the superiority of optimized algorithm stable with respect to the types of pattern sets?
4. Does the optimized algorithm scale well with respect to the number of streams and the number of patterns?

We implement two methods in C/C++ as our methods of comparison: the naive method and our predication-based optimization method. For our experiments, we use WinXP on an Intel Pentium 4 processor of 1.8GHZ with 256 MB of main memory.

**Data Generation** We generate the streaming data sets using the random walk model. For stream  $s$ ,

$$s_i = u_0 + \sum_{j=1}^i u_j, \quad i = 1, 2, \dots, L$$

where all  $u_j$  ( $j \geq 0$ ) are uniformly distributed, independent variables. Variable  $u_0$  gives an integer value in  $[-50, 50]$  and each  $u_j$ ,  $j > 0$ , gives an integer value in  $[-5, 5]$ .

A data set is modeled by two parameters: *numStream*, the number of streams in the data set, and *lenStream*, the length of each stream.

**Pattern Set Generation** We use our own pattern generator to generate a set of interested composite correlation patterns for a given streaming data set.

To generate a pattern set, the first step is to generate the composite correlation patterns without considering the threshold value range for each correlation term. The second step is to generate the corresponding threshold range for each correlation term.

The first step is modeled by four parameters: *numPattern*, the number of composite correlation patterns in the set, *zPair*, the Zipf parameter for the frequency distribution of stream pairs, *minTerm*, the minimum number of correlation terms in a composite correlation pattern, and *maxTerm*, the maximum number of correlation terms in a composite correlation pattern.

The *zPair* parameter is not as straightforward as other parameters and needs a little more explanation. For a given stream pair  $(s^i, s^j)$  and a pattern set  $P$ , we

call the number of occurrences of term  $c_l \leq \text{corr}(s^i, s^j) \leq c_h$  in  $P$  the *occurrence frequency* of pair  $(s^i, s^j)$  in  $P$ .

For example, suppose  $P = \{p_1, p_2, p_3\}$ , where

$$p_1 = (0.8 \leq \text{corr}(s^1, s^2) \leq 0.9) \wedge (0.85 \leq \text{corr}(s^1, s^3) \leq 0.95),$$

$$p_2 = (0.6 \leq \text{corr}(s^1, s^2) \leq 0.7) \wedge (0.75 \leq \text{corr}(s^2, s^3) \leq 0.85),$$

$$p_3 = (0.5 \leq \text{corr}(s^1, s^2) \leq 0.8) \wedge (0.45 \leq \text{corr}(s^3, s^4) \leq 0.75),$$

then the occurrence frequencies of stream pairs  $(s^1, s^2)$ ,  $(s^1, s^3)$ ,  $(s^2, s^3)$ , and  $(s^3, s^4)$  are 3, 1, 1, and 1, respectively. It is easy to understand that in most applications, the distribution of the occurrence frequency for stream pairs is usually skewed, i.e., some pairs occur more frequently than other pairs in the pattern set. As in a lot of previous work (e.g., [10, 13]), we use Zipf distribution to model the skewed distribution of occurrence frequency for stream pairs [17]. A higher *zPair* value corresponds to fewer very high frequencies, i.e., a few stream pairs appear in most of the patterns while other stream pairs rarely appear.

In the second step, we generate the corresponding threshold range for each correlation term. Since we are usually interested in high correlation value, we pick the threshold range from the high ends.<sup>4</sup> Note that the effectiveness of our predication-based optimization method does not depend on any specific choice of the threshold range. To figure out what is the “reasonable” threshold range for a given stream pair  $(s^i, s^j)$ , we compute the minimum negative correlation value (*minNegCorr*), the maximum negative correlation value (*maxNegCorr*), the minimum positive correlation value (*minPosCorr*), and the maximum positive correlation value (*maxPosCorr*) for the pair. For positive correlation, we always set  $c_h = \text{maxPosCorr}$  and  $c_l = c_h - \alpha(\text{maxPosCorr} - \text{minPosCorr})$ , where  $\alpha$  is a uniformly distributed random variable in  $[\text{minRangeSize}, \text{maxRangeSize}]$ .<sup>5</sup> Similarly, for negative correlation, we always set  $c_l = \text{minNegCorr}$  and  $c_h = c_l + \alpha(\text{maxNegCorr} - \text{minNegCorr})$ .

**Parameter Setting** We show our experimental results on several representative data sets and pattern sets in Figure 2–6. Unless otherwise specified, the default values for the parameters used in the experiments are as listed in Table 1.

In the setting given by Table 1, the number of patterns that are evaluated true is around 30 at each basic window among the given 10,000 patterns. This is consistent with our assumption that the monitored events should be rare events. The CPU time that the algorithms spent for each basic window is around 30ms and 15ms for the naive algorithm and the optimized algorithm, respectively. Since the basic window has 10 data values, it is generally true that the algorithm

<sup>4</sup> By high correlation value, we mean the magnitude (absolute value) of the correlation value is large. High ends thus correspond to the correlation values that are close to -1 (for high negative correlations) and 1 (for high positive correlations).

<sup>5</sup> Parameters *minRangeSize* and *maxRangeSize* are specified by the user and  $0 \leq \text{minRangeSize} \leq \text{maxRangeSize} \leq 1$ .

<i>Parameter</i>	<i>Meaning</i>	<i>Default Value</i>
<i>numStream</i>	number of streams in the data set	2,000
<i>lenStream</i>	length of each stream	10,000
<i>w</i>	length of sliding window	100
<i>b</i>	length of basic window	10
<i>numPattern</i>	number of patterns in pattern set	10,000
<i>minTerm</i>	minimum number of correlation terms in a pattern	2
<i>maxTerm</i>	maximum number of correlation terms in a pattern	5
<i>zPair</i>	Zipf parameter for the frequency distribution of stream pairs	1.3
<i>minRangeSize</i>	minimum range size factor	0.2
<i>maxRangeSize</i>	maximum range size factor	0.4
<i>predFreq</i>	prediction frequency	400

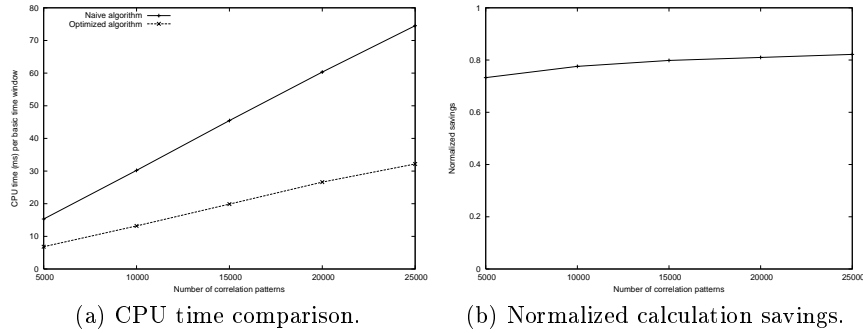
**Table 1.** Parameter setting for experiments

can handle (with the optimized algorithm) a stream rate of roughly up to 700 values per second for each stream under the default setting (with 2000 streams, among other parameters).

**Experimental Results** In accordance with the goal of the experiments, we performed five groups of tests. In each group of tests, we pick one of the parameters shown in Table 1 and vary it in different ways while keeping the other parameters fixed to these shown in Table 1. We then measure two metrics. The first is the CPU time comparison of the naive algorithm and the optimized algorithm, and the second is the comparison of the number of correlation calculations by these two algorithms. We randomly generate data and queries as described earlier and for each combination of parameters, we perform five runs and use the average value of the five runs as the value reported in the figures.

Figure 2 reports the result of a set of tests where we only change the number of correlation patterns we deal with. In Figure 2(a), the CPU time is measured for each case. In Figure 2(b), the two algorithms are compared in terms of the number of correlation calculations. The value in the figure is the result of the formula  $(n-o)/n$ , where  $n$  ( $o$ , respectively) is the number of correlation calculations performed in the naive algorithm (optimized algorithm, respectively).

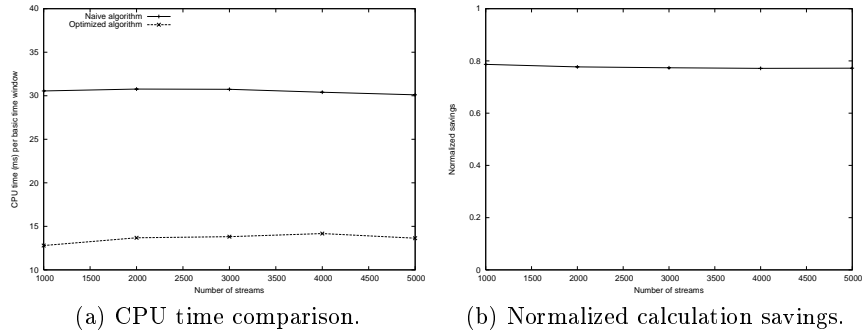
From the figure, for the savings in terms of number of correlation calculations, the optimized algorithm only perform around 20-30% of the calculations that the naive algorithm does. In terms of CPU time, however, the savings is not as much due to the overhead of ranking process of the stream pairs (Step 1 in Figure 1). Also observe that the savings tend to become smaller when the number of patterns goes up, although the CPU time goes up slower for the optimized algorithm than the naive algorithm. It is due to the fact that when the number of correlation patterns increases while the number of streams stay constant, there is a tendency that the correlation patterns have fewer shared correlation terms. Thus, the effect of Step 2.2 of Figure 1 decreases. The savings will mostly be achieved in terms of the evaluation order of the correlation terms in a pattern.



**Fig. 2.** Scalability with number of correlation patterns.

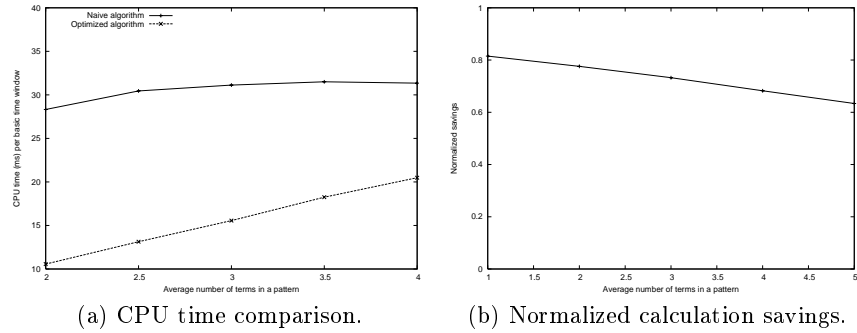
Since the number of terms is not large (on average each pattern has 3.5 terms in the default setting), the naive algorithm will eventually perform similarly as the optimized algorithm when the number of patterns continue to go up.

Figure 3 gives the test results when the number streams is varied. Both the CPU time and the number of correlation calculations do not go up with the increase of the number of the streams. This is true for both the naive algorithm and the optimized algorithm. This is due to the fact that when the number of streams increases while the number of patterns stays the same, the number of pairs of streams that appear in patterns do not change much (due to the skewness of the selection of stream pairs). Hence, the overall performance of the algorithms is not too sensitive to the number of streams.



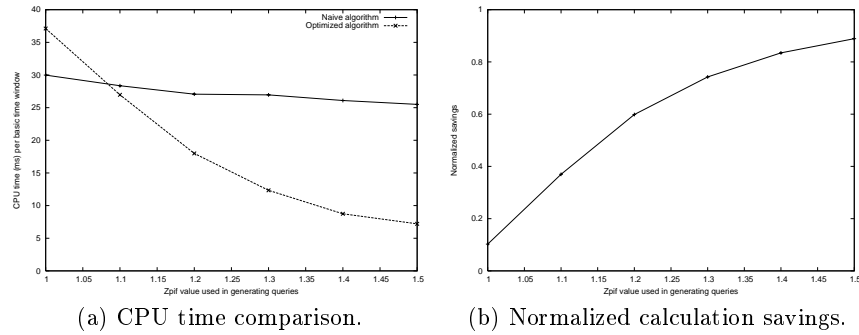
**Fig. 3.** Scalability with number of streams.

In Figure 4, we give the test results when we change the average number of correlation terms in a pattern. The results are as expected that the savings decrease as the number of terms in a pattern goes up. Indeed, in this case, since other parameters are fixed, the number of stream pairs will increase and the effect of Step 2.2 will decrease.



**Fig. 4.** Scalability with number of terms in patterns.

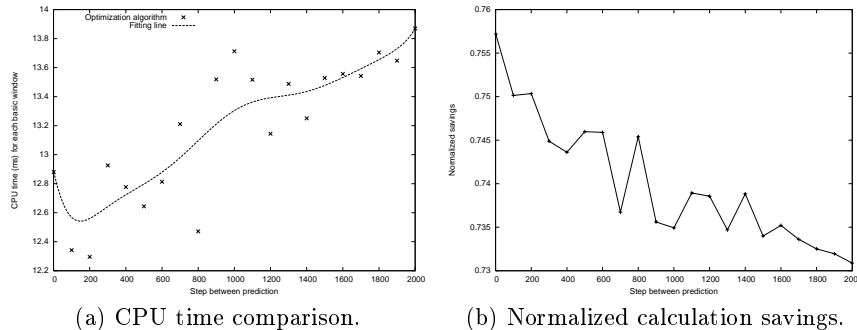
The skewness of our patterns also has an effect on the performance of our algorithms. With less skewed pattern set, the overhead will overcome the savings, while more skewed pattern set is, the more saving we have. Experiment result in Figure 5 confirms this general trend. Note that when the Zipf value is 1.0, the naive algorithm outperforms the optimized algorithm in terms of the CPU time, while in terms of the number of calculated correlations, the optimized algorithm still wins. The reason is that the overhead of the optimized algorithm in this case cannot be compensated by the savings in the number of correlation calculations. We believe, in practice, the skew level of the pattern set is usually high.



**Fig. 5.** Effect of skewness of patterns.

As the last experiment result, we show in Figure 6 the effect of the frequency of prediction step (Step 1 of Figure 1). The general trend of the savings is consistent with our observations, namely, too frequent and too infrequent prediction step are both counterproductive. In this figure, the trend is not as clear cut as other figures and we use a fitting line in Figure 6(a) to highlight the trend. In Figure 6(b), the trend is a little clearer, i.e., in general, the more frequent (smaller step) the prediction is performed, the more saving we obtain. How-

ever, the savings may not compensate the overhead incurred. Therefore, there is usually a point of the “best return”.



**Fig. 6.** Effect of prediction frequency.

## 5 Conclusions

In this paper, we present a prediction-based method that can efficiently evaluate composite correlations for multiple streaming time series. We show, through experiments, that our method significantly improves the performance of the evaluation process comparing to the naive method.

While we have focused on monitoring patterns in conjunctive form, our method can easily be extended to deal with disjunctions. As a result, our method can be used to evaluate patterns in general composite forms.

There are two basic building blocks for our method: (1) an efficient algorithm for computing the correlation value for a given pair of streaming time series, and (2) an accurate and effective prediction mechanism for predicting the future correlation value of a given stream pair. We did not discuss (1) in this paper and used a very simple prediction method for (2). To improve the performance of our method, we can certainly plug in any advanced algorithm for (1), e.g., the approximation algorithm proposed in [16]. The prediction mechanism is more domain specific and we refer the interested readers to [7, 8, 12, 6, 15] for more insights.

An interesting research problem is how to handle the tradeoff between the overhead and computational saving introduced by applying the predication and ranking step in a better way. While we used a fixed predication frequency in our experiments, a better approach should be adjusting the predication frequency adaptively: the predication and ranking step should be applied when the performance drops to certain level due to inaccurate predication.

## References

1. S. Babu and J. Widom. Continuous queries over data streams. *SIGMOD Record*, 30(3):109–120, 2001.
2. J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: a scalable continuous query system for Internet databases. In *SIGMOD Conference*, pages 379–390, 2000.
3. Y. Chen, G. Dong, J. Han, B. W. Wah, and J. Wang. Multi-dimensional regression analysis of time-series data streams. In *VLDB Conference*, pages 323–334, 2002.
4. L. Gao and X. S. Wang. Continually evaluating similarity-based pattern queries on a streaming time series. In *SIGMOD Conference*, pages 370–381, 2002.
5. L. Gao and X. S. Wang. Improving the performance of continuous queries on fast data streams: Time series case. In *Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD)*, 2002.
6. T. V. Gestel, J. Suykens, D.-E. Baestaens, A. Lambrechts, G. Lanckriet, B. Vandaele, D. B. Moor, and J. Vandewalle. Financial time series prediction using least squares support vector machines within the evidence framework. *IEEE Transactions on Neural Networks*, 12(4):809–821, 2001.
7. L. Györfi, G. Lugosi, and G. Morvai. A simple randomized algorithm for sequential prediction of ergodic time series. *IEEE Transactions on Information Theory*, 45(7):2642–2650, 1999.
8. I. Kim and S.-R. Lee. A fuzzy time series prediction method based on consecutive values. In *Fuzzy Systems Conference Proceedings, 1999. FUZZ-IEEE '99*, volume 2, pages 703–707, 1999.
9. S. Madden and M. J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *ICDE Conference*, 2002.
10. Y. Matias, J. S. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 448–459, Seattle, WA, June 1998.
11. B. Plale and K. Schwan. Optimizations enabled by a relational data model view to querying data streams. In *Proc. of 15th International Parallel and Distributed Processing Symposium*, page 20, 2001.
12. S. Policker and A. Geva. A new algorithm for time series prediction by temporal fuzzy clustering. In *Proceedings. 15th International Conference on Pattern Recognition*, volume 2, pages 728–731, 2000.
13. V. Poosala, Y. E. Ioannidis, P. J. Haas, and E. Shekita. Improved histograms for selectivity estimation of range predicates. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, Montreal, Canada, May 1996.
14. D. Terry, D. Goldberg, D. Nichols, and B. Oki. Continuous queries over append-only databases. In *SIGMOD Conference*, pages 321–330, 1992.
15. L. Wang, K. K. Teo, and Z. Lin. Predicting time series with wavelet packet neural networks. *Proc. International Joint Conference on Neural Networks*, 3:1593–1597, 2001.
16. D. S. Yunyue Zhu. Statstream: Statistical monitoring of thousands of data streams in real time. In *Proceedings of the 28th International Conference on Very Large Data Bases*, pages 358–369, 2002.
17. G. K. Zipf. *Human Behaviour and the Principle of Least Effort*. Addison-Wesley, Reading, MA, 1949.