# Adaptive Query Processing for Time-Series Data

Yun-Wu Huang
IBM T.J. Watson Research Center
30 Saw Mill River Road
Hawthorne, NY 10532
ywh@us.ibm.com

Philip S. Yu
IBM T.J. Watson Research Center
30 Saw Mill River Road
Hawthorne, NY 10532
psyu@us.ibm.com

## ABSTRACT

Traditional query process for time-series data transforms data from the time domain into the frequency domain. It is less controllable by the end users, therefore isn't well suited for queries that find patterns with many dynamically specified user constraints. For these queries we present a method to search time-series data which first transforms time sequences into symbol strings using change ratio between contiguous data points in time series. Next, a suffix tree is built to index all suffixes of the symbol strings. The focus of this paper is to demonstrate how this method can adapt to the processing of the dynamically constrained time-series queries.

## Keywords

Time-series Query Processing, Data Mining, Pattern Matching.

## 1. INTRODUCTION

Many real-life database applications manage temporal data that are physically stored in time sequence. The examples of time-series data include the prices of mutual funds and stocks, the production outputs, weekly sale totals, and the daily temperatures.

In many of these applications, it is highly desirable to find patterns in their respective time-series data set. The result may help to improve the processes such as analyses, predictions, and data mining [2]. The followings are examples of time-series pattern matching queries:

**Q1**: *Find all stocks that behave "similarly" to a certain pattern.*

**Q2**: *Find all factories in any n-day time periods in which the defect rates of their perspective factories show a consecutive upward trend.*

**Q3**: *Find all head-and-shoulder patterns in all regions' weekly sale totals based on 3-day moving averages.*

The example query **Q1** is a similarity query for time-series data [1, 3, 7, 15]. It can be used to help the prediction of future stock or mutual fund performance. The query **Q2** can be used as a warning system to detect a potential production problem. The query **Q3** provides a useful marketing tool in analyzing the sale patterns among different regions. It is based on the moving average [6] to generalize trends.

Many traditional approaches in time-series query processing are based on transforming data from the time domain into the frequency domain to optimize the indexes so that more efficient search and retrieval can be achieved [1, 3, 7, 15, 16]. As a result, these approaches are well suited for searching patterns, such as the similarity-based queries (e.g., example query **Q1**).

We in this paper explore solutions when no known patterns exist. For example, the upward trend constraint in **Q2** is very general and includes shapes of various degrees. Therefore, methods for similarity-based pattern matching [1, 3, 7, 15] which focus on matching a specific shape may not be suitable for processing **Q2**. The example query **Q3** is based on weekly totals and pre-built indexes based on daily totals may not be effective in processing this query. Other examples include matching based on different moving average window sizes, matching combinatorial patterns (e.g., patterns represented by regular expressions), matching patterns with different degrees of accuracy specified for different sub-patterns, etc.

In general, we focus time-series query processing with various dynamically specified constraints and present a new method called IMPACTS (the Interactive Matching of Patterns with Advanced Constraints in Time-Series databases). IMPACTS is divided into two stages, the off-line preprocess stage and the on-line query stage (Figure 1).

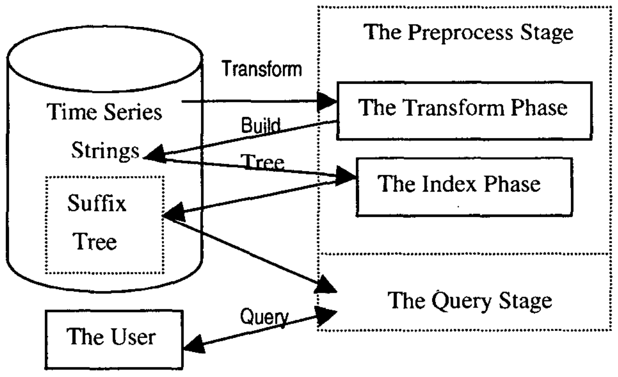

**Figure 1. The Different Stages of IMPACTS.**

The pre-process stage of IMPACTS has two phases: the *transform* phase and the *index* phase (Figure 1). In the transform phase, each sequence of real numbers in the time-series data is transformed into a string of symbols. This is achieved by first segmenting the entire range of change ratios between any two consecutive time points into a finite number of non-overlapping smaller ranges (segments). Next, a unique symbol is used to represent each segment, thereby transforming a time sequence into a symbol string.

In this paper, we present the different stages of IMPACTS as well as algorithms (based on the suffix tree structure) to compute time-series pattern matching queries with the abilities to dynamically process constraints such as the multiple degrees of accuracy, different moving average window sizes. We also describe how IMPACTS can be used to resolve advanced constraints such as a dynamically specified aggregate period and combinatorial patterns. Lastly, we present a performance evaluation to show that our method can be much faster than the naive approach.

The rest of this paper is organized as follows: We first discuss related work in Section 2. Next, we present the pre-process stage in Section 3 and the query stage in Section 4 respectively. We present our performance evaluation in Section 5 and conclude the paper in Section 6.

## 2. RELATED WORK

We categorize recent work in time-series pattern matching into two general approaches. The first approach maps time sequences into frequency domain; the second processes the time sequences directly in time domain. The frequency domain approach, pioneered by [1], in general computes a Discrete Fourier Transform [10, 14] for each sequence and selects the first few coefficients to index their respective original sequences. Sequences with matching coefficients are considered similar. The indexing mechanism is typically through a multi-dimensional index structure such as R-tree [9] or R*-tree [5].

While in [1], the focus is on whole sequence matching, the work in [7] generalizes the process to allow subsequence matching. In [15], moving average, time warping, and reversing are formulated and the indexing methods are further examined for approximate subsequence matching.

The recent work in time domain [3, 16] focus on segmenting long sequences into smaller subsequences for approximation. A multi-dimensional index structure, such as R-tree [9] or R*-tree [5], is then used to index the small segments. In [3], an innovated approach is introduced based on atomic segments, called *windows*, which can be normalized with similar windows respectively stitched to form pairs of large similar subsequences. In [16], based on the divide-and-conquer approach, subsequences are identified and approximated by families of real-valued functions.

The IMPACTS system also operates in time domain. But it differs from all the above approaches (both frequency and time domains) [1, 3, 7, 15, 16] in one significant way. The IMPACTS can preserve, to a very high degree of details, the shapes of the original sequences in its index structure (the suffix tree). A major advantage of detail preservation is the flexibility to dynamically process a diverse class of different time-series matching queries such as those based on different degrees of accuracy, moving window sizes, aggregate periods, and vague trends (see Section 4). For example, time-series matching queries with different moving window sizes, aggregate periods, or vague trends cannot be processed by the aforementioned approaches [1, 3, 7, 15, 16] without the restructure of their respective index structures. Therefore, the ability to dynamically process a diverse class of different matching queries makes IMPACT more suitable as an interactive time-series pattern matching approach.

In [4], a shape-describing query language called SDL was proposed for time-series shape matching. An index structure was proposed but no performance data was given. The SDL model can be adopted by IMPACTS to formulate some of its query classes. The focus of this paper however is on the system aspect of IMPACTS, not on query language. The suffix tree index deployed by the IMPACTS system is used extensively in the string-matching problem [11, 12, 13, 17].

## 3. THE PRE-PROCESS STAGE

The first stage of IMPACTS is the preprocess stage in which the transform phase transforms the time-series data into symbol strings and the index phase build suffix tree indexes based on these strings.

### 3.1 The Transform Phase

Transforming a time series into a symbol string can be achieved with various ways. In this paper, we present one method that is based on computing the change ratio from one time point to the next time point and dividing the variances of all change ratios into discrete segments such that change ratios that fall into one segment are represented by a unique symbol. We use an example (Figures 2) to illustrate this process. First, we define three parameters below:

- The parameter `min` is the lower bound of all change ratios between any two consecutive time points among all time series in the database.
- The parameter `max` is the upper bound all change ratios between any two consecutive time points among all time series in the database.
- The parameter `numSegment` represents the number of segments selected to divide the range between `min` and `max`. While all segments together fill the entire range between the `min` and `max` parameters, no two distinct segments overlap. In the transformation process, a unique symbol is mapped to each segment

In Figure 2, `min` and `max` are set to *-47%* and *63%* while the `numSegment` parameter is set to *11*. We equally divide the entire range between `min` and `max` by `numSegment` with each unit range set to *20%*. In this example, *11* unique symbols are used to model the segments respectively and any change ratio will fall into one of the segments, therefore will be represented by one symbol. The larger `numSegment` is, the more accurately the transformation process captures the shape of a time series. However, a large number of unique symbols may more space to model each symbol, resulting in a larger and less space-efficient suffix tree.

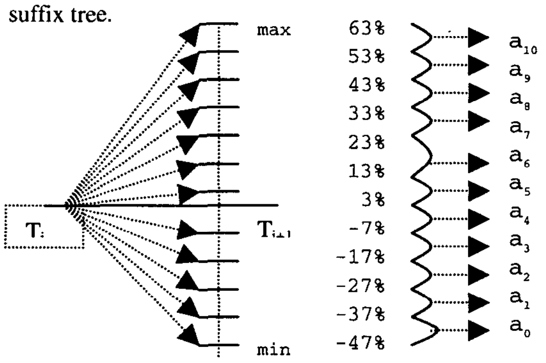

**Figure 2. A Transformation Example.**

## 3.2 The Index Phase

A suffix tree [11, 12, 13] is a trie-like data structure that compactly stores all suffixes of a string such that each suffix is represented by a path from the root node to a certain node at a lower level. The compactness is achieved by suffixes having the same prefix share a node such that the path from the root node to this node represents the shared prefix. Because all suffixes of a string are each represented by a unique path in the suffix tree, finding a pattern in a string corresponds to traversing the suffix tree downwards along the path that matches the target pattern. Such a structure insures high efficiency in matching string patterns, therefore the suffix tree is widely used in the problem of string matching [11, 12, 13].

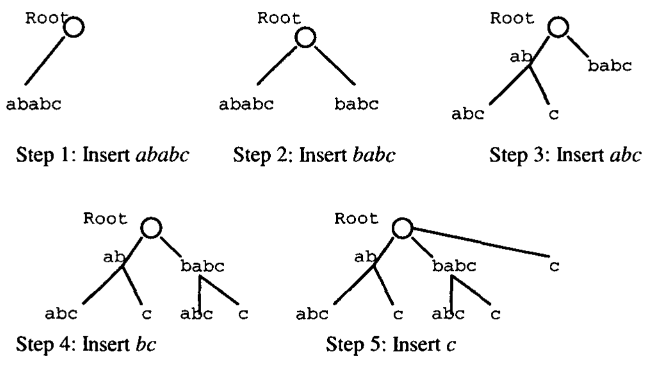

**Figure 3: Constructing A Suffix Tree with *ababc*.**

In the index phase, a generalized suffix tree is constructed to index the suffixes of all the transformed symbol strings for fast pattern matching. In Figure 3, we show the construction of a suffix tree based on the string *ababc*. The five steps in Figure 3 represent the shape of the tree after inserting the suffixes *ababc*, *babc*, *abc*, *bc*, and *c* respectively. Note that node splitting occurs in steps 3 and 4 because of the sharing of prefixes *ab* and *b* respectively.

Inserting suffixes from multiple strings into a suffix tree creates a side effect that makes its structure more complex. The string along the path to a node can represent a suffix of an input string or a prefix of a suffix of an input string. For example in Figure 4, the path from root to the node *ab* in the middle branch represents the string *bab*. However, *bab* is a suffix of String 2 as well as a prefix of the suffix *babc* of String 1. To preserve the knowledge that the middle *ab* node is an end node of a path representing a suffix of an input string, we associate this node with a list of string IDs that have a suffix represented by this node.

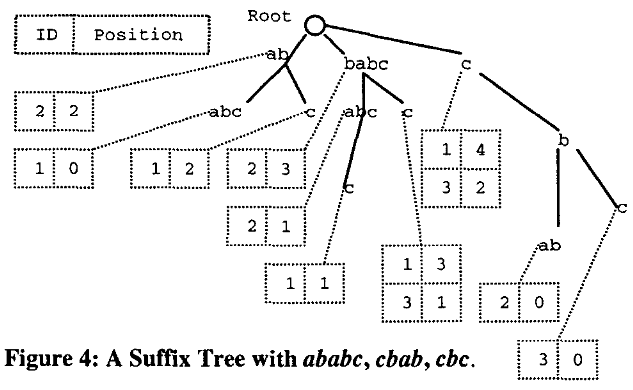

**Figure 4: A Suffix Tree with *ababc*, *cbab*, *cbc*.**

In our implementation, along with each string ID in the list, we also maintain the starting position in this string of the corresponding suffix. For example, the list associated with the middle *ab* node has a 2-tuple entry (Figure 4). The attributes 2 and 1 in this entry indicates that the suffix represented by the path from root to this node (i.e., *bab*) starts at position 1 in String 2.The incorporation of the ID list helps to improve the efficiency in retrieving the matching suffixes.

## 4. THE QUERY STAGE

IMPACTS exploits the suffix tree structure and provides methods to process pattern matching queries on time series with dynamic user constraints such as the degree of accuracy and the average moving window size. In this section, we first describe how simple and dynamically constrained pattern matching queries are computed based on the suffix tree structure. Next, we discuss how IMPACTS can be used as an interactive pattern-matching tool to process a diverse class of advanced time-series pattern matching queries.

## 4.1 Simple String Matching Queries

To match a time-series pattern, the IMPACTS system first transforms the target time-series into a symbol string using the same transformation process discussed in Section 3.1. Note that it is possible that a change in the target sequence is beyond the min and max boundaries defined for the data set. In this case, we can use two additional symbols to respectively represent these two out-of-bound cases.

Next, IMPACTS traverses, in a downward fashion, the suffix tree containing all the suffixes of the symbol strings. During the traversal, with each encountered tree node, the IMPACTS tries to match the prefix of the target string with the string associated with this node. If a match is found, the IMPACTS continues to traverse the children nodes of this node with a new target string created by removing the matching prefix from the original target string.

If the string associated with the encountered node does not match any prefix of the target string, this search thread is discarded. A string that matches the original target is found when a search thread reaches a node whose associated string matches with the entire updated target string.

Note that without specifying a dynamic constraint such as the degree of accuracy or an average moving window size, the pattern matching of IMPACTS is exactly the same as string matching on a suffix tree [11, 12, 13]. With dynamically specified query constraints, the search process of IMPACTS becomes more complex.

## 4.2 Degree of Accuracy Constraint

In our transformation model, the two pre-set parameters `min` and `max` define the entire range any movement from one number to the next in the sequence must fall into (see Section 3.1). The segmentation process divides this range into equal but non-overlapping segments with each segment represented by a unique symbol (see Figure 2). The `min`, `max`, and `numSegment` together determine the lower and upper bounds represented by each symbol.

In the case of simple string matching, the accuracy of the result is determined by the range that each symbol models. High accuracy is achieved by very small segment range with the `numSegment` parameter set to a large value. In this model, two symbols are considered a match if and only if they are the same identical symbol. To process dynamic query constraints with different degrees of accuracy, IMPACTS relaxes this rule and allows for fuzzy matching in which a match for a target symbol can be a set of symbols, including the target symbol itself.

More precisely, let $r$ be the dynamically specified degree of accuracy, the match of a symbol $s$ are any symbols representing segments that are within $1 - r$ from the segment represented by s. For example, in Figure 2, let the dynamically specified degree of accuracy be *15%*. The match for the symbol $a_5$ are symbols $a_3$, $a_4$, $a_5$, $a_6$, and $a_7$. This is because there exists a point in the segment associated with any of these symbols that is with *15%* away from a point within the segment associated with the target symbol $a_5$.

## 4.3 Average Moving Window Size Constraint

The average moving approach is a popular method in financial and stock analysis [6] to generalize trends. The result of applying an average moving window to a time series is a smoother curve with the smoothness enhanced by a wider moving window

To process a time-series query with a moving average constraint, IMPACTS first transforms the target pattern into the moving average representation based on the user specified constraint (e.g., the window size). Next, IMPACTS traverses the suffix tree in a depth-first order. During tree traversal, IMPACTS computes the moving average for the strings along the search threads on the fly and then matches the dynamically computed moving average with the moving average represented by the target sequence.

Initially, each search thread may have traversed strings which have less symbols than the dynamically specified moving average window size, say $k$. At this initial stage, IMPACTS queues up (queue size = $k$) these traversed prefixes along all search threads. When a search thread has traversed a string of $k$ symbols on the suffix tree, it computes the average of the upper bounds and the average of the lower bounds associated with these $k$ symbols. IMPACTS then checks to see if the first number in the moving average representation of the target sequence is within these upper and lower bound averages. If it is, IMPACTS declares it a match and goes on to traverse the next symbol on the tree for this thread. At this point, IMPACTS drops the first symbol in the queue and puts the next traversed symbol in the queue. New upper and lower bound averages are computed and matched with the next number in the moving average representation of the target sequence, and so on.

If at any time during the matching process a mismatch happens, IMPACTS discards this search thread. If during traversal, all numbers in the moving average representation of the target sequence falls within the upper and lower bound averages, then the string of this search path matches the target sequence.

## 4.4 Aggregation Constraint

A time series can be based on a fine-grained time unit such as day whereas a pattern-matching query can be specified with a dynamic constraint that is based on weekly totals. While to our knowledge no current systems can efficiently processing this kind of queries, IMPACTS can compute this query class in a dynamic fashion. For example, to search aggregate pattern based on weekly totals on the suffix tree with daily totals, IMPACTS traverses downwards each search threads while at the same time aggregates the lower and upper bounds respectively of the traversed symbols. When it encounters a symbols representing the proper ending time point (e.g., the last day of the week), IMPACTS matches the aggregated upper bounds and lower bounds with the corresponding symbol in the target string. If a match is found, IMPACTS traverses the children nodes of the encountered node and starts aggregating again, and so on.

## 4.5 Biased Similarity Constraint

Finding similar pattern based on biased criteria can be a very useful function. For example, a query finding a group of stocks or mutual funds that behave similarly to a certain pattern, but are in general more skewed upwards (or downwards) is one such query. IMPACTS can process this type of queries by setting up biased matching criteria. For example, when IMPACTS determines a match for a target symbol, it can allow for a wider degree of accuracy for symbols representing movements that are more upward skewed than the ones representing movements that are less upward skewed. In other words, IMPACTS uses two degrees of accuracy, one for upward symbols and one for downward ones.

## 4.6 Vague Trend Constraint

Searching a vague pattern such as the example query **Q2** in Section 1 may generate results of many different shapes. The state-of-the-art similarity-based time-series matching approaches [1, 3, 7, 15] typically focus on matching more precise shapes, therefore are less suited than IMPACTS for processing this class of queries. For example, to search a continuous downward trend, the IMPACTS traverses the suffix tree and only follows the symbols representing downward movements.

## 4.7 Mismatch Tolerance Constraint

IMPACTS can incorporate a constraint that allows for a certain number of mismatches. For example, a pattern matching query can be constrained to find all similar patterns allowing at most $k$ mismatches. To process this kind of queries, IMPACTS searches the suffix tree and keeps a mismatch counter for each search thread. When a mismatch counter increases beyond the allowable value, its corresponding search thread is discarded.

## 4.8 Inconsistent Accuracy Constraints

This class of queries means that different sections along the target string have a different accuracy constraint. While no current systems can efficiently process queries of this class, IMPACTS can dynamically process them in the same way as it computes the queries with consistent accuracy constraints, with only a minor extension. The extension is that during tree traversal, IMPACTS keeps track of what section of the target string that it is currently matching and uses the corresponding degree of accuracy for matching as specified by the query constraints.

## 4.9 Combinatorial Constraint

IMPACTS can be adapted to search a combinatorial pattern similar to one modeled by a regular expression. For example, the query manager, in matching a combinatorial pattern, can starts a multi-thread search over the suffix tree while keeping track of the current state for each search thread (much like the state of the finite state machine represented by an regular expression). A search is discard if, at any node, it violates the pattern specified by the target or when it reaches the bottom of the tree without having completely traversed the target pattern.

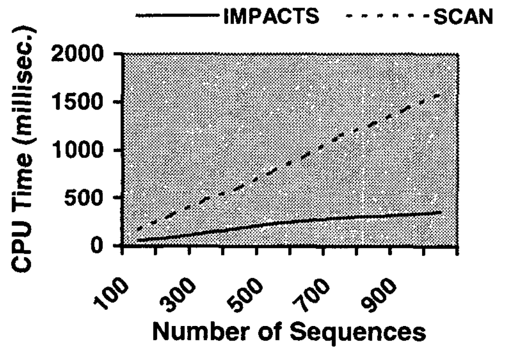

**Figure 4. Search Time vs. Number of Sequences.**

## 5. PERFORMANCE EVALUATION

We compare pattern-matching efficiency between IMPACTS and the sequential scanning method based on synthetic sequence data. Because of space limitation, we only show a small set of our results. We first randomly generated sequences with a length of *500* and varied the number of sequences from *100* to *1000.* For the second set of experiments, we limited the number of sequences to *500* and varied the sequence length from *100* to *1000.* For both sets of experiments, we set the moving average window size to *5* and the allowable error ration to *0.3%.* Next, we process pattern matching with randomly selected target sequences and collected the averaged results. Both sets of results show that pattern matching by IMPACTS is much more efficient than by the sequential scanning method.

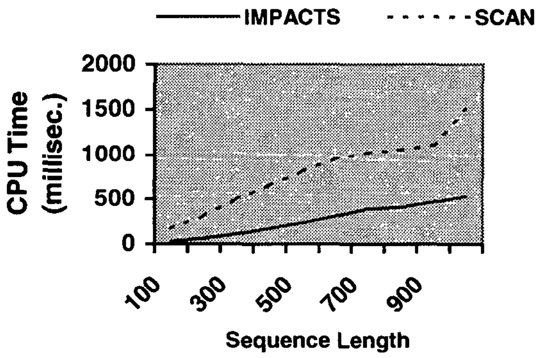

**Figure 5. Search Time vs. Sequence Length.**

## 6. CONCLUSIONS

We consider the problem of query processing for time-series data with various dynamically specified constraints. We present the IMPACTS system that maps atomic movements in a time series into a finite set of symbols to transform time series into symbol strings. Next, IMPACTS uses a suffix tree to index prefixes of these symbol strings. In this paper, we show that IMPACTS can be used to process queries with dynamically specified constraints such as moving average window sizes, aggregate time units, vague trends, and combinatorial patterns. We also present a portion of our performance studies which shows that IMPACTS can be much more efficient in processing queries with dynamic constraints than the sequential scanning method.

## 7. REFERENCES

[1] Agrawal, R., Faloutsos, C., and Swami, A. Efficient similarity search in sequence databases. In Proc. of 4th Intl. Conf. On Foundation of Data Organization and Algorithms, 1993.

[2] Agrawal, R. and Imielinski, T. Data mining: a performance perspective. IEEE Trans. On Knowledge and Data Engineering, (6):914-925, 1993.

[3] Agrawal, R., Lin, K.I., Sawhney, H.S., and Shim, K. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In Proc. of 21st VLDB, 490-501, 1995.

[4] Agrawal, R., Psaila, G., Wimmers, E.L., and Zait, M. Querying shapes of histories. In Proc. Of 21st VLDB, 502-514, 1995.

[5] Bechmann, N., Kriegel, H., Schneider, R., and Seeger, B. The R*-tree: an efficient and robust access method for points and rectangles. In Proc. ACM SIGMOD, 322-332, 1990.

[6] Edwards, R.D. and Magee, J. Technical Analysis of Stock Trends. John Magee, 1969.

[7] Faloutsos, C., Ranganathan, M., and Manolopoulos, Y. Fast subsequence matching in time-series databases. In Proc. ACM SIGMOD, 419-429, 1994.

[8] The Green Line Mutual Funds. http://www.tdbank.ca, 1998.

[9] Guttman, A. R-tree: a dynamic index structure for spatial searching. In Proc. ACM SIGMOD, 45-57, 1984.

[10] Hamming, R.W. Digital Filters. Prentice-Hall, 1977.

[11] Hui, L.C.K. Color set size problem with applications to string matching. In Combinatorial Pattern Matching, Lectures Notes in Computer Science, 230-243, Springer-Verlag, 1992.

[12] Landau, G.M. and Vishkin, U. Fast parallel and serial approximate string matching. Journal of Algorithms, (2):157-169, 1989.

[13] McCreight, E.M. A space-economical suffix tree construction algorithm. JACM, 262-272, 1976.

[14] Oppenheim, A.V. and Schafer, R.W. Digital Signal Processing. Prentice-Hall, 1975.

[15] Rafiei, D. and Mendelson, A. Similarity-based queries for time series data. In Proc. ACM SIGMOD, 13-23, 1997.

[16] Shatkay, H. and Zdonik, S.B. Approximate queries and reprresentations for large data sequences. In Proc. 12th Intl. Conf. On Data Engineering, 536-545, 1996.

[17] Wang, J.T.L., Chirn, G.W., Marr, T.G., and Shapiro, B. Combinatorial pattern discovery for scientific data: some preliminary results. In Proc. ACM SIGMOD, 115-125, 1994.