Discovering and Matching Elastic Rules from Sequence Databases

Sanghyun Park and Wesley W. Chu

Department of Computer Science
University of California, Los Angeles
Los Angeles, CA 90095, USA
{shpark, wwc}@cs.ucla.edu

Abstract. This paper presents techniques for discovering and matching rules with *elastic patterns*. Elastic patterns are ordered lists of elements that can be stretched along the time axis. Elastic patterns are useful for discovering rules from data sequences with different sampling rates. For fast discovery of rules whose heads (left-hand sides) and bodies (right-hand sides) are elastic patterns, we construct a trimmed suffix tree from succinct forms of data sequences and keep the tree as a compact representation of rules. The trimmed suffix tree is also used as an index structure for finding rules matched to a target head sequence. When matched rules cannot be found, the concept of *rule relaxation* is introduced. Using a cluster hierarchy and relaxation error as a new distance function, we find the least relaxed rules that provide the most specific information on a target head sequence. Experiments on synthetic data sequences reveal the effectiveness of our proposed approach.

1 Introduction

Rule discovery from sequential data is a data mining technique for trend prediction [3][7]. There have been several approaches [1][6][9][14] to discover useful rules from patterns occurring frequently in data sequences. A pattern is defined as a partially ordered collection of elements. According to the constraints on the arrangement of elements, patterns can be classified as serial patterns and parallel patterns [9].

As a subset of serial patterns, we can think of *elastic patterns* where elements can be stretched along the time axis by replicating themselves. Elastic patterns AB and ABC are interpreted as A^+B^+ and $A^+B^+C^+$, respectively, using the notation of a regular expression. $\langle A, B\rangle$ and $\langle A, A, B, B, B\rangle$ are instances of an elastic pattern AB while $\langle A, C, B\rangle$ is not. Elastic patterns are useful for discovering rules from data sequences whose sampling rates may vary. For example, consider medical data sequences that record the body temperatures of patients. Some data sequences may have temperature values taken every day while others may have values taken every week. Furthermore, even within a single data sequence, time intervals between neighboring temperature values can vary

Z.W. Raś and S. Ohsuga (Eds.): ISMIS 2000, LNAI 1932, pp. 400–408, 2000.

non-linearly. These sequences cannot be compared directly without considering stretches or compressions of elements along the time axis.

The rules whose heads (left-hand sides) and bodies (right-hand sides) are elastic patterns are called *elastic rules*. Given elastic patterns α and β, elastic rules have the format '$\alpha \rightarrow \beta$' that is interpreted as "if there occurs a sequence which is an instance of α, then it will be followed by a sequence which is an instance of β". Time intervals are not associated with elastic patterns because these patterns are flexible on the time axis.

There are many techniques [1][6][9][14] to discover rules with serial patterns. Many of them use the relationship between patterns and their sub-patterns. Given the serial patterns AB occurring 200 times and $ABAC$ occurring 150 times, they extract the rule '$AB \rightarrow AC$ with confidence $\frac{150}{200}(= 0.75)$'. Infrequent patterns whose numbers of occurrences are below a threshold are ignored because infrequent patterns are considered insignificant. To find frequent patterns, they first find short frequent patterns and then combine them to generate longer candidate patterns. Candidate patterns are checked whether they are frequent or not. These combining and checking steps are repeated until all frequent patterns are found. Therefore, repeated readings of data sequences are unavoidable.

Once rules are discovered from data sequences, they may be used to predict the future trend of a target head sequence $\vec{q}$ via the process of rule matching. We say that a rule is matched to $\vec{q}$ when each element of the rule head is equal to the corresponding element of $\vec{q}$. However, if there are large number of rules, it is not a trivial task to find rules efficiently that are matched to $\vec{q}$.

There are some occasions when we fail to find rules matched to a target head sequence $\vec{q}$. This failure often occurs when $\vec{q}$ is not a frequent sequence. For those infrequent target head sequences, we can introduce the concept of *rule relaxation*. Based on a cluster hierarchy, a rule R is relaxed to R' by replacing some elements of R with elements denoting more general concepts or broader range values. Given a target head sequence $\vec{q}$ and a rule R that is not matched to $\vec{q}$, we can relax R to R' so that R' *covers* $\vec{q}$. We say that a rule covers $\vec{q}$ when each element of the rule head represents the same range as or broader range than the one represented by the corresponding element of $\vec{q}$. Among many relaxed rules that can cover $\vec{q}$, we are interested in finding the least relaxed rules since they describe $\vec{q}$ more accurately than the other relaxed rules.

In this paper, we investigate the problems of discovering and matching elastic rules for data sequences with different sample rates. An efficient rule discovering algorithm is developed and algorithms for exact and relaxed rule matching algorithms are presented.

2 Background

2.1 Suffix Tree

A suffix tree [13] is an index structure that has been used as a fast access method to locate substrings (or subsequences) that are exactly matched to a target string

(or a target sequence). The suffix tree structure is based on *tries* and *suffix tries*. A trie is an indexing structure used for indexing sets of keywords of varying sizes. A suffix trie is a trie whose set of keywords comprises the suffixes of sequences. Nodes of a suffix trie with a single outgoing edge can be collapsed, yielding a suffix tree. We use the notation PN for the parent node of N, and the notation $label(N_i, N_j)$ for the labels on the path connecting nodes N_i and N_j.

2.2 Type Abstraction Hierarchy

Type Abstraction Hierarchy (TAH) [5] is a data-driven multi-level cluster hierarchy that uses relaxation error as a goodness measure for generating clusters. For a cluster $C = \{x_1, x_2, ..., x_n\}$ of n elements, the relaxation error for C is defined as $RE(C) = \sum_{i=1}^{n} \sum_{j=1}^{n} P(x_i)P(x_j) \mid x_i - x_j \mid$ where $P(x_i)$ and $P(x_j)$ are occurring probabilities of x_i and x_j, respectively. The algorithms for generating binary and n-ary TAHs are given in [5]. TAH is easier to implement than the maximum-entropy clustering method and generates more accurate clusters than the equal-length interval clustering method. Figure 1 shows an example TAH built from a distribution of data sequences whose elements take values within the range of $[0, 7.0)$. The relaxation error and the value range are stored in each node, and the nodes are labeled with unique symbols.

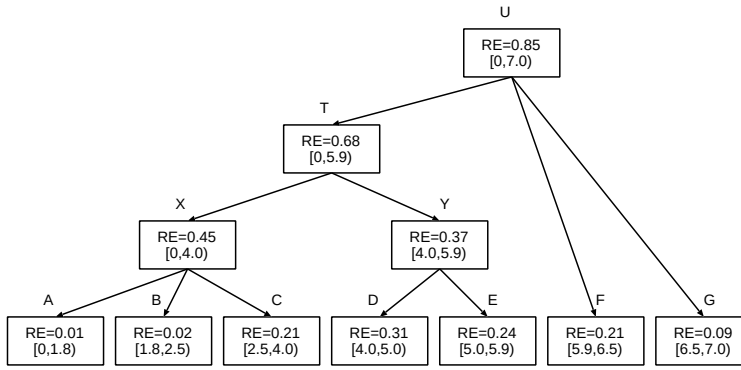


Fig. 1. An example of TAH. Each node is labeled with a unique symbol. The value range and the corresponding relaxation error are stored at each node.

3 Rule Discovery

In this section, we propose an efficient method to discover elastic rules from data sequences via a suffix tree. We assume that the TAH has been generated from data sequences and distinct symbols have been assigned to the TAH nodes.

The support value of the pattern α is defined as the number of suffixes having α as their prefixes. SUP_{min} is the minimum support value that is used to filter

out infrequent patterns. We also define the relative support value of the pattern α as $RSUP(\alpha)$ = (the number of suffixes having α as their prefixes) / (the total number of suffixes). For the applications where the total number of data sequences and their lengths may vary, the relative support is better than the (absolute) support.

The problem of elastic rule discovery is defined as follows: Given a database with M sequences $\vec{x}_1, \vec{x}_2, ..., \vec{x}_M$ and the minimum support value SUP_{min}, discover rules composed of elastic patterns whose supports are at least SUP_{min}. This elastic rule discovery consists of the following five steps.

Step 1. Converting numeric elements to symbol elements: We convert each numeric element of data sequences into the symbol of the corresponding leaf node of the TAH. The symbolized representation of $\vec{x}$ is denoted as $S(\vec{x})$. For example, based on the TAH in Figure 1, a data sequence $\vec{x} = \langle 3.4, 3.0, 3.7, 2.3, 2.1, 4.3 \rangle$ is converted to $S(\vec{x}) = \langle C, C, C, B, B, D \rangle$.

Step 2. Compaction: We convert the symbolized data sequence $S(\vec{x})$ into the compact representation $C(S(\vec{x}))$ by replacing consecutive elements that have the same value with a single element of that value. This step is for considering the property of elastic patterns. For example, $S(\vec{x}) = \langle C, C, C, B, B, D \rangle$ is converted to $C(S(\vec{x})) = \langle C, B, D \rangle$. We use the notation $\vec{X}$ for $C(S(\vec{x}))$.

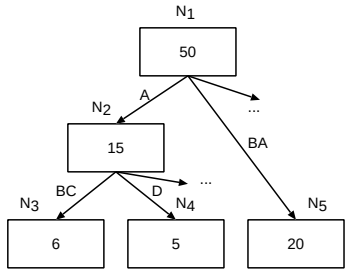
Step 3. Suffix tree construction: From the set of M converted data sequences $\vec{X}_1, ..., \vec{X}_M$, we build a suffix tree using either McCreight's algorithm [8] or incremental disk-based algorithm [4].

Step 4. Trimming: We compute the support values of the nodes and trim out the nodes whose support values are less than SUP_{min}. The support values of internal nodes are obtained by summing up the support values of their children nodes. The support values of the leaf nodes are the same as the number of suffixes represented by the leaf nodes. The trimmed suffix tree is called the *rule tree*.

Step 5. Rule extraction: We compute the confidence values of nodes and then extract rules. The expression for computing the confidence value of the node N is $confidence(N) = Support(N)/Support(PN)$ where PN is the parent node of N. If the number of labels on the path from PN to N is L, we extract L rules as shown in the following:

$R_1 : label(rootNode, PN) \rightarrow label(PN, N)$
$R_2 : label(rootNode, PN) \bullet (label(PN, N)[1:1]) \rightarrow label(PN, N)[2:L]$
$R_3 : label(rootNode, PN) \bullet (label(PN, N)[1:2]) \rightarrow label(PN, N)[3:L]$
...
$R_L : label(rootNode, PN) \bullet (label(PN, N)[1:L-1]) \rightarrow label(PN, N)[L:L]$

where $label(PN, N)[i : j]$ is the subsequence of $label(PN, N)$ including elements in positions i through j ($i \leq j \leq L$), and '$\bullet$' is the binary operator for concatenating two sequences. If N is the root node, then $label(rootNode, PN)$ becomes the empty sequence $\langle\rangle$. The confidence of R_1 is the same as confidence(N) while the confidences of R_2, R_3, ..., and R_L are 1. Figure 2 shows a part of a rule tree and the rules extracted from the tree. The values in the nodes represent their support values.



(a) A part of a rule tree

```
<> -> A  with confidence 15/50
<> -> BA with confidence 20/50
A  -> BC with confidence 6/15
A  -> D  with confidence 5/15
AB -> C  with confidence 1
B  -> A  with confidence 1
```

(b) Rules extracted from a rule tree

Fig. 2. An example of a rule tree and the corresponding rules.

4 Exact Rule Matching

Exact rule matching is defined as follows: Given a rule tree, a type abstraction hierarchy, and a target head sequence $\vec{q}$, find the rules matched to $\vec{q}$. Our approach for exact rule matching consists of two steps.

Step 1. Search for exactly matched rule head: Using the rule tree as an index structure, we find the rule head $\vec{h}$ that is exactly matched to a target head sequence. Algorithm 1 shows the exact matching algorithm RTI-E (Rule Tree Index for Exact matching). We use the notation $\vec{Q}$ for $C(S(\vec{q}))$. The algorithm traverses the rule tree and returns a pair (N, p) that represents the matched rule head $\vec{h} = label(rootNode, PN) \bullet (label(PN, N)[1 : p])$. The first call to the algorithm has two arguments: rootNode and $\vec{Q}$.

Step 2. Rule extraction from exactly matched rule head: Using the relationship between the exactly matched rule head and its following subsequences, we extract the rules. Let us assume that RTI-E has returned the pair (N, p) and the length of $label(PN, N)$ is L. If $p < L$, then the matched rule is '$label(rootNode, PN) \bullet (label(PN, N)[1 : p]) \rightarrow label(PN, N)[p + 1 : L]$ with confidence 1'. Otherwise, the number of matched rules is the same as the number of children of N. For each child node CN of N, the matched rule is '$label(rootNode, N) \rightarrow label(N, CN)$ with confidence(CN)'.

Input : node N, target head sequence $\vec{Q}$
Output: child node CN, length of matched prefix

Visit the node N;
Select the child node, CN, where $label(N, CN)$ is matched to the prefix of $\vec{Q}$;
Remove the matched prefix from $\vec{Q}$;
if $\vec{Q}$ *becomes empty* **then** return a pair (CN, the length of a matched prefix);
else call RTI-E(CN, $\vec{Q}$);

Algorithm 1: Exact matching algorithm RTI-E

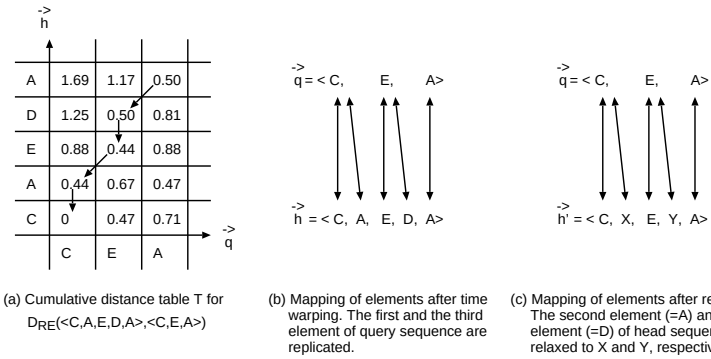
5 Relaxed Rule Matching

Relaxed rule matching is defined as follows: Given a rule tree, a type abstraction hierarchy, and a target head sequence $\vec{q}$, find the least relaxed rules that cover $\vec{q}$.

Since a rule head $\vec{h}$ whose length is not equal to $|\vec{q}|$ may be stretched and relaxed to cover $\vec{q}$, we propose a relaxation-error based time warping distance function $D_{RE}(\vec{h}, \vec{q})$ as a similarity measure of $\vec{h}$ and $\vec{q}$. $D_{RE}(\vec{h}, \vec{q})$ stretches $\vec{h}$ and $\vec{q}$ non-linearly to find the best element mappings that minimize the difference of $\vec{h}$ and $\vec{q}$. Let $\vec{h}[i]$ be mapped to $\vec{q}[j]$. Then, the distance of this mapping is defined as $RE(\vec{h}[i], \vec{q}[j]) - RE(\vec{h}[i])$ where $RE(\vec{h}[i], \vec{q}[j]))$ is the relaxation error of the lowest node containing both $\vec{h}[i]$ and $\vec{q}[j]$, and $RE(\vec{h}[i])$ is the relaxation error of the lowest node containing $\vec{h}[i]$. The detailed description of $D_{RE}(\vec{h}, \vec{q})$ is given in [10]

$D_{RE}(\vec{h}, \vec{q})$ can be calculated efficiently by the dynamic programming technique [2] based on the recurrence relation $r(x, y)$. ($x = 1, ..., |\vec{h}|$, $y = 1, ..., |\vec{q}|$). The final cumulative distance, $r(|\vec{h}|, |\vec{q}|)$, is the amount of relaxation needed for $\vec{h}$ to cover $\vec{q}$. Using the cluster hierarchy (Figure 1), Figure 3 shows the cumulative distance table T for $D_{RE}(\vec{h} = \langle C, A, E, D, A\rangle, \vec{q} = \langle C, E, A\rangle)$ and the element mappings after time warping and relaxation. In the following, we present the two-step approach for relaxed rule matching.

Step 1. Search for the nearest rule head: To generate the least relaxed rules, we first traverse the rule tree to find the rule head $\vec{h}$ that requires the least relaxation to cover a target head sequence $\vec{q}$. The similarity matching algorithm RTI-S (Rule Tree Index for Similarity matching) is given in Algorithm 2. Note that a target head sequence $\vec{q}$ is converted to the compact representation $\vec{Q}$ before beginning the search process. The algorithm maintains three global variables during its execution: $\vec{Q}$, the nearest rule head $\vec{h}$ found so far, and its distance $MinDist$ from $\vec{Q}$. The first call to the algorithm has two arguments: rootNode and emptyTable. RTI-S reduces the search time by applying the branch-pruning approach [11] and by allowing the cumulative distance table to be shared by rule heads that have the same prefix.



(a) Cumulative distance table T for D_{RE}(<C,A,E,D,A>,<C,E,A>)

(b) Mapping of elements after time warping. The first and the third element of query sequence are replicated.

(c) Mapping of elements after relaxation. The second element (=A) and the fourth element (=D) of head sequence are relaxed to X and Y, respectively.

Fig. 3. Cumulative distance table T for $D_{RE}(\vec{h} = \langle C, A, E, D, A\rangle, \vec{q} = \langle C, E, A\rangle)$ and the mapping of elements after time warping and relaxation.

Input : node N, cumulative distance table T

Visit the node N;
for *each child node* CN **do**
  Build a new cumulative distance table $newT$, by adding rows corresponding to $label(N, CN)$ on T;
  Find a nearer rule head $\vec{h}$ from $newT$ and update $MinDist$;
  If further traverse-down the tree is necessary, call RTI-S(CN, $newT$);

Algorithm 2: Similarity matching algorithm RTI-S

Step 2. Rule extraction from the nearest rule head: After finding the rule head $\vec{h}$ most similar to $\vec{Q}$, we generate the least relaxed rules from $\vec{h}$ and its following subsequences. This step begins with extracting the rules from $\vec{h}$ using the method explained in Section 4. Then, we convert symbols of rule heads and bodies into their relaxed symbols according to the mapping of $\vec{h}$ and $\vec{Q}$, and get the compact representations of rules. Finally, the rules having the same head and body are merged and their confidence values are recomputed.

6 Experiments

To study the effectiveness of our proposed methods, we performed experiments on the *random walk* synthetic data sequences [10]. We used the relative minimum support value $RSUP_{min}$ to control the number of discovered rules.

6.1 Rule Discovery

We used the total elapsed time as a performance measure of our rule discovery algorithm. First, we increased the number of sequences from 100 to 10,000 while keeping their average length constant at 200. Then, we changed the average

length of sequences from 100 to 1,000 while maintaining the number of sequences at 500. As shown in Figures 4 and 5, the total elapsed times increase linearly as the number of and the average length of data sequences grow. The figures also show that the linearity is maintained with different $RSUP_{min}$ values.

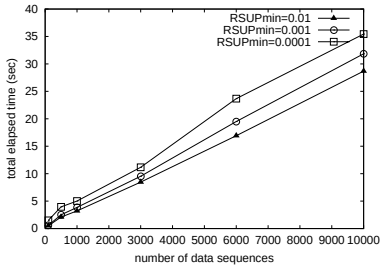


Fig. 4. Total elapsed time for discovering elastic rules with selected numbers of data sequences.

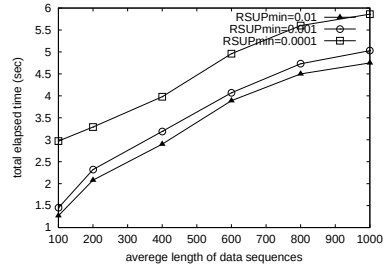


Fig. 5. Total elapsed time for discovering elastic rules with selected average length of data sequences.

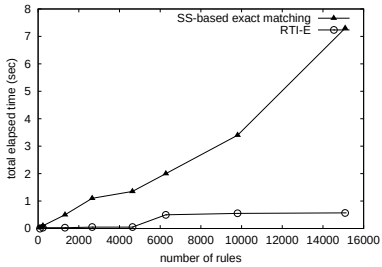


Fig. 6. Performance comparison between sequential scanning and RTI-E for exact rule matching.

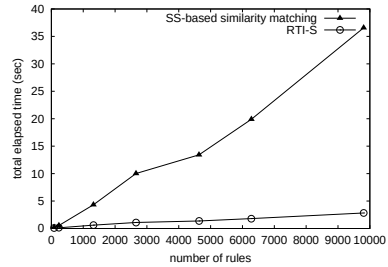


Fig. 7. Performance comparison between sequential scanning and RTI-S for relaxed rule matching.

6.2 Rule Matching

For 500 data sequences with the average length 400, Figure 6 shows the average search times of RTI-E and SS(Sequential- Scanning)-based exact matching algorithm for increasing numbers of rules. The search times of SS-based exact matching algorithm increase linearly with the number of rules while the search times of RTI-E remain relatively constant. Figure 7 shows the average search times of RTI-S and SS-based similarity matching algorithm. The performance gain of RTI-S increases as the number of rules increases.

7 Conclusion

In this paper, we proposed a method to discover elastic rules from sequence databases. We also presented efficient techniques to find matched rules and to derive the least relaxed rules. We focused on data sequences consisted of univariate numeric values. If elements are non-numeric, we employ an encoding scheme that converts non-numeric elements to numeric elements.

Experiments on synthetic data sequences revealed that: 1) our rule discovering algorithm is linear to both the total number of and the average length of data sequences, and 2) our exact and relaxed rule matching algorithms are several orders of magnitude faster than sequential scanning.

References

1. R. Agrawal, and R. Srikant, "Mining Sequential Patterns", *Proc. IEEE ICDE*, 1995.
2. D. J. Berndt, and J. Clifford, "Finding Patterns in Time Series: A Dynamic Programming Approach", *Advances in Knowledge Discovery and Data Mining*, AAAI/MIT, 1996.
3. P. S. Bradley, U. M. Fayyad, and O. L. Mangasarian, "Data Mining: Overview and Optimization Opportunities", *Microsoft Research Report MSR-TR-98-04*, 1998.
4. P. Bieganski, J. Riedl, and J. V. Carlis, "Generalized Suffix Trees for Biological Sequence Data: Applications and Implementation", *Proc. Hawaii Int'l Conf. on System Sciences*, 1994.
5. W. W. Chu, and K. Chiang, "Abstraction of High Level Concepts from Numerical Values in Databases", *Proc. of AAAI Workshop on Knowledge Discovery in Databases*, 1994.
6. G. Das, K. Lin, H. Mannila, G. Renganathan, and P. Smyth, "Rule Discovery from Time Series", *Proc. International Conference on Knowledge Discovery and Data Mining*, 1998.
7. U. M. Fayyad, "Mining Databases: Toward Algorithms for Knowledge Discovery", *Data Engineering Bulletin 21(1)*, 1998.
8. E. M. McCreight, "A Space-Economical Suffix Tree Construction Algorithm", *Journal of the ACM*, Vol. 23, No. 2, 1976
9. H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovering Frequent Episodes in Sequences", *Proc. International Conference on Knowledge Discovery and Data Mining*, 1995.
10. S. Park and W. W. Chu, "Discovering and Matching Elastic Rules from Sequence Databases", *UCLA Technical Report UCLA-CS-TR-200012*, 2000.
11. S. Park, W. W. Chu, J. Yoon, and C. Hsu, "Efficient Searches for Similar Subsequences of Different Lengths in Sequence Databases", *Proc. IEEE ICDE*, 2000.
12. L. Rabinar, and B. Juang. *Fundamentals of Speech Recognition*, Prentice Hall, 1993.
13. G. A. Stephen, *String Searching Algorithms*, World Scientific Publishing, 1994.
14. J. T.-L. Wang, G.-W. Chirn, T. G. Marr, B. Shapiro, D. Shasha, and K. Zhang, "Combinatorial Pattern Discovery for Scientific Data: Some Preliminary Results", *Proc. ACM SIGMOD*, 1994.