Mining Semi-Structed Data by Path Expressions

Taniguchi, Katsuaki
Department of Informatics, Kyushu University

Sakamoto, Hiroshi
Department of Informatics, Kyushu University

Arimura, Hiroki
Department of Informatics, Kyushu University

Shimozono, Shinichi
Department of Informatics, Kyushu University

他

出版情報：DOI Technical Report. 190, 2001-05. Department of Informatics, Kyushu University
バージョン：
権利関係：

Mining Semi-Structured Data by Path Expressions

Katsuaki Taniguchi[†], Hiroshi Sakamoto[†], Hiroki Arimura[†*], Shinich Shimozono[‡] and Setsuo Arikawa[†]

[†]Department of Informatics, Kyushu University
Fukuoka 812-8581, Japan
{k-tani, hiroshi, arim, arikawa}@i.kyushu-u.ac.jp
[*]PRESTO, Japan Science Technology Co., Japan
[‡]Department of Artificial Intelligence, Kyushu Institute of Technology
Iizuka 820-8502, Japan
sin@ai.kyutech.ac.jp

Abstract

A new data model for filtering semi-structured texts is presented. Given positive and negative examples of HTML pages labeled by a labelling function, the HTML pages are divided into a set of paths using the XML parser. A path is a sequence of *element nodes* and *text nodes* such that a text node appears in only the tail of the path. The labels of an element node and a text node are called a *tag* and a *text*, respectively. The goal of a mining algorithm is to find an interesting pattern, called *association path*, which is a pair of a tag-sequence t and a word-sequence w represented by the *word-association pattern* [1]. An association path (t, w) *agrees with* a labelling function on a path p if t is a subsequence of the tag-sequence of p and w matches with the text of p iff p is in a positive example. The importance of such an associate path α is measured by the *agreement* of a labelling function on given data, i.e., the number of paths on which α agrees with the labelling function. We present a mining algorithm for this problem and show the efficiency of this model by experiments.

Correspondence:
Dr. Hiroshi Sakamoto
Department of Informatics, Kyushu Univ.
hiroshi@i.kyushu-u.ac.jp
phone: +81-92-642-2693, fax: +81-92-642-2698

1 Introduction

In the *information extraction*, it is one of the central problems in Web mining to detect the occurrences or the regions of useful texts. In case of the Web data, this problem is particularly difficult because we can not represent a rich logical structure by the limited tags of the HTML. The framework of *wrapper induction* introduced by Kushmerick [13] is a new approach to handle this difficulty. The most interesting result of his study is to show the effectiveness and efficiency of simple wrappers with string delimiters in the information extraction tasks. Together with his work, we can find other extracting models, for example, in [8, 9, 10, 11, 15, 17].

The target class, called *HTML pages*, of the wrapper induction model is restricted such that a page is defined by a finite repetition of a sequence of *attributes*. The attributes are the data which an algorithm has to extract. In a learning model, a learning algorithm takes an input of labeled examples such that the labels indicate whether they are *positive* data or *negative* data. The strategy is useful to learn a *concept* for the wrapper class.

However, in case that a concept class is hard to learn by a small number of examples, the model may not be effective. This difficulty is critical in the point of implementation since the labelling examples are actually made by human inspection. Thus, we would like to present a mining model to decide which potion of a given data is important and an automatic process to construct a large labelling sample.

The aim of this paper is to find rules for filtering semi-structured texts according to users interests. An HTML/XML file can be considered as an ordered labeled tree. We assume that each node is either an *element node* and a *text node*. Each node has two types of labels called the *name* and *value*. An element node corresponds to a tag. The name of the node is the tag name like `<HTML>`, `<br>`, and `<a>`, and the value of the node is empty. A text node corresponds to a potion of a plain text in an HTML and the name is the reserved string $\sharp Text$, respectively. The value of a text node is the text.

A filtering rule is a sequence $s = \langle \alpha_1, \ldots, \alpha_k, \beta \rangle$, where α_i is a tag name, β is a *word-association pattern* [1] which is a string consisting of several words and the wild card $*$. A word-association pattern *matches with a string* if there is a possible substitution for all $*$. Given the s and a semi-structured text, using an XML parser, we can easily construct the tree structure and decompose the tree into the set P of paths. Each path contains at most one text node in the tail. The semantics of the filtering rule s for P is defined as follows. For each $p \in P$, s matches with p if $\alpha_1, \ldots, \alpha_k$ is a subsequence of the sequence of tag names of p and the tail of p is a text node such that β matches with the value of the node.

Such a filtering rule is considered as a simple decision tree to extract texts from paths in HTML trees. Each α_i represents a test on a node. Unless the test is failed, we continue the test to the next test α_{i+1}. Finally, the value of the text node is extracted according to the pattern β. In other words, this rule is a pair of *tag patterns* and association patterns $\langle \alpha, \beta \rangle$, where a tag pattern is a

sequence $\alpha = (\alpha_1, \ldots, \alpha_k)$ of tag names such that these tags frequently appear in positive examples together with the association pattern. Such a filtering rule is called an *association path* in this paper. We can use this notion for a measure to decide the importance of keyword in a text. We show the efficiency of the association paths by experiments.

This paper is organized as follows. In Section 2, we define the data model for HTML pages, HTML trees, and path expressions. In Section 3, we review the definition of the word-association pattern in [1] and formulate the mining problem, called ASSOCIATION PATH problem, of this paper. Next we describe a mining algorithm which finds an association path for given a large collection of HTML texts. In Section 4, we show several experimental results. In the first experiment, the set of positive examples is a collection of HTML texts containing a keyword "TSP" and the set of negatives is that containing "NP". These keywords mean the *travelling sealsman problem* and *NP-optimization problem* on the computational complexity theory, respectively. The aim is to find an association path to characterize the notion TSP comparing to NP. In this experiment, the algorithm found some interesting association paths. For the next experiment, we choose the keyword "DNA" for positive examples. Compared to the first result, the algorithm found few interesting paths. In Section 5, we conclude this study.

2 The Data Model

In this section, we introduce the data model considered in this paper. First, we begin with the notations used in this paper. $I\!N$ denotes the set of all nonnegative integers. An *alphabet* Σ is a set of finite symbols. A finite sequence $\langle a_1, \ldots, a_n \rangle$ of elements in Σ is called *string* and it is denoted by $w = a_1 \cdots a_n$ for short. The *empty string* of length zero is ε. The set of all strings is denoted by Σ^* and let $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. For string w, if $w = \alpha\beta\gamma$, then the strings α and β are called a *prefix* and a (*suffix*) of w, respectively. For a string s, we denote by $s[i]$ with $1 \leq i \leq |s|$ the i-th symbol of s, where $|s|$ is the length of s.

For an HTML file, the HTML trees are the ordered node-labeled trees defined as follows. For each tree T, the set of all nodes of T is a finite subset of $I\!N$, where the 0 is the root. A node is called a *leaf* if it has no child and otherwise called an *internal node*. If nodes $n, m \in I\!N$ have the same parent, then n and m are *siblings*. A sequence $\langle n_1, \ldots, n_k \rangle$ of nodes of T is called a *path* if n_1 is the root and n_i is the parent of n_{i+1} for all $i = 1, \ldots, k-1$. For a path $p = \langle n_1, \ldots, n_k \rangle$, the number k is called *the length of* p and the node n_k is called *the tail of* p.

With each node n, the pair $NL(n) = \langle N(n), V(n) \rangle$, called *the node label of* n, is attached, where $N(n)$ and $V(n)$ are strings called the *node name* and *node value*, respectively. If $N(n) \in \Sigma^+$ and $V(n) = \varepsilon$, then the node n is called the *element node* and the string $N(n)$ is called the *tag*. If $N(n) = \sharp Text$ for the reserved string $\sharp Text$ and $V(n) \in \Sigma^+$, then n is called the *text node* and the $V(n)$ called the *text value*. We assume that every node $n \in I\!N$ is categorized to the element node or text node.

If a page P contains a beginning tag of the form <*tag*> and P contains no ending tag corresponding to it. Then, the tag <*tag*> is called an *empty tag in* P. If a page P contains a string of the form $t_1 \cdot w \cdot t_2$ such that t_1, t_2 are either beginning or ending tags and w is a string not containing any tag, then the string w is called a *text in* P.

An HTML file is called a *page*. A page P corresponds to an ordered labeled tree. For the simplicity, we assume that the P contains no comments, which is any string beginning the string `<!-` and ending the string `->`.

Definition 1 For a page P, we define the HTML tree P_t recursively as follows.

1. If P contains an empty tag of the form <*tag*>, then P_t has the element node n such that it is a leaf of P, $N(n) = tag$, and $V(n) = \varepsilon$.
2. If P contains a text w, then P_t has the text node n such that it is a leaf P, $N(n) = \sharp Text$, $V(n) = w$.
3. If P contains a string of the form `<tag>`s`</tag>` for a string $s \in \Sigma^*$, then P_t has the subtree $n(n_1, \ldots, n_k)$, where $N(n) = tag$, $V(n) = \varepsilon$ and $n_1, \ldots, n_k$ are the roots of the trees $t_1, \ldots, t_k$ which are obtained from the w by recursively applying the above 1, 2 and 3.

Next we define the functions to get the node names, node values, and HTML attributes from given nodes and HTML trees defined above. These functions are useful to explain the algorithms in the next section. These functions return the values indicated below and return *null* if such values do not exist.

- $Parent(n)$: The parent of the node $n \in I\!N$.
- $ChildNodes(n)$: The sequence of all children of n.
- $Name(n)$: The node name $N(n)$ of n.
- $Value(n)$: The concatenation $V(n_1) \cdots V(n_k)$ of all values of the leaves $n_1, \ldots, n_k$ of the subtree rooted at n in the left-to-right order.

Recall that $V(n)$ is not empty only if n is text node. Thus, $Value(n)$ is equal to the concatenation of values of all text nodes below n. Let P_t be an HTML tree for a page P and let $N = \{0, \ldots, n\}$ be the set of nodes in P_t. For nodes $i, j \in N$, if there is a sequence $p_{i,j} = \langle i_1, \ldots, i_k \rangle$ of nodes in N such that $i_1 = i$, $i_k = j$, and $i_\ell = Parent(i_{\ell+1})$ for all $1 \leq \ell \leq k-1$, then the $p_{i,j}$ is called the *path* from i to j. If i is the root, then $p_{i,j}$ is denoted by p_j for short. For each path $p = \langle i_1, \ldots, i_k \rangle$ of P_t, we also define the following useful notations.

- $Name(p)$: The sequence $\langle Name(i_1), \ldots, Name(i_k) \rangle$.
- $Value(p)$: $V(n_k)$.

Definition 2 Let P_t be an HTML tree over the set N of nodes. Let $p = \langle i_1, \ldots, i_n \rangle$ be a path of P_t and let $Name_t = \{Name(n) \mid n \in N\}$. A sequence $\alpha = \langle name_1, \ldots, name_m \rangle$, $(name_i \in Name_t)$ is called a *path expression over* $Name_t$. It is called that *the* α *matches with the* p if there exists a subsequence $j_1, \ldots, j_m$ of p such that $Name(j_\ell) = name_\ell$ for all $1 \leq \ell \leq m$.

In the next section, we define a measure of the matching of the path expressions with the paths of HTML trees. We also define the finding problem of a path expression to maximize the measure.

3 Mining HTML texts

In this section we first define the problem to find an expression, called an *association pattern*, for filtering semistructured texts. The pattern is a pair of a *word-association pattern* and a path expression. The semantics of the patterns is defined by the matching semantics of the word-association patterns and the path expressions.

3.1 The problem

A *word-association pattern* [1] π over Σ is a pair $\pi = (p_1, \ldots, p_d; k)$ of a finite sequence of strings in Σ^* and a parameter k called *proximity* which is either a nonnegative integer or infinity. A word-association pattern π *matches* a string $s \in \Sigma^*$ if there exists a sequence $i_1, \ldots, i_d$ of integers such that every p_j in π occurs in s at the position i_j and $0 \leq i_{j+1} - i_j \leq k$ for all $1 \leq j < d - 1$. The notion (d, k)*-pattern* refers to a d-word k-proximity word-association pattern $(p_1, \ldots, p_d; k)$.

Let $S = \{s_1, \ldots, s_m\}$ be a finite set of strings Σ^* and let ψ be a labeling function $\psi : S \to \{0, 1\}$. Then, for a string $s \in S$, we say that a word-association pattern π *agrees with* ψ *on* s if π matches s iff $\psi(s) = 1$.

Given (Σ, S, ψ, d, k) of an alphabet Σ, a finite set $S \subset \Sigma^*$ of strings, a labeling function $\psi : S \to \{0, 1\}$, and positive integers d and k, the problem MAX AGREEMENT BY (d, k)-PATTERN [1] is to find a (d, k)-pattern π such that it maximizes the *agreement* of ψ, i.e., the number of strings in S on which π agrees with ψ.

Definition 3 An *association path* is an expression of the form $pe\#\pi$, where the α is a path expression such that its tail is $\sharp Text$, the π is a word-association pattern, and the $\#$ is the special symbol not belonging to any α and π. Let $p = \alpha\#\pi$ be an association pattern and p' be a path in a tree. It is said that *the* p *matches the* p' if α matches p' and π matches $Value(p')$.

For a finite set T of HTML trees, let

$$Text_T = \{\langle Name(p), Value(p) \rangle \mid p \text{ is a path of } t \in T,\ Value(p) \neq \varepsilon\}$$

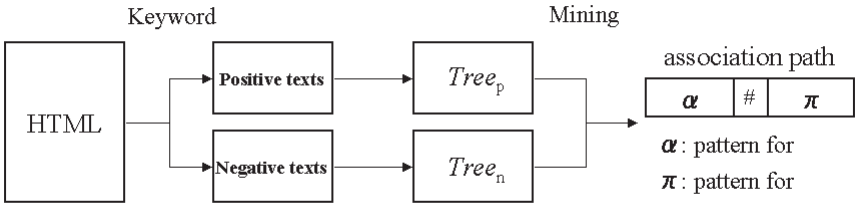
The intuitive meaning of p appearing in $Text_T$ is a path p of an HTML tree such that the tail of p is a text node. Let $Name_T$ be the set of $Name(p)$ and let $Value_T$ be the set of $Value(p)$ in $Text_T$.

Definition 4 ASSOCIATION PATH
An instance is $(\Sigma, Text_T, \psi, d, k)$ of an alphabet Σ, a set $Text_T$ of pairs for a finite set T of HTML trees, a labeling function $\psi : Value_T \to \{0, 1\}$, and positive integers d, k. A solution is an association path $\alpha\#\pi$. The string π is a (d, k)-pattern for a solution of the max agreement problem for input $(\Sigma, Value_T, \psi, d, k)$. The string α is a (d, k)-pattern for a solution of the max agreement problem for input $(\Sigma, Name_T, \psi', d, k)$, where ψ is defined by $\psi'(Name(p)) = 1$ iff $\psi(Value(p)) = 1$. The goal of the problem is to maximize the sum of the agreements of ψ and ψ' over all association paths $\alpha\#\pi$.

3.2 The algorithm

To find association paths, the data of HTML texts are transformed to path expressions as follows. Given a large set S of HTML texts, it is divided into two disjoint sets S_1 and S_2 by a labeling function. The labeling function is considered as a keyword or phrase by a user, i.e., any text in S is labeled by 1 if it contains the keyword and labeled by 0 otherwise. Next all texts in S_1 and S_2 are parsed to HTML trees and let Pos be the set all paths from S_1 and Neg be the set of all paths from S_2. Figure 1 shows the process of our algorithm briefly.

Figure 1: The process of mining algorithm.



Then, the mining algorithm is described as follows.

Algorithm **Path-Find**$(\Sigma, Text, \psi, d, k)$
/* Input: a set of HTML pages P over Σ, a labeling function ψ, non negative integers d, k */
/* Output: a solution of ASSOCIATION PATH for the input*/

1. Let P_1 be the set of all pages in P labeled by 1 and let $P_2 = P \setminus P_1$. For the set T_1 of HTML trees of P_1, compute the set Pos of all paths of trees in P_1 and the set Neg of all paths of all trees in P_2.

2. Let $Pos = \{p_i \mid 1 \leq i \leq m\}$ and $Neg = \{q_j \mid 1 \leq j \leq n\}$ $(m, n \geq 0)$. Compute the sets $Name_{Pos} = \{Name(p) \mid p \in Pos\}$, $Value_{Pos} = \{Value(p) \mid p \in Pos\}$, $Name_{Neg} = \{Name(q) \mid q \in Neg\}$, and $Value_{Neg} = \{Value(q) \mid q \in Neg\}$.

3. Find a (d, k)-pattern π of the max agreement problem for $\langle Value_{Pos}, Value_{Neg} \rangle$, and find a (d, k)-pattern α of the max agreement problem for $\langle Name_{Pos}, Name_{Neg} \rangle$.

4. Output the pattern $\alpha \# \pi$ which maximizes the sum of the agreement of α and π.

We estimate the running time of the **Path-Find**. This algorithm finds an association path for only the paths whose tails are the text nodes, i.e., the paths of the form $p = \langle n_1, \ldots, n_k \rangle$, the n_i $(1 \leq i \leq k-1)$ is an element node and the n_k is a text node. Thus, for such paths p, we regard the mining problem as the problem to find two phrases α from the strings $Name(p)$ and β from the strings $Value(p)$ for constant parameters d of the number of phrases of texts and k of the distance of phrases.

If the maximum number of phrases in a pattern is bounded by a constant d then the max agreement problem for (d, k)-patterns is solvable by *Enumerate-Scan* algorithm [19], a modification of a naive generate-and-test algorithm, in $O(n^{d+1})$ time and $O(n^d)$ scans although it is still too slow to apply real world problems.

Adopting the framework of optimized pattern discovery, we have developed an efficient algorithm, called *Split-Merge* [1], that finds all the optimal patterns for the class of (d, k)-patterns for various statistical measures including the classification error and information entropy.

The algorithm quickly searches the hypothesis space using dynamic reconstruction of the content index, called a *suffix array* with combining several techniques from computational geometry and string algorithms.

We showed that the Split-Merge algorithm runs in *almost linear time in average*, more precisely in $O(k^{d-1} N (\log N)^{d+1})$ time using $O(k^{d-1} N)$ space for nearly random texts of size N [1]. We also show that the problem to find one of the best phrase patterns with arbitrarily many strings is MAX SNP-hard [1]. Thus, we see that there is no efficient approximation algorithm with arbitrary small error for the problem when the number d of phrases is unbounded.

4 Experimental results

In this section, we show the experimental results. The text data is a collection from the *ResearchIndex* [1] which is a scientific literature digital library. A positive data is the set Pos of HTML pages containing the keyword "TSP" and a negative data is the set Neg of HTML pages containing the keyword "NP". The set Neg consists of many topics of computational complexity problems and Pos is concerned with one of the most popular NP-hard problems *Travelling Salesman Problem* not properly contained in Neg. The aim of this experiment is to find an association path which characterizes TSP with NP.

By this experiment on the collection of 8.4MB, the algorithm **Path-Find** finds the best 600 patterns at the entropy measure in seconds for $d = 2$ and three minutes for $d = 3$ with $k = 10$ words using 200 mega-bytes of main memory on IBM PC (PentiumIII 600 MHz, gcc++ on Windows98). The following figure is the result.

Figure 2: The association paths found in the experiments, which characterize the Web pages on the TSP problem from these on NP-optimization problm. The parameters are $(2, 10)$ for (d, k), where α is a path and π is a phrase.

Rank	Association path $\alpha\#\pi$
5	< i font p body html> # <tsp >
38	< i font p body html> # <for the >
90	< i font p body html> # <the tsp >
171	< i font p body html> # <local search >
213	< i font p body html> # <traveling >
276	< i font p body html> # <tsp and other >
394	< i font p body html> # <euclidean tsp >
455	< i font p body html> # <other geometric problems >
552	< i > # <approximation schemes for euclidean >

Our system found several interesting association paths which may be difficult for human user to find by inspection. The phrase "local search" in Rank 171 indicates the *local search heuristics* for TSP such as [14]. In this path, the tag `<i>` and `<font>` (font style and size) in the left hand side indicates the

[1] http://citeseer.nj.nec.com/

importance of the phrase `<local search>` in the right hand side. The phrase "tsp and other" in Rank 276 is a substring of the title of the outstanding paper written by Arora [2] in 1996 on the approximation algorithm for Euclidean TSP. The *euclidean* graph is an important *geometric* structure to construct an *approximation* algorithm for TSP. These keywords appear in Rank 394, 455, and 552, respectively.

Finally, we show other experimental results. The positive sample is a set of HTML pages containing the keyword "DNA" and the negative sample is the same to above experiment. By this experiment on the collection of 9.3MB, the algorithm finds the best 600 patterns at the entropy measure in seconds for $d = 2$ with $k = 10$. The result is shown in Figure 3. Our system found few of association paths containing interesting keywords like "sequence" "computer" and "molecular". In this result, several paths containing the *ancer tag*. Unfortunately, interesting keywords are not found in such paths.

Figure 3: Other result of the experiments for the DNA from NP-optimization problem. The parameters are also $(2, 10)$ for (d, k), where α is a path and π is a phrase.

Rank	Association path $\alpha\#\pi$
23	`<a > # <dna >`
136	`<i font p body html > # <dna sequences >`
199	`<i font p body html > # <molecular >`
360	`<i font p body html > # <computer >`
395	`<a > # <computation >`
444	`<a body html > # <computing >`

5 Conclusion

We introduced a new method for mining from HTML texts and present an algorithm to find an association rule which is a pair of association patterns over tag sequences and text sequences. By experiments on HTML data of scientific literature, the algorithm found interesting association paths from positive and negative examples on the traveling salesman problem and the other NP optimization problems.

References

[1] Shimozono, S., Arimura, H., and Arikawa, S. Efficient discovery of optimal word-association patterns in large text databases. *New Generation Computing* 18:49-60, 2000.

[2] Arora, S. Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. *Proc. 37th IEEE Symposium on Foundations of Computer Science*, 2-12, 1996.

[3] Abiteboul, S., Buneman, P., and Suciu, D. Data on the Web: From relations to semistructured data and XML, Morgan Kaufmann, San Francisco, CA, 2000.

[4] Angluin, D. Queries and concept learning. *Machine Learning* 2:319–342, 1988.

[5] Buneman, P., Davidson, S., Hillebrand, G., and Suciu, D. A query language and optimization techniques for unstructured data. *University of Pennsylvania, Computer and Information Science Department, Technical Report MS-CIS* 96-09, 1996.

[6] Cohen, W. W. and Fan, W. Learning Page-Independent Heuristics for Extracting Data from Web Pages, *Proc. WWW-99.* 1999.

[7] Craven, M., DiPasquo, D., Freitag, D., McCallum, A., Mitchell, T., Nigam, K., and Slattery, S. Learning to construct knowledge bases from the World Wide Web, *Artificial Intelligence* 118:69–113, 2000.

[8] Freitag, D. Information extraction from HTML: Application of a general machine learning approach. *Proc. the 15th National Conference on Artificial Intelligence*, 517-523, 1998

[9] Grieser, G., Jantke, K. P., Lange, S., and Thomas, B. A unifying approach to HTML wrapper representation and learning, *Proc. the 3rd International Conference, DS2000*, Lecture Notes in Artificial Intelligence 1967:50–64, 2000.

[10] Hammer, J., Garcia-Molina, H., Cho, J., and Crespo, A. Extracting semistructured information from the Web. *Proc. Workshop on Management of Semistructured Data*, 18–25, 1997.

[11] Hsu, C.-N. Initial results on wrapping semistructured web pages with finite-state transducers and contextual rules. *Proc. 1998 Workshop on AI and Information Integration*, 66–73, 1998.

[12] Kamada, T. Compact HTML for small information appliances. *W3C NOTE 09-Feb-1998.* `www.w3.org/TR/1998/NOTE-compactHTML-19980209`, 1998.

[13] Kushmerick, N. Wrapper induction: efficiency and expressiveness. *Artificial Intelligence* 118:15–68, 2000.

[14] Lin, S., and Kernighan, B. W. An effective heuristic algorithm for the travelling salesman problem. *Operations Research* 21:498-516, 1973.

[15] Muslea, I., Minton, S., and Knoblock, C. A. Wrapper induction for semistructured, web-based information sources. *Proc. Conference on Automated Learning and Discovery*, 1998.

[16] Sakamoto, H., Arimura, H., and Arikawa, S. Identification of tree translation rules from examples. *Proc. the 5th International Colloquium on Grammatical Inference*, LNAI 1891:241–255, 2000.

[17] Thomas, B. Anti-unification based learning of T-Wrappers for information extraction, *Proc. AAAI Workshop on Machine Learning for IE*, 15–20, AAAI, 1999.

[18] Valiant, L. G. A theory of the learnable. *Comm. ACM* 27:1134–1142, 1984.

[19] Wang, J. T., Chirn, G. W., Marr, T. G., Shapiro, B., Shasha, D., and Zhang, K. Combinatorial pattern discovery for scientific data: Some preliminary results. *Proc. SIGMOD'94*, 115-125, 1994.