An Optimal Algorithm for Querying Tree Structures and its Applications in Bioinformatics

Hsiao-Fei Liu[1], Ya-Hui Chang[2]*, and Kun-Mao Chao[1]

[1]Department of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan, 106
{r92019, kmchao}@csie.ntu.edu.tw

[2]Department of Computer Science
National Taiwan Ocean University, Keelung, Taiwan, 202
yahui@cs.ntou.edu.tw

Abstract

Trees and graphs are widely used to model biological databases. Providing efficient algorithms to support tree-based or graph-based querying is therefore an important issue. In this paper, we propose an optimal algorithm which can answer the following question: "Where do the root-to-leaf paths of a rooted labeled tree Q occur in another rooted labeled tree T?" in time $O(m + Occ)$, where m is the size of Q and Occ is the output size. We also show the problem of querying a general graph is NP-complete and not approximable within n^k for any $k < 1$, where n is the number of nodes in the queried graph, unless P = NP.

1 Introduction

Trees and graphs have been widely used to model data with complicated structures or relationships, such as XML, Web, and structured documents [11]. The applications in life sciences also arise naturally, such as the biological databases representing molecular graphs, taxonomy, and pathways. The issue of querying tree-based or graph-based databases efficiently is therefore very important and attracts a lot of attention [3, 4, 6, 12].

In this research, we first investigate how to query tree-based biological databases, such as the newly released KEGG Glycan database for glycan structures which contains thousands of tree-structured entries [1]. More formally, given a database composed of a rooted labeled tree[1] T and a query composed of a rooted labeled tree Q, we intend to identify where each root-to-leaf path of Q occurs in T. This problem is named as the $TRPF$ problem (Tree version of Root-to-leaf Path Finding problem). An algorithm with optimal querying time for the $TRPF$ problem is proposed.

We also consider the situation where the database is represented as a complex graph, such as the regulatory pathway. It can be first converted into a directed labeled graph by the following steps: 1) Determine the similarity between genes by the sequence alignment approach [8] and give the same label to similar genes. 2) Construct a corresponding pseudo-node $\bar{g}$ labeled with $\neg L(g)$ for each gene g labeled with $L(g)$. 3) If there exists an inhibition edge from g_1 to g_2, delete that inhibition edge and add an activation edge from g_1 to $\bar{g_2}$. Figure 1 shows an example of converting part of the regulatory pathway "map04210hsa" in KEGG [9] to a directed labeled graph.[2] Based on a set of regulatory pathways which have been represented as directed labeled graphs, some interesting paths can be extracted and stored collectively as a tree, as illustrated in Figure 2. For an unfamiliar pathway, we could get more insight on its structure and functionality by pointing out where those paths stored in the collected tree occur in that pathway.

The application stated above motivates the more

*To whom all correspondence should be sent.

[1]If the database is a forest, we can attach a pseudo root to connect all the trees.

[2]CASP2, CASP7, CASP8 are assumed to be similar and labeled with C. The other nodes are labeled with their first character. Pseudo-nodes without edges are omitted in this figure.

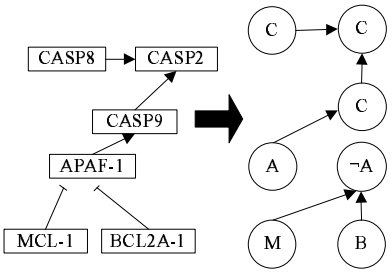

Figure 1: Representing a pathway as a directed labeled graph.

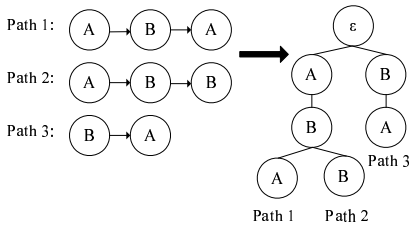

Figure 2: Representing a set of paths as a tree.

general problem, where the database to search is a directed labeled graph G, and we want to identify where the root-to-leaf paths of a rooted labeled tree Q occur in G. This is named as the Root-to-leaf Path Finding (RPF) problem. We show this problem is NP-complete and is not approximable within n^k for any $k < 1$, where n is the number of nodes in G, unless P = NP. Since the RPF problem is hard in general, a feasible solution in practice might be converting the RPF problem to a slightly-less-accurate $TRPF$ approximation by considering the spanning tree of the queried graph.

The rest of this paper is organized as follows. Section 2 discusses some related researches. The formal definitions of the RPF and $TRPF$ problems are given in Section 3. Section 4 describes an algorithm for the $TRPF$ problem and Section 5 proves the hardness of the RPF problem. Section 5 concludes the paper with a few remarks.

2 Related Work

Multi-disciplinary research results are relevant to this research, and we describe a few here. The algorithm proposed in this paper is mainly based on the approach in [10], which applies the suffix tree for efficient document retrieval. The well-established genomic databases represent data as a sequence of codes and querying is supported by sequence alignment algorithms [8]. Powerful query languages are studied for next-generation database applications, such as in the mobile environment [2].

There is also a huge amount of research results in the field of XML query processing which is related to our research, since the XML document and the XML query are usually represented as trees. However, their researches differ from ours in several ways due to the characteristics of XML. For example, we allow the root of the path in the query tree to match any point of the data tree, but the simple path in the XML query tree needs to match the root of the XML data tree. We will only discuss the research results of some representative papers in the following for comparison.

The Index Fabric indexing structure is proposed in [4] to accelerate the searching of root-to-leaf paths in the XML data tree which satisfy the query. Their method is based on layered Patricia tries to efficiently handle a large amount of disk-based data. The ViST indexing structure proposed in [12] is based on the notation of suffixes and uses the tree structure as the unit of querying to avoid join operations. Some researchers represent an XQuery expression as the generalized tree pattern (GTP), and the problem of evaluating an XQuery expression is reduced to the problem of finding matches for its GTP representation [3]. The time complexity of evaluating XPath is discussed in [6]. The full-fledged XPath 1.0 expressions could be processed in time $O(|D|^4 * |Q|^2)$, where $|D|$ is the size of the database and $|Q|$ is the size of the query, and the most common *Core XPath Fragment* can be efficiently processed in time $O(|D| * |Q|)$.

3 Preliminaries

We define the problem formally and provide the background knowledge for solving the problem.

Given a finite alphabet $\Sigma = \{l_1, l_2 \dots l_{|\Sigma|}\}$, we call $G = (V, E, L)$ a directed labeled graph if (V, E) is a directed graph and $L : V \rightarrow \Sigma$ is a labeling function. Similarly, we call $T = (V, E, r, L)$ a rooted labeled tree if (V, E, r) is a rooted tree and L is a labeling function. Both the directed labeled graph and the rooted labeled tree are called directed labeled structures. Given two directed labeled structures G_1 and G_2, we say a path p' of G_2 occurs at G_1's node v iff there exists a simple path p of G_1 starting from v such that the concatenation of labels on p is the same as p'. For convenience, we let $labels(p)$ denote the concatenation of labels on a simple path p. If v is a node of a rooted labeled tree, $\sigma(v)$ denotes the path from the root to v.

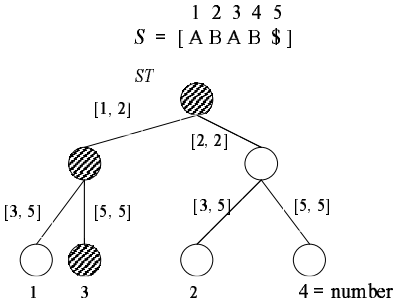

Figure 3: An example of a suffix tree.

Problem 1 *The Root-to-leaf Path Finding* (RPF) *problem: Given a directed labeled graph* $G = (V, E, L)$ *and a rooted labeled tree* Q *with leaves* $(l_1 \cdots l_\ell)$*, the goal is to output* $(Occ_1 \cdots Occ_\ell)$*, where* $Occ_i = \{v | \sigma(l_i)$ *occurs in* G *at node* $v \in V\}$ *for* $i = 1 \cdots \ell$*.*

Problem 2 *The Tree version of the RPF* ($TRPF$) *problem: Given a rooted labeled tree* $T = (V, E, L, r)$ *and a rooted labeled tree* Q *with leaves* $(l_1 \cdots l_\ell)$*, the goal is to output* $(Occ_1 \cdots Occ_\ell)$*, where* $Occ_i = \{v | \sigma(l_i)$ *occurs in* T *at node* $v \in V\}$ *for* $i = 1 \cdots \ell$*.*

The query Q in both problems is in the form of a rooted labeled tree. The difference is that the database to search is a directed labeled graph in *Problem 1*, and is a rooted labeled tree in *Problem 2*. In the following, we introduce the data structure used to solve the $TRPF$ problem.

Definition 1 *Let* S *be a string ended with a special symbol "$" which is not in the alphabet. A suffix tree* ST *for* S *is a rooted tree with* $|S| - 1$ *leaves numbered from 1 to* $|S| - 1$*, and each edge of* ST *is labeled with a pair of integers such that the following two conditions are satisfied: (1) If* (g, h) *and* (i, j) *are the labels of two edges out of the same node,* $S[g] \neq S[i]$*. (2) For each leaf node* l *of* ST *with the number* i*, if* $(i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k)$ *are the edge-labels on the path from the root to* l*, then the concatenation of* $S[i_1 \ldots j_1], S[i_2 \ldots j_2], \ldots, S[i_k \ldots j_k]$ *spells out* $S[i \ldots |S|]$*.*

Figure 3 shows the suffix tree for the string [A B A B $], where the path from the root to the leaf numbered with k will correspond to the k_{th} suffix of S. For example, the shaded path corresponds to the third suffix of S, *i.e.*, [A B $]. A linear time algorithm of constructing a suffix tree for an input string could be found in [7].

4 Algorithms for $TRPF$

Now we introduce the algorithms for solving TRPF. **Algorithm Preprocessing** constructs the suffix tree with indexing structures based on the input tree in $O(n^2)$, where n is the number of nodes in the input. **Algorithm Querying** can then identify the paths of the query tree in the suffix tree in $O(m + Occ)$, where m is the number of nodes in Q and Occ is the output size. We will first explain the algorithms and then analyze the time complexity.

4.1 Preprocessing

The queried tree T is first augmented to T' as in Figure 4.[3] A suffix tree ST will be built based on the concatenation of all root-to-leaf paths of T', and each leaf node ℓ of ST is "colored" with the positive integer $\underline{k}$ if ℓ corresponds to the path of T' starting from the node with ID $\underline{k}$. All the colors on leaves of ST are collected into an array C from the leftmost leaf to the rightmost leaf, and each leaf is labeled by $[i, i]$ if its color is represented by the i_{th} entry of C. By *depth-first* traversal of ST, we then label each internal node with $[l, r]$, if its leftmost descendant leaf has the label $[l, l]$ and the rightmost descendant leaf has the label $[r, r]$. The last step is to build an indexing structure on C, such that for any interval $[l, r]$, we can output the set of distinct colors, *i.e.*, distinct node ID's, in $C[l, r]$ efficiently. The detailed steps of the algorithm are listed as follows:

Algorithm PREPROCESSING
Input: A rooted labeled tree T on Σ .
Output: (ST, S, C).
0 for each leaf ℓ of T
1 add a new child with label \$ to ℓ;
2 end for
3 $T' \leftarrow T$;
4 $P \leftarrow$ concatenate all root-to-leaf paths of T';
5 $S \leftarrow labels(P)$;
6 $ST \leftarrow$ build the suffix tree of S;
7 for each leaf ℓ of ST /* from left to right*/
8 color ℓ with $P[num]$ if $\ell's$ number is num;
9 $C[i] \leftarrow P[num]$ /* i has initial value 1*/
10 assign label $[i, i]$ to ℓ;
11 $i \leftarrow i + 1$;
12 end for
13 for each internal node v
14 $[l, l] \leftarrow v's$ left-most-descendant-leaf's label;
15 $[r, r] \leftarrow v's$ right-most-descendant-leaf's label;
16 assign label $[l, r]$ to v;

[3] The symbol "$" is assumed to be not in the alphabet.

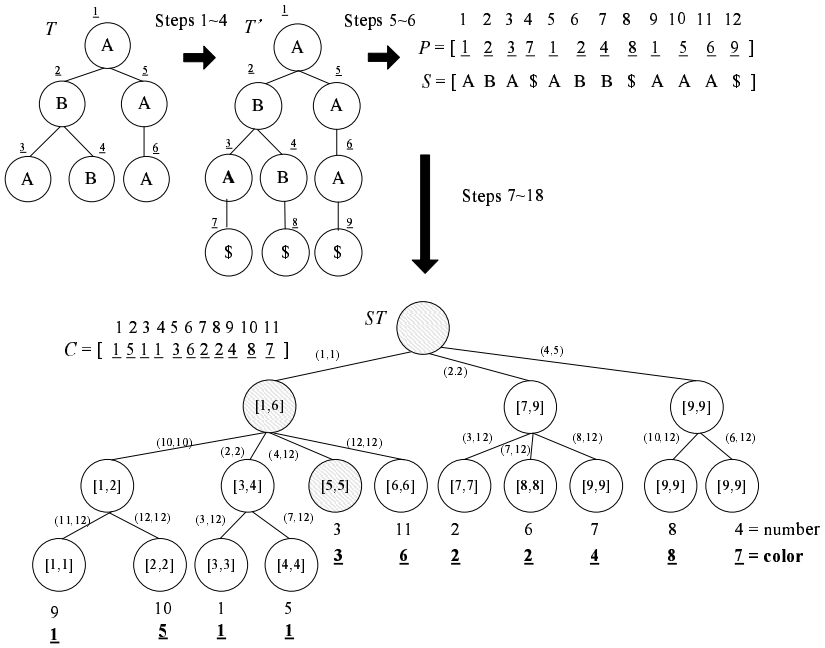

Figure 4: An illustrative example of Algorithm PREPROCESSING.

17 end for
18 build the indexing structure on C;
19 output (ST, S, C);

Figure 4 shows an illustrative example. Note that the root-to-leaf path in ST where the leaf node is numbered with i, corresponds to the path starting from the node $P[i]$ in the tree T'. For example, the leaf node of the shaded path in ST is associated with the number 3 and colored with $\underline{3}$. This path represents [AABBAAA$] and exactly corresponds to the following path in T': $\underline{3} \rightarrow \underline{7}$.

The time complexity of the algorithm is analyzed in the following. Lemma 1 guarantees the efficiency of building the supplement indexing structure on C (line 18). Based on Lemma 1, the time complexity is shown to be quadratic.

Lemma 1 [10] *Let C be a positive integer array. If $C[i] \leq |C|$ for all i, there exists an $O(|C|)$ time algorithm to build an index structure on C such that for any interval $[l, r]$, we can output the set of distinct numbers in $C[l, r]$ within time $O(d)$, where d is the output size.*

Theorem 1 *The time complexity of Algorithm Preprocessing is $O(n^2)$, where n is the number of nodes in the queried tree.*

Proof. T' has at most n root-to-leaf paths and the length of each root-to-leaf path is at most $n+1$, so $|P| = |S| = O(n^2 + n) = O(n^2)$. It follows that the time complexity of building the corresponding ST is $O(n^2)$. The size of C is equal to the number of leaves in ST, so $|C| = O(n^2)$. According to Lemma 1, it takes $O(|C|) = O(n^2)$ to preprocess C. The total time complexity could be concluded as $O(n^2)$. □

4.2 Querying

Given a rooted labeled tree Q on Σ with leaves $l_1 \cdots l_\ell$, we describe how to find $(Occ_1 \cdots Occ_\ell)$, where $Occ_i = \{v | \sigma(l_i)$ occurs in T at node $v \in V\}$ for $i = 1 \cdots \ell$. Without loss of generality, we assume that the root of Q has the label ε which denotes the empty string and is not in Σ.

The first step of the algorithm is to match Q with ST, which is done by traversing Q in the depth-first order as follows: First set p as the position of the root of ST. Whenever we step downward to a node v in Q, we also step downward in ST from p to find the position p' in ST, such that p' corresponds to $\sigma(v)$. If p' is found, we set the *match point* of v as p', and reassign p as p'. If no such p' exists, we backtrack from v to process other nodes, and assign p as the match point of $v's$ parent, if v is not the root of Q; otherwise the procedure stops.

After finding all the match points for Q, suppose that m_k is the match point for the leaf l_k. We will

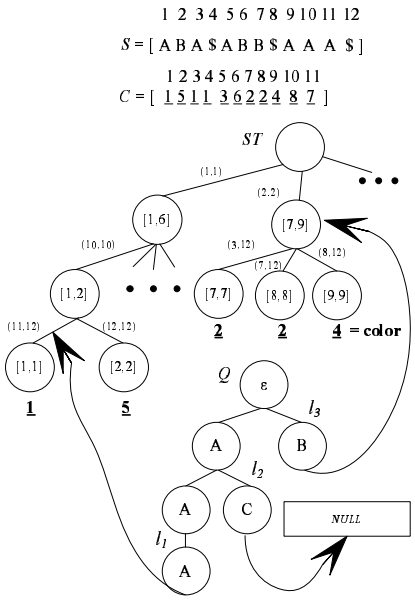

Figure 5: An illustrative example of Algorithm QUERYING.

identify the nearest node below (including) m_k in ST and suppose its label is $[i_k, j_k]$. We can then obtain Occ_k by determining the distinct colors on m_k's descendant leaves, which will be the distinct colors in $C[i_k, j_k]$.

Algorithm QUERYING
Input: $Q = (V', E', r', L')$ with leaves $\{ l_1 \cdots l_\ell \}$.
Output: $(Occ_1 \cdots Occ_\ell)$.

```
1   match the tree Q with ST;
2   for k := 1 to l do
3       m_k ← match point of l_k in ST ;
4       if m_k ≠ NULL
5           N ← the nearest node not above m_k ;
6           [i_k, j_k] ← label of N;
7       else
8           [i_k, j_k] ← [0, 0];
9       end if
10  end for
11  for each leaf k ∈ {1 ··· ℓ}
12      Occ_k ← distinct colors in C[i_k, j_k];
13  end for
14  output (Occ_1 ··· Occ_ℓ);
```

Continue the example as illustrated in Figure 4. Suppose the query tree is the one shown in the bottom of Figure 5. Take the leaf node l_3 as an example. Its corresponding match point in ST will be the node labeled with [7, 9]. Since $C[7, 9] =$ [2 2 4], we output the distinct colors from the range and obtain $occ_3 = \{2, 4\}$.

Theorem 2 *The time complexity of Algorithm Querying is $O(m + Occ)$, where m is the size of the query and Occ is the output size.*

Proof. The time to match Q with ST is $O(m)$. According to Lemma 1, it takes $O(d_k)$ time to output the set of distinct colors in $C[i_k, j_k]$ for each k, where d_k is the output size. Therefore, the total time complexity is $O(m) + O(d_1) + \cdots + O(d_\ell) = O(m + Occ)$, where Occ is the output size. □

5 Analysis of RPF

We define the Decision version of the RPF ($DRPF$) problem as follows: Given a directed labeled graph G, a rooted labeled tree Q with leaves $(l_1 \cdots l_\ell)$, and a positive integer k, is $(|Occ_1| + |Occ_2| + ... + |Occ_\ell|) \geq k$?

Theorem 3 *The DRPF problem is NP-complete.*

Proof. It is clear $DRPF \in$ NP, so we only have to show how to reduce the Hamiltonian path problem [5] to $DRPF$. Given a graph $G = (V, E)$, let $G' = (V, E', L)$ be a directed labeled graph such that: 1) $\{v_1, v_2\} \in E$ iff (v_1, v_2) and $(v_2, v_1) \in E'$ for all $v_1, v_2 \in V$ and 2) each node in G' has the same label. Let Q be a rooted labeled tree such that: 1) Q has only one leaf, 2) each node of Q has the same label as G', and 3) the number of nodes in Q is $|V|$, so Q is just a path of length $|V|$. It is easy to see that G has a Hamiltonian path iff $DRPF(G', Q, 1)$ is "yes". □

We define the Optimization version of the RPF ($ORPF$) problem as follows: Given a directed labeled graph G to be queried and a rooted labeled tree Q with leaves $(l_1 \cdots l_\ell)$, $S_1 \cdots S_\ell$ is a feasible solution iff if $v \in S_i$ then $\sigma(l_i)$ occurs in G at node v for $i = 1 \cdots \ell$. The goal is to find a feasible solution such that the cost $(|S_1| + |S_2| + \cdots + |S_\ell| + 1)$ is maximun.

Theorem 4 *Let n be the number of nodes in the queried graph. The ORPF problem is not approximable within $\rho(n)$, where $\rho(\cdot)$ is any function with two constants c_1 and c_2 such that $x^y \geq \rho(x^{y+1})$ if $x > c_1$ and $y \geq c_2$.*

Proof. Assume there is a polynomial time $\rho(n)$-approximation algorithm A for $ORPF$. We shall describe a polynomial time algorithm for the Hamiltonian path problem, so that we can conclude the assumption is wrong unless P = NP.

Given a graph G, let n be the number of nodes in G. If $n \leq c_1$ then we can determine whether G has a Hamiltonian path by enumerating all simple paths in G. Otherwise we construct G' and Q as in the proof of Theorem 3. Obtain G'' by copying G' n^{c_2}-times. Let $C = \text{cost}(A(G'', Q))$. If $C > 1$ then return "Yes" else return "No" .

It is clear this algorithm runs in polynomial time and is correct when $n \leq c_1$. We now show this algorithm is also correct when $n > c_1$. Let C^* be the cost of the optimal solution of $ORPF(G'', Q)$. According to our assumption, we know $(C^*/C) \leq \rho(n^{c_2+1})$. If G has a Hamiltonian path, then $C^* \geq n^{c_2} + 1 > \rho(n^{c_2} + 1)$. Therefore, C has to be greater than 1 to make $(C^*/C) \leq \rho(n^{c_2} + 1)$. If G does not have a Hamiltonian path, clearly $C^* = C = 1$. □

Corollary 1 *Let n be the number of nodes in the queried graph. The ORPF problem is not approximable within n^k for any $k < 1$, unless P = NP.*

6 Concluding Remarks

We expect that trees and graphs will play an important role in biological data archives in the post-proteomic era. In this paper, an efficient algorithm is proposed to solve the $TRPF$ problem and the RPF problem is proven to be hard. We close this paper by mentioning some directions for future research: 1) improving the preprocessing time for the $TRPF$ problem, and 2) finding an efficient algorithm for the RPF problem when the queried graph is acyclic.

Acknowledgements. We thank the anonymous reviewers for their valuable comments. We also thank Atsuko Yamaguchi for providing the working manuscript on the KEGG Glycan database queries and Hsueh-I Lu for his superb lecture notes on suffix trees. Hsiao-Fei Liu and Kun-Mao Chao were supported in part by an NSC grant 92-2213-E-002-059. Ya-Hui Chang was supported in part by an NSC grant 92-2213-E-019-010.

References

[1] K. F. Aoki, A. Yamaguchi, Y. Okuno, T. Akutsu, N. Ueda, M. Kanehisa, and H. Mamitsuka, Efficient Tree-Matching Methods for Accurate Carbohydrate Database Queries. *Genome Informatics*, 14: 134–143, 2003.

[2] Ya-Hui Chang, A Graphical Query Language for Mobile Information Systems. *SIGMOD Record*, 32(1): 20–25, 2003.

[3] Z. Chen H. V. Jagadish L. V. S. Lakshmanan S. Paparizos, From Tree Patterns to Generalized Tree Patterns: On Efficient Evaluation of XQuery. In *Proceedings of the 29th Very Large Data Base Conference*, 2003.

[4] B. F. Cooper, N. Sample, M. J. Franklin, G. R. Hjaltason, M. Shadmon, A Fast Index for Semistructured Data. In *Proceedings of the 27th Very Large Data Base Conference*, 2001.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 2/e. The MIT Press, 2001.

[6] G. Gottlob, C. Koch, R. Pichler, XPath Query Evalution: Improving Time and Space Efficiency. In *Proceedings of the 19th IEEE International Conference on Data Engineering*, 2003.

[7] D. Gusfield, Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press, 1999.

[8] X. Huang, and K.-M. Chao, A Generalized Global Alignment Algorithm. *Bioinformatics*, 19: 228–233, 2003.

[9] M. Kanehisa, S. Goto, S. Kawashima, Y. Okuno, and M. Hattori, The KEGG Resource for Deciphering the Genome. *Nucleic Acids Research*, 32(1):D277–D280, 2004.

[10] S. Muthukrishnan, Efficient Algorithms for Document Retrieval Problems. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, 657–666, 2002.

[11] D. Shasha, J. T. L. Wang, R. Giugno, Algorithmics and Applications of Tree and Graph Searching. In *Proceedings of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database System*, 39–52, 2002.

[12] H. Wang S. Park W. Fan P. S. Yu, ViST: A Dynamic Index Method for Querying XML Data by Tree Structures. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2003.