5th European Semantic Web Conference (ESWC2008), Tenerife, Spain, June 1-5, 2008.

Graph Summaries for Subgraph Frequency Estimation

Angela Maduko[1], Kemafor Anyanwu[2], Amit Sheth[3], Paul Schliekelman[4],

[1]Department of Computer Science, University of Georgia
[2]Department of Computer Science, North Carolina State University
[3]Kno.e.sis Center, Wright State University
[4]Department of Statistics, University of Georgia
maduko@cs.uga.edu, kogan@ncsu.edu, amit.sheth@wright.edu, pdschlie@stat.uga.edu

Abstract. A fundamental problem related to graph structured databases is searching for substructures. One issue with respect to optimizing such searches is the ability to estimate the frequency of substructures within a query graph. In this work, we present and evaluate two techniques for estimating the frequency of subgraphs from a summary of the data graph. In the first technique, we assume that edge occurrences on edge sequences are position independent and summarize only the most informative dependencies. In the second technique, we prune small subgraphs using a valuation scheme that blends information about their importance and estimation power. In both techniques, we assume conditional independence to estimate the frequencies of larger subgraphs. We validate the effectiveness of our techniques through experiments on real and synthetic datasets.

Keywords: Frequency estimation, Graph summaries, Data summaries

1 Introduction

Graphs are increasingly used to model data on the Web, the emerging Semantic Web and complex biochemical structures such as proteins and chemical compounds. They offer a representation amenable to analysis and knowledge extraction. Structure search that matches a query graph over a graph database, is a common technique for retrieving information from graphs. In biochemistry, search for common features in large sets of molecules is used for drug discovery and drug design studies. These searches, which return all graphs that contain the query graph, can be computationally challenging. As demonstrated in [12] and [15], path or subgraph indexes help to cope with this difficulty. The idea is to use indexed fragments of the query graph to retrieve a set of data graphs, from which those containing the whole query graph are found using subgraph isomorphism tests. The frequency of the fragments in the query graph plays a crucial role in optimizing these searches. As an illustration, the left part of Figure 1 shows a database of graphs, with the subgraphs in Figure 1h and Figure 1i indexed. Given the query graph of Figure 1g, the graphs in Figure 1b–1f that contain the indexed fragments are retrieved. With subgraph isomorphism tests, only Figure 1b and 1f are found to contain the entire query graph. One strategy for optimizing this process is to reduce the number of isomorphism tests performed, which depends on

the frequency of the indexed subgraphs. Figure 1h and Figure 1i have 9 and at least 18 occurrences in the selected graphs respectively. Thus the isomorphism tests can be performed by matching Figure 1h first, then expanding to the rest of the query graph.

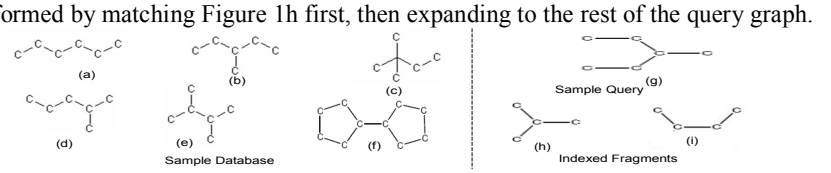

Figure 1: A Sample Graph Database, Its Indexed Fragments and a Query Graph

Most proposed query languages for querying graph representations of semi-structured data, such as RDF [5], support structure search as a primary query paradigm. In storage schemes for graph databases that organize graphs as edge relations, the embeddings of a given query graph in the database are computed using join operations. For such scenarios, subgraph frequency estimates are needed to determine the cardinality of intermediate join results during query optimization.

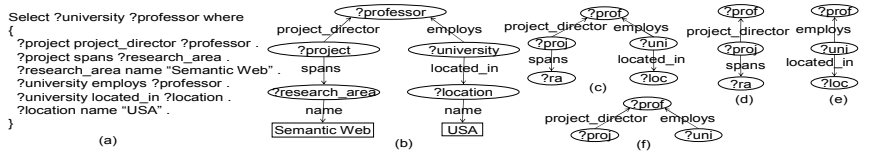

Figure 2: A SPARQL Query, Its Graph and Subgraph Patterns

To illustrate, the SPARQL [10] query (namespaces are omitted) of Figure 2a (with graph pattern shown in Figure 2b) asks for professors employed in universities in the United States who direct a Semantic Web project. Three join operations (shown in Figure 2c) are needed to process this query if edge relations is the storage scheme for the data. To optimize this query, the optimizer will need accurate estimates of the frequency of subpatterns of Figure 2c, such as those shown in Figure 2d, e and f.

In this work, we focus on efficient techniques for estimating the frequency of subgraph patterns in a graph database. Noting that (1) the number of possible subgraphs in a graph database could be exponential and (2) it is more expedient if the estimates are computed without disk accesses, since they are needed at optimization time, we focus on techniques that summarize subgraphs in the graph database so as to fit in the available memory. Obviously, such a summary will be useful only if it captures the correlations among subgraphs. Our choice of subgraphs is strengthened by the observations in [13], [15] and [16], where it is shown that subtrees and subgraphs perform better than paths in capturing correlations among graphs and trees, respectively. However, the number of unique subgraphs greatly exceeds the number of paths in a graph. It is thus infeasible to examine all subgraphs, and efficient pruning techniques are needed. We propose two summaries that differ in their pruning techniques. The Maximal Dependence Tree (MD-Tree) and the Pattern Tree (P-Tree).

The pruning technique of the MD-Tree is based on the observation that high-order statistical dependencies often exist among subgraphs. It may be prohibitive to keep all such dependencies; thus we attempt to capture the most informative dependencies in the given space. The pruning technique of the P-Tree is based on two insights: (1) the frequency of a graph may be close to that of a function of its subgraph; and (2) information about the importance of subgraphs could lead to characterizing some as more important than others. For example, frequent subgraphs from a query workload

are more important than infrequent ones for tuning purposes. We prune the P-Tree by blending the significance of patterns for estimation and for tuning purposes.

This paper is structured as follows: Section 2 formally defines the problem we address in this work and briefly discusses background work. Section 3 discusses the proposed summaries while Section 4 presents the experimental evaluation of the summaries. In Section 5 we discuss related work and conclude the paper in Section 6.

2 Preliminaries

Data Model. A collection of connected graphs or a graph database can be viewed as a large graph with several connected components. We use the term "graph" to refer to such a large graph. We focus on a directed labeled graph model that represents named binary relationships (edges labeled with names) between entities (nodes) such. Such named relationships can be viewed as triples (entityA relationship1 entityB).

Definition 1. Let L and T be finite sets of labels. We define a graph G as a 4-tuple (V, E, λ, τ).V and E are sets of nodes and edges of G respectively, $\lambda : (V \cup E) \rightarrow L$ is a many-to-one function that maps nodes/edges of G to labels in L and $\tau : 2^V \rightarrow T$ is a multivalued type function that maps sets of nodes of G to labels in T, (i.e. $L \cap T \neq \varnothing$) so that for a node v, τ(v) may be perceived as a conceptual entity to which v belongs.

Note that, both τ and λ map nodes not perceived as members of a conceptual entity to the same label so that for such a node v, $\tau(v) = \lambda(v)$. Our graph model captures many semantic data models, particularly the RDF model[5].

Definition 2. Given graphs G = (V, E, λ, τ) and G′ = (V′, E′, λ′, τ′), we say that G′ is embedded in G if there is an injective function $f : V' \rightarrow V$ such that:

- $\tau'(v) = \tau(f(v)), \forall v \in V'$
- $\forall(u, v) \in E', (f(u), f(v)) \in E$ and $\lambda'(u, v) = \lambda(f(u), f(v))$

Problem Definition. Given graphs G and G′, the frequency of G′ in G is the number of unique embeddings it has in G. The problem we address is stated succinctly as:
Given a graph G and a space budget B, create a summary of size at most B, for obtaining accurate estimates of the frequencies of graphs embedded in G.

2.1 Background

With minimal modifications, efficient pattern-mining algorithms such as gSpan [14] can be used to discover subgraphs and count their frequencies. This technique uses a canonical label for a graph for computing subgraphs frequencies. We now briefly review the *minimum DFS code* [14] canonical label of a graph, adopted in this work.

DFS Coding. This technique transforms a graph into an edge sequence called DFS code, by a DFS traversal. Each edge (u, v) in the graph is represented by a 5-tuple <i, j, l_i, $l_{(i,\ j)}$, l_j>, where integers i and j denote the DFS discovery times of nodes u and v, l_i, l_j and $l_{(i,\ j)}$ are the labels of u, v, and the edge (u, v), respectively. The edges are ordered by listing those in the DFS tree (tree edges) in order of their discovery. The

rest are inserted into the ordered list as follows: For a tree edge (u, v), all non-tree edges from v come immediately after (u, v); if (u_i, v_j) and (u_i, v_k) are two non-tree edges, (u_i, v_j) is listed before (u_i, v_k) only if $j < k$. A graph may have many DFS codes, so the minimum, based on a linear ordering of all its DFS codes, is its canonical label. Details of the DFS coding and gSpan algorithm can be found in [14].

3 Approach

In this section, we present our proposed summaries. We discuss the Maximal Dependence Tree in section 3.1, then the Pattern Tree in section 3.2. To create each summary, we generate and count the frequencies of all subgraphs of length at most maxL, using a slight modification of gSpan and input graph G = (V, E, λ, τ). We represent each edge e = (u, v) in G by a 5-tuple <i, j, λ(e), τ(u), τ(v)>, where integers i and j are the DFS discovery times of nodes u and v and λ and τ are the functions defined in Definition 1, with λ(.) and τ(.) (i.e., the ranges of λ and τ) mapped to unique integers. The sequence of edges/quintuples obtained after the algorithm is run, represents the structure of subgraphs of G; thus, given any two edges $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$, if $v_1 = v_2$, it follows that $\tau(v_1) = \tau(v_2)$. [14] discusses the minimum DFS code in the context of undirected labeled graphs. For directed labeled graphs, we ignore edge directions during the DFS traversal so as to maintain the connectivity of the graph. However the directions are kept implicitly in the quintuples. We use the term pattern ambiguously to refer to a subgraph as well as to its minimal DFS code.

Example 1. *Figure 3a shows a directed labeled graph of conference paper information. The same graph is shown in Figure 3c, with nodes and edges assigned integer ids as shown in Figure 3b. To obtain the edge sequence for the subgraph au1 authorOf pub1, pub1 submittedTo conf1, pc1 pcMember of conf1, we begin DFS with the edge authorOf as it is lexicographically the smallest label. DFS proceeds as indicated by the boxed discovery ids to yield (1,2,5,1,4) (2,3,7,4,3) (4,3,6,3,2). Note that the direction of the edge labeled "pcMemberOf" is implicit in the sequence. Figure 3d shows all patterns of length at most 3 and their frequencies.*

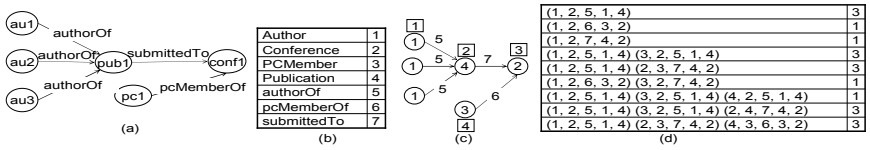

Figure 3: Unique Edge Sequences for Subgraphs

Note that, in our directed graph model, an edge, for example (5, 1, 4) in Figure 3, may appear in a pattern of length at least two, in one of three possible directions: forward, as in (1, 2, 5, 1, 4); backward, as in (3, 2, 5, 1, 4); or self-loop, as in (2, 2, 5, 1, 4).

3.1 Maximal Dependence Tree (MD-Tree)

To motivate the MD-Tree, we observe that edges in certain positions in patterns may largely determine their probabilities. For example, the three patterns of length 3 in

Figure 3d have the edge in the first position in common so that the edge in the second position rather than the first, will exert a greater influence in their probabilities. Our MD-Tree approach exploits the existence of such edge positions. If none exists, we assume edges occur independently at each position. We construct the MD-Tree through an adaptation of the Maximal Dependence Decomposition [2] technique.

Notations.

Let $\mathbf{N_E}$ denote the number of unique edge labels in a graph G = (V, E, λ, τ) that is N_E is size of the mapping (λ(e), τ(u), τ(v)), for each edge e = (u, v) in E. Further, let β be an integer in the range [1, 3] such that β is: (a) 1 if all edges in E are forward edges, (b) 2 if E also contains backward edges and (c) 3 if in addition to forward edges, E also contains self loop edges or both backward and self loop edges. We denote a set of patterns of length k by $\boldsymbol{P_K}$ and define **freq** as a function with domain the power set of patterns of length at most maxL and range the set of positive integers. If X is a single pattern, freq(X) maps to the frequency of the pattern but if X contains more than one pattern, freq(X) maps to the sum of the frequencies of patterns in X. We denote the probability with which edge x occurs at position y by $\mathbf{Pr(x_{(y)})}$.

We begin the discussion of the MD-Tree by introducing *weight matrices* for a set of pattern. A weight matrix WM_K for P_K is a $\beta N_E \times k$ matrix whose rows represent the possible edge patterns that may appear in patterns in P_K and columns represent positions in which the edge types may occur. Cell (i,j) contains the probability that edge i occurs at position j. Given P_K and the assumption that edges occur independently at any position in the patterns, a weight matrix for P_K suffices for estimating the frequency of any pattern $p = (e_1, e_2, \ldots, e_k)$ in P_K as $freq(p) = freq(P_K) \times Pr(e_{1(1)}) \times Pr(e_{2(2)}) \times \ldots \times Pr(e_{k(k)})$. To construct a weight matrix WM_K for P_K, we obtain the row indices by assigning unique integer ids to the possible edge patterns that may appear in patterns of length at least two, in multiples of β. We assign integer x to edge type y such that if x modulo β is 0, 1 or 2 then x identifies y in the forward or backward directions or self-loop, respectively. The column indices are the k positions in which edge patterns may occur.

(1, 2, 5, 1, 4) (3, 2, 5, 1, 4)	3
(1, 2, 5, 1, 4) (2, 3, 7, 4, 2)	3
(1, 2, 6, 3, 2) (3, 2, 7, 4, 2)	1

(a)

1	(5, 1, 4)
2	(5, 1, 4)
3	(6, 3, 2)
4	(6, 3, 2)
5	(7, 4, 2)
6	(7, 4, 2)

(b)

	1	2
1	6/7	0
2	0	3/7
3	1/7	0
4	0	0
5	0	3/7
6	0	1/7

(c)

Figure 4: A Weight Matrix for Patterns of Length 2

Example 2. *To construct WM_2 with dimension 6×2 (i.e. β is 2, since there are no self loop edges) for the patterns in* ***Figure 4****a, we assign integer ids to the edge types (5, 1, 4), (6, 3, 2) and (7, 4, 2) as shown in* ***Figure 4****b. Next, we compute the entries for each cell (i, j) in WM_2 as shown in* ***Figure 4****c. Thus, cell (2, 1) holds the probability that edge (5, 1, 4) occurs in a backward direction at position 2 etc. Under the independence assumption, the frequency of (1, 2, 5, 1, 4)(3, 2, 5, 1, 4) is estimated as 7(6/7)(3/7) i.e. 18/7, which rounds to 3.*

Definition 3. Base MD-Tree. Given the sets $P_1, P_2, \ldots, P_{maxL}$ of patterns of length at most maxL, a base MD-Tree for the patterns in P_i $1 \le i \le maxL$ is a triple (R_T, V_T, E_T) where $R_T \in V_T$ is the root of the tree and V_T and E_T are the sets of nodes and edges of the tree, such that $|V_T - R_T| = |E_T| = maxL$. All nodes in $V_T - R_T$ are ordered children of

R_T such that child i is associated with the weight matrix WM_i, for patterns in P_i. Each edge (R_T, i) is labeled with freq(P_i), the total frequency of all patterns in P_i.

If the independence assumption does not hold, a refinement process on the base MD-Tree is required to capture edge dependencies. Given P_K, if it is known that the occurrence of an edge at position i, $1 \le i \le k$, $i \ne m$, depends on the edge at position m, we estimate the frequency of a pattern $p = (e_1, e_2, \ldots, e_k)$ in P_K as:

$$\text{freq}(P_K) \times \Pr(e_{m(m)}) \times \prod_{i=1, i \ne m}^{k} \Pr(e_{i(i)} \mid e_{m(m)}), \quad (1)$$

where $\Pr(e_{i(i)} \mid e_{m(m)})$ is the conditional probability that e_i occurs at position i given that e_m occurred at position m. The base MD-Tree is modified to reflect this dependence. We refer to the modified tree as a *refined MD-Tree*.

Definition 4. Refined MD-Tree. A refined MD-Tree is a triple (R_T, V_T, E_T) where $R_T \in V_T$ is the root and V_T and E_T are the sets of its nodes and edges respectively. V_T can be partitioned into two disjoint non-empty sets V_{Tleaf} and $V_{Tnon\text{-}leaf}$ such that for $v \in V_{Tleaf}$ or $v \in V_{Tnon\text{-}leaf}$, v is a leaf node or non-leaf node, respectively. Weight matrices are associated only with leaf nodes and every non-leaf node has βN_E ordered children except the root, which has maxL children.

For ease of exposition, we illustrate the refinement process with an example.

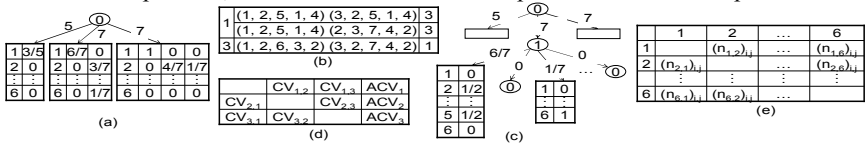

Figure 5: Refining the Base MD-Tree of patterns of length 2, at position 1

Example 3. *Suppose that edge types at position 1 influence those at position 2, for patterns of length 2. We refine the second child (which we denote as v_2) of the root of the base MD-Tree of* ***Figure 5****a as follows. First, we create βN_E (in this case 6) ordered children nodes for node v_2, one for each edge type. Next, we obtain the length 2 patterns used to create the weight matrix associated with v_2 i.e., the patterns in* ***Figure 4****a. We then partition these patterns with respect to the occurrence of the 6 edge types at position 1. As shown in* ***Figure 5****b, only the partitions for edge types 1 and 3 are non-empty. Using the patterns in these partitions, we create two new 6×1 weight matrices for child nodes 1 and 3 of v_2 respectively. We label the edges to these nodes 6/7 and 1/7 i.e. the probabilities that the edge types with ids "1" and "3" occur at position 1 in patterns in their respective partitions. Finally, we delete v_2's weight matrix and assign it the split position 1.* ***Figure 5****c shows the refined MD-Tree.*

Given P_K, we refine its base MD-Tree by finding the position in its patterns that most influences others, by chi-square association tests for edge types at all pairs of positions i and j, $1 \le i,j \le k$, $i \ne j$. The test statistic is given by:

$$\sum_{m=1}^{\beta N_E} \sum_{n=1}^{\beta N_E} \frac{(O_{m,n} - E_{m,n})^2}{E_{m,n}} \text{ where } E_{m,n} = \left(\sum_{a=1}^{\beta N_E} O_{m,a} \sum_{b=1}^{\beta N_E} O_{b,n} \right) \Big/ \sum_{a=1}^{\beta N_E} \sum_{b=1}^{\beta N_E} O_{a,b}. \quad (2)$$

$O_{m,n}$ is the sum of the frequency of patterns in P_K for which edge types m and n occur at positions i and j respectively. We clarify this with an illustration.

Example 4. *To find a position of maximal dependence for a set of patterns of length 3, first, we create a 3×4 matrix as shown in* ***Figure 5****d. To compute* $CV_{1,2}$ *i.e. the chi-square value for cell (1,2) for instance, we create a 6×6 matrix as shown in* ***Figure 5****e, where cell i,j contains the number of times the edge types with integer ids i and j occur at positions 1 and 2 in the patterns respectively.* $CV_{1,2}$ *is the value of the test statistic for this 6×6 matrix. Next, we obtain the aggregate chi-square value (ACV) stored in the fourth column of each row by summing the chi-square values in each row of the 3×4 matrix. Then we find the maximum ACV over the three rows. Suppose it is* ACV_2*, we then conclude that position 2 has the greatest influence on others but only if at least one of* $CV_{2,j}$ *is statistically significant.*

If v is a leaf node of a base or refined MD-Tree T, we say that v is significant if there is a position j of maximal dependence in the set of patterns used to create the weight matrix of v. We denote an MD-Tree that is completely refined (has no significant nodes) as a *Complete MD-Tree*. A complete MD-Tree is ideal for estimating the frequency of patterns but its size may exceed the budget. Our *optimal MD-Tree* then is a refined MD-Tree that fits the budget and that gives the best estimates of pattern frequencies. We now formalize the problem of finding the optimal MD-Tree.

Given a complete MD-Tree (R_T, V_T, E_T), let $T' = (V', E')$ be a tree such that $V' \subseteq V_T$, where $V' = \{R_T, v_1, v_2, \ldots, v_m\}$ contains all significant nodes of V_T and every edge (u, v) in E' is an edge in E_T. Also, let $S = (0, s_{v1}, s_{v2}, \ldots, s_{vm})$ be the size increment induced on the MD-Tree when v_i is refined and let $I = (0, i_{v1}, i_{v2}, \ldots i_{vm})$ be the impact of node v_i, given by $\max(ACV)/C_v$, rounded to the nearest integer. C_v is the number of columns of the weight matrix associated with v. The problem is to find a tree $T'' = (V'', E'')$, $T'' \subseteq T'$ rooted at R_T, such that $\sum_j (S_{vj}) \leq B$ and $\sum_j (i_{vj})$ is maximized. This problem is an instance of the Tree Knapsack Problem, which is known to be NP-hard. Given x_j, an indicator variable with value 1 if v_j is selected as part of the optimal solution or 0 otherwise, TKP is formulated as:

$$\text{Maximize } \sum_j^m i_{vj} x_j \text{ constrained on } \sum_j^m s_{vj} x_j \leq B, \; x_{pred(j)} \geq x_j, \tag{3}$$

where pred(j) is the predecessor of j in T'. With this reformulation, we employ a greedy approximation with $O(|V'|^2)$ running time. Given $T' = (V', E')$, vectors S and I and the size budget B, our greedy approximation creates the tree $T'' = (V'', E'')$ by keeping maximal impact subtrees of T' that fit the budget.

Frequency Estimation Using the MD-Tree. Given an optimal MD-Tree (R_T, V_T, E_T), let λ_V map nodes in V_T to the integers or weight matrices they are associated with and and λ_E map edges in E_T to integers or real numbers they are labeled with. Let the function id on edge patterns return the integer id of its edge type. Let $p' = e_1, e_2, \ldots, e_k$ be the edge sequence of a graph $G' = (V', E', \lambda, \tau)$ of length k. To estimate the frequency of P', we first check that the structure of p' exists in the structural summary given in Definition 1, which we keep along with the MD-Tree. If so, beginning from the k^{th} child v of R_T, we estimate freq(p') as:

$$\lambda_E(R_T, v)\left(\prod_{i=1}^{j} \lambda_E(v^i, v^{i+1})_{\lambda_V(V^i)}\right)\left(\prod_{r=1, r \notin S}^{k} (\lambda_V(v^{j+1}))_{(id(e_r), r)}\right). \tag{4}$$

In this product, the subscript r of an edge $(v, v')_r$ denotes the r^{th} edge of node v. The integer j is the number of edges of the optimal MD-Tree found on the path from the root to a leaf node as defined by the subscripts on the edges, so that the node v^{j+1} is a leaf. The subcripts (r, r′) are integers indices for accessing cell (r, r′) of the weight matrix associated with node $\lambda_V(v^{j+1})$. The set *S* holds labels of all nodes on the path from R_T to v^{j+1} so that at v^{j+1}, any integer in the range [1, maxL] not in the set *S* did not label any node on this path. The depth of the MD-Tree is at most maxL; thus the time complexity for estimating pattern frequencies is O(maxLlog(maxL)).

Example 5. *To estimate the frequency of the pattern $p = e_1, e_2$ given by (1,2,5,1,4) (3,2,5,1,4) from* ***Figure 5****c, we first access v_1, the second child of the root. Since $\lambda_V(v_1) = 1$, we insert 1 into set S and set the frequency of p (freq(p)) to freq(*p*) which is 7. Recall from* ***Figure 4****b that $id(e_1) = id(1, 2, 5, 1, 4)$ is 1. So, we access v_2, the node on which the first edge of v_1 is incident. Next, we multiply freq(p) by $\lambda_E(v_1, v_2)$ given by 6/7, resulting in 6. Then we obtain $\lambda_V(v_2)$ i.e. the weight matrix WM_2 of v_2. S contains the integer 1, so the lone column of WM_2 must index position 2 of patterns in P_2. Further, $id(e_2) = id(3,2,5,1,4)$ is 2; thus we access the cell that represents the index (2, 2) in WM_2 to obtain 1/2. We then multiply freq(p) by 1/2 to obtain 3.*

3.2 Pattern Tree (P-Tree)

The idea of our P-Tree approach is to identify sets of patterns with almost the same edge patterns, such that for a set P_K, the frequencies of patterns in P_K are within δ of that of at least one pattern in P_K say p. Given p, the frequencies of patterns in P_K can be estimated within δ error thus we can safely eliminate all patterns but p from the summary. Given $P = P_1, P_2, ..., P_{maxL}$ a set of patterns of length at most maxL, the unpruned P-Tree for *P* is a prefix tree of patterns in *P*. Its nodes are labeled with edge patterns so that a pattern in *P* is obtained by concatenating node labels on a path from the root. Also, each node is associated with the frequency of the pattern it represents.

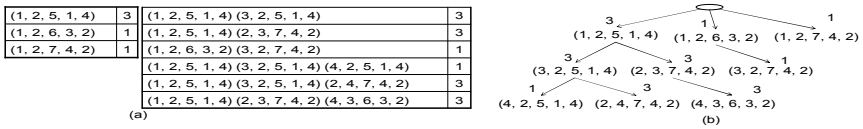

Figure 6: Pattern Tree

If the size of the P-Tree exceeds the budget, it must be systematically pruned so as to avoid a large increase in its overall estimation error. The question is *which nodes are to be pruned and in what order?* We answer this question with the concepts of *observed* and *estimation* values of patterns. We begin by introducing some notations.

Notations.

In addition to the notations introduced in section 3.1, we define the function **children** whose domain and range are the set of patterns of length at most maxL and the power set of patterns of length at most maxL respectively, such that for a pattern p, children(p) maps to the set of children of p in the P-Tree.

Definition 5. Observed Value of a Pattern. Let $P = (p_1, p_2, \ldots, p_m)$ be a set of patterns with frequencies $(freq(p_1), freq(p_2), \ldots, freq(p_m))$. Let T, a positive integer, be

an importance threshold T and let $P_{OI} = (p_{OI1}, p_{OI2}, \ldots, p_{OIm})$ be a vector such that $0 \leq p_{OIi} \leq 1$ for every $p_{OIi} \in P_{OI}$ and p_{OIi} defines the importance of pattern p_i. We define the observed value of p_i (p_{OVi}) as the number of patterns that are less important than p_i that is the number of patterns p_j in *P* such that $p_{OIi} > p_{OIj}$.

We do not assume any particular technique for computing the importance of a pattern. However, for the purpose of tuning the summary to favour frequent patterns, it can simply be computed as the ratio of its frequency to that of the most frequent.

To motivate the estimation value of patterns, we note that if there is a match for a pattern $p = e_1, e_2, \ldots, e_k$ in the tree, its frequency freq(p) is the integer associated with the matched node labeled e_k. If e_k is contracted, we guess freq(p) as $p'_{GR} \times \text{freq}(p')$, where $p' = e_1, e_2, \ldots, e_{k-1}$ is the parent of p and p'_{GR} is the growth rate of p', under the assumption that children of p' have a uniform frequency distribution. When the children of p' are to be contracted, we keep its growth rate given by $N/(m \times \text{freq}(p'))$, where m is the number of children of p' and N is their total frequency. Thus the frequency of each child is estimated as N/m. We keep the growth rate and not N/m, for ease of propagation as we will discuss later. We keep one growth rate for p' for all its children, to avoid overly increasing the size of the tree as patterns are pruned. To validate our uniformity assumption, we prune the P-Tree by deleting the children of patterns that are uniformly or nearly uniformly distributed. To do this, we let the random variable Y define the occurrence of a child of p'; then we measure the evenness of the probability distribution of Y using its *entropy* [10] $H(Pr_Y)$, given by $-\sum_j Pr_Y(p_j)\log_2(Pr_Y(p_j))$. We compute the probability of the occurrence of any child p of a pattern p' as its proportion to the total frequency of children of p' (i.e., freq(p)/N, where N is the sum of $\text{freq}(p_j)$ for all p_j in children(p')). The entropy of a probability distribution is maximized if the distribution is uniform. Thus to measure the uniformity of Y, we normalize $H(Pr_Y)$ by a division by its maximum entropy. We denote this ratio as p'_{ENT} for pattern p' in the tree. If p' has one child, we set p'_{ENT} to 1.

Definition 6. Estimation Value of a Pattern. Given a set of patterns $P = (p_1, p_2, \ldots, p_m)$ with frequencies $(\text{freq}(p_1), \text{freq}(p_2), \ldots, \text{freq}(p_m))$ and some $\varepsilon \geq 0$, the estimation value of p_i (p_{EVi}) is given by:

$$p_{ENTi} \times \frac{\left(\left| \{ p_j \mid p_j \in P_{Ci} \text{ and } |(\text{freq}(p_i) \times p_{GRi}) - \text{freq}(p_j)| \leq \varepsilon \} \right| \right)^h}{|P_{Ci}|} . \quad (5)$$

By definition, p_{ENT} is at most 1. It is 1, if the distribution of the children of p is uniform. If the exponent h is 1, the second term of the product measures how closely p estimates all its children within ε error. Thus if p_i and p_j both have three children and p_i estimates only two within ε while p_j estimates just one within ε, this value will be higher for p_i (2/3) than for p_j (1/3). However, if p_i has six children and estimates only two within ε, then the value will be 1/3 for both p_i and p_j, although p_i estimates more children outside ε than p_j. We set h to 1.5 to prevent the numerator from overly dominating the denominator. To find the optimal ε, beginning with exponent 0 and base 2, we recursively increment the exponent until we get to 2^i, such that enough patterns can be pruned to meet the budget. Then, we search between 2^{i-1} and 2^i for the value that allows for pruning the fewest patterns. We now combine the observed and estimation values of patterns to obtain a single value for pruning the P-Tree.

Definition 7. Let $P = (p_1, p_2, \ldots, p_m)$ be the set of patterns in the P-Tree and p_{EVmax}, the maximum expected value of patterns in P. Given a constant $c > 0$, the value of a pattern p_j is given by:

$$p_{Vj} = (1 + p_{EVj})(1 + p_{OVj}) + i p_{EVmax} . \quad (6)$$

where i is an indicator variable whose value is 1 if $p_{OVj} \geq c$ and 0 otherwise. The additive constants ensure that the value of a pattern is non-zero when either its observed or its estimation value is zero. The second term allows for tuning the P-Tree by boosting the values of important patterns, to delay their contraction.

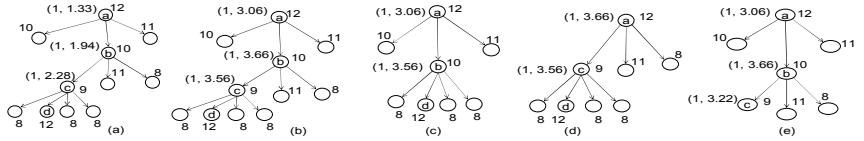

Figure 7: Contracting Nodes of the Pattern Tree

Example 6. *Figure 7a shows a subtree of a P-Tree. The 2-tuple (p_Z, p_V) for each internal node is its growth rate and its value, computed at $\varepsilon = 1$ and $c = 0$, with no importance information (i.e. p_{OVi} is zero for all patterns). The growth rate of node "a" is given by (10+10+11)/(3×12) ≈ 1, so its frequency (12) estimates that of one child (11). With exponent 1.5, the second term of the equation in Definition 7 is 0.333. The entropy of the frequency distribution of its children is 10/31 × $\log_2$(31/10) + 10/31 × $\log_2$(31/10) + 11/31× $\log_2$(31/11) = 1.583, maximized at ($\log_2$(3)) = 1.585 with ratio 1.583/1.585 = 0.999. Its estimation value is 0.999(0.333) = 0.333, so its value is 1+0.333 = 1.333. In Figure 7b, the values are computed at $\varepsilon = 2$. In Figure 7c, d, and e, the children of nodes b, a, and c have been contracted at $\varepsilon = 2$, with total estimation errors of 4, 5, and 6. Our technique will result in the contraction of Figure 7c since node b has the largest value.*

When children(p_i) are to be contracted, if the children of any node in children(p_i) have been contracted, the average of the growth rates of such nodes are computed and associated with p_i. Thus, a node in the pruned P-Tree may have at most maxL growth rates, ordered in increasing order of the original depths of their sources in the P-Tree.

Frequency Estimation Using the Pattern Tree. Given $G' = (V', E', \lambda, \tau)$, we obtain its edge sequence $p = e_1, e_2, \ldots, e_k$ and check that its structure exists in the structural summary given in Definition 1, which we keep along with the P-Tree. If so, we match p against the P-Tree. If we find a complete match for p, we return the frequency of the matched node e_k in the P-Tree. If we find a partial match, we consider the last matched node v_j in the P-Tree. If it matches e_k, we return its frequency, which is the exact frequency of p if no descendant of v_j was contracted. If it matches e_i $i < k$, we use its frequency to estimate that of the contracted node that originally matched e_k. Estimating the frequency of e_k requires estimating and propagating those of its k-i-1 immediate contracted ancestors. If $\xi_1, \xi_2, \ldots, \xi_r$, $k \leq r \leq$ maxL are the growth rates kept with v_j and freq(p_j) is the frequency of v_j, we estimate the frequency of e_k as:

$$\text{freq}(p_j) \times \prod_{r=1}^{k-i} \xi_r . \quad (7)$$

Example 7. *To estimate the frequency of (a, b, c, d) from Figure 7c, we find the partial match (a, b, _, d) and we return 12 since d is matched. With the P-Tree of Figure 7e, we find the matches (a, b, c, _). We return 9, since the growth rate of c is 1.*

3.3 Estimating the Frequency of Large Patterns.

Given a subgraph $G' = (V', E', \lambda, \tau)$ with $|E'| > \text{maxL}$, as always, we check that the structure of $p = e_1, e_2, \ldots, e_k$ exists in the structural summary given in Definition 1. If so, we partition G' into $G'_1, G'_2, \ldots, G'_{|E|-\text{maxL}+1}$ non-disjoint connected subgraphs such that G'_i intersects G'_{i-1} in all but one edge. Let G''_i denote the intersecting edges of G'_i and G'_{i-1}. Next, we obtain the edge sequences $p'_1, p'_2, \ldots, p'_{|E|-\text{maxL}+1}$ and $p''_2, p''_3, \ldots, p''_{|E|-\text{maxL}+1}$ for the subgraphs $G'_1, G'_2, \ldots, G'_{|E|-\text{maxL}+1}$ and $G''_2, G''_3, \ldots, G''_{|E|-\text{maxL}+1}$, respectively. As in [13], we assume conditional independence to estimate the frequency of G' as follows:

$$\text{freq}(p') = \text{freq}(p'_1) \times \prod_{r=2}^{|E|+\text{maxL}-1} \frac{\text{freq}(p'_r)}{\text{freq}(p''_r)}. \quad (8)$$

Since G' may be partitioned into $G'_1, G'_2, \ldots, G'_{|E|-\text{maxL}+1}$ in several different ways, we select the partition for which frequency estimates of the patterns $P'_1, P'_2, \ldots, P'_{|E|-\text{maxL}+1}$ are obtained along the deepest paths, that is paths with the maximum total split nodes in the MD-Tree or along paths with the fewest contracted nodes in the P-Tree.

4 Experimental Evaluation

In this section, we (1) show the efficiency of the proposed techniques in terms of the accuracy of the estimates and (2) evaluate the situations in which one technique may be preferred over the other.

Datasets. We used part of the SwetoDBLP [18] dataset, which follows a Zipfian distribution. We also experimented on a synthetic graph generated from TOntoGen [6] RDF graph generator, in which edge types/labels are uniformly distributed across their corresponding source and destination node types.

Table 1: Dataset Properties

	SwetoDBLP	TOntoGen
# Nodes	1037856	200001
# Edges	848839	749825
# Unique edge labels	87	9
Avg node degree	1	7

Implementation Details. Implementation is in C++ with experiments performed on a 1.8GHz Dual AMD Opteron processors and 10GB RAM. We created sparse matrices using sparseLib++ [17] libraries and used BRAHMS [18] to parse the graphs.

Summaries. The unsummarized size of all patterns of length three for the SwetoDBLP dataset is 6036340 bytes and the size of the unpruned P-Tree and MD-Tree are 245000 and 259200 bytes, giving a 95% reduction in size. We summarized the P-Tree and MD-Tree, constructing two sets of summaries with budgets 10KB,

25KB and 50KB, with one set tuned for frequent patterns. On the other hand, the TOntoGen dataset has fewer unique edge labels, so it had fewer unique patterns of lengths at most three with an unsummarized size of 10890 bytes. Its unsummarized P-Tree and MD-Tree are 4916 and 7554 bytes, with at least a 30% reduction. We summarized the P-Tree and MD-Tree, constructing two sets of summaries of sizes 1000 bytes, 1250 bytes, and 1500 bytes, with one set tuned for frequent patterns. For both datasets, we used a 5% significance level and a β value of 3 for constructing the MDTree summaries.

Time Analysis. The time for discovering all patterns of length maxL is the most time-consuming part of our approach. Fortunately, it is a preprocessing step and depends on the connectedness of the dataset but will typically take a couple of hours. The time needed for constructing the P-Tree and MD-Tree summaries is in the order of tenths of seconds whereas the estimation time is negligible, running in tens of milliseconds.

Query Workloads. We used two sets of three workloads: (1) the positive workload has patterns with non-zero frequencies; (2) the frequent workload has those with frequencies of at least 500; (3) the negative workload has patterns with zero frequencies. One set contains patterns of length at most three and the other has patterns of length at least four and at most six. All workloads contain 500 randomly selected patterns as appropriate for the workload.

Error metrics. We used the absolute error metric $|freq(p) - freq(p^)|$ to measure the estimation error ($freq(p)$ and $freq(p^)$ are the true and estimated frequencies of p). In our charts, we measure the overall estimation error by cumulating on the y-axis, the percentage of patterns estimated with at most ε error, for $\varepsilon \in [0, E]$ and E, the observed maximum error. The error values (in logscale) are shown on the x-axis.

Accuracy of Positive Queries. Figure 8a – c show the accuracy of estimates from summaries of the SwetoDBLP dataset using 10KB, 25KB and 50KB space. With the 10KB summary, about 20% of queries in the workload are estimated with very high accuracy (with error value 0 or 1). With the 50KB summary, at least 50% of queries are estimated with high accuracy. Recall that the 10KB and 50KB summaries can only hold about 4% and 20% of the patterns in the unsummarized P-Tree (MD-Tree). Thus, their performances are encouraging. The P-Tree performs better than the MD-Tree, since the accuracy of the base MD-Tree assumes edge patterns occur independently while that of the refined MD-Tree depends on the existence of single points of dependence among edge patterns. Although these assumptions do not hold in all sets of patterns for this dataset, the MD-Tree still exhibits an encouraging performance. **Figure 8**g shows the accuracy of the estimates obtained from the summaries of the TOntogen dataset. For lack of space, we show only the summary constructed at 1500 bytes space which can hold only about 30% of the patterns in the original unsummarized P-Tree (MD-Tree). The P-Tree that estimates 40% of the patterns with very high accuracy exhibits an encouraging performance. Although the dataset was created by assigning edge types/labels to node types in a uniform manner, our assumption of uniform growth rate of patterns does not necessarily hold. The MD-Tree, on the other hand, does not perform as well, because the optimal MD-Tree constructed is the base MD-Tree and the assumptions of independence of edge patterns upon which the base MD-Tree rests, does not hold for this dataset.

Accuracy of Frequent Queries. Figure 8d–f show the accuracy of the estimates for the summaries of the SwetoDBLP dataset, tuned to favour frequent patterns. As

expected, the accuracy of the tuned P-Tree surpasses that of the untuned P-Tree. We tuned the MD-Tree by increasing the impact of significant nodes that also had more frequent patterns over others. However, as the performance of the MD-Tree shows, the TKP greedy algorithm may yet prune such a node if its subtree does not fit the budget. **Figure 8**h shows the results from the tuned summaries of the TOntogen dataset. For lack of space, we only show the 1500 byte tuned summary. The tuned P-Tree performed better while the performance of the MD-Tree remained unchanged since the optimal MD-Tree constructed is the base MD-Tree and the independence assumption does not hold for this dataset.

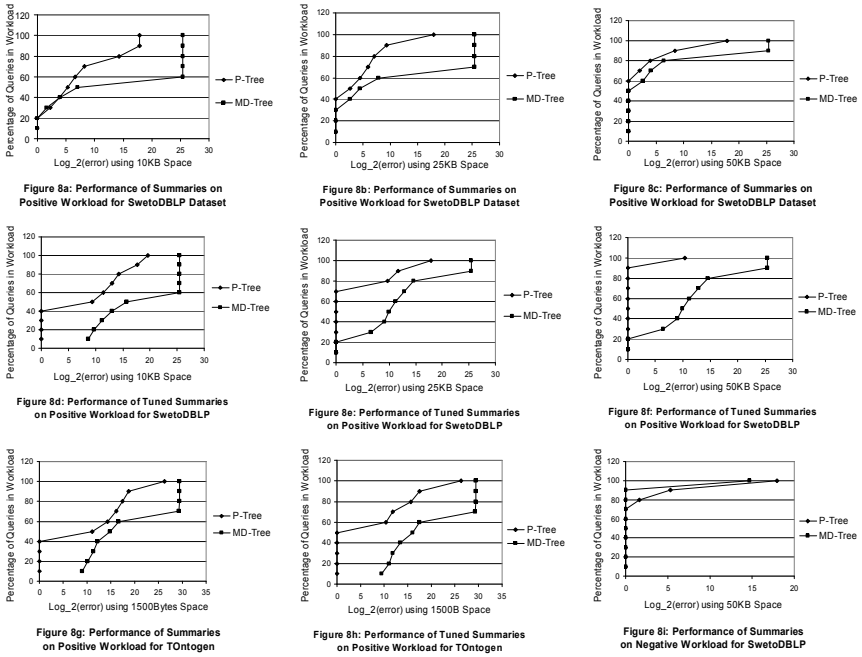

Figure 8a: Performance of Summaries on Positive Workload for SwetoDBLP Dataset

Figure 8b: Performance of Summaries on Positive Workload for SwetoDBLP Dataset

Figure 8c: Performance of Summaries on Positive Workload for SwetoDBLP Dataset

Figure 8d: Performance of Tuned Summaries on Positive Workload for SwetoDBLP

Figure 8e: Performance of Tuned Summaries on Positive Workload for SwetoDBLP

Figure 8f: Performance of Tuned Summaries on Positive Workload for SwetoDBLP

Figure 8g: Performance of Summaries on Positive Workload for TOntogen

Figure 8h: Performance of Tuned Summaries on Positive Workload for TOntogen

Figure 8i: Performance of Summaries on Negative Workload for SwetoDBLP

Figure 8: Comparing the Accuracy of Estimates Obtained from the Summaries

Accuracy of Negative Queries. Figure 8i show the accuracy of the estimates from the summaries of the SwetoDBLP dataset, using 50KB space on the negative workloads. For of lack of space, we show only the result of this largest summary, which represents the worst case scenario for negative queries. Since we encode the structure of the graph in the patterns, we are mostly able to detect patterns whose structures do not exist. Thus both the P-Tree and MD-Tree exhibit very good performances. However, some non-zero estimates are obtained due to spurious paths and cycles that may be introduced in the structure.

For lack of space, we do not show the results of the query workload of longer patterns. However, the results are consistent with those of the shorter patterns albeit with larger estimation errors.

5 Related Work

To the best of our knowledge, ours is the first work on summarizing graphs for subgraph frequency estimation. Work most closely related are techniques that summarize XML data for selectivity estimation for path expressions [1] and twigs[8][9][13]. A fundamental difference between these techniques and ours is the data and query models. All the techniques except [9] assume tree-structured XML data. More important, all the techniques are proposed for either path or twig queries so that it is unclear how they apply to arbitrary graph-structured queries. Our estimation value for patterns is similar in spirit to the notion of δ-derivable twigs introduced in [13] for pruning twigs whose estimated frequencies are within δ error of their true frequencies. However, the technique of [13] may blindly prune a pattern that, if left unpruned, may have caused more twigs to be pruned, thereby reducing the summary size further. In contrast, our value-based approach makes a more informed choice of patterns to be pruned.

Several efforts have been made in using graph-indexing schemes to reduce the cost of processing graph containment queries, over a collection of many disconnected graphs. In these approaches, a graph containment query is processed in two steps. The first step retrieves a candidate set of graphs that contain the indexed features of the query graph. The second step uses subgraph isomorphism to validate each candidate graph. GraphGrep [12] uses a path-based indexing approach that selects all paths of up to length l_p as the indexing feature. The size of the candidate set obtained in the first step could be large since paths do not keep graph structure. To cope with this, GIndex [15] uses frequent graph fragments as the indexing feature. To reduce the large (potentially exponential) number of frequent fragments, only discriminative frequent fragments are kept. Noting that the set of frequent graph fragments contain many more tree than non-tree structures, Tree+Δ [16] indexes frequent trees, reducing the large index construction time of GIndex due to graph-mining. On demand, Tree+Δ further reduces the size of the candidate set by selecting a small portion of discriminative non-tree features related to query graphs only. In complement, our work allows allows for optimizing the subgraph isomorphism tests, in the second step, using estimates of the cardinalities of both indexed and non-indexed fragments of the query. In addition, our technique can also be applied to a large connected graph.

6 Conclusions and Future Work

Structure querying is important for eliciting information from graphs. Optimizing structure queries requires estimating the frequency of subgraphs in a query graph. In this work, we presented two techniques for summarizing the structure of graphs in limited space. The Pattern Tree is relatively stable for all datasets but performs best when graph patterns that share a common sub-graph pattern co-occur. The MD-Tree performs best when single points of dependence exist among subgraphs. As our experiments showed, the untuned MD-Trees had more encouraging results for the SwetoDBLP dataset than the tuned MD-Trees for the same dataset. This is mostly

because our current representation of sparse matrices as a sparse vector and two one-dimensional arrays is feasible only when the sparse matrix is at most half-filled, otherwise, the space overhead results in the pruning of deep maximal impact subtrees. In the future, we will explore more compact alternatives that reduce this space overhead and characterize the performance of the MD-Tree for datasets that have multiple points of dependence. We will also provide a comprehensive evaluation of the benefits of our summaries in terms of speeding up structure queries. In addition, we will look into estimating patterns in graphs such as RDF graphs, which may have subsumption hierarchies on the edges. Further we will investigate techniques for gracefully accommodating updates to the data graph into our summaries.

Acknowledgments. This work is funded by NSF-ITR-IDM Award #0325464 and #071444.

7 References

1. Aboulnaga, A., Alameldeen, A., Naughton, J.: Estimating the Selectivity of XML Path Expressions for Internet Scale Applications. In VLDB, 2001.
2. Burge, C. Identification of Complete Gene Structures in Human Genomic DNA. Ph.D. Thesis, Stanford University, Stanford, CA. 1997
3. Dehaspe, L., Toivonen, H., King, R. D.: Finding Frequent Substructures in Chemical Compounds. In KDD, 1998.
4. Desphande, M. Kuramochi, M. Wale, N.: Frequent Substructure-Based Approaches for Classifying Chemical Compounds. In TKDE. Vol. 17, No. 8. Aug. 05.
5. Klyne, G., Carroll, J. J.: RDF Concepts and Abstract Syntax. W3C Recommendation. (Revised) February 2004. http://www.w3.org/TR/rdf-syntax-grammar/
6. Perry, M. TOntoGen: A Synthetic Data Set Generator for Semantic Web Applications. In SIGSEMIS Bulletin.
7. Pei, J., Dong, G., Zou, W., Han, J.: On Computing Condensed Frequent Pattern Bases. In ICDM, 2002.
8. Polyzotis, N., Garofalakis, M., Ioannidis, Y.: Selectivity Estimation for XML Twigs. In ICDE, 2004.
9. Polyzotis, N., Garofalakis, M.: Statistical Synopses for Graph-Structured XML Databases. In SIGMOD, 2002.
10. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. W3C Working Draft. 19th April 2005. http://www.w3.org/TR/rdf-sparql-query/
11. Shannon, C.E. A Mathematical Theory of Communication, Bell Syst. Tech. Journal 27, 379-423, 623-656. 1948.
12. Shasha, D., Wang, J. T. L., Giugno, R. Algorithmics and Applications of Tree and Graph Searching. In PODS, 2002
13. Wang, C., Parthasarathy, S., Jin, R.: A Decomposition-Based Probabilistic Framework for Estimating the Selectivity of XML Twig Queries. In EDBT, 2006.
14. Yan, X., Han, J.: gSpan: Graph-Based Substructure Pattern Mining. In ICDM, 2002.
15. Yan, X., Yu, P. S., Han, J. Graph Indexing: A Frequent Structure-based Approach. In SIGMOD, 2004.
16. Zhao, P., Yu, J. X., Yu, P. S.: Graph Indexing: Tree + Delta >= Graph. In VLDB, 2007.
17. http://math.nist.gov/sparselib++/
18. http://lsdis.cs.uga.edu/projects/semdis/brahms
19. http://lsdis.cs.uga.edu/projects/semdis/swetodblp