Propagation-Vectors for Trees (PVT): Concise yet Effective Summaries for Hierarchical Data and Trees[*]

Venkata S. Cherukuri
Computer Science and Engineering Department
Arizona State University,Tempe,AZ 85287
Venkata.Cherukuri@asu.edu

K. Selçuk Candan
Computer Science and Engineering Department
Arizona State University,Tempe,AZ 85287
candan@asu.edu

ABSTRACT

Summarization of hierarchical data and metadata is a fundamental operation in applications in many domains. In particular, similarity search of hierarchical data, such as XML, would benefit greatly from concise and indexable summaries. This is especially true in P2P scenarios, where the search needs to be done in a distributed fashion on multiple peers. This situation requires summaries which are small, yet effective in identifying potential peers that need to be further explored. In this paper, we propose a method, called *propagation-vectors for trees (PVT)* which constructs very concise and accurate summaries of hierarchical data, such as XML trees. We then show how to use this summary to perform similarity search on summarized data. The proposed summarization scheme relies on a label-propagation mechanism, which constructs an n-dimensional vector from a given tree with n unique data labels. Experimental results have shown that the constructed PVT summaries capture the structure of the input trees very accurately, the representations are highly concise, and that the search based on these summaries are faster than the existing approaches.

Categories and Subject Descriptors: C.2.4 [COMPUTER-COMMUNICATION NETWORKS][Distributed Systems]: Distributed databases; H.3.1 [INFORMATION STORAGE AND RETRIEVAL][Content Analysis and Indexing]: Indexing methods

General Terms: Algorithms, experimentation.

Keywords: Tree summarization, P2P search.

1. MOTIVATION AND RELATED WORK

With the popularity of XML in scientific, business, and other domains, the amount of data represented and exchanged in tree(-like) forms is increasing. Thus, being able to perform quick (albeit, dirty) look-ups in large tree-collections is a critical capability in many application domains. This is for example the case in P2P data management scenarios, where the search needs to be done in a distributed fashion on multiple peers. Such P2P data management systems are gaining in popularity because of their decentralized and distributed nature, which provides a number of advantages, such as high robustness, better use of the resources, better scalability, and the lack of need for integrated-administration. Consider, for example, the *the Digital Archaeologial Record (tDAR) [15, 20, 21]* project which is developing a *knowledge-based archaeological data integration system (KADIS)* for systematically collected distributed archaeological data. The goal of tDAR/KADIS disciplines to locate, search, and query across distributed archaeological datasets, gathered using incommensurate recording protocols [15]. This requires methods for (1) locating relevant and comparable observations from numerous archaeological datasets, (2) ad hoc data integration where the semantic demands of the query are reconciled with the semantic content and schemas of the available datasets, and (3) for resolve conflicts in concept-oriented query processing that arise from inconsistent recording strategies (represented as taxonomies) [20, 21].

The first step in the above process involves identification of peers in the knowledge network that have the most similar schemas, taxonomies, or simply most compatible data sets. Since the cost of the integration steps (including schema or taxonomy alignment and conflict resolution) needed for query processing could be very high, the selection of good knowledge peers is extremely important. On the other hand, the amount of time spent to select knowledge peers and the amount of metadata indexed (or exchanged between peers during search), both need to be minimal to prevent this initial process from becoming a bottleneck to the effective use of the available data sets. In this paper, we focus on source descriptions (i.e., meta-data, such as taxonomies), which are tree-structured[1]. When looking for a compatible knowledge peer, a tDAR node searches for nodes that have similar tree-structured source descriptions. One way to achieve this is to perform some form of a tree edit distance computation. There has been a lot of work done in computing the edit distances between trees in the past [1, 6, 17, 18, 26, 28, 31, 30, 32]. Two of the most common algorithms that are used to compute edit distances between trees are by Zhang-

[*]This project is supported by NSF grants "Design of Dense RFID Systems for Indexing in the Physical World across Space, Time, and Human Experience" (0725910) and "Archaeological Data Integration for the Study of Long-Term Human and Social Dynamics" (0624341)

[1]We are currently extending the PVT method to graph-structured data.

Sasha [22] and Klein [16]. But the complexities of these algorithms are high: O(n^4) and O(n^3 logn), when comparing trees with n nodes, respectively. Consequently, search for matching peers using tree edit distances quickly becomes infeasible. A potentially more scalable approach to search is to construct easy-to-compare (and easy-to-index) source descriptions and then use them to measure tree similarities. In the literature, there are various tree summarization algorithms [9, 10]. For example, according to the experiments done by Davood *et al.* [22], comparing XML trees using their summaries, did a much better job in clustering XML documents, generated from the same DTD, than by using edit distance values computed by tree edit-distance based algorithms proposed by Nierman and Jagadish [19], Shasha [27], and Chawate [3]. DataGuides [9] was one of the first approaches which attempted to construct structural summaries of hierarchical structures. Though the number of nodes in the summary are less than in the original tree and that they capture the structure well, the constructed summaries are also either graphs or trees. Using such tree or graph structured summaries for computing similarities of the original trees still requires edit distance computations.

To overcome the short coming of tree or graph edit-distance approaches, Davood *et al.* [22] suggest another method of constructing tree summaries. In this approach, one constructs the summary of a given tree as a vector of all $\langle path, frequency\rangle$ pairs, where each path is unique in the hierarchy. From an efficiency perspective, this approach does better than the previous approaches since it results in a vector rather than a graph and comparing two vectors is much more efficient when computing the similarities. But a major drawback is that the number of string comparisons they have to do for a path set of size $ml(l + 1)/2$, where m is the number of root paths and l is the length of each path, is around $O(ml^2)$. While more efficient than computing tree-edit distances, this is still costly for many applications. Helmer [11] suggests another approach which uses the concept of *information distance*. In particular Helmer [11] uses the Ziv-Lempel Encoding and the Ziv-Merhav crossparsing to compute the similarity between the summaries of tree based sources. Flesca et al. [7] represent the structure of a tree as a time series in which each occurrence of a node-label in a given context corresponds to an impulse, thereby representing the entire document as a numerical sequence. These numerical sequences (i.e., time series) are then cheaply compared by using their discrete fourier transforms. The cost of this approach is $O(NLogN)$, where N is the maximum number of nodes in the documents to be compared. Buttler [2] summarizes the structure of the given document by reducing the paths in a document to hash values, which are then compared to the hash summaries of other documents using set union and set intersection operators. But again, in the worst case scenario, this method is has super-linear cost. Nierman and Jagadish [19] proposed a variant of the traditional tree-edit distance algorithm for summary construction. In the traditional tree-edit distance algorithms, only three basic operations were allowed to convert one tree to another : *relabel*, which relabels a given node in a tree to a new label, *insert*, which inserts a new node into the tree, and *delete*, which deletes a single node from a tree. In addition to these three basic operations, they introduced two new operations *insert tree* which inserts a new subtree and *delete tree* which deletes an entire subtree. They assume

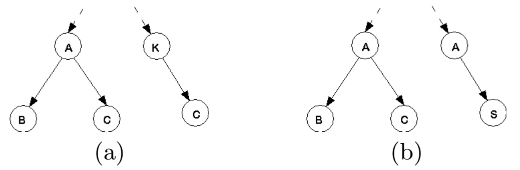


Figure 1: (a) Ancestor-supplied context and (b) descendant-supplied context for node differentiation

unit cost for each of these operations. Given two trees A and B and the set S of all the allowable sequences of edit operations, the edit distance is defined as the minimum of the sum of the costs of each sequence in S. While allowing insertions and deletions of entire sub-trees, this algorithm still requires $O(|A||B|)$ time, where $|A|$ and $|B|$ are the number of nodes of A and B, and thus impractical in general.

As we show in Section 3, even with these existing summarizing approaches, either the search effectiveness is low or the complexity to compute the similarities is high. Thus, the challenge is to find a method which would compute the similarity between trees in a very quick and effective fashion.

In this paper, we propose a method, called *propagation-vectors for trees (PVT)* which constructs very concise and accurate summaries of hierarchical data, such as taxonomies or XML data trees, through a structurally-informed label-propagation scheme. We then show how to use this summary to perform similarity search on summarized data: the problem of computing the similarity between trees is reduced to the problem of computing similarities between summary vectors. The matching process is very fast (linear in n, the number of nodes). While creation of PVT summaries themselves may be costlier, this can be performed off-line and summaries can be indexed for quick-lookup. Especially in decentralized search applications, the fact PVT summaries are concise (linear in the number of unique labels) and also comparison of the resulting summaries is fast (again linear in the sizes of the trees) makes PVT summaries highly advantageous. Furthermore, experiments (reported in Section 3) showed that these summaries are more accurate then the existing schemes in capturing the structure of the trees. between tree based structures.

The rest of the paper is organized in the following manner: In Section 2, we explain the PVT approach and how it can be used to create summaries. Section 2.3 explains how to use these summaries in P2P environments. Section 3 gives a detailed explanation of the experimental set up and discusses the results. In Section 4, we present our conclusions.

2. PVT SUMMARIES

As its name suggests, the *propagation-vectors for trees* (PVT) approach relies on label propagation in the process of obtaining tree summaries. It primarily relies on the following observation

> A node in a given hierarchy clusters all its descendants and essentially acts as a *context* for the descendant nodes (Figure 1(a)). Similarly, descendants of a given node may also act as a context for the node (Figure 1(b)), differentiating the node from others that are similarly labeled.

Consequently, one way to differentiate nodes from each other is to infer the contexts imposed on them by their neigh-

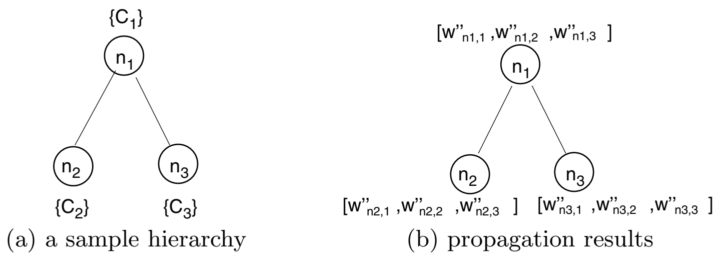

{C1}
n1
n2
n3
{C2}
{C3}
(a) a sample hierarchy
[w"n1,1 ,w"n1,2 ,w"n1,3]
[w"n2,1 ,w"n2,2 ,w"n2,3]
[w"n3,1 ,w"n3,2 ,w"n3,3]
(b) propagation results

Figure 2: Label/weight propagation based mapping of the nodes of a tree onto a multi-dimensional space

bors, ancestors, and descendants in the given hierarchy, enrich (or annotate) the nodes using vectors representing these contexts, and compare these context-vectors along with the label of the node (Figure 2).

2.1 Mapping Nodes of a Tree onto Vectors

Mapping a tree node into a vector (representing the node's relationship to all the other nodes in the tree) requires a way to quantify the structural relationship between the given node and the others in the tree. [24] proposes that the distance between two nodes can be defined as the number of edges on the path between two nodes in the taxonomy. This approach, however, ignores the various structural properties of the tree, including the variations of the local densities in the tree. To overcome this shortcoming, [23] associates weights to the edges in the hierarchy: the edge weight is affected both by its depth in the hierarchy and the local density in the taxonomy. To capture the effect of the depth, [29] estimates the distance between two concepts c_1 and c_2, in a tree-structured taxonomy by counting the number of edges between them, and normalizing this value using the number of edges from the root of the hierarchy to the closest common ancestor of c_1 and c_2.

Concept Propagation /Concept Vector(CP/CV) [14] was originally developed to measure the semantic similarities between terms/concepts in a given taxonomy. Given a user supplied concept hierarchy, $C = \mathcal{H}(N, E)$ with c concepts, it maps each node into a vector in the concept-space with c dimensions. These concept vectors are constructed by *propagating* concepts along the concept hierarchy in such a way that they preserve the semantic relationships among all parent/child concept-nodes. Spreading activation is a general scheme used for propagation of knowledge on data represented in the form of graphs. In spreading activation, activation of one concept in a given node in the graph will spread to several or many related nodes. Spreading activation is used heavily in information retrieval [5] and Web mining [8]. For example, [8] presents a method to improve Web pages annotations using spreading activation (of available annotations) over the Web graph.

Unlike generic spreading activation schemes, in CP/CV, concept vectors (CVs) are generated using a concept propagation (CP) process. CP quantifies the degree of generality of the concepts relative to its neighbors and uses this to inform each node about its neighbors. The result of the iterative propagation is a set of concept vectors (CVs), one for every concept. Each vector represents the relationship of the corresponding concept node with the rest of the nodes.

Measuring Structural Relationship between Parent and Child. Unlike basic spreading activation approaches,

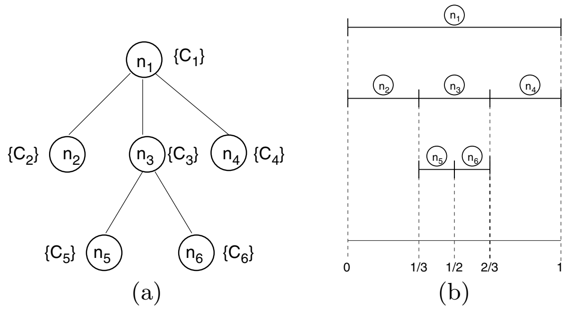


Figure 3: (a) A sample concept hierarchy and (b) the corresponding split of the unit concept range (CP/CV assumes uniform splits in the absence of any prior or external/corpus-based knowledge)

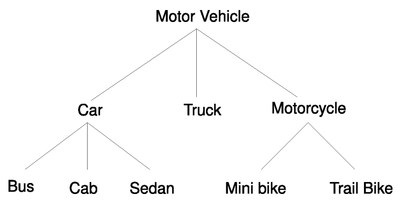
which generally pick a fixed spreading ratio between a pair of nodes, CP/CV spreads concept-labels of the nodes in a given hierarchy in a way that preserves the inherent structural knowledge. Figure 3 provides a sample concept hierarchy and shows how the concepts in the hierarchy share the corresponding concept range. The root node of the hierarchy is the most general concept and occupies the entire range, while the deeper nodes are more specific and have smaller shares. Note that, in the absence of any prior or external/corpus-based knowledge, the children of a given concept-node are assumed to split the concept range of the parent uniformly. In particular, given a node, n_{parent}, and one of its children, n_{child}, CP/CV measures the structural relationship between n_{child} and n_{parent} as $G^{tree}(n_{child}, n_{parent}) = \frac{1}{num\_children(n_{parent})}$. As in [23], this captures the local density of the tree around n_{parent}.

Measuring the Contextual Relationship between Parent's Concept Vector and Child' Concept Vector. In order to translate this structural information into comparable concept vectors, CP/CV also measures the contribution of n_{child} and n_{parent} using the extended Boolean model [25] interpretation of vector spaces. In particular, according to the extended boolean model, if $\mathcal{O} = [0, \ldots, 0]$ denotes the origin of an (m-dimensional vector space, then $\mathcal{O}$ can be interpreted as $(\neg c_1 \wedge \neg c_2 \wedge \neg \ldots \wedge \neg c_m)$ or equivalently as $\neg(c_1 \vee c_2 \vee \ldots \vee c_m)$. Since $\mathcal{O}$ corresponds to $\neg(c_1 \vee c_2 \vee \ldots \vee c_m)$, the contribution of the concepts to a vector V can be measured by $|V - \mathcal{O}|$; i.e., the length, $|V|$, of the vector V. Thus, according to this interpretation of the vectors in the vector space, after the propagation between a node, n_{parent}, and one of its children, n_{child}, the resulting structural relationship between them can also be measured as $G^{vs}(n_{child}, n_{parent}) = \frac{|V_{n_{child}}|}{|V_{n_{parent}}|}$, in terms of the corresponding vectors.

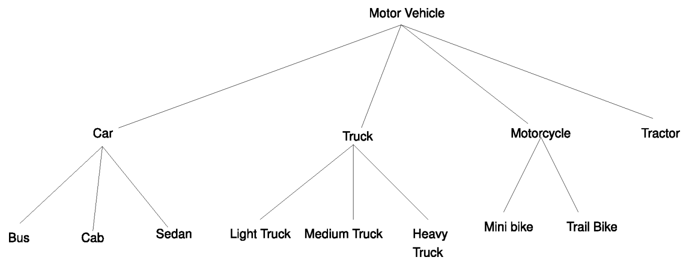
Propagation. CP/CV leverages this (tree-node/vector-space-node) duality of the nodes in the hierarchy to pick a degree of propagation between n_{parent} and n_{child} such that

$$G^{tree}(n_{child}, n_{parent}) = G^{vs}(n_{child}, n_{parent}).$$

CP/CV is an iterative process, repeated until all concepts have chance to get propagated across all nodes: Before the propagation process starts, concept-vectors of the nodes are simply initialized with the concepts corresponding to each



(a) Tree 1



(b) Tree 2

Figure 4: Two taxonomy segments

node; i.e., if the node n_i in the concept hierarchy corresponds to the concept c_i, then the initial concept-vector of this node is $V_{n_i} = [0, 0, \ldots, 1, \ldots, 0]$, where the only non-zero weight is associated with the concept-dimension, c_i.

Let us consider a parent node, *par*, its children, $chld_i \in Chld(par)$, and the corresponding concept-vectors, $V_{par} = [w_{par,1}, w_{par,2}, \ldots, w_{par,m}]$ and $V_{chld_i} = [w_{chld_i,1}, w_{chld_i,2}, \ldots, w_{chld_i,m}]$. The *propagation degrees* ($\alpha_{par \rightarrow chld_i}$ and $\alpha_{chld_i \rightarrow par}$) between the parent node and its children are such that after propagation, we obtain *concept-vectors*, $V'_{par} = [w'_{par,1}, w'_{par,2}, \ldots, w'_{par,m}]$ and $V'_{chld_i} = [w'_{chld_{i,1}}, w'_{chld_i,2}, \ldots, w'_{chld_i,m}]$, where

$$w'_{par,k} = w_{par,k} + \left(\sum_{chld_i \in Chld(par)} \alpha_{child_i \rightarrow par} \times w_{chld_i,k} \right),$$

and for all $chld_i \in Chld(par)$ we have

$$w'_{chld_i,k} = w_{chld_i,k} + \alpha_{par \rightarrow chld_i} \times w_{par,k}.$$

CP/CV arbitrarily sets the *propagation degrees* from children to parent (i.e., $\alpha_{child_i \rightarrow par}$) to 1.0 and solves for the *corresponding* propagation degrees from parent node to its children (i.e., $\alpha_{par \rightarrow chld_i}$) relying on the *preservation* principle described above.

By propagating concepts between a parent node and its children, CP/CV enables the nodes' concept-vectors to get enriched based on their relationships relative to each other. The concept propagation process is performed iteratively on the entire hierarchy until all concepts have a chance to be propagated across the entire tree. The details of the CP/CV process can be found in [14]. In this paper, we note that the concept vector of a given node represents its structural relationship to all the other nodes in the tree. We further note that, due to the root node's special position in the hierarchy, it can be safely said that the vector corresponding to the root is also an accurate summary of the entire hierarchy.

2.2 PVT: Root Vectors as Tree Summaries

As described above, the CP/CV technique [14] was originally developed to measure the similarities between concept-nodes in a given taxonomy. Each node in the given hierarchy is represented by a vector (CV), obtained through a concept-label propagation (CP) process. For example, Figures 4 and 5 shows sample concept (IS-A) hierarchies and the CV for the root nodes. When an application needs the degree of similarity between any pair of concept-nodes in the hierarchy, the corresponding CP/CV vectors are compared.

Unlike the original use of CP/CV, in comparing the concept-nodes in a single hierarchy, in this paper we propose

Keyword	Weight
Motor Vehicle	0.165
Car	0.152
Bus	0.053
Cab	0.053
Sedan	0.053
Truck	0.218
Motorcycle	0.168
Mini bike	0.069
Trail Bike	0.069

(c)CV of the root of Tree 1

Keyword	Weight
Motor Vehicle	0.121
Car	0.116
Bus	0.039
Cab	0.039
Sedan	0.039
Truck	0.116
Light Truck	0.039
Medium Truck	0.039
Heavy Truck	0.039
Motor cycle	0.129
Mini bike	0.052
Trail Bike	0.052
Tractor	0.170

(d) CV of the root of Tree 2

Figure 5: The CP/CV keyword vector for the root of the trees in Figure 4

to use the vectors associated to the nodes of *two* hierarchies to compare the two hierarchies themselves. In this, we face the following problems:

- while the vectors corresponding to the nodes of a single hierarchy all have the same dimensions (i.e., they are in the same space), this is not necessarily the case for vectors of nodes from different hierarchies.
- in order to compare two hierarchies using the sets of vectors corresponding to their nodes, we need to decide vectors of which nodes from one hierarchy will be compared to the vectors of which nodes from the other.

The PVT method handles the first challenge above by mapping the vector spaces corresponding to the two trees onto a larger space containing all the dimensions of the given two spaces. Since in many hierarchies (such as XML data), we can have many nodes with the identical label, before integrating the spaces of the given two trees, the dimensions corresponding to repeating labels of the individual trees are collapsed into a single dimensions. Say an attribute repeats four times in an XML document. The original vectors for the root node would contain four different entries corresponding to the nodes having same attribute label. But, in PVT, we need one unique dimension which effectively represents all these four nodes. To achieve this PVT computes the magnitude of the combined label as the square root of the sum of the squares of the individual components. More generally, let S be the set of nodes in a given tree, all with the same label. The combined weight, w_S, is computed as $w_S = \sqrt{(\sum_{n_i \in S} {w_{n_i}}^2)}$. Note that after the collapse of the dimensions corresponding to the identically-labeled nodes in S, the magnitude of the new vector remains the same as that of the original vector. Thus we have transformed the original vector from one space to another space with reduced dimensions, but keeping its energy (or the distance from the origin, $\mathcal{O}$, the same. Since CP/CV (which is leveraged by PVT to obtain the node vectors) quantifies the relationship between the nodes in the hierarchy using the magnitudes of the corresponding vectors, this means that the relationships between the nodes have been preserved throughout this transformation.

A naive solution to the second challenge would be simply to compare two sets of vectors corresponding to the nodes of the given two trees. In order to reduce the complexity of the process, on the other hand, PVT relies on the special position of the root node in the hierarchies. Since the vector corresponding to the root node represents the context provided to it through all its descendants (i.e., the entire hierarchy), the vector representation for the root node could be considered as a structural summary for the entire tree. Thus, the vector corresponding to the root of a given hierarchy can be used for computing similarity of the hierarchy to others. For example, consider the two related taxonomy segments and the corresponding vectors in Figures 4 and 5. The compositions of the vectors are also similar.

2.3 Use of PVT Summaries in Peer Search

Since PVT summaries are less complex than the corresponding tree-structured source descriptions, yet capture their structures, they can act as the representatives of the original source descriptions. Once the summary of the source description has been constructed, as described in the previous section, the next step is to find those peers that have similar source descriptions. Since each peer calculates the summary of its meta data (source description) off-line and then stores the resulting summary vector to be used in similarity comparisons, peer discovery involves looking for information peers with matching summary vectors and then picking the closest nodes in terms of their source descriptions. The PVT summary vectors can be compared using different similarity/difference measures: cosine similarity (measuring the angles between the vectors),

$$sim_{cosine}(\vec{v_1}, \vec{v_2}) = cos(\vec{v_1}, \vec{v_2}),$$

average KL divergence (which treats the vectors as probability distributions and measures the so-called relative entropy between them),

$$\frac{\Delta_{KL}(\vec{v_1}, \vec{v_2}) + \Delta_{KL}(\vec{v_2}, \vec{v_1})}{2} = \frac{1}{2}\sum_{i=1}^{n} v_{2i} log\frac{v_{2i}}{v_{1i}} + v_{1i} log\frac{v_{1i}}{v_{2i}},$$

and intersection similarity (which considers to what degree $\vec{v_1}$ and $\vec{v_2}$ overlaps along each dimension)

$$sim_{intersection}(\vec{v_1}, \vec{v_2}) = \frac{\sum_{i=1}^{n} min(v_{1i}, v_{2i})}{\sum_{i=1}^{n} max(v_{1i}, v_{2i})}$$

are candidates. We note that, the cosine similarity will give more weight to dimensions with high values (these tend to correspond to those tags higher up in the hierarchy - i.e., closer to the root), whereas average KL distance and intersection similarity will essentially normalize each dimension, thus will likely give equal weight to all tags, independent of where they occur. In the experiments section, we evaluate the above measures for PVT-based search.

For a given tree source, the PVT summary (the vector corresponding to the root node), consists of only the unique labels in the data. Thus, if n is the total number of node labels in a given hierarchy and u is the number of unique node labels, the space complexity for our approach is $O(u)$. For a large XML document generated from a DTD where there are a large number of repeating node labels, the vector for the root node, consisting of only the unique node labels, is likely to be much smaller. In the worst case, when all the nodes in the given hierarchy are different, then the space complexity for our approach is equal to the number of nodes in the hierarchy, i.e., $O(n)$. Compare this for example with the four different methods, *tags*, *pairwise*, *full path* and *family order*, proposed in [11] to generate structural summaries of the documents. For each of these approaches, the space that is required to store the summaries is $O(n)$, since all the repeating nodes appear in the summary. Consequently, especially for large data trees where $n >> u$, PVT would outperforms this scheme.

While the creation of PVT summaries require $O(dn)$ exchanges, where d is the depth of the tree, between the parent/child node pairs [14], this can be performed offline and summaries can be indexed for quick-lookup. Especially in decentralized P2P search applications, the facts that PVT summaries are concise (linear in the number of unique labels) and also comparison of the resulting summaries is fast (again linear in the sizes of the trees) make PVT summaries highly advantageous. Experimental results in Section 3 show that summaries constructed through PVT accurately represent the trees. In addition to this, PVT summaries are much smaller than those which are obtained by other competing methods, such as [11]. This implies that, the similarity comparisons using this method are much faster and also the amount of space that these summaries occupy is smaller. This is a significant improvement in peer-to-peer information search scenarios compared to some other methods which compute easy-to-compare summaries, which might in the worst cases be bigger than the actual tree itself.

3. EXPERIMENTS

This section describes discusses the experimental results.

3.1 Experimental Set-up

3.1.1 Document Collections

Helmer data set: In [11], Helmer presents comparisons of various summarization algorithms. In order to leverage the results reported in that paper as a way to compare PVT to other algorithms reported in the literature (including the approaches proposed Helmer, as well as with others reported in that paper, Nierman and Jagadish [19], Flesca *et. al* [7], and Buttler [2]) we also ran experiments on the Helmer data set [11]. These data sets consist of both real as well as synthetic data. The real data sets are made up of 57 documents from the XML Version of the Sigmod Record, 60 documents from the heterogeneous track at INEX 2005 and 34 music sheets encoded in XML. For the synthetic data sets, Helmer uses the two DTD's from Flesca *et. al* [7], DFT1 and DFT2, along with four other data sets: *element data set*, *frequency data set*, *position data set*, *depth data set*. As the names of these last four data sets suggest, they were generated by varying: tag names, frequency of appearance of different elements, position of the elements within the document, depth of the document from cluster to cluster respectively. For each data set eight different clusters were generated and each cluster had ten documents. For a detailed description of these data sets please refer to [11].

Table 1: XML generator settings for Data Sets DS1...DS4 (each with 700 documents)

	DS1	DS2	DS3	DS4
r	4	4	8	8
i	0.75	1	0.75	1

Distinct DTD data set: As shown in The document structures used in the Helmer data set [11] are very specific. Thus, to make sure that the results are not specific to these data sets abut are further generalizable, we have also experimented with more randomized data sets.

In order to observe the performance of summarization process in further detail, we first created four new data sets: *DS1*, *DS2*, *DS3*, *DS4*. For each data set, we used seven DTDs[2] containing different elements. We generated random documents from these DTDs using an XML Generator [12]. For different data sets, we varied the parameters r (maximum number of times a node can appear as a child of its parent) and i (implied odds, i.e 1/probability that an implied/optional attribute will not appear in the data) for each of the data sets (Table 1). For each data set and for each of the seven DTDs, 100 documents were created. Hence, each data set contains 700 documents.

Partially Hybrid DTD data set: We also created four *partially hybrid* datasets, *pHyb1*, *pHyb2*, *pHyb3*, and *pHyb4*, each of which contains 300 documents generated from three DTDs: two original DTDs and a third which is a partially hybrid DTD, generated by combining only the top-level elements of the two DTDs at the root level.

Fully Hybrid DTD data set: In addition to creating partially hybrid DTDs (which only involve root level combination), we also created another set of datasets, *fHyb1*, *fHyb2*, *fHyb3*, and *fHyb4*, in which the Hybrid DTD corresponding to the data sets are generated by combining the elements from the source DTDs not only at the root level but also at the lower levels. Thus, thus the resulting DTDs combine features of the original DTDs at multiple levels. As in the partially hybrid DTD data sets, each fully Hybrid DTD data set contains 300 documents, 100 each from the source DTDs and 100 from the hybrid DTD.

3.1.2 Quality Measure

One intuitive quality measure, also leveraged by the other papers in the literature, is to measure and compare the clustering performance of different schemes: one can cluster a given set of documents based on the document similarities computed using different schemes and then count the number of documents that were put into the wrong cluster; *the lower the mis-clusterings, the better the comparison scheme.*

Nierman [19] and Helmer [11] use hierarchical agglomerative clustering (HAC), a popular clustering method, to cluster documents in their experiments. The goal of a hierarchical clustering algorithm is to create a tree or a hierarchy of clusters, from the fine-grained to the coarse-grained [13]. The lowest level of the cluster hierarchy (i.e the leaves), consist of documents in their own clusters, where as the highest level (i.e the root cluster) consists of all the documents in the given set. The hierarchical agglomerative clustering, HAC, is a bottom up hierarchical clustering approach where we start off with each document in a separate cluster and we repeatedly merge the closest pair of clusters until all the documents are members of the same cluster. Central to the HAC algorithm is the process of identifying the closest pair of clusters at every iteration. Given a pair of clusters, C_1 and C_2, we used the average link measure to compute the distance between them: $\delta(c_1, c_2) = \frac{\Sigma_{\mathbf{x}\in C_x}\Sigma_{\mathbf{y}\in C_y}\delta(\mathbf{x},\mathbf{y})}{|C_x||C_y|}$, where $\delta(\mathbf{x},\mathbf{y})$is the distance between the documents $\mathbf{x}$ and $\mathbf{y}$ respectively[3]. Once all the pairwise cluster distances are found, HAC merges the pair of clusters with the minimum distance into a single cluster. This process of clustering is then repeatedly applied until only one cluster, containing all documents, is obtained. Nierman and Jagadish [19] define the number of *mis-clusterings* as the minimum number of documents that have to be moved from one cluster to another so that all the documents belonging to a particular DTD are in the same cluster. Since this definition has been used in various previous works (including [11, 22, 19]), we compared the performance of our approach with the other approaches based on this definition of quality.

3.2 Results and Discussion

Helmer data set: Using the experimental set up described above, we first conducted experiments on the Helmer data set to compare the mis-clusterings returned by PVT summaries with those returned by the other competing approaches. Table 2 shows the mis-clusterings of our approach as compared with the other approaches. These results shows that the PVT approach (which summarizes the structure of a document by using the CV of the root node) does much better than most other approaches on this data set. In particular the PVT approach based on the KL Divergence performs better than the cosine and the intersection similarity. Its worthwhile to note that different DTDs in the data sets, *SIGMOD*, *INEX*, and *Music* have only a few elements in common. Thus, there are no mis-clusterings for these when using PVT. Interestingly, some of the other summarization schemes return mis-clusterings even under these conditions.

Path shingles return a comparable mis-clustering ratio, but only when the expensive full-path scheme is used. Among all the algorithms, Ziv-Merhav cross-parsing is the most competitive (especially on the particular data sets used in [11]), performing as good or better than the PVT-*kLdistance* approach. Notably, however, both Ziv-Merhav and gzip return mis-clusterings even for data sets (such as SIGMOD, INEX, Music) containing documents from DTDs with mostly distinct element names.

Distinct DTD data set: In order to study the impact of this last observation on more general data sets, we next considered the *distinct DTD data set* containing a larger number of documents, obtained from DTDs with fully distinct elements. Table 3 shows the mis-clusterings returned by our approach vs. Ziv-Merhav and gzip. As can be seen here, while PVT perfectly distinguishes documents, Ziv-Merhav cross-parsing results in very large number of mis-clusterings.

While surprising, this poor performance on such distinct documents actually stems from the way Ziv-Merhav cross-parsing is used in [11]. Based on the algorithm, while cross-parsing two documents doc_a and doc_b, the non-occurrence

[2]`dri.dtd`, `blastxml.dtd`, `roamops-phonebook.dtd`, `dsml.dtd`, `giml.dtd`, `837institutional.dtd`, `giml.dtd`, `qaml-xml.dtd`, all taken from `http://www.cs.ualberta.ca/~drafiei/dtds.html`

[3]In PVT, clustering is achieved both by using *similarity*:cosine and intersection as well as *distance*: KL Divergence between the PVT summaries of the given pair of documents, $\mathbf{x}$ and $\mathbf{y}$.

Table 2: Number of mis-clusterings for various approaches (the results, except for PVT, Ziv-Merhav, and gzip), are those reported in [11]. The Ziv-Merhav and gzip results reported here are better than the ones in [11], because in a private exchange, the authors provided us with an improved implementation

	SIGMOD	**INEX**	**Music**	**DFT1**	**DFT2**	**Elem**	**Freq**	**Pos**	**Depth**	**Overall**
Num. of docs	57	60	34(*)	140	140	80	80	80	80	
Tree-edit[19]	1	0	13	26	0	0	0	5	0	6.0%
DFT[7]										
direct ML	0	3	9	52	1	27	9	32	33	22.1%
pairwise ML	0	4	7	39	3	10	0	42	22	18.1%
Path Shingles[2]										
tags	0	0	0	42	0	0	0	14	48	13.8%
pairwise	0	0	1	6	0	0	0	6	39	6.9%
full path	0	0	0	6	0	0	0	0	30	4.8%
gzip[11]										
simple	1	2	0	15	19	3	29	15	44	17.0%
tags	0	0	0	7	20	0	16	0	6	6.5%
pairwise	0	0	0	7	38	0	20	0	26	12.1%
full path	0	0	1	2	38	0	24	0	3	9.1%
family order	0	0	0	23	34	0	13	8	0	10.4%
Ziv-Merhav[11]										
tags	0	2	0	1	0	0	0	0	0	0.4%
pairwise	0	2	0	6	0	0	7	0	20	4.7%
full path	0	2	0	6	0	0	7	0	0	2.0%
family order	0	0	0	1	0	0	0	2	0	0.4%
PVT										
cosine sim.	0	0	0	10	0	0	12	5	4	4.13%
intersection similarity	0	3	0	6	0	0	12	11	2	4.53%
KL distance	0	0	0	6	0	0	0	0	3	1.19%

(*) For PVT, only 33 out of 34 the Music documents are used. While summaries themselves are compact, during summarization process, PVT keeps all vectors of all nodes in the memory. Thus, summarization of one of the 34 documents, with >30K nodes, required more memory than available. We are currently implementing a secondary-storage supported, less-memory intensive version for summarizing very large trees

Table 3: Comparison of the number of misclusterings on the distinct DTD data set

	DS1	**DS2**	**DS3**	**DS4**	**Overall**
Num. of docs	700	700	700	700	
gzip					
tags	34	5	32	79	5.35%
pair wise	6	0	36	23	2.32%
full path	11	0	25	27	2.25%
family order	5	0	25	3	1.17%
Ziv-Merhav					
tags	34	25	26	25	3.92%
pair wise	181	169	190	156	24.85%
full path	178	174	190	161	25.10%
family order	149	116	126	116	18.10%
PVT					
Cosine	0	0	0	0	0%
Intersection	0	0	0	0	0%
KL Distance	0	0	0	0	0%

Table 4: Comparison of the number of misclusterings on the partially hybrid DTD data set

	pHyb1	**pHyb2**	**pHyb3**	**pHyb4**	**Overall**
Num. of docs	300	300	300	300	
gzip					
tags	13	8	59	36	9.66%
pair wise	19	0	71	35	10.41%
full path	19	1	63	21	8.66%
family order	21	0	66	9	8%
Ziv-Merhav					
tags	77	35	77	62	20.92%
pair wise	67	35	84	0	15.5%
full path	67	35	89	0	15.91%
family order	58	29	27	36	12.5%
PVT					
cosine	0	19	0	0	1.58%
Intersection	19	0	37	0	4.67%
KL Distance	49	0	19	0	5.66%

of any prefix of doc_a, in doc_b affects the distance between the two documents. Ideally, given three documents which are totally different from each other, doc_1, doc_2 and doc_3, the distance between all pairs of these documents should all be the same, i.e 1 when normalized. But, since each non-occurrence of a prefix counts towards the total dissimilarity score, the dissimilarity between documents which are totally different from each other depends only on the length of the documents. The gzip approach performs better than Ziv-Merhav, but still results in a large number of mis-clusterings.

Partial and Fully Hybrid DTD data sets: Since distinguishing documents from DTDs with fully distinct element sets serves little purpose, we also used *partially and fully hybrid DTD data sets.* These contain DTDs obtained by merging various DTDs. Table 4 and Table 5 show the number of mis-clusterings returned by our approach and by [11] on the *partially hybrid* and the *fully hybrid* data sets.

Table 5: Comparison of the number of misclusterings on the fully hybrid DTD data set

	fHyb1	**fHyb2**	**fHyb3**	**fHyb4**	**Overall**
Num. of docs	300	300	300	300	
gzip					
tags	48	28	32	0	9%
pair wise	42	10	84	0	11.33%
full path	63	17	38	0	9.83%
family order	29	10	33	0	6%
Ziv-Merhav					
tags	119	0	57	43	18.25%
pair wise	139	0	45	52	16.41%
full path	139	0	50	0	15.75%
family order	94	0	41	52	15.58%
PVT					
cosine	40	0	48	0	7.33%
Intersection	98	0	43	0	11.75%
KL Distance	40	0	34	0	6.16%

Table 6: Comparison of run times for computing all-pairs similarities for 700 documents

	Time (seconds)
gzip	
tags	3359
pairwise	3439
full path	3466
family order	3424
Ziv-Merhav	
tags	529
pairwise	478
full path	795
family order	485
PVT	228

For the *partially hybrid* data sets, the PVT approach easily outperforms the other approaches. Once again, Ziv-Merhav returns large degrees of mis-clusterings. The gzip scheme performs, again, better than Ziv-Merhav cross-parsing, but the degree of mis-clusterings are significantly worse than that of PVT.

In the case of the *fully hybrid* data sets, once again, PVT performs consistently better than Ziv-Merhav and gzip. On all data sets, PVT performs much better than the Ziv-Merhav cross-parsing. The difference between gzip and PVT, on the other hand, drops. In particular, family order implementation of gzip and KL-distance based implementation of PVT are competitive. As shown next, however, gzip's performance comes at the cost of execution time.

Execution times: In addition to capturing the structures of the tree based sources accurately, the PVT based summaries are smaller than the other schemes. As a consequence, not only the space overhead for storing these summaries is small relative to the storage requirements of other schemes, but also, thanks to PVT summaries being more concise than the alternatives, distance computations between pairs of data trees based on their PVT summaries are also relatively more efficient.

Table 6 shows the time taken by Ziv-Merhav, gzip, and PVT approaches to, given summaries of the documents, to compute the pairwise similarities for the 700 document set. As can be seen from this table, pair-wise similarity comparisons using the PVT approach is relatively faster. Note that, for these comparisons, we used the implementation provided to us by the authors of [11], who have indicated that their implementation was not optimized for efficiency. Therefore, these must be considered only as ball-park figures.

4. CONCLUSION

In this paper, we presented a tree summarization method to enable nodes in an information network to find information peers, both quickly and accurately. The proposed *propagation vectors for trees* (PVT) method constructs summaries using a label (and weight) propagation scheme which maps each node in the tree into a multi-dimensional vector space, where each dimension corresponds to a unique data label. PVT then uses the vector representation of the root node as the summary. These summaries are the used to compute similarities between the trees. Experiments have showed that the PVT summaries are highly accurate and the similarity computations are faster than the existing approaches. Future work includes deployment in an actual P2P scenario and analysis of the impact on the P2P network.

5. REFERENCES

[1] P. Bille. Tree edit distance, alignment distance and inclusion. IT Univ. of Copenhagen TR-2003-23, 2003

[2] D.Buttler. A short survey of document structure similarity algorithms. ICIT 2004.

[3] S. S. Chawathe. Comparing hierarchical data in external memory. VLDB, 1999.

[4] R. Cilibrasi and P. Vitanyi. Clustering by compression. *IEEE Transactions on Information Theory*, 2005.

[5] F. Crestani. Application of spreading activation techniques in informationretrieval. *Artif. Intell. Rev.*, 11(6):453–482, 1997.

[6] M. Farach and M. Thorup. Sparse dynamic programming for evolutionary-tree comparison. *SIAM Journal on Computing*, 26(1):210–230, 1997.

[7] S. Flesca, G. Manco, E. Masciari, L. Pontieri, and A. Pugliese. Fast detection of xml structural similarity. *IEEE TKDE*, 17(2):160–175, 2005.

[8] F.Gelgi, S.Vadrevu, H.Davulcu. Improving web data annotations with spreading activation. WISE 2005.

[9] R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases, VLDB'97.

[10] R. Goldman and J. Widom. Approximate dataguides. In *Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats*, 1999.

[11] S. Helmer. Measuring the structural similarity of semistructured documents using entropy. VLDB 2007.

[12] IBM XML Generator. http://www.alphaworks.ibm.com/tech/xmlgenerator.

[13] Hierarchical Clustering. http://www.cs.rpi.edu/∼zaki/dmcourse/notes /chapter12.pdf

[14] J. W. Kim and K. S. Candan. Cp/cv: Concept similarity mining without frequency information from domain describing taxonomies. CIKM 2006.

[15] K. Kintigh. The promise and challenge of archaeological data integration. *American Antiquity*, 71(3):567–578, 2006.

[16] P. Klein. Computing the edit-distance between unrooted ordered trees. *Proceedings of the 6th Annual European Symposium*, number 1461, 1998.

[17] S. Lu. A tree-to-tree distance and its application to cluster analysis. *IEEE Trans. PAMI*, 1:219–224, 1979.

[18] F. Luccio and L. Pagli. Approximate matching for two families of trees. *Inf. Comput.*, 123(1):111–120, 1995.

[19] A. Nierman and H. V. Jagadish. Evaluating structural similarity in XML documents. WebDB 2002.

[20] Y. Qi, K. S. Candan, and M. L. Sapino. Ficsr: feedback-based inconsistency resolution and query processing on misaligned data sources. SIGMOD 2007.

[21] Y. Qi, K. S. Candan, M. L. Sapino, and K. W. Kintigh. Integrating and querying taxonomies with quest in the presence of conflicts. SIGMOD 2007.

[22] D. Rafiei, D. L. Moise, and D. Sun. Finding syntactic similarities between xml documents. DEXA 2006.

[23] R. Richardson and A. F. Smeaton. Using WordNet in a knowledge-based approach to information retrieval. Technical Report CA-0395, Dublin, Ireland, 1995.

[24] R.Rada, H.Mili, E.Bicknell, and M.Blettner. Development and application of a metric on semantic nets. *IEEE Transactions on Systems, Man and Cybernetics*, 19(1), 1989.

[25] G. Salton, E. A. Fox, and H. Wu. Extended boolean information retrieval. *Commun. ACM*, 26(11):1022–1036, 1983.

[26] S.Selkow. The tree-to-tree editing problem. *Information Processing Letters*, 6(6):184–186, 1977.

[27] D. Shasha, and K. Zhang Approximate Tree Pattern Matching *Pattern Matching Algorithms*, 341–371, 1997

[28] K.-C. Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, 1979.

[29] Z. Wu and M. Palmer. Verb semantics and lexical selection. In *32nd. Annual Meeting of the Association for Computational Linguistics*, pages 133 –138, 1994.

[30] K. Zhang. The editing distance between trees: Algorithms and applications. *PhD thesis, Courant Institute, Departement of Computer Science, 1989.*

[31] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, 1989.

[32] K. Zhang, J. T. L. Wang, and D. Shasha. On the editing distance between undirected acyclic graphs and related problems. SoCPM 1995.