TOSS: An Extension of TAX with Ontologies and Similarity Queries

Edward Hung Yu Deng V.S. Subrahmanian
Computer Science Department
University of Maryland
College Park, MD 20742
{ehung, yuzi, vs}@cs.umd.edu

ABSTRACT

TAX is perhaps the best known extension of the relational algebra to handle queries to XML databases. One problem with TAX (as with many existing relational DBMSs) is that the semantics of terms in a TAX DB are not taken into account when answering queries. Thus, even though TAX answers queries with 100% precision, the recall of TAX is relatively low. Our TOSS system improves the recall of TAX via the concept of a *similarity enhanced ontology* (SEO). Intuitively, an ontology is a set of graphs describing relationships (such as isa, partof, etc.) between terms in a DB. An SEO also evaluates how *similarities* between terms (e.g. "J. Ullman", "Jeff Ullman", and "Jeffrey Ullman") affect ontologies. Finally, we show how the algebra proposed in TAX can be extended to take SEOs into account. The result is a system that provides a much higher answer quality than TAX does alone (quality is defined as the square root of the product of precision and recall). We experimentally evaluate the TOSS system on the DBLP and SIGMOD bibliographic databases and show that TOSS has acceptable performance.

Keywords

XML Databases, Ontologies, Semantic Integration of Heterogeneous Data, Similarity Enhancement.

1. INTRODUCTION

Recent years have seen a strong upsurge of interest in XML databases. TAX[8] is one of the best known attempts to develop an algebra for XML databases. As in the case of the relational algebra, TAX does not attempt to take the semantics of words and strings into account when answering queries. For example, a query that wishes to find all papers in the DBLP database written by "J. Ullman" will not find bibliographic references to papers by "J.D. Ullman" or by "Jeffrey Ullman." This is because TAX does not use any notion of *similarity* between search terms to answer queries. Likewise, TAX cannot answer queries of the form "Find all papers having at least one author from the US government." Such a query may be useful in order to identify papers that potentially have no copyright restrictions. However, few authors if any will list their affiliations as "US Government." They are more likely to list their affiliations as "US Census Bureau" or "US Army" and so on. As a consequence, TAX will miss these answers. Problems such as these are not really an artifact of TAX's design - rather, they are caused by a general lack of *lexical semantics* in answering queries. Most commercial relational DBMSs would not perform such reasoning either when answering queries.

The net effect of this shortcoming of TAX (as well as most commercial DBMSs) is that such systems have high precision[1] (all the answers to a query are correct) but poor recall[2] (not all the answers that should be returned are in fact returned). *The primary goal of this paper is to extend and enhance the semantics of TAX so that the resulting system returns high quality answers.* The quality of an answer is the square root of the product of the precision and recall of the answer[14].

In this paper, we introduce two mechanisms to alleviate these two problems.

1. We first introduce the concept of an *ontology* to capture inter-term lexical relationships. In the above example, lexical relationships such as part_of reflect the fact that "US Census Bureau" is part_of "US government". Likewise we may say that "Google" isa "web search company" isa "Computer company" isa Company (in case people want to ask the DBLP database for a list of all authors of papers written by someone in a web search company). An *ontology extended semistructured instance* (OES instance) consists of a semistructured instance (e.g. an XML instance) and an associated ontology.
2. When answering queries spanning multiple OES instances (as in the case of join, union, intersection, etc.), there is a need to *merge* the ontologies associated with these instances to avoid any potential semantic ambi-

SIGMOD 2004 June 13-18, 2004, Paris, France.

[1] Formally, the precision of an answer is the number of correct answers returned divided by the number of answers returned.

[2] Formally, the recall of an answer is the number of correct answers returned by the algorithm divided by the total number of correct answers that should have been returned.

guities such as different terms are used for the same concept (problem of synonymy) or the same term for different concepts. We adapt the algorithms in [3, 2] to merge ontologies taking such context into account.

3. As shown in the examples above, we need methods to identify similarities between terms (even if the terms are not necessarily semantically identical). For example, the strings "J. Ullman", "J.D. Ullman" and "Jeffrey D. Ullman" may all refer to the same DB researcher. However, a lexical semantical system such as WordNet [7] cannot match all these individuals together. Rather, some notion of similarity is needed (and many such as Levenstein distance, Monge-Elkan distance [12], Jaro metric[9], Jaccard similarity[5], rule-based similarity where a set of domain-specific rules are used to define what is similar to what are available in the text processing literature). *We do not propose to reinvent notions of string/lexical similarity in this paper.* However, there is the delicate issue of how to take similarities into account when merging ontologies together. In other words, does the fact that "J. Ullman" is deemed to be similar to "Jeff Ullman" and the fact that Jeff Ullman is a Professor at Stanford allow us to infer the "J.Ullman" is a Professor at Stanford? We show that once ontologies are merged together *without* taking similarity concepts into account, we can apply a *similarity enhancement* operation to the merged ontology to capture such notions of similarity. The resulting similarity enhanced ontology (SEO for short) can be generated using *any* notion whatsoever of similarity between strings with a specified threshold.
4. We show how to extend the TAX algebra to answer queries w.r.t. such a similarity enhanced ontology. Our algebra is called TOSS.
5. Finally, we develop a prototype implementation of TOSS on top of the Apache Xindice XML database system[18]. We ran experiments to measure the following properties: (i) how does the quality of answers returned by TOSS (with different similarity threshold) compare against the quality of answers returned by TAX? (ii) how is the performance of TOSS queries compared with that of TAX when we vary the size of the semistructured instance being queried and/or the size of the ontology used in executing the query? (iii) how is the performance of TOSS queries affected by the similarity threshold? Our experiments were run on the entire SIGMOD XML proceedings data set[13] and on a part of the DBLP data set[6].

This paper is organized as follows. In Section 2, we first introduce the basics of TAX and its problems using illustrative examples. We then give a birdseye view of the TOSS architecture and a brief introduction to its components in Section 3. In Section 4, we quickly describe the ontologies and their integration, and then show how to enhance an ontology with similarity measures. We then extend the definitions of semistructured data model and the TAX algebra to handle ontology and similarity concepts in Section 5. In Section 6, we describe briefly the implementation of our TOSS system and report our experimental results. The related work and conclusion are given in Section 7.

2. PRELIMINARIES AND MOTIVATING EXAMPLES

In this section, we provide a quick overview of TAX[8]. We start with the notion of a semistructured instance, followed by the concept of a pattern tree used to express queries in TAX. The DBLP[6] and SIGMOD[13] bibliographies on the web store data in XML format. Throughout this paper, we will use a small subset of each of these data sources to illustrate why we may get (intuitively) inadequate answers from TAX and how the ontology and similarity mechanisms proposed in this paper can overcome these problems. These sample sets are introduced in Figures 1 (DBLP) and 2 (SIGMOD). Section 2.1 also shows how TAX answers queries. Section 2.2 explains some problems with the answers returned by TAX.

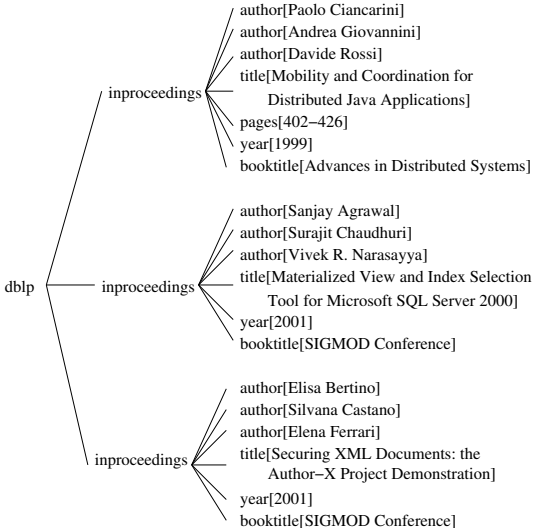

Figure 1: A DBLP example

2.1 Semistructured Instance and TAX

TAX assumes XML documents to be in ordered tree structures.

DEFINITION 1. *A* **semistructured instance** I *over a set* $\mathcal{O}$ *of objects, a set* $\mathcal{L}$ *of strings called* labels, *a set* $\mathcal{T}$ *of types, and a domain* $dom(\tau)$ *for each type* $\tau \in \mathcal{T}$, *is a triple* $I = (V, E, t)$ *where:*

1. $G = (V, E)$ *is a set of rooted, directed trees where* $V \subseteq \mathcal{O}$ *and* $E \subseteq V \times V$.
2. t *is a mapping such that for each object* $o \in V$, $t(o, \textbf{string})$ *assigns a type in* $\mathcal{T}$ *with:*
   - *the tag of* o, *i.e.* $o.tag$, *if* $\textbf{string} = tag$,
   - *the content of* o, *i.e.* $o.content$, *if* $\textbf{string} = content$

Intuitively, $o.tag$ is the label of the edge between o and its parent. We call an object's tag and content its *attributes.* As is well known, a semistructured instance can be represented as a labelled directed graph. A *semistructured database* (SDB) is a finite set of semistructured instances.

EXAMPLE 1. *Consider the DBLP example in Figure 1.* V *consists of all nodes in the tree and* E *consists of all*

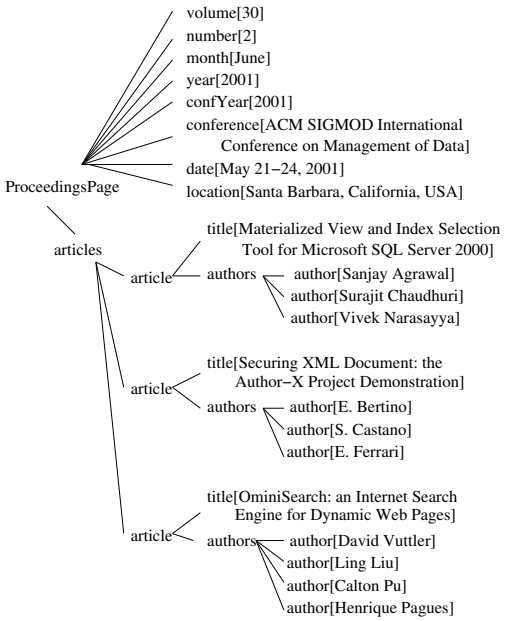

Figure 2: A SIGMOD example

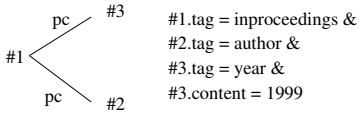

Figure 3: A pattern tree P1

edges. Let us use o to denote the first node "author" at the top of the figure. Then, $o.tag = author, t(o, tag) = \textbf{string}$, $o.content = Paolo\ Ciancarini$ *and* $t(o, content) = \textbf{string}$.

2.1.1 Embeddings and Witness Trees

TAX uses the concept of a *pattern tree* to express queries. We recapitulate the concepts of a pattern tree, an embedding and a witness tree from [8].

DEFINITION 2. *A pattern tree is a pair* $P = (T, F)$, *where* $T = (V, E)$ *is an object-labeled and edge-labeled tree such that:*

- *each object in* V *has a distinct integer as its label;*
- *each edge is either labelled pc (for parent-child) or ad (for ancestor-descendant);*
- F *is a selection condition*[3] *applicable to objects.*

[3]We do not repeat the formal definition of a selection condition in TAX here - some examples will be given shortly).

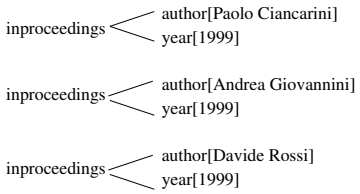

Figure 4: A selection result

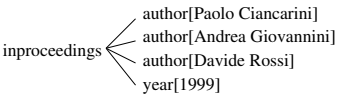

Figure 5: A projection result

EXAMPLE 2. *Figure 3 shows an example pattern tree, where* T *is the tree on the left and* F *is the condition on the right.*

Suppose SDB is a semistructured database and $P = (T, F)$ a pattern tree. An *embedding* of a pattern tree P into SDB is a total mapping $h : P \rightarrow \bigcup_{(V,E)\in \mathsf{SDB}} V$ from the nodes of T to those in SDB such that:

- h preserves the structure of T, i.e., whenever (u, v) is a pc (resp., ad) edge in T, $h(v)$ is a child (resp., descendant) of $h(u)$ in SDB.
- The image under the mapping h satisfies the selection condition F. (The reader may consult [8] for a formal definition of satisfaction).

Each embedding h of a pattern tree P into SDB induces a *witness tree* to the embedding, denoted $h^{\mathsf{SDB}}(P)$, which is defined as follows:

- a node n of SDB is in the witness tree if $n = h(u)$ for some node u in the pattern tree P.
- for any pair of nodes n, m in the witness tree, whenever m is the closest ancestor (of the nodes in the witness tree) of n in SDB, the witness tree contains the edge (m, n).
- the witness tree preserves order between nodes in SDB, i.e., for any two nodes in $h^{\mathsf{SDB}}(P)$, whenever m precedes n in the preorder enumeration of SDB, m precedes n in the preorder node enumeration of $h^{\mathsf{SDB}}(P)$ as well.

EXAMPLE 3. *Figure 4 shows all the possible witness trees from the pattern tree* $P1$ *in Figure 3 to the DBLP example in Figure 1.*

2.1.2 TAX

Selection in TAX. Consider the pattern tree $P1$ in Figure 3 and suppose SL is any set of nodes. In TAX, the selection query $\sigma_{P,SL}(\mathsf{SDB})$ will return all witness trees w.r.t. pattern query P and SDB. In addition, if a node n in SL appears in a witness tree above, then all descendants of n will also be added to the witness tree.

EXAMPLE 4. *Figure 4 shows the result of* $\sigma_{P1,\{\}}(dblp)$.

Projection in TAX. Projection takes a semistructured DB SDB, a pattern tree P and a projection list PL (a list of node labels appearing in P) as inputs. The projection operation $\pi_{P,PL}(\mathsf{SDB})$ returns tree(s) consisting of all nodes n selected from SDB such that for every node n in the result, there exists some witness tree $h^{\mathsf{SDB}}(P)$ and $n' \in PL$ where $h^{\mathsf{SDB}}(n') = n$.

EXAMPLE 5. *To find the authors of papers published in 1999, we can use the pattern tree* $P1$ *shown in Figure 3 and apply projection w.r.t. the semistructured DB shown in Figure 1. This query,* $\pi_{P1,\{\$1,\$2,\$3\}}(dblp)$, *returns the collection of trees in Figure 5.*

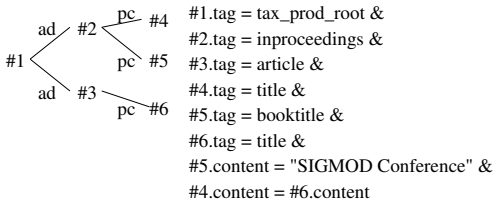

Figure 6: A pattern tree P2

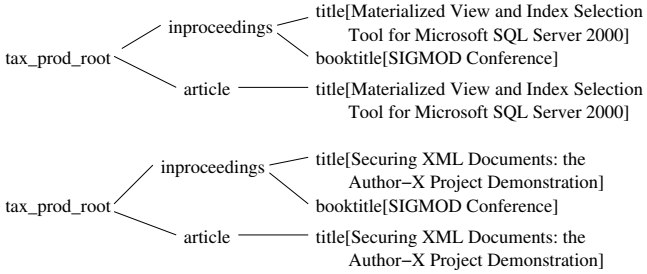

Figure 7: A join result

Product in TAX. The product $\mathsf{SDB}_1 \times \mathsf{SDB}_2$ of two semistructured DBs $\mathsf{SDB}_1, \mathsf{SDB}_2$ contains for each pair of trees $T_1 \in \mathsf{SDB}_1, T_2 \in \mathsf{SDB}_2$, a tree, whose root is a new node (called *tax_prod_root*), left child is the root of T_1 and right child is the root of T_2. Condition join consists of a product followed by selection.

EXAMPLE 6. *The join of the two XML instances in Figure 1 and Figure 2, by taking the product and applying a selection with the pattern tree shown in Figure 6, yields the result shown in Figure 7.*

2.2 Problems with TAX

When we examine the two databases in Figure 1 and Figure 2, we notice several things. First the object tags *booktitle* and *conference* refer to the same thing. Also, the name of the SIGMOD conference is stored differently in the DBLP and SIGMOD databases ("SIGMOD Conference" in DBLP DB but the full name in SIGMOD DB), but they both refer to the same thing. Likewise, names of the three authors of the paper titled "Securing XML Documents..." are stored differently: their first names are stored in full in DBLP but only initials are stored in SIGMOD. In general, there may be many people with the same last name and first initial of the first name, e.g., Marco Ferrari and Mauro Ferrari. Furthermore, there may be errors in data collection and input, e.g., Gian Luigi Ferrari and GianLuigi Ferrari are probably the same person.[4] Thus, it is reasonable to use some similarity measure d_s to indicate the degree of similarity between two strings. For example, d_s(Gian Luigi Ferrari, GianLuigi Ferrari) = 0.1 (i.e., very similar), d_s(Marco Ferrari, Mauro Ferrari) = 2.2 (quite similar) and d_s(Marco Ferrari, GianLuigi Ferrari) = 6.5 (much less similar).

Second, *cost* and *book-price* probably refer to the same concept. However, by looking at the objects, we really are at a loss to determine whether these fields use the same units or not. For example, *cost* may be in US dollars, while *book-price* may be in Euros. Our TOSS system can handle this as well via a clean extension of the TAX algebra.

A user asking queries spanning these two databases would probably like these semantic ambiguities resolved. For example if we wish to find all conference papers published by Mauro Ferrari by looking at both DBLP and SIGMOD DBs, the system must know that: (1) the tag names *booktitle* and *conference* are identical, (2) the short names and full names of SIGMOD conference refer to the same thing. (3) It should be also smart enough to consider sufficiently similar names to be the same.

[4]All the four names above can be found in DBLP.

3. BIRDSEYE VIEW OF TOSS ARCHITECTURE

Figure 8 shows the architecture of our TOSS system. TOSS is built on top of Xindice database system[18] and consists of three components: Ontology Maker, Similarity Enhancer and Query Executor.

1. The *Ontology Maker* associates an ontology with each semistructured instance $I \in \mathsf{SDB}$. It uses WordNet [7] to automatically identify isa, equivalent, and part_of relationships between terms in an SDB. These can be edited further and refined by a database administrator to incorporate additional expertise she may have. These lead to a set of *interoperation constraints* describing relationships between the terms in two ontologies. It will then use an ontology merging algorithm proposed in [3, 2] to merge all the ontologies in a given semistructured database SDB together under the specified interoperation constraints – this leads to a single *fused ontology* for SDB.
2. The *Similarity Enhancer* can utilize *any* measure of semantic similarity between terms (e.g. any of the many methods proposed in the information retrieval literature such as [5]) to enhance the fused ontology described above with similarity – the result is a single *similarity enhanced (fused) ontology* (SEO) for SDB.
3. After SEO is precomputed, the *Query Executor* transforms a user query into a query that takes the single similarity enhanced (fused) ontology into account. It implements the ontology extended algebra described in Section 5 and returns results by accessing the remote or local Xindice system.

We will describe each of the components in this birdseye view of TOSS's architecture in the remainder of this paper.

4. SIMILARITY ENHANCED ONTOLOGIES

In this section, we define ontologies and then recapitulate how [2] integrates ontologies based on graph merging algorithms in [3]. We then show how to enhance an ontology with similarity measures.

4.1 Ontologies

Suppose $(S, \leq)$ is a partially ordered set. A *hierarchy for* $(S, \leq)$ is the Hasse diagram for $(S, \leq)$. Note that the Hasse diagram for a partial order $(S, \leq)$ is a directed acyclic graph whose set of nodes is S. The Hasse diagram of $(S, \leq)$ has a *minimal* set of edges such that there is a path from u to v in the Hasse diagram iff $u \leq v$.

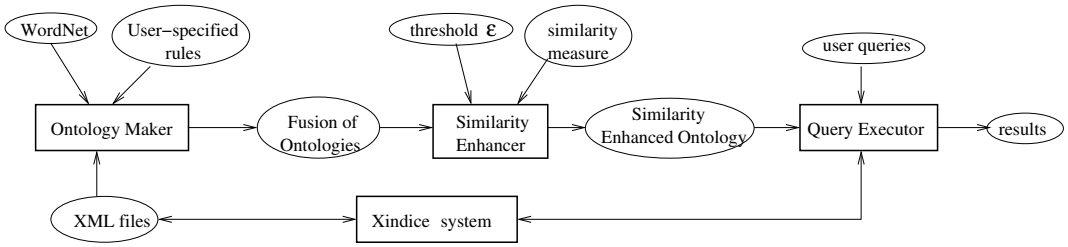

Figure 8: Architecture of TOSS system

EXAMPLE 7. *Consider the set* $S = \{article, author, title\}$. *A partial ordering reflecting the* part_of *relation may say that author is part of article and title is part of article. In addition, everything is part of itself. In this case, the natural definition of the* part_of *relation, denoted* $\leq$ *is given by the set* $\{(author, article), (title, article), (article, article), (author, author), (title, title)\}$. *There is only* one *hierarchy associated with this partial ordering, viz. the set* $\{(author, article), (title, article)\}$.

DEFINITION 3. *Suppose* Σ *is some finite set of strings and* S *is some set. An* ontology w.r.t. Σ *is a partial mapping* Θ *from* Σ *to hierarchies for* S.

Throughout the rest of this paper, we will assume without loss of generality that Σ *and* S *are arbitrary but fixed. We will further assume that* Σ *contains distinguished strings called* isa *and* part_of *(representing the usual* isa *and* part_of *relationships) and that* Θ(isa) *and* Θ(part_of) *are always defined.*

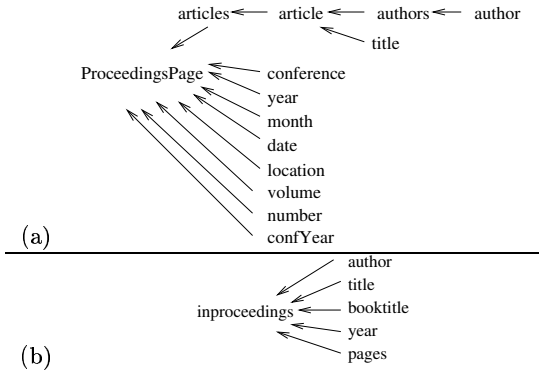

Figure 9: Simplified examples of ontologies associated with (a) SIGMOD and (b) DBLP

EXAMPLE 8. *Suppose* $\Sigma = \{$part_of$\}$. *Figures 9 (a) and (b) show the simplified examples of ontologies associated with SIGMOD and DBLP respectively.*

4.2 Integrating Ontologies

Suppose our SDB $= \{I_1, \ldots, I_n\}$. Suppose, for the sake of simplicity, that $\Sigma = \{$part_of$\}$. Suppose $(H_j, \leq_j)$ is the part_of hierarchy associated with I_j. To answer queries spanning these different semistructured instances, we need to take into account the relationships between terms in these hierarchies. These are captured by interoperation constraints (specified by the database administrators) defined below.

DEFINITION 4 (INTEROPERATION CONSTRAINTS). *Suppose* $(H_i, \leq_i), 1 \leq i \leq n$ *are* n *different hierarchies and suppose* $i \neq j$. *Then* $(x : i = y : j), (x : i \leq y : j), (x : i \not\leq y : j), (x : i \neq y : j)$ *are called* interoperation constraints.

NOTE: As the constraint $x : i = y : j$ can be written as two constraints $x : i \leq y : j$ and $y : j \leq x : i$, we assume that equality constraints are not present.

EXAMPLE 9. *We may write* booktitle:1 = conference:2 *to indicate that* booktitle *(in DBLP) is the same as* conference *in the SIGMOD collection.*

DEFINITION 5 (INTEGRATION). *Suppose* $(H_i, \leq_i), 1 \leq i \leq n$ *are* n *different hierarchies and suppose* **IC** *is a finite set of interoperation constraints. A hierarchy* $(H, \leq)$ *is said to be an* integration *of* $(H_i, \leq_i), 1 \leq i \leq n$ *iff there are* n *injective mappings* $\psi_1, \ldots, \psi_n$ *from* $H_1, \ldots, H_n$ *respectively to* H *such that:*

1. $(\forall i \in \{1, \ldots, n\}) x \preceq_i y \rightarrow \psi_i(x) \leq \psi_i(y)$.
2. $(\forall x \in H_i)(\forall y \in H_j)(x : i \underline{op} y : j) \in \mathbf{IC} \rightarrow \psi_i(x) \underline{op} \psi_j(y)$.

In this case, $(H, \leq), \psi_1, \psi_2, \ldots, \psi_n$ *is a* witness *to the integrability of* $(H_i, \leq_i), 1 \leq i \leq n$ *in the presence of interoperation constraints* **IC**.

Intuitively, an integration of two hierarchies is a new hierarchy. Each member of the hierarchies being merged must be associated with a member of the integrated hierarchy. Axiom (1) above says that the integrated hierarchy must preserve the ordering associated with each of the input hierarchies. Axiom (2) above says that they must preserve the interoperation constraints. It is important to note that there could be many different ways of integrating hierarchies.

DEFINITION 6 (GRAPH ASSOCIATED WITH A SET OF HIERARCHIES). *Suppose* $(H_i, \leq_i), 1 \leq i \leq n$ *are* n *different hierarchies and suppose* **IC** *is a finite set of interoperation constraints. The* hierarchy graph $\mathcal{G}$ *associated with* $(H_i, \leq_i), 1 \leq i \leq n$ *is defined as follows:*

- *The set of nodes is the set* $\{x : i \mid x \in H_i\}$;
- *The set of edges is the union of:*
  - $\{(x : i, y : i) \mid x, y \in H_i \wedge x \leq_i y\}$
  - $\{(x : i, y : j) \mid x : i \leq y : j \in \mathbf{IC}\}$.

EXAMPLE 10. *Consider integrating the two hierarchies of Figures 9 (a) and (b) w.r.t. the following five interoperation constraints conference:1 = booktitle:2, title:1 = title:2, author:1 = author:2, year:1 = year:2, confYear:1 = year:2,*

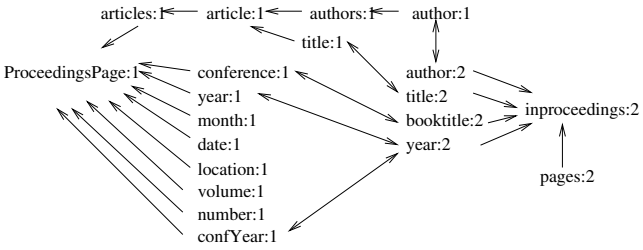

Figure 10: Hierarchy Graph associated with SIGMOD and DBLP

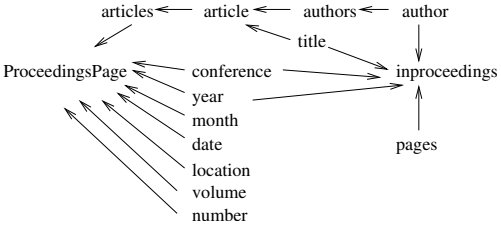

Figure 11: Canonical fusion of the ontologies of SIGMOD and DBLP

which means that conference in the first hierarchy (SIGMOD) is equal to booktitle in the second hierarchy (DBLP), etc. Figure 10 shows the hierarchy graph associated with the two ontologies. Note that there may be many different ways of integrating hierarchies. [3, 2] describe how to find an integration, which is called the canonical hierarchy. We call this integration the canonical fusion (or just fusion) Θ of ontologies $\Theta_1, \ldots, \Theta_n$ via a witness (which maps a node of a hierarchy onto a node in the canonical hierarchy). Figure 11 shows the fused ontology.

We would like to mention that there are various works in the literature on integrating ontologies that have a very rich structure (e.g. definitions where an ontology is any arbitrary first order logic theory, and so on) [4]. However, to date, we have seen no extension that uses these very rich ontology definitions to answer queries over even relational sources. In this paper, we have only considered simple ontologies expressible as graphs.

4.3 Similarity Enhanced Ontologies

In the rest of this section, we assume that given a semistructured database SDB, an ontology associated with each $I \in$ SDB and a set of interoperation constraints IC, we have computed the fusion of all the ontologies w.r.t. IC. We will now show how to enhance such a fused ontology using similarity notions. As is common in many aspects of information retrieval, similarity between terms is measured using a distance function[5].

DEFINITION 7. *A* **string similarity measure** *d_s is any function which takes two strings X, Y and returns a non-negative real number such that:*

- $\forall X,\ d_s(X,X) = 0$.
- $\forall X, Y,\ d_s(X,Y) = d_s(Y,X)$.

A string similarity measure d_s is **strong** *iff for all strings X, Y, Z, $d_s(X,Y) + d_s(Y,Z) \geq d_s(X,Z)$.*

A **similarity measure** *d is any function which takes nodes A, B as input and returns a non-negative real number such that $d(A,B) = min_{X \in S_A, Y \in S_B} d_s(X,Y)$, where d_s is a string similarity measure, S_A, S_B are the sets of strings contained in nodes A, B respectively, and for any node A, if $X, Y \in S_A$, then $d_s(X,Y) = 0$. d is strong iff d_s is strong and satisfies the previous condition.*

Intuitively, two nodes get more and more similar as the distance between them gets smaller.

The following result says that for strong similarity measures, we can avoid trying all combinations of strings between two nodes.

LEMMA 1. *Suppose S_A, S_B are the sets of strings contained in nodes A, B respectively. If a similarity measure d is strong, then $\forall X, P \in S_A, Y, Q \in S_B$, $d_s(X,Y) = d_s(P,Q)$. Therefore, $\forall X \in S_A, Y \in S_B$, $d(A,B) \equiv d_s(X,Y)$.*

Many different similarity measures can be used. The information retrieval community has proposed various string distances such as Levenstein distance[5], Monge-Elkan distance[12], Jaro metric[9], and token-based distance like Jaccard similarity[6] and cosine similarity, etc[5]. In addition, products such as WordNet[7] define similarity measures between terms in English and various foreign languages (but not between proper nouns). In certain domains, rule based methods can also be used to specify similarity between proper nouns (in our SIGMOD/DBLP application for example, we could write a set of rules describing when two names are considered similar). There are also *statistical* approaches to checking similarity between strings used in methods such as latent semantic indexing. *Our goal in this paper is not to invent a new notion of similarity between terms - this has already been well done by the IR community. Instead, the TOSS framework can plug in any such similarity implementation and use it to answer queries in a manner that increases the quality of an answer.*

DEFINITION 8 (SIMILARITY ENHANCEMENT). *Suppose H is an integrated hierarchy, d is a similarity measure and $\epsilon \geq 0$. The pair (H', μ) is called a* similarity enhancement *of H w.r.t. d, ϵ iff H' is a hierarchy and μ is a function from H to $2^{H'}$ ($2^{H'}$ is the power set of H') such that*

1. *$\forall A, B \in H, A_\mu \in \mu(A), B_\mu \in \mu(B)$, if there is a path from A to B in H, then there is a path of length zero or more from A_μ to B_μ in H'. Let us define $\mu^{-1} : H' \to 2^H$ such that $\mu^{-1}(A') = \{A \mid A \in H \wedge A' \in \mu(A)\}$. Similarly, $\forall A', B' \in H'$, if there is a path for A' to B' in H', then $\forall A \in \mu^{-1}(A')$, $B \in \mu^{-1}(B)$, there is a path from A to B in H.*
2. *$\forall A, B \in H, A_\mu \in \mu(A), B_\mu \in \mu(B)$, if $A_\mu = B_\mu$, then $d(A,B) \leq \epsilon$.*
3. *$\forall A, B \in H$, if $d(A,B) \leq \epsilon$, then $\exists A_\mu \in \mu(A), B_\mu \in \mu(B)$ such that $A_\mu = B_\mu$.*
4. *$\forall A', B' \in H'$, $\mu^{-1}(A') \cap \mu^{-1}(B') \neq \mu^{-1}(A')$.*

The first condition ensures that the similarity enhanced hierarchy preserves the original partial orderings in H and does

[5]The Levenstein distance is strong. It assigns a unit cost to every edit operation.

[6]Jaccard similarity between two word sets S, T is $|\frac{S \cap T}{S \cup T}|$.

not include unwarranted orderings in the result. The second condition ensures that all nodes mapped into the same node are sufficiently similar to each other - the threshold ϵ, specified by the database administrator, says two terms are sufficiently similar if the distance between them is ϵ or less.

Our similarity enhancement says that two strings are similar if and only if the two strings are jointly present in some node in the similarity enhancement. That is why we have the third condition.

For example, let us consider a hierarchy H consisting of nodes A, B, C. If $d(A,B) \leq \epsilon$, $d(A,C) \leq \epsilon$, but $d(B,C) > \epsilon$, then we must have one and only one enhancement: we should have a node containing $\{A,B\}$ AND a node containing $\{A,C\}$. We define the similarity enhancement in this way such that if a query is on a string in A, then we can find the result of all similar strings in the nodes containing $\{A,B\}$ and $\{A,C\}$ easily. If we just have two nodes containing $\{A,B\}$ and $\{C\}$ then we do not know that strings in C are similar to those in A from the resulting ontology. In this case, we will need to either (i) compare all nodes with A to find those similar; or (ii) use an index to find those nodes similar to A quickly. But then, we can just use (i) or (ii) directly and do not need to use the similarity enhancement.

The fourth condition removes redundant nodes which are a subset of some other node in the similarity enhancement.

By definition, an ontology is not allowed to have cycles. This may mean, on occasion, that no similarity enhancement exists *for some similarity measures and some thresholds* ϵ.

DEFINITION 9 (SIMILARITY INCONSISTENCY). *Suppose H is a hierarchy, d is a similarity measure and $\epsilon \geq 0$. The triple (H, d, ϵ) is* similarity consistent *(resp. similarity inconsistent) iff there exists (resp. does not exist) a similarity enhancement of H w.r.t. d, ϵ.*

The following theorem states that all similarity enhancements of a particular hierarchy (w.r.t a similarity measure d and a threshold ϵ) are equivalent.

THEOREM 1. *(Equivalence of Similarity Enhancements) Suppose $H = (S, \preceq)$ is a hierarchy, d is a similarity measure and $\epsilon \geq 0$. If (H'_1, μ_1) and (H'_2, μ_2) are similarity enhancements of H, then there exists an one-to-one mapping $M : S'_1 \rightarrow S'_2$ such that $M \circ \mu_1 = \mu_2$ and $\forall A, B \in S'_1, (A,B) \in \preceq_1$ iff $(M(A), M(B)) \in \preceq_2$, where $H'_1 = (S'_1, \preceq_1)$ and $H'_2 = (S'_2, \preceq_2)$.*

Proof sketch: Conditions (2) – (4) of Definition 8 equivalently define the set of nodes and the mapping in the similarity enhancement as follows. First, we define $S'' = \{V \mid \forall A, B \in V, A \in S \wedge B \in S \wedge d(A,B) \leq \epsilon\}$. Then, define $S'' \leftarrow S'' - \{V \mid \exists V' \in S'' \wedge V \subset V'\}$. Thus, S'' is unique. Next, we can define a mapping $M_1 : S \rightarrow 2^{S''}$ s.t. $M_1(A) = \{V \mid A \in V\}$.

For any similarity enhancement, say, (H'_1, μ_1), we can define a one-to-one mapping $M_2 : S'' \rightarrow S'_1$ s.t. $M_2(B) = C$ iff $\forall A \in B$, $C \in \mu_1(A)$.

Similarly, for (H'_2, μ_2), we can define a one-to-one mapping $M'_2 : S'' \rightarrow S'_2$ s.t. $M'_2(B) = C$ iff $\forall A \in B$, $C \in \mu_2(A)$.

Now we can define $M : S'_1 \rightarrow S'_2$ s.t. $M(C_1) = C_2$ iff $\forall A \in \{A \mid C_1 \in \mu_1(A)\}$, $C_2 \in \mu_2(A)$. Thus, $M \circ \mu_1 = \mu_2$.

Because $M \circ \mu_1 = \mu_2$, $\mu_1^{-1} \circ M^{-1} = \mu_2^{-1}$.

From condition (1) in Definition 8, $\forall A, B \in S'_1$, $(A,B) \in \preceq_1$ iff $\forall a \in \mu_1^{-1}(A)$, $b \in \mu_1^{-1}(B)$, $(a,b) \in \preceq$. Similarly, $\forall A, B \in S'_2$, $(A,B) \in \preceq_2$ iff $\forall a \in \mu_2^{-1}(A)$, $b \in \mu_2^{-1}(B)$, $(a,b) \in \preceq$. Thus, $\forall A, B \in S'_1$, $(M(A), M(B)) \in \preceq_2$ iff $\forall a \in \mu_2^{-1}(M(A))$, $b \in \mu_2^{-1}(M(B))$, $(a,b) \in \preceq$. But $\forall X \in S'_1$, $\mu_2^{-1}(M(X)) = \mu_1^{-1} \circ M^{-1}(M(X)) = \mu_1^{-1}(X)$. Thus, $\forall a \in \mu_2^{-1}(M(A))$, $b \in \mu_2^{-1}(M(B))$, $(a,b) \in \preceq$ is equivalent to $\forall a \in \mu_1^{-1}(A)$, $b \in \mu_1^{-1}(B)$, $(a,b) \in \preceq$. Therefore, $\forall A, B \in S'_1$, $(M(A), M(B)) \in \preceq_2$ iff $(A,B) \in \preceq_1$. □

Figure 12 presents the SEA algorithm (short for **Similarity Enhancement Algorithm**) to find a similarity enhancement of a hierarchy w.r.t. d, ϵ.

```
algorithm SEA(H, d, ε)
/* Input: hierarchy H = (S, ≼), similarity measure d,
          real number ε ≥ 0 */
/* Output: H' = (S', ≼'), μ */
1)   S' ← ∅
2)   for each s ∈ S,
3)       S' ← S' ∪ {s}
4)       for each s' ∈ S',
5)           if ∀a ∈ s', d(s,a) ≤ ε, then s' ← s' ∪ {s}
6)           else if ∃a ∈ s', d(s,a) ≤ ε, then
7)               S' ← S' ∪ {{s} ∪ {a | a ∈ s' ∧ d(s,a) ≤ ε}}
8)       S' ← S' − {s' | ∃s'' ∈ S' ∧ s' ⊂ s''}
9)   for each s ∈ S,
10)      define μ(s) = {s' | s' ∈ S' ∧ s ∈ s'}
11)  ≼' ← ∅
12)  for each (a,b) ∈ ≼,
13)      ≼' ← ≼' ∪ {(c,d) | c ∈ μ(a) ∧ d ∈ μ(b) ∧ c ≠ d}
14)  if check-acyclic(H') == TRUE, then
15)      return H' = (S', ≼'), μ
16)  else return "Similarity inconsistent"
```

Figure 12: Similarity Enhancement Algorithm SEA

In Line 14, the function check-acyclic(H') checks whether H' is acyclic or not. We can modify a depth-first search to detect cycles in $O(| \preceq' |)$ time where $| \preceq' |$ indicates the number of edges in the hierarchy of the similarity enhancement. The time complexity of the **SEA** algorithm is $O(|S| \cdot |S'|(|S| + |S'|) + | \preceq ||S'|^2)$.

THEOREM 2. *(Correctness of Algorithm) Given a hierarchy H, a similarity measure d and a real number $\epsilon \geq 0$, the SEA algorithm in Figure 12 returns a similarity enhancement (H', μ) of H if one exists.*

Proof sketch: Lines 5 – 7 produce nodes such that the second condition of a similarity enhanced hierarchy is satisfied. Lines 3, 5 – 7 ensure that the third condition is satisfied because line 3 ensures that, for every $s \in S$, there exists some node in S' that contain s, and lines 5 – 7 ensures that $\forall s_1, s_2 \in S$, if $d(s_1, s_2) \leq \epsilon$, then there must exist some node in S' to contain both of them. Line 8 removes redundant nodes to ensure that the fourth condition is satisfied. Lines 11 – 13 create $\preceq$ to satisfy the first condition. Lines 9 – 10 define μ according to the definition. □

EXAMPLE 11. *Figure 13(a) shows a toy ontology consisting of a* isa *hierarchy. Suppose we use Levenstein distance as the similarity measure d and a threshold $\epsilon = 2$. Because $d(relation, relational) = 2$, $d(model, models) = 1$, the algorithm SEA will form two new nodes in the similarity enhancement, containing $\{relation, relational\}$ and $\{model, models\}$ respectively. SEA will also remove the four redundant nodes relation, relational, model and models. Then it defines μ, which maps each node in the original hierarchy to itself in the result, except for the four nodes removed, which are mapped to the two new nodes respectively. $\preceq'$ is defined as shown in Figure 13(b).*

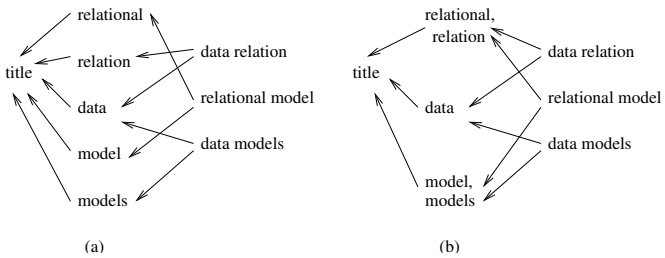

Figure 13: (a) An example ontology and (b) its similarity enhancement

5. ONTOLOGY EXTENDED SEMISTRUCTURED DATA MODEL

In this section, we extend the semistructured data model and the TAX algebra to handle ontology and similarity concepts. *Due to space restrictions, in the rest of this section, we will assume that the set* Σ *equals* {isa}*, i.e. only one hierarchy is associated with each ontology. The results of the paper can be easily extended to incorporate arbitrary hierarchies such as* part_of.

Types, Domain Values, and Hierarchies. We assume there exists a set $\mathcal{T}$ of strings called *types*. Each type $\tau \in \mathcal{T}$ has an associated set $\mathsf{dom}(\tau)$ called its *domain*. The members of $\mathsf{dom}(\tau)$ are called *values* of type τ. When $\mathcal{T}_i \subseteq \mathcal{T}$, we will often use $\mathsf{dom}(\mathcal{T}_i)$ to denote $\bigcup_{\tau \in \mathcal{T}_i} \mathsf{dom}(\tau)$.

For example, we may have $\mathcal{T} = \{\mathsf{int}, \mathsf{string}\}$. We may have $\mathsf{dom}(\mathsf{int})$ be the set of all integers and $\mathsf{dom}(\mathsf{string})$ be the set of all strings over $\{a, \ldots, z\}$. However we may also have in $\mathcal{T}$ a type like mm whose domain is the set of all non-negative integers (representing millimeters).

A hierarchy of the form $H = (\mathcal{T}_H, \leq_H)$ where $\mathcal{T}_H$ is a set of types is called a *type hierarchy*.

Suppose $\tau \in \mathcal{T}_H$ is a type and $a \in \mathsf{dom}(\tau)$ is an arbitrary value of type τ. Clearly, we could also have a type called a with $\mathsf{dom}(a) = \{a\}$. Thus, each value of a type may also be viewed as a type. For example, consider the $o.tag = author$ in Example 1. Extended with the ontology, $t(o, tag) = author$. Given any type hierarchy $H = (\mathcal{T}_H, \leq_H)$ and a $\tau \in \mathcal{T}_H$, let $\mathsf{below}_H(\tau) := \{\tau' \mid \tau' \leq \tau\} \cup \mathsf{dom}(\tau)$. Intuitively, below_H extends the partial ordering of the hierarchy H to include the values of the types.

SEO Semistructured Instance. In this section, we show how to extend a semistructured instance to include type, ontology, and similarity information. Recall that a semistructured instance is defined as $I = (V, E, t)$ where t associates a type in $\mathcal{T}$ with each attribute (*tag* and/or *content*) of each object $o \in V$.

Intuitively, an extended instance associates *types* with each object attribute. This is important because a type could be something like USD representing US Dollars or cm representing (intuitively) centimetres. It is also associated with an ontology (which for now is just a hierarchy H_{isa}, a distance function d and a real number $\epsilon > 0$).

An *ontology extended semistructured instance* is a quadruple $(V, E, t, H_{\mathsf{isa}})$, where H_{isa} is an isa hierarchy[7] and the following constraints are satisfied:

- $\forall$ object $o, t(o, tag) \in \mathcal{T}_{\mathsf{isa}}$ and $o.tag \in \mathsf{below}_{H_{\mathsf{isa}}}(t(o, tag))$
- $\forall$ leaf object $o, t(o, content) \in \mathcal{T}_{\mathsf{isa}}$ and $o.content \in \mathsf{below}_{H_{\mathsf{isa}}}(t(o, content))$

An SEO *(similarity enhanced ontology extended) semistructured instance* is a quadruple $(V, E, t, H_{\mathsf{isa}}')$ where (H_{isa}', μ) is the similarity enhancement of H_{isa} for some μ w.r.t. similarity measure d and threshold ϵ.

Conversion Functions. As the reader has already seen, as different units might be used in instances, there is often a need to *convert* from one unit to another. In this section, we adapt the concept of a conversion function introduced in [3, 2] and give a brief description below.

For each pair of types τ_i and τ_j, we assume there exists at most one *conversion function* $\tau_i 2\tau_j : \mathsf{dom}(\tau_i) \rightarrow \mathsf{dom}(\tau_j)$. We assume that conversion functions are total. Moreover, we assume that the set of conversion functions enjoys the following closure conditions and constraints:

- for each $\tau \in \mathcal{T}$, $\tau 2\tau$ exists and equals the identity function,
- if both $\tau_1 2\tau_2$ and $\tau_2 2\tau_3$ exist, then $\tau_1 2\tau_3$ exists and equals the composition $\tau_2 2\tau_3 \circ \tau_1 2\tau_2$. (Note that as $\tau_1 2\tau_3$ is assumed to be unique, all the possible conversion function compositions that convert τ_1's values into τ_3's values must be equivalent.)[8] Moreover, it is not necessary to specify $\tau_1 2\tau_3$ explicitly. The system may automatically compose it from the other functions.
- for all hierarchies H, if $\tau_1 \leq_H \tau_2$ then there exists a conversion function $\tau_1 2\tau_2$.

5.1 TOSS Algebra

Throughout this section, we assume without loss of generality that we are querying some semistructured database SDB and that the ontologies of semistructured instances in SDB have been fused together as described in Section 4.2 and that the fused ontology was then enhanced with similarity using the SEA algorithm shown in Figure 12. All algebraic operators therefore assume the existence of this similarity enhanced (fused) ontology.

5.1.1 SEO *Embeddings and Witness Trees*

Selection conditions are used in pattern trees which are used in both selection and projection. In this section, we define selection conditions and show how to extend the definitions of embedding and witness trees to support ontologies and similarity.

Simple (or *atomic*) *conditions* have the form $X\, op\, Y$, where $op \in \{ =, \neq, <, \leq, >, \geq, \sim,$ `instance_of`, isa, `subtype_of`, `above`, `below` $\}$, and X, Y are *terms*, that is, attributes, types, or *typed values* $v{:}\tau$, with $v \in \mathsf{dom}(\tau)$. Types will sometimes be omitted in typed values when they can be unambiguously derived from the context. Here $\sim$ represents the similarTo operation which will return true if both operands are sufficiently similar (i.e., $\leq \epsilon$ for a given similarity measure d).

Selection conditions can now be defined as follows:

[7]When we allow Σ to contain not just isa but other strings as well, the H_{isa} term needs to be replaced with Θ with obvious corresponding changes to the rest of the paper. For the sake of simplicity and due to space restrictions, in this section, we only consider the case where one hierarchy is used.

[8]For some applications such as foreign exchange, converting from Euro to Pound is not identical to converting from Euro to USD to Pound. This is a tradeoff between accuracy and simplicity. The database administrator may choose to use the above intuitive strategy or specify every conversion function $\tau_1 2\tau_2$ explicitly.

- all simple conditions are selection conditions,
- if c_1, c_2 are selection conditions then $c_1 \wedge c_2$, $c_1 \vee c_2$, $\neg c_1$ are selection conditions,
- nothing else is a selection condition.

EXAMPLE 12. *Suppose we want to find the titles of all papers in DBLP related to Microsoft (independently of the field in which Microsoft appears). We may change the selection condition in the pattern tree in Figure 3 to the following and apply it to* dblp*:* $\#1.tag = inproceedings \wedge \#2.tag = title \wedge \#3.tag$ part_of $inproceedings \wedge \#3.content =$ "$\star$ *Microsoft*$\star$" *(where* $\star$ *is a wild card).*

In order to ensure an embedding to be correct w.r.t an SDB with an associated similarity enhanced ontology, we need to redefine the satisfaction of selection conditions.

In the rest of this section, let $EI = (V, E, t, H_{\mathsf{isa}})$ be an arbitrary but fixed ontology extended or SEO instance.[9] In the context of EI, the type of a term X w.r.t. a mapping h[10] from a pattern to a tree collection is defined as follows:

$$\mathsf{type}(X)^h = \begin{cases} t(h(X), tag) & \text{if } X \text{ is the tag of an object} \\ t(h(X), content) & \text{if } X \text{ is the content of an object} \\ \tau & \text{if } X = \tau \text{ or } X = v{:}\tau. \end{cases}$$

When $\mathsf{type}(X), \mathsf{type}(Y)$ are both in a type hierarchy T and the least upper bound of $\mathsf{type}(X)$ and $\mathsf{type}(Y)$ in T exists, then it is called the *least common supertype* of X and Y.

We say that a simple condition $X\ op\ Y$ with $op \in \{=, \neq, <, \leq, >, \geq\}$ is *well-typed* if X and Y have a least common supertype τ and the conversion functions $\mathsf{type}(X)^h 2\tau$ and $\mathsf{type}(Y)^h 2\tau$ exist. Simple conditions with other operators are always well-typed. A selection condition is *well-typed* if all its simple subconditions are.

The *value* of a term X w.r.t. a mapping h from a pattern to a tree collection is

$$X^h = \begin{cases} h(X).tag & \text{if } X \text{ is the tag of an object} \\ h(X).content & \text{if } X \text{ is the content of an object} \\ \tau & \text{if } X = \tau \\ v & \text{if } X = v{:}\tau. \end{cases}$$

The image (witness tree WI) under the mapping h (from a pattern P to a collection C), i.e., $h^C(P)$, *satisfies* a well-typed condition c in the context of the ontology extended instance EI (denoted $EI, WI \models c$) in the following cases:

- $c = X\ op\ Y$, $op \in \{=, \neq, <, \leq, >, \geq, \sim\}$, τ is the least common supertype of X and Y, and $(\mathsf{type}(X)^h 2\tau)(X^h)$ op $(\mathsf{type}(Y)^h 2\tau)(Y^h)$ is true. For $A \sim B$, the condition is true iff $\exists$ a node containing both of them in the similarity enhancement.
- $c = X$ `instance_of` Y, $Y^h \in \mathcal{T}$, $\mathsf{type}(X)^h \leq_H Y^h$, and $X^h \in \mathsf{dom}(Y^h)$.
- $c = X$ `subtype_of` Y, $X^h \in \mathcal{T}$, $Y^h \in \mathcal{T}$, $X^h \leq_H Y^h$.
- $c = c_1 \wedge c_2$, $EI, WI \models c_1$ and $EI, WI \models c_2$.
- $c = c_1 \vee c_2$, either $EI, WI \models c_1$ or $EI, WI \models c_2$.
- $c = \neg c_1$, $EI, WI \not\models c_1$.
- $c = X$ `below` Y, $EI, WI \models X$ `instance_of` $Y \vee X$ `subtype_of` Y.
- $c = X$ `above` Y, $EI, WI \models Y$ `below` X.

[9]The semantics of both ontology extended and SEO instances are similar except that an ontology extended instance uses an ontology but a SEO instance uses a SEO and it allows similarity operations.

[10]Strictly speaking, h maps a node in the pattern tree to a node in a tree collection. However, X in $h(X)$ is not a node but a node's attribute. For convenience, here we abuse the notation such that, suppose $X = y.b$ where y is a node and b is an attribute, we use $h(X)$ to denote $h(y)$.

5.1.2 TOSS Algebra

In this section, we are able to give a succinct definition of the TOSS algebra. Note that this algebra is easy to implement on top of any semistructured DBMS system.

Suppose $EI_1 = (V_1, E_1, t_1, H_1), \ldots, EI_z = (V_z, E_z, t_z, H_z)$ are ontology extended instances. Suppose $\mathcal{F}'$ is a fusion of $H_1, \ldots, H_z$ via witness $tr_{\mathcal{F}'}$, $(\mathcal{F}, \mu)$ is the similarity enhancement of $\mathcal{F}'$, $tr_{\mathcal{F}} = \mu \circ tr_{\mathcal{F}'}$. $\Theta_{\mathcal{F}}$ is the ontology that associates the hierarchy $\mathcal{F}$ with isa.

For all algebraic expressions Exp, $[Exp]_{\mathcal{F}}$ is inductively defined as follows.

- If Exp is an instance EI_i, then $[Exp]_{\mathcal{F}} = (tr_{\mathcal{F}}(V_i), E_i, tr_{\mathcal{F}}(t_i), \Theta_{\mathcal{F}})$, which is an SEO instance. Here $tr_{\mathcal{F}}(V_i)$ (resp. $tr_{\mathcal{F}}(t_i)$) maps the tags and contents of objects $o \in V_i$ (resp. their types) to terms in $\Theta_{\mathcal{F}}$. In this case, Exp is always well-typed.
- (Selection) If Exp is $\sigma_{P,SL}(Exp')$ and the selection condition F in P is well-typed in the context of $[Exp']_{\mathcal{F}}$, then $[Exp]_{\mathcal{F}}$ returns the set of witness trees WI such that $[Exp']_{\mathcal{F}}, WI \models F$. SL specifies the nodes whose descendants should be included in the final output. Exp is well-typed if Exp' and F are.
- (Projection) If Exp is $\pi_{P,PL}(Exp')$ and the condition F in P is well-typed in the context of $[Exp']_{\mathcal{F}}$, then $[Exp]_{\mathcal{F}}$ keeps the nodes in $[Exp']_{\mathcal{F}}$ which match some pattern node in P (and also in PL) for some embedding $h : P \to C$ (such that the witness tree under h satisfies F in the context of $[Exp']_{\mathcal{F}}$) and their hierarchical relationships but eliminates all other nodes. Exp is well-typed if Exp' and F are.
- (Cross product) If Exp is $Exp_1 \times Exp_2$ then $[Exp]_{\mathcal{F}}$ contains for each pair of trees $T_i \in [Exp_1]_{\mathcal{F}}, T_j \in [Exp_2]_{\mathcal{F}}$, a tree, whose root is a new node, left child is the root of T_i and right child is the root of T_j. If Exp_1, Exp_2 are well-typed, then Exp is.

The join operator can be expressed in the standard way as a composition of $\sigma_{P,SL}$ and $\times$. For the remaining set theoretic algebraic operators we need to specify when two data trees should be considered identical as described in [8]: two data trees T_1, T_2 are *equal* iff there exists an isomorphism $\iota : T_1 \to T_2$ between the two sets of nodes that preserves edges and order, and futhermore, for every value-based atom of the form "attribute θ value" (θ is one of $=, \neq, >$, etc), the atom is true at node u in T_1 iff it is true at node $\iota(u)$ in T_2.

- (Union, Intersection and Difference) If $Exp = Exp_1$ op Exp_2 where $op \in \{\cup, \cap, -\}$, then $[Exp]_{\mathcal{F}}$ is the standard result of $[Exp_1]_{\mathcal{F}}$ op $[Exp_2]_{\mathcal{F}}$. Exp is well-typed if Exp_1, Exp_2 are.

PROPOSITION 1. *For all algebraic expressions Exp over* $EI_1 = (V_1, E_1, t_1, H_1), \ldots, EI_z = (V_z, E_z, t_z, H_z)$ *and the similarity enhanced fusion* $\mathcal{F}$ *of* $H_1, \ldots, H_z$, $[Exp]_{\mathcal{F}}$ *is an* SEO *instance.*

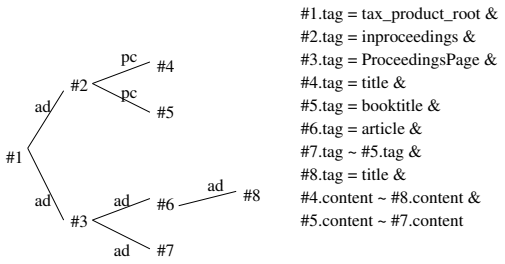

Figure 14: A pattern tree P3

EXAMPLE 13. *Suppose we want to find the papers in SIGMOD DB such that the title of that paper is similar to the title of some SIGMOD conference paper recorded in DBLP. A join of both databases by taking product of them and a selection with the pattern tree shown in Figure 14 can solve this task, i.e.,* $\sigma_{P3,\{\}}(dblp \times ProceedingsPage)$. *The result will contain two trees corresponding to the papers titled "Materialized View ..." and "Securing XML ..." respectively.*

6. IMPLEMENTATION AND EXPERIMENTS

The TOSS system is implemented in Java and currently consists of 24402 lines of code. TOSS is built on top of the Xindice[18] database system. TOSS consists of the following components: (1) the *Ontology Maker* automatically takes XML files as input and uses WordNet[7] and user-specified rules to automatically generate ontologies, and then integrate them to obtain their fusion; (2) the *Similarity Enhancer* automatically finds the similarity enhancement of the ontology according to the threshold ϵ and the similarity measure specified by the database administrators among a variety of possible choices (supported by a JAVA toolkit available in [5]); (3) the *Query Executor* implements the ontology extended algebra, transforms a user query into a query that takes ontological information into account, and accesses the remote or local Xindice system.

Recall and precision. We obtained the recall and precison of results returned by TOSS and TAX by evaluating 12 selection queries on 3 data sets (each containing 100 random papers from DBLP). Each query contains 1 isa, 1 similarTo and 3 tag matching conditions. For isa and similarTo conditions, "contains" and exact match are used for TAX respectively. The precision and recall of TOSS and TAX results for each query are calculated by checking against semantically correct results generated manually. A query result contains 1 to 38 papers.

Figure 15(a) shows that TAX always get 100% precision but low recall (lower than 0.5 for 75% of queries) because the exact match in TAX guarantees that the results it returns are correct but it misses most of the correct results. Its recall is 1 for 3 queries whose semantically correct results contain 3 or fewer papers. In contrast, the average precision and recall of TOSS (with $\epsilon = 3$) results are 0.942 and 0.843. For most of queries, it maintains its precision close to 1 with much higher recall. For the query with lowest TOSS precision, by comparing with TAX recall, it shows that it takes a tradeoff of 1/3 of precision degradation for 3 times of recall increase. For TOSS (with $\epsilon = 2$), the average precision and recall are 0.987 and 0.596. The higher precision and lower recall compared with the one of $\epsilon = 3$ are expected because the SEO obtained using a lower ϵ can only merge very similar terms together, missing some correct answers.

Figure 15(b) shows the square root of product of recall and precision as a quality measure[14] of TOSS and TAX results against the square root of TAX recall for the corresponding query. It clearly shows that TOSS (ϵ=3) outperforms TAX for all queries (except the 3 queries mentioned above). Figure 15(c) shows how much the recall is improved by TOSS compared with TAX normalized by the precision. In TOSS (ϵ=3), most of the queries get their normailzed recall more than doubled. Again, the performance of TOSS (ϵ=2) lies between TOSS (ϵ=3) and the original TAX with the similar reason.

Scalability experiments. In all our scalability experiments, we used the DBLP and SIGMOD proceeding data. The total DBLP file size is 188,566,002 bytes. We selected all proceeding records and removed unnecessary spaces. Then we truncated the file such that it has 4,753,774 bytes containing 3712 papers due to the 5MB maximum data size limitation of Xindice. The SIGMOD data contains 16 proceedings pages having a total size of 712KB. The experiments were run on a PC with 1.4GHz CPU, 524MB memory and Windows 2000 Professional platform.

The times taken in all experiments include the following: (i) time to parse a pattern tree and rewrite the pattern tree into XPath queries, (ii) time to execute the XPath queries in the Xindice system, (iii) time to parse the result returned from the Xindice system and convert the result to the form defined by TAX.

Scalability of selection. Figure 16 (a) shows the results of evaluating TOSS and TAX conjunctive selection queries, each of which contains 2 isa and 4 tag matching conditions, on the DBLP data. For isa conditions, exact match is used for TAX. We tested the scalability of selection queries by varying the size of ontologies (only for TOSS) as well as the size of XML files.

The reader can easily see that our algorithms handle data size of approximately 4.75 MB in 12.5-14 seconds, almost independent of the ontology size. The slope of the curves are more or less linear - approximately 500 KB can be processed in one second. As the last XML data file has 4,753,774 bytes, very close to the 5MB maximum data size of Xindice, there was a sudden slope up at the last point.

The difference between the lowest TOSS curve in the figure and TAX is in 0.4064-3.1544 seconds. The difference between the highest TOSS curve and TAX is in 0.6228-4.1402 seconds. When data size grows, the difference increases because there are more accesses to the ontology. The bigger difference is also due to the empty result returned by TAX.

Scalability of join. We also ran scalability experiments on join. Each query contains 5 tag matching and 1 similarTo conditions. For similarTo conditions, exact match is used for TAX. Figure 16 (b) shows the scalability of join of the DBLP and SIGMOD data as the total size of the two XML files varies upto a little over 3.7 MB.

As the total size of the data being joined increased, we had a linear increase in the time taken to do the joins except the two last data points. These two data points show a faster increase because Xindice system returns intermediate results in a super-linear time and dominates the total execution time when the size of intermediate result is large. For example, in the experiment of the last data point, the total execution time was 17 seconds, where 12.1 seconds is

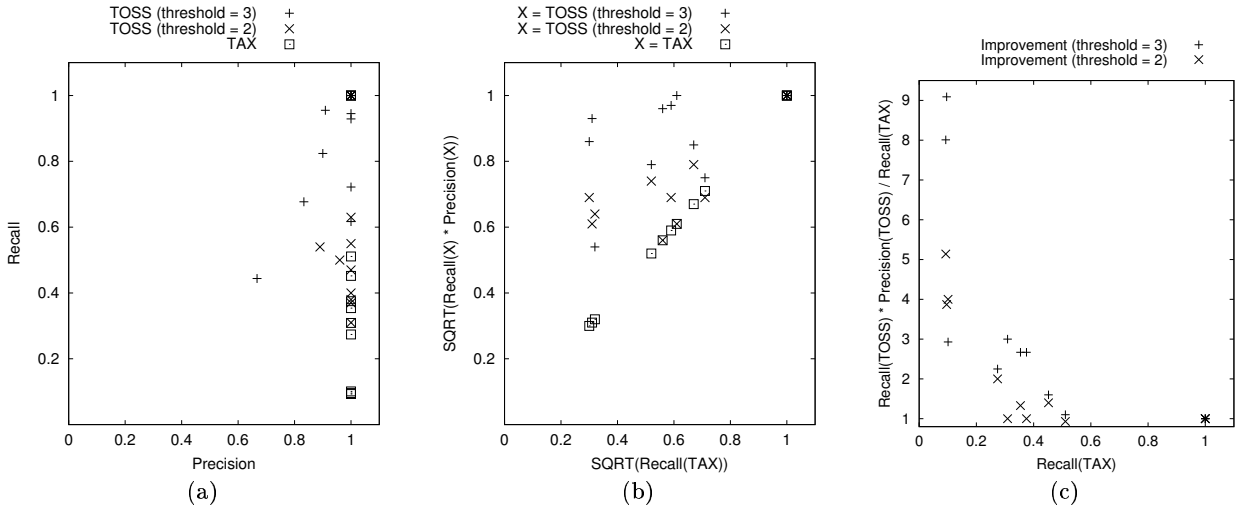

Figure 15: (a) Recall against precision for TOSS and TAX results, (b) Square root of product of recall and precision of TOSS and TAX results against the square root of corresponding TAX recall for each query, (c) Improvement factor of TOSS recall compared with TAX recall normalized by precision

from Xindice and 4.9 seconds from our code.

The difference between the lowest TOSS curve in the figure and TAX is in 0.3078-1.3 seconds. The difference between the highest TOSS curve and TAX is in 0.532-2.7238 seconds. The difference increases with the data file size because the number of accesses to the ontology increases.

TOSS computation time vs ϵ. Finally, we conducted experiments to show how the evaluation time of conjunctive selection queries and join queries are affected by the different values of ϵ (in generating SEO). The selection experiment was on a dblp file with 4,753,774 bytes with ontology size of 1003 terms. The join experiment was on a dblp file of 3,071,430 bytes and the sigmod data of 692,188 bytes with ontology size of 1769 terms. Figure 16 (c) shows that both execution time increase approximately linearly to the increase of ϵ because when ϵ increases, each node will contain more similar terms in average and thus more time is needed to output a larger result.

7. RELATED WORK AND CONCLUSIONS

Our work builds on two important recent contributions in databases, viz. the TAX algebra for semistructured databases by Jagadish et. al.[8] and the need to use ontologies for query processing by Wiederhold's group[15, 16].

As mentioned several times in this paper, TAX is the starting point for our work. The landmark TAX paper proposes a formal theoretical basis for semistructured data, together with an algebra to query semistructured data sources using the concept of a pattern tree.

Wiederhold's group was the first to notice that ontologies can be used to improve the quality of answers to queries. They proposed an *ontology algebra* [17, 10, 11]. Their algebras consisted of logical statements [10] using a LISP style syntax. Their later work included graphs which were combined in their ONION system using the three set operations of union, intersection and difference.

TOSS is fundamentally different from both these works in many ways. First, in TOSS, ontologies are merged under some *interoperation constraints* – no such constraints were considered in [17, 10]. However, this is important because terms in one ontology may have a relationship with terms in another. Although a similar concept was considered in [11], their integrated ontologies were not concise. And of course, ontologies are not used at all in TAX. Second, TOSS does not just merge ontologies together - it extends the semistructured DB model and algebra so that semistructured instances with associated ontologies can be queried, whereas [17, 10, 11] only merges together ontologies without directly addressing the impact of how the query algebra is affected. Third, our framework introduces for the first time, the concept of similarity search in semistructured databases - something that neither the TAX work nor Wiederhold's prior work did. Fourth, we have developed experimental results showing that TOSS greatly improves the quality of answers compared to TAX.

In a very recent work[1], Al-Khalifa et al. introduced a bulk algebra TIX (modified from TAX) to integrate information-retrieval style query processing into a traditional pipelined query evaluation engine for an XML database. TIX uses a scored pattern tree instead of a pattern tree as in TAX. User-defined score functions are given to some nodes in a scored pattern tree to specify how to compute scores of nodes using IR-style conditions. They also proposed new access methods for efficient score generations. By extending existing operators in TAX and introducing new operators, an IR-style query can be expressed in TIX to find relevant results, with weighting and ranking support. Instead of returning a number of result trees that match with the pattern tree as in TAX, TIX further processes the result trees in order to return only their appropriate parts which allows the users to get what they want more easily. This is an IR feature that we didn't consider. Another difference is that they uses traditional IR methods for scoring such as term frequencies. Their scoring can also involve non-local information. We didn't measure relevance (of a part of a document) in the IR sense. Instead, we use both the (fused) ontologies and various similarity measures such that both the lexical and semantical information are considered to define the relationship between any two terms. We also precompute an SEO during integration of different XML databases instead of their run-time handling of relevance of results. On the

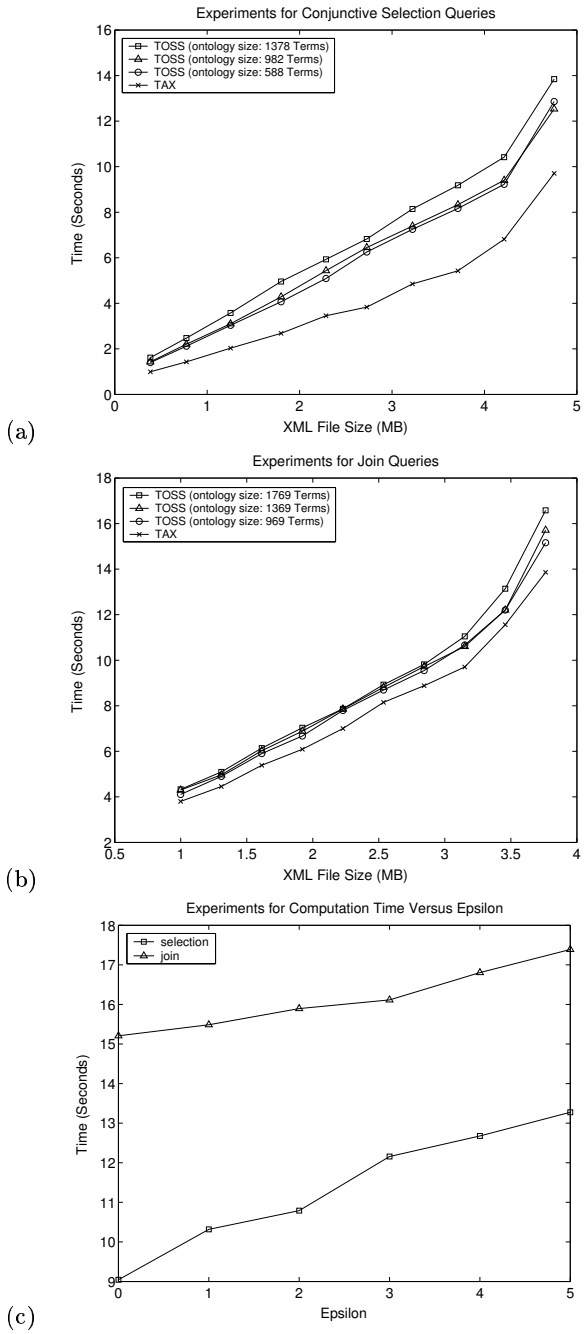

Figure 16: Performance of TOSS and TAX for (a) selection queries, (b) join queries. (c) TOSS computation time of selection and join against ϵ.

other hand, TIX does not consider the integration of XML documents.

Acknowledgements

This work was supported in part by the Army Research Lab under contracts DAAL0197K0135 and DAAD190320026, the CTAs on Advanced Decision Architectures and Telecommunications, by ARO contract DAAD190310202, and by NSF grants 0205489 and IIS0329851.

8. REFERENCES

[1] S. Al-Khalifa, C. Yu, and H. V. Jagadish. Querying structured text in an xml database. In *Proc. ACM SIGMOD Conf. on Management of Data*, San Diego, CA, 2003.

[2] P. Bonatti, Y. Deng, and V. S. Subrahmanian. An ontology-extended relational algebra. In *Proceedings of the IEEE International Conference on Information Reuse and Integration (IEEE IRI 2003)*, 2003.

[3] P. Bonatti, M. L. Sapino, and V. S. Subrahmanian. Merging heterogeneous security orderings. *Journal of Computer Security*, 5(1):3–29, 1997.

[4] D. Calvanese, G. D. Giacomo, and M. Lenzerini. A framework for ontology integration. In *Proc. of the First Semantic Web Working Symposium*, pages 303–316, 2001.

[5] W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A comparison of string metrics for matching names and records. In *Proc. of the First Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, 2003.

[6] DBLP XML records. Available at `http://dblp.uni-trier.de/xml/`, Nov 2003.

[7] G. A. Miller et. al. WordNet - a lexical database for english. Cognitive Science Laboratory, Princeton University. Available at `http://www.cogsci.princeton.edu/~wn/w3wn.html`, 2000.

[8] H. V. Jagadish, L. V. S. Lakshmanan, D. Srivastava, and K. Thompson. TAX: A tree algebra for XML. In *Proc. DBPL Conf*, Rome, Italy, 2001.

[9] M. A. Jaro. Probabilistic linkage of large public health data files. *Statistics in Medicine*, 14:491–498, 1995.

[10] D. Maluf and G. Wiederhold. Abstraction of representation for interoperation. *Lecture Notes in AI*, 1315, 1997.

[11] P. Mitra, G. Wiederhold, and M. Kersten. A graph-oriented model for articulation of ontology interdependencies. In *Proceedings Conference on Extending Database Technology 2000 (EDBT'2000)*, Konstanz, Germany, 2000.

[12] A. Monge and C. Elkan. The field-matching problem: algorithm and applications. In *Proc. of the Second International Conference on Knowledge Discovery and Data Mining*, 1996.

[13] SIGMOD Record in XML. Available at `http://www.acm.org/sigmod/record/xml/`, Nov 2002.

[14] V. G. Voiskunskii. Evaluation of search results: A new approach. *Journal of the American Society for Information Science*, 48(2), Feb 1997.

[15] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, pages 38–49, Mar 1992.

[16] G. Wiederhold. Intelligent integration of information. In *Proc. 1993 ACM SIGMOD Conf. on Management of Data*, pages 434–437, 1993.

[17] G. Wiederhold. Interoperation, mediation and ontologies. In *International Symp. on Fifth Generation Computer Systems, Workshop on Heterogeneous Cooperative Knowledge Bases, ICOT*, pages 33–48, 1994.

[18] Apache Xindice XML database. Available at `http://xml.apache.org/xindice/`.