A Lightweight Framework for Exchanging Web Data

Filipe Mesquita
Federal University of Amazonas
Manaus, Amazonas, Brazil
fsm@dcc.ufam.edu.br

Denilson Barbosa
University of Calgary
Calgary, Alberta, Canada
denilson@ucalgary.ca

Eli Cortez
Federal University of Amazonas
Manaus, Amazonas, Brazil
ecv@dcc.ufam.edu.br

Altigran S. da Silva
Federal University of Amazonas
Manaus, Amazonas, Brazil
alti@dcc.ufam.edu.br

ABSTRACT

We propose a lightweight framework for data exchange that is suitable for non-expert and casual users sharing data on the Web and/or through peer-to-peer systems. Unlike previous work, we consider a minimalistic data model and schema formalism that are suitable for describing online data and propose algorithms for mapping such schemas as well as for translating the corresponding instances. Also our solution requires minimal overhead and setup costs (e.g., we consider data stored in tables, XML or CSV files) comparing to existing data exchange systems, making it very attractive in our setting. We report experimental results indicating that our method works well with real Web data from various domains.

General Terms

ALGORITHMS, EXPERIMENTATION.

Keywords

XML, data exchange, Web data management.

1. INTRODUCTION

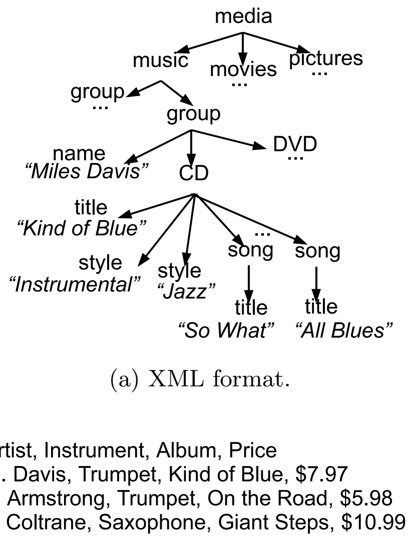
The past few years have witnessed a drastic increase in the amount of data *collections* maintained and shared by non-expert users through easy-to-use services on the Web or peer-to-peer (P2P) data sharing systems [12, 18]. There are many such services available today, both on focused topics (e.g., the Internet Book Database[1] and the Recipe Tavern[2]), as well as generic services (e.g., GoogleBase[3], FreeBase[4], Kijiji[5], *craigslits*[6]). We refer to these data as data collections, as opposed to databases, because these collections are not created nor maintained as traditional databases; for instance, most of them are not kept inside a DBMS. Instead, these collections are kept in files or in online data sharing services, whose access interfaces are rudimentary when compared with a declarative query interface. Usually, these collections are represented as CSV (comma-separated-values) or XML files, while a few Web data sharing services allow users to store their data in a relational format (e.g., Web Office[7] and DabbleDB[8]).

One defining characteristic of the data sharing services mentioned above is that they allow the users to store their data organized in any way they wish, while offering a set of predefined (and sometimes customizable) schemas from different application domains. This flexibility lowers the entry-level cost for one to share data, but naturally leads to a myriad of schemas describing very similar application domains, making it harder for one to integrate them afterwards [15]. Nevertheless, given the abundance of data collections available, it would be highly desirable to be able to integrate them in the same easy way with which one can define their own data collections.

We consider the problem of exchanging data between such collections, that is, translating data from one *source* collection into data that conforms to the schema of a *target* collection, in a way that is suitable for non-expert users. To illustrate the problem consider the example in Figure 1, showing data collections about music in XML and CSV formats. Notice that they use distinct labels for the same kind of data, as well as different structure. Moreover, in general, schema information is implicit, i.e., a carefully designed DTD or XML Schema may not always be available. Furthermore, often the input data cannot be fully embedded in the target data collection; that is, only a part of the input schema can be matched correctly to the target schema. For instance, consider exchanging data from the collection in Figure 1(b) into the collection in Figure 1(a). Observe that `Artist` and `Album` match `name` and `title`, respectively, while `Instrument` and `Price` have no counterpart in the target collection.

The data exchange problem consists in, given data structured under a source schema, restructure and translate it to a target schema [8]. While this problem has attracted con-

[1] `http://www.ibookdb.net`
[2] `http://www.recipetavern.com`
[3] `http://base.google.com`
[4] `http://www.freebase.com`
[5] `http://www.kijiji.com`
[6] `http://www.craigslist.org`
[7] `http://www.weboffice.com`
[8] `http://dabbledb.com/`



(a) XML format.

Artist, Instrument, Album, Price
M. Davis, Trumpet, Kind of Blue, $7.97
L. Armstrong, Trumpet, On the Road, $5.98
J. Coltrane, Saxophone, Giant Steps, $10.99

(b) CSV format.

Figure 1: Example data collections.

siderable attention recently, the bulk of this work considers a very different setting in which the data are kept in databases and tools are used to help translating the data from one source into another. Notice that this approach is completely unrealistic in the setting we consider here. First of all, non-expert users do not have the skills nor the resources to set up databases and use mapping tools for finding the correspondences between them. Also, given the large number of data collections and the high heterogeneity among them, the effort invested in using a standard database solution would be unacceptable. Finally, most of the exchanges in this setting move only small portions of a data collection at a time, and it is quite possible that two peers may exchange data once and never again. Therefore, the traditional solution to the data exchange problem requires considerable investment and effort to be practical in our setting.

Outline and contributions. In this paper we propose a lightweight data exchange framework tailored for non-expert and casual users sharing semi-structured data on the Web or in P2P systems. More specifically, we discuss a minimalistic generic hierarchical data model as well as a schema formalism that capture essential features of XML and tabular data (Section 3), and present the data exchange problem on those terms. We then discuss our Data Fitting algorithm, which restructures instances of our data model according to a target schema, without any user intervention (Section 4). We present experimental results on real Web data from several domains showing that our approach is very promising (Section 5). Conclusions and future work are given in Section 6.

2. RELATED WORK

The data exchange problem consists in, given data structured under a source schema, restructure and translate it to a target schema. Fagin et al. [8] laid down the foundations of the data exchange problem; in particular, they studied different semantics for data exchange and their complexity. Fuxman et al. [9] study the problem in the context of two peers sharing data; they consider the case when peers specify what data they are willing to receive from others. Libkin [13] studies the data exchange problem in the presence of incomplete information. Arenas and Libkin [1] consider the exchange of XML data where the source and target schemas are XML DTDs. These works have laid out the theoretical underpinning of the data exchange problem, focusing mostly on complexity results.

There has been considerable work on schema mapping; Rahm and Bernstein provide a thorough survey [21]. Unlike in the data exchange scenario, the goal here is finding the actual mapping from a source schema into a target schema. Cupid [14] and Similarity Flooding [16] exploit schema information, including the labels of schema elements, to derive mappings. Our experiments show that this approach alone does not work well in our setting. Other methods exploit the actual data values to derive associations between schema elements [4]. As we show later, combining schema and value information yields very acceptable results in our setting.

There has been work on actually translating the data once the schema mapping is found. The Clio tool (see [20] and references therein) is a system that generates such mappings in several languages, converting between XML and relational data seamlessly. Unlike Clio, which requires considerable setup investment and user intervention, our solution is targeted to non-expert and casual users who may not have the expertise nor the time to define and carefully debug mappings. Thus, we focus on a simpler data model and constraint language than what is handled in Clio and other similar tools.

3. FRAMEWORK

In this section we discuss the data exchange problem in light of a simple, generic data model and schema formalism which are rich enough for the setting we consider in this paper. We show how to convert XML data into instances of our data model and vice-versa; we also relate our schema formalism to Document Type Definitions (DTDs) [3]. We focus on XML because it is the preferred encoding format for exchanging data on the Web. Moreover, it is expressive enough to represent other forms of data as well, such as tabular data (i.e., a spreadsheet) and relational data.

3.1 Data Model

We consider a generic tree data model called *FDM*, with two kinds of nodes for representing *entities* and their *attributes*. Intuitively, entities represent real world objects while attributes describe those entities. As usual, attributes can only assume atomic values from a given domain.

An instance of the *FDM* data model is a labeled tree with two kinds of nodes for representing entities and attributes, respectively, and a distinguished entity node called the *root* of the instance. Only attribute nodes have a *value*, which is a literal of a given domain (i.e., strings, numbers, dates, etc.). Figure 2 shows a document with the artist entity *Norah Jones*, and one of he CDs, *Not Too Late*, which in turn has two songs.

Context. The *context* of an entity e in an instance I is defined by the sequence of entity labels spelled out in the path from the root of I to e. The context of an attribute is the same of the entity where that attribute is defined. For example, the context of the song entities in Figure 2 is artist.CD.

3.2 *FDM* Schema Graph

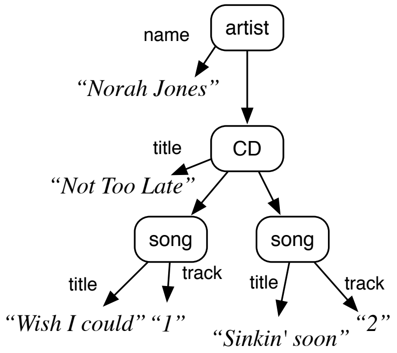


Figure 2: Example of an entity. Entities are represented by round rectangles while attributes are textual nodes stemming out of entities; attribute values are shown in italics.

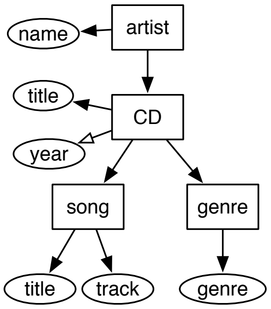
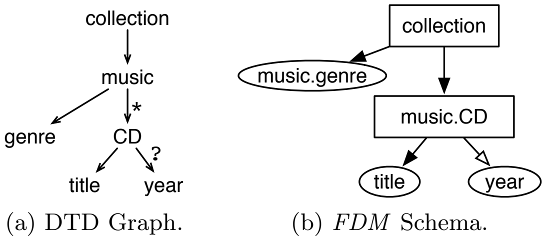


Figure 3: An *FDM* schema. Boxes represent entity types, while ovals represent attributes. The arrows indicate the attributes of the entities and the way they can be nested. Hollow arrows indicate optional attributes.

We use a simple schema formalism, similar in nature to DataGuides [11], to describe the attributes of different entities and the ways in which these entities can be nested in one another. We assume that an entity label and a context defines a *type*. That is, we assume that two entities with the same label appearing in the same context must have the same attributes.

An *FDM* schema is a tree $G = (V, E, r)$ in which the correspond to entity types and attribute names; r is an entity defining the root of V and E is the set of edges between nodes in E. An edge from entity e_1 into entity e_2 in a schema graph indicates that: e_2 is a sub-entity of e_1; an instance of e_1 may be associated with zero or more instances of e_2. Figure 3 shows an *FDM* schema for the instance in Figure 2.

Converting DTDs into FDM schemas. We abstract a DTD into an *FDM* schema as follows. Recall that the DTD graph [23] of a DTD is a graph in which vertices correspond to element tags in the DTD and an edge $x \to y$ is defined iff the DTD allows elements of tag y to appear in the content of elements of tag x; moreover, an edge $x \to y$ is labeled with a ?, *, or + if y is optional in x, can occur zero or more times in x, or at least once in x, respectively. For simplicity, we replace all + edges by * edges. The root of the DTD graph is the element tag of the root element in the document (specified by the DOCTYPE clause). Given a DTD graph G, an *FDM* schema S is produced as follows. Intuitively, leaf nodes in G will be mapped into attributes in S, while the root of G as well as its inner nodes on which the incident edge is labeled with * are mapped into entities. Inner nodes in which the incident edge is not labeled with * are *inlined*; that is, their labels are used as prefixes of the entities or attributes that appear below them in G. For instance, the music node in the DTD graph of Figure 4(a) is inlined in the *FDM* Schema (Figure 4(b)). Finally, leaf nodes in G in which the incident edge is labeled * are modeled as entities with a homonymous attribute.



(a) DTD Graph. (b) *FDM* Schema.

Figure 4: Example DTD Graph and corresponding *FDM* schema.

More precisely, we create an entity type in S for the root node of G, and for every node in G in which there is an incident edge labeled with *. If x is a leaf node in G that is mapped into an entity e_1 in S, we add an attribute to e_1 with the same label x. Let x and y are distinct nodes in G that are mapped into entities e_1 and e_2, respectively, in S such that: x is an ancestor of y, and there is no other node in the path $x \rightsquigarrow y$ that is mapped into an entity in S. We add an edge $e_1 \to e_2$ to S and we use the labels of the nodes between x and y as prefixes to the label of e_2. Finally, every leaf node in G that is not mapped yet becomes an attribute of the entity corresponding to its closest node in G.

Note that *FDM* schemas are less expressive than XML DTDs and relational schemas. In particular, our formalism does not capture recursive DTDs easily. This simplification is intentional. We argue that our framework is expressive enough to capture the essence of the data exchange problem in our setting, as those discussed in Section 1.

Converting between XML and FDM. Converting an XML document into an instance of *FDM* is straightforward. The conversion in the opposite direction is also not hard; all one needs to do is expand the *inlined* elements accordingly. Note that converting from XML into *FDM* and back is a lossy process, as *FDM* is not an ordered model. However, if one has an *FDM* instance I that conforms to a schema derived from a DTD D (as discussed above), one can translate I into a valid document w.r.t. D, by ordering the elements accordingly.

4. DATA FITTING

We now describe the Data Fitting method for restructuring an instance of *FDM* that conforms to a source schema S into another one that conforms to a target schema T. We first discuss how to find mappings between a source schema S and a target schema T, and then move to how to translate an instance of S according to T.

4.1 Attribute Matching

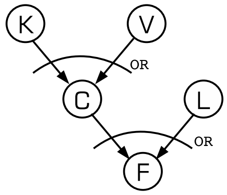


Figure 5: Combining similarity components: F is the final similarity between two attributes, C and L are the content and label similarity scores, respectively. K and V and the keyword-based and value-based similarity scores.

The first step in mapping schemas is finding correspondences between their attributes. Let A and B be two attributes from a source instance I_S under schema S and a target instance I_T under schema T, respectively. Intuitively, the similarity between A and B depends on two components: their *content similarity* (C) and their *label similarity* (L). The content similarity estimates to which extent the values in the domain of A overlap with the values in the domain of B, based on the actual values present in the source and target instances. The label similarity estimates how close the labels (within their context) of A and B are to each other.

We model similarity scores as probabilities and use the formal framework of Bayesian networks [19] to combine them as follows (see Figure 5; ignore nodes K and V for the moment.). The *final similarity* between A and B, denoted by F, depends on the content and label similarity between them. Moreover, we assume that C and L influence F through a disjunctive operator $or(\cdot,\cdot)$, also known as *Noisy-OR-Gate* [19]:

$$F(A,B) = or(C(A,B), L(A,B))$$

Informally, by using the disjunctive operator we mean that either parent node (C and L) is likely to activate F (i.e., significantly increase the function's final score). This disjunctive operator is particularly useful when any individual factor is likely to activate F alone, regardless of other factors [19]. Formally, the disjuntive operator is defined as follows:

$$or(x,y) = 1 - [(1-x)\cdot(1-y)]$$

where x and y are probabilities.

4.1.1 Content similarity

We treat numeric and textual attributes differently when computing the C score. For numeric attributes, we consider a simple yet effective approach: we assume that the values in the target attribute B follow a Gaussian distribution. The similarity between A and B is defined as the mean value of the probability density function for each value in A. We normalize this function by the maximum probability density, which is reached when a given value is equal to the mean. Thus, we define the content score for numeric attributes as follows:

$$C(A,B) = \frac{1}{|A|}\sum_{v\in A} e^{-\frac{v-\mu}{2\sigma^2}}$$

where σ and μ are standard deviation and mean, respectively, of the values of B.

Textual attributes, on the other hand, require more work. As illustrated in Figure 5, the content similarity for textual data type is computed combining the keyword-based (K) and value-based (V) similarity scores, i.e.,

$$C(A,B) = or(K(A,B), S(A,B))$$

Keyword-based similarity. The keyword-based similarity measures the overlap of individual words appearing in the content of A and B. We take two factors into account: (a) the proportion of keywords in A that occurs at least once in values of B, and (b) how likely the keywords in A are to appear in values of B:

$$K(A,B) = \frac{1}{2}\left[\sum_{k\in A\cup B}\frac{w_k(A)}{w_{max}(A)} + 1 - \prod_{k\in A\cup B} 1 - w_k(B)\right] \tag{1}$$

where $w_k(A)$ and $w_k(B)$ are the weight of keyword k relative to attribute A and B, respectively; and $w_{max}(A) = \sum w_k(A)\forall k \in A$.

The first component of Equation 1 accounts for factor (a) and is estimated by the normalized sum of weights of keywords that occurs in both A and B. The weights are computed by the well-known *TF-IDF* weighting scheme according the distribution of keywords in the attribute A and the source instance S. Our goal in using the weighting term $w_k(A)$ is to privilege high overlap with keywords that are rare in S but common in values of A:

$$w_k(A) = \mathit{tf}_k(A)\cdot\log\left(1 + \frac{N_S}{\mathit{att}(S,k)}\right),$$

where $\mathit{tf}_k(A)$ is the term-frequency of k among values of A, N_S is the total number of attributes in the source schema S and $\mathit{att}(S,k)$ is the number of attributes in the source instance I_S containing k. In other words, $w_k(A)$ will be higher if k is frequent in values of A and does not appear everywhere in the target instance I_T.

The second component in Equation 1 combines the likelyhood of each keyword in A being a typical keyword in B using the disjuntive operator. This same idea for measuring the similarity of a keyword to a attribute was successfully applied in the context of keyword-based search over relational databases [17] and citation metadata extraction [7]. The weighting term $w_k(B)$ measures how likely a keyword in A is to appear in a value of B:

$$w_k(B) = \frac{\log(\mathit{val}(B,k))}{\log(V_B)}\cdot\left(1 - \frac{\log(\mathit{att}(T,k))}{\log(N_T)}\right) \tag{2}$$

where $\mathit{val}(B,k)$ returns the number of values of attribute B where k occurs, V_B is the total of values of B, $\mathit{att}(T,k)$ counts to attributes in I_T containing k among its values and N_T is the total number of attributes in T.

Value-based similarity. While the keyword-based similarity is appropriate when there is little or no overlap between the values of A and B, the value-based similarity takes advantage of such overlap. Intuitively, we evaluate how many values in A occur as values in B, combining the result for each value by the disjunctive operator. That is:

$$V(A,B) = 1 - \prod_{v\in A} 1 - \frac{log(o_v(B))}{log(|A|)}$$

where $o_v(B)$ is 1 if value v occurs as value of B, or 0 otherwise; and $|A|$ is the number of values of A.

We consider two values as equal if they contain the same keywords (i.e., we remove stopwords from them). In order to speed up the computation, we represent each value by the MD5 signature of its terms.

4.1.2 Label Similarity

We compute the label similarity between attributes A and B taking into account their context (recall Section 3). We don't compare labels directly; instead we use stemming and some simple heuristics to extract the relevant keywords in the. For instace, "running_time" is represented by {"run", "time"}. We will call these set of keywords as the *label descriptor* of the attribute.

We estimate the similarity between a pair of label descriptors using the "soft" version of the cosine measure in the vector space model, named soft TF-IDF [5]. Unlike the traditional cosine measure, the soft TF-IDF relaxes the requirement that terms must match exactly and yields better results in our setting. The soft TF-IDF model also considers similar keywords by using a string matcher. In this way, given two label keywords a and b, such that $|a| \leq |b|$, we define the string similarity as $s(a, b) = |a|/|b|$ if a is prefix or suffix of b, or 0 otherwise.

Thus, the label similarity measure is computed as follows. Let $close(\theta, A, B)$ be the set of keyword pairs (a, b) where $a \in A$ and $b \in B$ such that $s(a, b) > \theta$ and $s(a, b) = \max_{b' \in B} s(a, b')$.

$$L(A, B) = \frac{\sum\limits_{(a,b) \in close(\theta, A, B)} w(a, A) \cdot w(b, B) \cdot s(a, b)}{\sqrt{\sum\limits_{a \in A} w(a, A)^2} \cdot \sqrt{\sum\limits_{b \in B} w(b, B)^2}}$$

where $w(a, A)$ and $w(b, A)$ is the weight of label keywords a and b regarding to attributes A and B, respectively.

We take into account two factors to compute the weight of a keyword: (1) the level of keyword in the path from the root entity to the attribute and (2) how rare is the keyword among the attributes in schema. More formally, we define:

$$w(a, A) = level(a, A) \cdot log(IDF_a)$$

where IDF_a is the inverse of the fraction of attribute label descriptors in the underlying schema that contain a.

4.2 Finding Mappings

Once we define a similarity measure for pairs of attributes, the next step is to find those pairs of attributes that do in fact match. We say that attributes A and B match when their similarity $F(A, B)$ is higher than a given threshold (we use 0.5 in this work). From a pairwise computation, we build an *attribute multimapping* [16] $\mathcal{M}$ that is a relation associating each attribute in S to all those that match it in T. However, we only consider attributes of compatible datatypes; moreover, for textual attributes, we also require that their *length* be compatible. For instance, we want to avoid mapping an attribute with movie reviews into another one with movies titles (even though their datatypes are the same and they share common values, as movie titles are likely to appear in reviews). Thus, considering a textual attribute X, let $\hat{X}$ be the distribution of lengths of values in X, $E(\hat{X})$ be the mean value of $\hat{X}$ and $std(\hat{X})$ be the standard deviation of $\hat{X}$. We keep a mapping from A into B if the difference between the mean values of $\hat{A}$ and $\hat{B}$ is within one standard deviation of $\hat{B}$. More precisely, we require that $|E(\hat{A}) - E(\hat{B})| \leq \max(std(\hat{B}), \varepsilon)$, where ε is a tolerance threshold (in our tests we found that $\varepsilon = 1.5$ works well in practice).

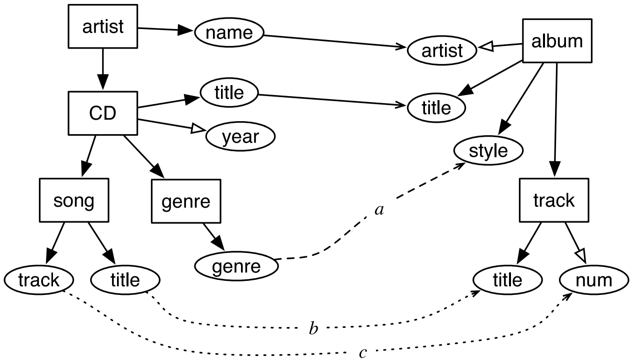


Figure 6: Pairwise attribute mappings.

Given this attribute multimapping $\mathcal{M}$, we can move to mapping entities. To accomplish this, we first generate an *entity multimapping* $\mathcal{M}'$ from $\mathcal{M}$ in which entities $E_1 \in S$ and $E_2 \in T$ are mapped if an attribute of E_1 is mapped to an attribute of E_2 in $\mathcal{M}$. For instance, in Figure 6, the entity `artist` is mapped to `album`, since there is a attribute match between them, `name` → `artist`.

We compute the similarity between entities E_1 and E_2 by using the disjoint operator over the set of mapped attribute pairs between E_1 and E_2, denoted $\mathcal{P}(E_1, E_2)$:

$$F'(E_1, E_2) = 1 - \prod_{(A,B) \in \mathcal{P}(E_1, E_2)} 1 - F(A, B)$$

where A and B are attributes belonging to E_1 and E_2, respectively, and $F(A, B)$ counts for the similarity between A and B.

4.2.1 Mapping conflicts

It is possible that $\mathcal{M}$ induces conflicting mappings between entities in $\mathcal{M}'$. To see this, consider Figure 6, which dictates that the artist, CD and genre information will be merged together into a single album entity. As the target schema does not allow more than a style per album, the values of artist and title must be duplicated for every CD with more than one genre in the source instance. Now consider mapping the track entities: note that we must also repeat all tracks as sub-entities for each duplicate album, which leads to high redundancy. We solve this situation by avoiding pairs connecting a set of source entities to a path of target entities unless this set is connected by a single path as well. More precisely, we are looking for a entity mapping μ' that map entity paths to entity trees. For instance, in Figure 6 we must choose between mapping the genre information (a) or the track information (b and c) because the entities CD, genre and song in the source do not lie in a single path, but in a tree. In this case, we say that a has a conflict with b and c.

Thus, given the entity multimapping $\mathcal{M}'$, we need to find the best subset of entity pairs with no conflicts to generate a entity mapping μ'. As it turns out, this is an NP-complete optimization problem. To see this, let $G(V, E)$ be a graph where V contains pairs of entities in $\mathcal{M}'$ and E contains an edge u, v iff $u, v \in V$ conflict with each other. We want to find an entity mapping that is contained in $\mathcal{M}'$ (i.e., a subset of V) with maximal score and without any conflicts; that is, we want to *remove* from $\mathcal{M}'$ those entity mappings that cause conflicts and have low scores. This is equivalent of finding a minimun-weigth vertex cover in G [10]. We use a simple greedy heuristic in our work. First, we define a score for each vertex by decreasing its original score (given by F') by scores of its adjacent vertices. We then remove the vertex with smallest score and update the scores of its neighbor vertices until no edges are left in the graph. The remaining set of vertices compose μ' as the best subset of entity pairs of $\mathcal{M}'$.

4.2.2 The final attribute mapping

We are now ready to discuss how we arrive at the final attribute mapping μ that associates attributes in S into attributes in T. Note that, unlike $\mathcal{M}$, μ is a function. Moreover, as customary [21], we require μ to be injective; that is, each attribute in S is mapped to at most one attribute in T, and vice-versa. We obtain μ from $\mathcal{M}$ and μ' as follows. Given match attributes A and B in $\mathcal{M}$ and its respective entities E_1 and E_2 in μ', we multiply the attribute similarity $F(A, B)$ by the entity similarity $F'(E_1, E_2)$. Here, the entity similarity acts as a structural score, by privileging attribute mappings between high scored pair of entities. Notice that if E_1 and E_2 are not in μ' then attribute pair A and B is not considered. Finally, we use the *best filter* algorithm [16] to produce μ. That is, we chose the best available candidate pairs from $\mathcal{M}$ until all attributes are mapped.

4.3 Translating instances

Once a mapping $\mu : S \to T$ is defined, the last step of the Data Fitting process is to restructure the source instance I_S by applying the transformations defined in μ. This does not imply only relabeling but may also involve structure changes. For instance, consider the entity genre illustrated in Figure 6, which originally is descedent of entity CD. However, in the target schema attributes of genre and CD are mapped to a single entity album.

This process is similar to the content creation and structuring/tagging steps for publishing relational data as [22]. In particular, we adapted the *path outer union* and *hash-based tagger* techniques. We start by extracting the content of I_S by decomposing it into a relation. The purpose of decomposing I_S is provide an intermediary representation of the data, where there is no particular nesting, such that it can be grouped and nested according to any other given structure. More precisely, consider a relation $R(B_1, B_2, \ldots, B_n)$, where each B_i is a attribute in T, such that each path in the source instance I_S from the root entity to a leaf-level entity represents a tuple $r_i \in R$. For each attribute A with value v in a path, we insert v as value of $\mu(A)$ in r_i, where $\mu(A)$ returns the mapped attribute B_i for A. Other attributes are `null`.

Next, we tag and struct the relational content in order to generate instances $I_1, I_2, \ldots, I_n$, such that I_i conforms to the target schema T. Let T'_i be a sub-tree of T that contains the entities presenting at least a attribute defined in tuple $r_i \in R$. We note that T'_i define the desired structure to tuple r_i. Therefore, for each r_i, we generate the target instances by reproducing entities in T'_i and associating the attributes and values found in r_i for each entity. To avoid duplicates, we use a main-memory hash table to look up whether an entity was already included in the final result.

5. EXPERIMENTS

We now present an experimental evaluation of our Data Fitting method carried out with real Web data. The experimental data was acquired from popular sites from four domains: movies, music, books and academic articles. For each domain, we chose representative websites and extracted data from them. Table 1 describes the data collections we use in this work, while Table 2 presents the sites we used to obtain them. All data used in our experiments is available at `http://www.ucalgary.ca/~denilson/fdm`.

Domain	Source Collection		Target Collection		Overlap
	Entities	Attr.	Entities	Attr.	
Movies	774	77	8,914	19	10
Music	714	40	10,000	4	4
Books	789	5	1,211	19	4
Articles	1,630	6	8,000	13	4

Table 1: Data collections used in the experiments. The Overlap column indicates the number of perfect matches between attributes in the source and target collections.

Domain	Source collections	Target collections
Movies	movies.yahoo.com	imdb.com
Music	pandora.com&itunes.com	musicbrainz.com
Books	books.google.com	dblp.uni-trier.de
Articles	sigmod.org/record	dblp.uni-trier.de

Table 2: Sites used in the experiments.

We implement our method by using inverted file indices [2] such that given a keyword or value signature we can retrieve the list of attributes where it occurs. We note that the time for building such indices dominates the whole experiment time, where the Data Fitting processing takes hundreds of milliseconds for each input. Furthermore, our greedy heuristic for solving the mapping conflicts, as described in Section 4.2.1, uses a Fibonacci heap [6] to remove the vertex with minimum score. Our implementation was done in Perl, and all experiments were run on a standard desktop machine (Pentium Core 2 Duo 2.13 GHz, 2 GB RAM).

As our main goal is to produce good mappings, we assess the *accuracy* of our method using the F-measure metric, which combines precision and recall and is commonly used in Information Retrieval experiments [2]. To do that, we manually inspected all data collections and defined the *correct* mappings between attributes and entities on a best effort basis. For instance, consider the combined plot for Movies in Figure 5, whose F-measure is 0.94 (0.97 of precision and 0.92 of recall). This means that, on average, our method chose less than one wrong pair (false positive) and missed less than one correct pair (false negative) in the final mapping, in the 50 runs of that experiment.

We now study the effectiveness of our Data Fitting approach with the different similarity measures discussed in Section 4 (recall Figure 5). For increased readability, we refer to the F, C, K, V and L scores as *combined*, *content*, *keywords*, *values* and *labels* in this section. Notice that K score (*keywords*) counts for numeric similarity as well, as opposite to *values*.

Effectiveness of the combined Data Fitting score. Figure 7 shows the average matching accuracy for different similarity measures. For each domain, we pick 50 samples of 10 "main" entities with their sub-entities as well (e.g., for the Movies domain with pick a movie with its actors, directors,

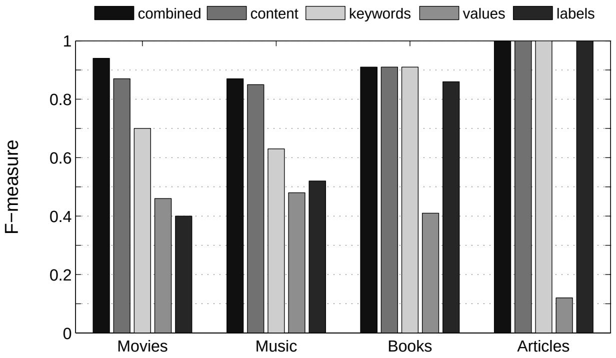


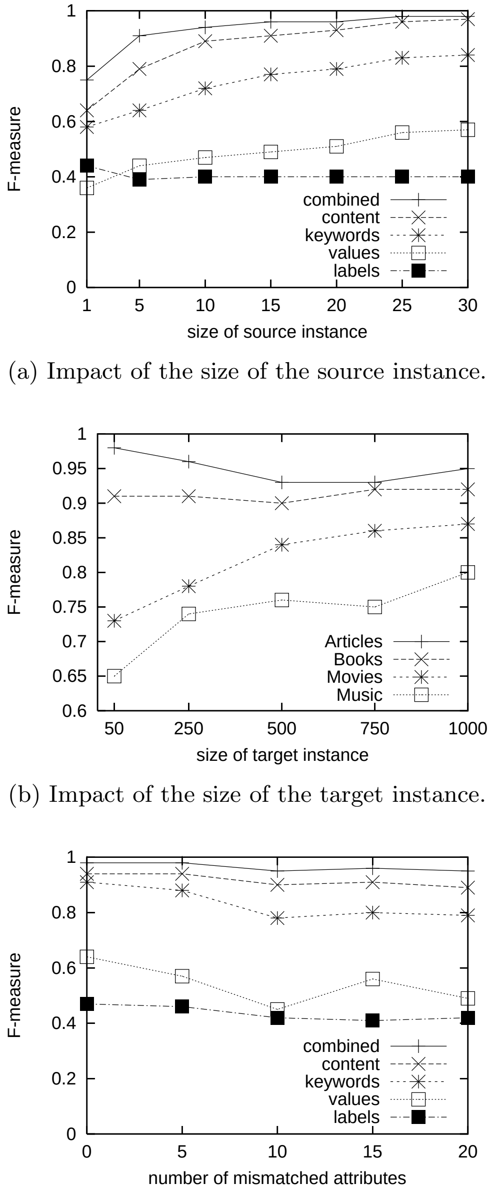
Figure 7: Accuracy of individual similarity measures across domains.

etc.), and use our Data Fitting method with different similarity measures. As the graph shows, the combined method we proposed (recall Section 4.1) outperforms all individual similarity measures; this is particularly evident for the most complex domains in our tests: Movies and Music.

Impact of source instance size. We use the Movies data collections in this experiment. Figure 8(a) compares the effectiveness of the Data Fitting method with varying sizes of the source instance; each plot shows the average accuracy of 20 runs, each with a different sample from the source movies collection. Note that the combined method again outperforms the others, particularly for smaller source instances (i.e., when exchanging fewer entities). The drop in performance of the *labels* approach is due to the fact that more optional attributes are present in larger samples.

Impact of the target instance size. Figure 8(b) shows how the F-scores of the *combined* similarity method vary as a function of the number of entities in the target data collection. Each plot shows the average accuracy of 5 runs, each with a different subset of the target collection in Table 1. In each run we use 20 samples from the corresponding source data collection, with 10 “main” entities each. Observe that the Data Fitting method performs very well regardless of collection size in simple collections (Articles and Books), which are likely to occur on the Web. For the more complex collections, as expected, the accuracy of the method improves as more entities are stored in the target collection.

Resilience to noise. We also studied the impact of spurious attributes in the source instance on the accuracy of our method, using the Movies data collections. Each plot shows the average accuracy of 20 runs, each with 10 movies. We start with only those attributes that have a perfect match into the target data collection and add *unmatched* attributes, and progressively add other attributes (with real data from the Web source) that have no match in the target collection. As one can see, the *combined* similarity suffers the least relative drop in accuracy of all measures, remaining almost perfect even when only 1/3 of the attributes in the source instance have a match in the target instance (recall from Table 1 that only 10 attributes match in the Movies data collections).



(a) Impact of the size of the source instance.

(b) Impact of the size of the target instance.

(c) Resilience to noise.

Figure 8: Accuracy results.

6. CONCLUSION

This paper introduced a lightweight data exchange framework sharing data on the Web or through P2P systems. Unlike previous solutions to the problem, our approach does not require the data to be stored inside database systems, nor the use of special-purpose schema mapping tools. Thus, our method is particularly attractive to for non-expert and casual users lacking the expertise or resources for setting up a complex data sharing environment. The data model and schema formalism used in our method are simple yet powerful enough for the setting considered. Finally, extensive experimental results with real Web data showed that our approach is effective and very promising.

There are several lines for future work. For instance, sometimes one may be interested in exchanging only small fragments of a large entity, and to associate them with other existing entities (e.g., a user adding a new CD to an existing artist). Thus, it would be interesting to define a means for the user to specify such update operations in a simple, intuitive way (i.e., without having to write complex XQuery update statements). Also, we would like to extend our model with simple constraints to enrich the data translation algorithm; in particular, we believe that uniqueness and referential constraints should be enough for most practical settings. Finally, it would be interesting to study how a data exchange tool based on a lightweight framework such ours fares against more sophisticated ones in the context of the Web.

Acknowledgments. This work was supported in part by grants from the Natural Science and Engineering Research Council of Canada; projects GERINDO (CNPq/CT-INFO 552.087/02-5) and SIRIAA (CNPq/CTAmazônia 55.3126/ 2005-9); UOL (www.uol.com.br), through the "UOL Bolsa Pes-quisa" program, Proc. num. 20060520151215a; and individual grants from the Alberta Ingenuity Fund (D. Barbosa), CAPES (F. Mesquita), and CNPq (303032/2004-9 A. S. da Silva).

7. REFERENCES

[1] M. Arenas and L. Libkin. XML data exchange: consistency and query answering. In *PODS '05: Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 13–24, New York, NY, USA, 2005. ACM Press.

[2] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison Wesley, 1999.

[3] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. *Extensible Markup Language (XML) 1.0.* World Wide Web Consortium, fourth edition, August 16 2006. `http://www.w3.org/TR/xml`.

[4] W. W. Cohen and H. Hirsh. Joins that Generalize: Text Classification Using WHIRL. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pages 169–173, 1998.

[5] W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A Comparison of String Distance Metrics for Name-Matching Tasks. In *Proceedings of IJCAI-03 Workshop on Information Integration on the Web*, pages 73–78, 2003.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* The MIT Press, Cambridge, MA, USA, 2nd edition, 2001.

[7] E. Cortez, A. S. da Silva, M. A. Gonçalves, F. Mesquita, and E. S. de Moura. FLUX-CIM: flexible unsupervised extraction of citation metadata. In *Proceedings of the 2007 conference on Digital libraries*, pages 215–224, New York, NY, USA, 2007. ACM Press.

[8] R. Fagin, P. G. Kolaitis, R. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *Theoretical Computer Scince*, 336(1):89–124, May 2005.

[9] A. Fuxman, P. G. Kolaitis, R. J. Miller, and W.-C. Tan. Peer data exchange. *ACM Trans. Database Syst.*, 31(4):1454–1498, 2006.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[11] R. Goldman and J. Widom. DataGuides: Enabling query formulation and optimization in semistructured databases. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB'97)*, Athens, Greece, August 25-29 1997.

[12] R. Huebsch, B. N. Chun, J. M. Hellerstein, B. T. Loo, P. Maniatis, T. Roscoe, S. Shenker, I. Stoica, and A. R. Yumerefendi. The Architecture of PIER: an Internet-Scale Query Processor. In *Second Biennial Conference on Innovative Data Systems Research (CIDR)*, pages 28–43, 2005.

[13] L. Libkin. Data exchange and incomplete information. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 60–69, New York, NY, USA, 2006. ACM Press.

[14] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic Schema Matching with Cupid. In *Proceedings of 27th International Conference on Very Large Data Bases*, pages 49–58, 2001.

[15] J. Madhavan, S. Cohen, X. L. Dong, A. Y. Halevy, S. R. Jeffery, D. Ko, and C. Yu. Web-Scale Data Integration: You can afford to Pay as You Go. In *Third Biennial Conference on Innovative Data Systems Research*, pages 342–350, 2007.

[16] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching. *18th International Conference on Data Engineering*, pages 117–128, 2002.

[17] F. Mesquita, A. S. da Silva, E. S. de Moura, P. Calado, and A. H. F. Laender. LABRADOR: Efficiently publishing relational databases on the web by using keyword-based query interfaces. *Inf. Process. Manage.*, 43(4):983–1004, 2007.

[18] W. S. Ng, B. C. Ooi, K.-L. Tan, and A. Zhou. Peerdb: A p2p-based system for distributed data sharing. In *Proceedings of the 19th International Conference on Data Engineering*, pages 633–644. IEEE Computer Society, 2003.

[19] J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kauffmann, 1988.

[20] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, and R. Fagin. Translating Web Data. In *Proceedings of 28th International Conference on Very Large Data Bases*, pages 598–609, 2002.

[21] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.

[22] J. Shanmugasundaram, E. Shekita, R. Barr, M. Carey, B. Lindsay, H. Pirahesh, and B. Reinwald. Efficiently publishing relational data as xml documents. *The VLDB Journal*, 10(2-3):133–154, 2001.

[23] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, and J. F. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *Proceedings of 25th International Conference on Very Large Data Bases*, pages 302–314, September 7-10 1999.