Query Relaxation Using Malleable Schemas

Xuan Zhou Julien Gaugaz Wolf-Tilo Balke Wolfgang Nejdl

L3S Research Center
Leibniz University Hanover
Hanover 30167, Germany
{zhou,gaugaz,balke,nejdl}@l3s.de

ABSTRACT

In contrast to classical databases and IR systems, real-world information systems have to deal increasingly with very vague and diverse structures for information management and storage that cannot be adequately handled yet. While current object-relational database systems require clear and unified data schemas, IR systems usually ignore the structured information completely. Malleable schemas, as recently introduced, provide a novel way to deal with vagueness, ambiguity and diversity by incorporating imprecise and overlapping definitions of data structures. In this paper, we propose a novel query relaxation scheme that enables users to find best matching information by exploiting malleable schemas to effectively query vaguely structured information. Our scheme utilizes duplicates in differently described data sets to discover the correlations within a malleable schema, and then uses these correlations to appropriately relax the users' queries. In addition, it ranks results of the relaxed query according to their respective probability of satisfying the original query's intent. We have implemented the scheme and conducted extensive experiments with real-world data to confirm its performance and practicality.

Categories and Subject Descriptors: H.3.3 [Information Search and Retrieval]: Query formulation

General Terms: Design, Algorithm.

Keywords: Malleable schema, Query relaxation.

1. INTRODUCTION

Recently there has been great interest in combining Database search and Information Retrieval technologies. The main reason is that most real-world applications have to cope with structured and unstructured data simultaneously, while neither DB nor IR tools supports a seamless way to meet that requirement. The most well-known examples are Personal Information Management (PIM) [19, 13] and Enterprise Information Management (EIM) [6] systems. In these applications, structured and unstructured data are always mixed: unstructured data usually include documents, images, user profiles, audio/video and all kinds of human consumable information, while structured data include enterprise data, but also properties associated with unstructured items (such as *title, creation date, origin, file format*) and various relationships among them (such as *author, reference*). Such property and relationship information can be defined by users, generated by specific applications or extracted using Information Extraction techniques. This information is usually very diverse and vague, so that it is difficult to be handled by classic DB systems, which rely on a clear and rigid data schema.

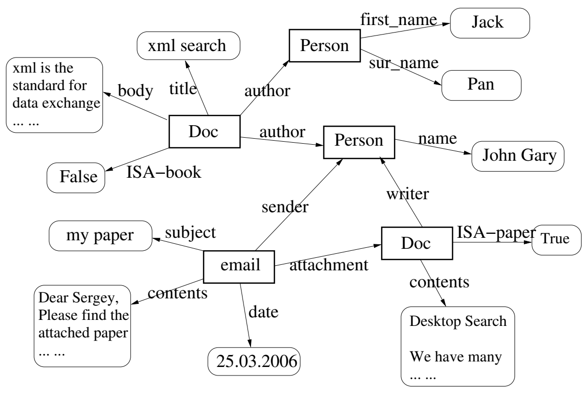

Figure 1: Data Modeled by Malleable schema

In [12], the authors proposed *malleable schemas* as a modeling tool for the diverse and vague data structures in the real world. The approach recognizes that the structure of a particular domain is usually extremely complicated, and by nature its model should allow some vagueness or redundancies in order to capture all intended semantics. In contrast to the predefined and rigid schemas in conventional DB systems, malleable schemas intentionally contain heterogenous and overlapping definitions of data structures that can be enriched and queried anytime. For example, Figure 1 shows a data instance modeled by a malleable schema. There exist various attributes and relationships. We can see that *name*, *first-name* and *surname* are overlapping attributes; so are *title*, *body* and *contents*, *author* and *writer*. While its schema seems to have a lot of redundancies, it allows us to effectively capture the diverse semantics in the domain.

Due to these characteristics, malleable schemas are a suitable tool for managing the vaguely structured data in various PIM and EIM systems. In this paper, we complement the initial work on malleable schemas in [12] by proposing a query scheme that enables users to effectively and efficiently search structured and unstructured information by exploiting malleable schemas for query relaxation.

The Challenges:

While a malleable schema is intended to capture heterogenous data structures (i.e. properties and relationships), a data instance modeled by the malleable schema is usually described by only a part of the structures based on its specific usage. When a user queries the data using a malleable schema, he has to appropriately relax the query in order to retrieve the complete set of relevant results. For example, a user might issue the following query to search for a person whose first name is "Daniel".

$Q1$: Select Person
    Where first-name = "Daniel"

Not all *persons* in the system will be represented with both attributes *first-name* and *surname*, though. As shown in Figure 1, some *persons* may only use a single attribute *name*. Thus, to find all the persons named "Daniel", the user needs to relax $Q1$ to $Q2$.

$Q2$: Select Person
    Where first-name = "Daniel"
    Or name Contains "Daniel"

Although the relaxation can find more relevant results, it might also introduce some irrelevant results, because not all $persons$ having *name* $\ni$ *"Daniel"* also satisfy *first-name = "Daniel"* ("Daniel" could be a surname). A reasonable tradeoff is to rank the query results according to their probabilities of relevance, such that the user's need can be satisfied as soon as possible. Considering $Q2$, the results satisfying *first-name = "Daniel"* should be returned prior to the results satisfying *name* $\ni$ *"Daniel"*, as the latter may contain irrelevant results.

In summary, to query data using malleable schemas, we should be able to:

Challenge One: (1) properly relax the query such that it allows to retrieve all relevant results; (2) rank query results according to their probabilities of matching the original query's intent.

To provide proper query relaxation, we must first identify the correlated elements in a malleable schema. For example, we need to know that *first-name* is a part of *name* in order to know that $Q1$ should be relaxed to $Q2$. Furthermore, to enable the ranking of query results, we need to quantify these correlations. If we know that the *title* of an *article* has a stronger correlation to its *abstract* than to its *body*, we can infer that the results of relaxing *title* to *abstract* are more relevant than the results of relaxing *title* to *body*.

Therefore, to address Challenge One effectively, we need to address the following challenge first:

Challenge Two: (1) identify correlated schema elements; (2) quantify these correlations.

This paper aims to handle these two challenges to enable effective query relaxation using malleable schema.

There is a significant body of work on schema matching [27, 10, 24, 18], focusing on how to discovering correspondences between two different schemas. These techniques utilize all kinds of resources, such as schema description, data instances and domain knowledge, to find the schema elements that are similar to each other. Some systems also assign weights or ratings to their outcomes, as quantifications of detected schema matches. However, the semantics of those matches and quantifications are different from the schema correlations needed in query relaxation (see Section 3), so that we cannot rely on them to obtain good rankings of query results. In real-world data, there are usually some objects that have been classified and stored in more than one way. In this paper, we present a novel scheme which utilizes these duplicates within the data sources to discover and quantify the correlations within a malleable schema.

There has been intensive research recently on approximate querying over structured data, like XML documents [15, 7, 3, 25, 23]. Similar to our work, they focus on query relaxation and ranking query results based on various similarities. However, as they only consider the explicit correlations given by the hierarchies of XML trees, they cannot be used to handle queries over malleable schemas. Our results are therefore a useful complement to that field of research.

Our Contribution:

The contribution of this paper can be summarized as follows:

1. We propose a query relaxation model that enables users to query vaguely structured information by exploiting malleable schemas (Section 2).
2. We propose a scheme that utilizes the duplicates in the database to find and quantify the correlations in a malleable schema (Section 3).
3. We devise a query engine that can efficiently perform query relaxation and return ranked results (Section 4).
4. We conduct extensive experiments on practical data crawled from Web sources like the IMDB collection and the Amazon catalog to show the effectiveness and performance of our approach in typical scenarios (Section 5).

Section 6 compares our approach against related work, and Section 7 concludes the paper as well as discussing directions for future research.

2. DATA / QUERY MODELS

We first present the data model that malleable schemas use to define vaguely structured data, and will then introduce a query model and a probabilistic model for query relaxation.

2.1 The Data Model

Following [12], we assume that malleable schemas use a Entity-Relationship data model, which expresses data by a number of entities, attributes and relationships. An entity is represented by a set of attributes, where each attribute is a binary relation between the entity and a value (usually a bag of words). A relationship is a binary relation between two entities. For simplicity, we do not consider relationships involving more than two entities, and we do not assign attributes to relationships. The types of entities, attributes and relationships are identified by keywords, which enables users to easily issue queries. Figure 1 shows some example data represented in this model. It includes entities such as *document*, *person* and *email*, attributes such as *name*, *title* and *date*, and relationships such as *author*, *sender* and *attachment*. We consider this data model appropriate and sufficient, because data in real PIM and EIM systems is usually organized around entities such as articles, photos, emails as well.

We will represent categorical attributes of an entity in pivoted form. In other words, we express that an entity belongs (or does not belong) to a category by assigning it an attribute like *ISA-category = true/false*. For example, Figure 1 states that the first *document* is not a book and the second *document* is a paper. As shown in Section 3, such a representation will enable us to find correlations between individual categories.

A malleable schema is created based on this data model. Different from a rigid schema in traditional relational database, a malleable schema does not need to be concise, but contains imprecise and overlapping definitions of attributes or relationships. In this way, a malleable schema can capture such heterogeneous data structures as in Figure 1.

2.2 The Query Model

Since in PIM and EIM systems user search is usually targeted on entities (like documents, emails and webpages), we use an entity oriented query model for malleable schemas. In this model the objective of a query is always a single entity. A user can express her needs by specifying the attributes of the entity or the relationships between this entity to other entities. For example, the user can issue the following query based on the schema in Figure 1,

$Q3$: Select Doc As E1
Where E1.title Contains "XML Query"
And E1.ISA-paper Contains "True"
And E1.author Contains E2
And E2.name Contains "Daniel"

which queries for an *paper* whose title contains "XML Query" and whose author is named "Daniel".

To simplify future analysis, in the following we restrict the comparison operator in a query to containment ($\ni$) only, and the connector between selection criteria to conjunctions only (no disjunction or negation). Hence, our queries can be expressed as a conjunction of predicates, each in the form of *"$A \ni t$"*, where A is an attribute or relationship and t is a term or entity. For example, $Q3$ can be written as:

$Q3 = \{E1|\mathit{E1.title}\ni\text{'}\mathit{XML}\text{'} \wedge \mathit{E1.title}\ni\text{'}\mathit{Query}\text{'} \wedge \mathit{E1.ISA\text{-}paper}\ni\text{'}\mathit{True}\text{'} \wedge \mathit{E1.author}\ni \mathit{E2} \wedge \mathit{E2.name}\ni\text{'}\mathit{Daniel}\text{'}\}$.

All entities satisfying these predicates will be correct answers to the query. While the expressiveness of this query model is limited, we believe it can satisfies user needs in most practical cases. Moreover, it can be straightforwardly extended to include more operators. For example, when disjunction is involved, we can first transform the query to disjunctive normal form and process each conjunctive clause separately.

2.3 A Probabilistic Query Relaxation Model

As each data instance uses only a subset of the attributes or relationships defined in a malleable schema, the predicates in a query have to be properly relaxed to retrieve all relevant results. Such relaxation is achieved by extending the types of attributes or relationships. For example, $Q3$ can be relaxed by extending *E2.name$\ni$'Daniel'* to *E2.first-name$\ni$'Daniel'*, which would still retrieve relevant results. By query relaxation, a query will be turned into a set of queries. We call those the *relaxed queries* of the original query. For example, the relaxed queries of $Q3$ contain:

$Q5 = \{E1|\mathit{E1.title}\ni\text{'}\mathit{XML}\text{'} \wedge \mathit{E1.title}\ni\text{'}\mathit{Query}\text{'} \wedge \mathit{E1.ISA\text{-}paper}\ni\text{'}\mathit{True}\text{'} \wedge \mathit{E1.author}\ni \mathit{E2} \wedge \mathit{E2}.\textbf{first-name}\ni\text{'}\mathit{Daniel}\text{'}\}$.

$Q6 = \{E1|\mathit{E1}.\textbf{subject}\ni\text{'}\mathit{XML}\text{'} \wedge \mathit{E1}.\textbf{subject}\ni\text{'}\mathit{Query}\text{'} \wedge \mathit{E1.ISA\text{-}paper}\ni\text{'}\mathit{True}\text{'} \wedge \mathit{E1}.\textbf{writer}\ni \mathit{E2} \wedge \mathit{E2.name}\ni\text{'}\mathit{Daniel}\text{'}\}$.

As stated earlier, because query relaxation might introduce irrelevant results, the system should return query results based on their probabilities of relevance. That means, given a query Q_0 that could be relaxed to $Q_1 \vee Q_2 \vee ... \vee Q_n$, we should return their results according to the probabilities $P(Q_0|Q_1), P(Q_0|Q_2), ..., P(Q_0|Q_n)$, where $P(Q_i|Q_j)$ represents the probability that a result of Q_j is also a relevant result of Q_i. As an example, $Q_0 = \{E|A \ni a \wedge B \ni b\}$ is relaxed to $Q_1 \vee Q_2$, where $Q_1 = \{E|A_1 \ni a \wedge B_1 \ni b\}$ and $Q_2 = \{E|A_2 \ni a \wedge B_2 \ni b\}$. If we know that $P(A \ni a \wedge B \ni b|A_1 \ni a \wedge B_1 \ni b) < P(A \ni a \wedge B \ni b|A_2 \ni a \wedge B_2 \ni b)$, we will return the results of Q_2 prior to the results of Q_1, because Q_2 will retrieve more relevant results than Q_1.

Thus, query relaxation requires estimating the correlation between the original query and each of its relaxed queries, i.e. $P(Q|Q_j)$. These conditional probabilities cannot be estimated straightforwardly, as there exist a huge number of query variances even in a simple malleable schema. We therefore use a probabilistic model that relies on two basic assumptions to make the computation of $P(Q|Q_j)$ feasible.

Assumption 1: Given a query $Q = \{E|A_1 \ni a_1 \wedge A_2 \ni a_2 \wedge ... \wedge A_k \ni a_k\}$, for any $i, j \in [1, k]$ such that $i \neq j$, $A_i \ni a_i$ is independent of $A_j \ni a_j$. If $Q' = \{E|A'_1 \ni a_1 \wedge A'_2 \ni a_2 \wedge ... \wedge A'_k \ni a_k\}$ is a relaxed query of Q, for any $i, j \in [1, k]$ such that $i \neq j$, $A'_i \ni a_i$ is independent of $A_j \ni a_j$. $\Box$

Assumption 1 states that all the predicates in a query are independent and their corresponding predicates in the relaxed query are also independent. This assumption is reasonable, because a user seldom uses two correlated predicates in one query. For instance, a user will not say "I am looking for a person whose name is 'Daniel' and whose first-name is also 'Daniel'", as this is too wordy. Similar assumptions have been widely used in IR models [28, 26]. Some techniques [18] even can identify synonyms by assuming that synonyms rarely co-occur in the same paragraph.

Assumption 2: For each pair of terms a and b, $P(A \ni a|A' \ni a) = P(A \ni b|A' \ni b)$. (Subsequently, we use $P(A|A')$ to denote $P(A \ni x|A' \ni x)$.) $\Box$

Assumption 2 states that the correlation between two predicates is independent of the values in the predicates. This is equivalent to saying that the correlations in a malleable schema apply to all data instances defined by the schema. Though this might not always be true, we believe it is a sufficiently good estimate of data correlation at the schema level. Functional dependencies in relational databases are an analogous concept.

Theorem 1: If Assumptions 1 and 2 hold, for any query $Q = \{E|A_1 \ni a_1 \wedge A_2 \ni a_2 \wedge ... \wedge A_k \ni a_k\}$ and one of its relaxed queries $Q' = \{E|A'_1 \ni a_1 \wedge A'_2 \ni a_2 \wedge ... \wedge A'_k \ni a_k\}$, $P(Q|Q') = P(A_1|A'_1)P(A_2|A'_2)...P(A_k|A'_k)$. $\Box$

Theorem 1 decomposes the correlation between two queries into the correlations (i.e. $P(A|A')$) between the attributes or relationships in a malleable schema. This makes the actual computation of query correlations much easier. In the next section, we introduce our method to discover and estimate the correlations between the attributes or relationships in a malleable schema.

3. DISCOVERING CORRELATIONS IN A MALLEABLE SCHEMA

As stated earlier, to achieve query relaxation we need to find and quantify the correlated attributes and relationships in a malleable schema. With the above query relaxation model, the task becomes estimating the conditional probability $P(A|A')$ (i.e. $P(A \ni x|A' \ni x)$) for any pair of attributes or relationships.

A brute force approach to estimate $P(A|A')$ is to examine the data instances where A and A' co-occur. For example, if we find that the terms in the attribute *first-name* also appear in the attribute *name* on all entities, we can estimate that *P(name|first-name)*= 1. Similarly, if only half of the terms in the *name* appear in *first-name*, we have *P(first-name|name)*= 0.5. However, in real cases, it is very unlikely that correlated attributes always co-occur on a common entity. For instance, if a *person* has been registered using attributes *first-name* and *surname*, usually it will not be registered for the attribute *name* anymore, because this is redundant. As a result, there will not be enough samples to make such statistical estimates. We propose to use the duplicates in the data sources instead. These duplicates are entities that are described by different attributes but refer to the same real-world concept, and are very common in multi-application systems such as PIM or EIM. Once being detected, they can serve as good samples for estimating the correlations in a malleable schema.

Therefore, discovering correlations in a malleable schema consists of two steps: (1) detecting duplicates in data; (2) using the duplicates to quantify the correlations in malleable schema.

Please note that the semantics of schema correlation is slightly different from that of schema match in classic data integration tasks. Schema match measures the likelihood that two attributes refer to the same real-world concept, while schema correlation measures how much the contents of two attributes overlap. As an example, *title* and *abstract* are very different concepts, but they are correlated in contents. Nevertheless, schema match can capture schema correlation to some extent, so that it could be a back-up solution when we do not have sufficient duplicates. We do not address it in this paper, though it can be an interesting problem for future research.

3.1 Duplicate Detection with Malleable Schema

The task of duplicate detection has been extensively studied in data integration and data cleaning [14]. However, most techniques assume a rigid schema and cannot be directly applied to malleable schemas. Especially when entities are described by different schema elements, detection results usually become very imprecise. Therefore, duplicate detection with malleable schemas should integrate with the process of discovering schema correlations. On one hand, knowing the correlations, we can more accurately detect the duplicates. On the other hand, the detected duplicates are quality evidences to infer the correlated attributes or relationships in the schema. We will therefore rely on a new algorithm that allows duplicate detection and discovering schema correlations to reinforce each other thus generating more precise results.

	title	subject	author	writer	pub-date	rec-date
$E1$	XML		Daniel		Jan 1999	
$E2$		XML		Daniel		Dec 2003
$E3$	DB		Ullman		Jul 1994	
$E4$		DB		Ullman		Nov 2001
$E5$	AI		Stuart		Nov 2001	
$E6$		Logic		Stuart		Nov 2001

Table 1: Duplicates under Malleable Schema

3.1.1 An Example

Table 1 shows an example of malleable schema data. It contains 6 entities that are described by 6 attributes. An inspection of the data gives us the impression that *title* vs *subject*, *author* vs *writer* and *pub-date* vs *rec-date* seem to be 3 pairs of correlated attributes, and $E1$ vs $E2$, $E3$ vs $E4$ and $E5$ vs $E6$ seem to be 3 pairs of duplicated entities. If we use the 3 pairs of duplicates to test the 3 attribute correlations, we discover that *pub-date* vs. *rec-date* are actually not correlated (one is publication date and the other is receiving date), because their values are different in most of the duplicates (i.e. $E1$ vs. $E2$ and $E3$ vs. $E4$). After eliminating this attribute correlation, we will further discover that $E5$ and $E6$ are no longer likely duplicates, because they are similar in only one pair of correlated attributes (*author* vs *writer*), which is not sufficiently convincing. In the end, we can conclude that only *title* vs *subject* and *author* vs *writer* are truly correlated, and $E1$ vs $E2$ and $E3$ vs $E4$ are authentic duplicates.

The above process can be summarized into the following algorithm:

1. **Duplicate detection:** based on current schema correlations, find all possible duplicates.
2. **Correlation detection:** based on current duplicates, reassess the schema correlations.
3. If the schema correlations did not change in step 2, stop the process. Otherwise, go to step 1.

Our algorithm performs duplicate detection and correlation detection iteratively until the resulting schema correlations and duplicates do not change anymore. Finally, the detected duplicates are much more precise than those detected without considering schema correlations. As a side effect, schema correlations will be disclosed, too. The following sections present our detailed algorithms to detect duplicates and schema correlations.

3.1.2 The DSCD Algorithm

Our algorithm for duplicate and schema correlation discovery (DSCD) is based on the observations above. When presenting the algorithm, we first ignore the relationships between entities and consider only attribute correlations.

Let $c_1, c_2, ..., c_n$ be the pairs of attributes that are likely to be correlated, where each c_i consists of two attributes represented by $c_i = (A_i, A'_i)$. Let $d_1, d_2, ..., d_m$ be the possible duplicate pairs, where each d_j consists of two entities represented by $d_j = (E_j, E'_j)$. C is a n-dimensional vector, where each element $C(i)$ is a weight indicating our belief of that c_i is a true correlation. D is a m-dimensional vector where each $D(i)$ indicates our belief of that d_i is a true duplicate. S is a $n \times m$ matrix called *evidence matrix.* Each element $S(i,j)$ measures the similarity between the attribute A_i on entity E_j and the attribute A'_i on entity E'_j, namely $S(i,j) = Sim(E_i.A_j, E'_i.A'_j)$. Obviously, if the attributes in c_j are truly correlated and the entities in d_i are true duplicates, this similarity should be high, and vice versa.

Given the belief of the schema correlations and the evidence matrix S we can deduce our belief in the duplicates. In particular, if the correlated attributes on two entities are more similar, we are more confident that the two entities are duplicates. This can be represented by

$$D(i) = \sum_{k=1}^{n} C(k) * S(i,k)$$

which can be written into

$$D = C \times S$$

Analogously, we can deduce C from D and S, too. In particular, if two attributes are more similar on the duplicates, we are more confident that the two duplicates are really correlated. This can be represented by

$$C(i) = \sum_{k=1}^{m} D(k) * S(k,i)$$

which could be written into

$$C = D \times S^T$$

Our DSCD algorithm performs the two deduction processes iteratively until our belief of schema correlations and our belief of duplicates are consistent with each other. As a result, each schema correlation and duplicate is given a weight indicating its likelihood to be true. The detailed algorithm is:

1. $C_0 = (0.5, 0.5, ..., 0.5)$ and $i = 1$;
2. $D_i = C_{i-1} \times S$;
3. $C_i = D_i \times S^T$;
4. If $C_i \neq C_{i-1}$, $i = i+1$ and go to step 2; otherwise, end the process and output C_i and D_i.

Since we have no proof about valid schema correlations in the beginning, we always assign them equal belief 0.5. With the algorithm going on, our beliefs of the schema correlations and duplicates will be repeatedly verified by the evidence matrix S and thus become increasingly clear. Combining he equations in Steps 2 and 3, we have

$$C_i = C_0 \times \left(S \times S^T\right)^i$$
$$D_i = C_0 \times S \times \left(S^T \times S\right)^{i-1}$$

Interestingly, the resulting equations are exactly the same as those of the Kleinberg HITS algorithm [20], which is used to compute page ranks in Web searches. Kleinberg shows that according to standard results of linear algebra, C_i is going to converge to the principal eigenvector of $S \times S^T$, and D_i is going to converge to the principal eigenvector of $S^T \times S$. This ensures that also the above algorithm will terminate.

Applying this algorithm to the example in Table 1, we have c_1 =(*title,subject*), c_2 =(*author,writer*), c_3 =(*pub-date, rec-date*), and $d_1 = (E1, E2)$, $d_2 = (E3, E4)$, $d_3 = (E5, E6)$. The evidence matrix S can be instantiated to the one in Table 2. ($E_i.author = E_j writer$ will have smaller similarity values because they are more probable to coincide than

	(title,subject)	(author,writer)	(pub-date,rec-date)
($E1$,$E2$)	1	0.7	0
($E3$,$E4$)	1	0.7	0
($E5$,$E6$)	0	0.7	1

Table 2: Matrix of Table 1

$E_i.title = E_i.subject$ and $E_i.pub\text{-}date = E_i.rec\text{-}date$.) The outcomes (after normalization) will be $C = (1.0, 0.89, 0.28)$ and $D = (1.0, 1.0, 0.56)$. As expected, c_1 and c_2 are more likely to be true than c_3, and d_1 and d_2 are more likely to be true than d_3.

3.1.3 The Similarity Measure

To instantiate an evidence matrix S, it is important to know how to measure the similarity between two attributes on two entities, namely $Sim(E_j.A_i, E'_j.A'_i)$. In the IR area, there exist a number of methods to measure the similarity between two bag of terms, such as the cosine similarity between TF×IDF vectors. However, we do not think those measures are adequate for application in our case, as they cannot capture the relationship between the compared attributes. For example, they cannot capture that *first-name* is actually a part of *name*. In order to preserve the relationship information, we consider the order of schema correlations and thus distinguish between (A_i, A'_i) and (A'_i, A_i). We also consider the order of similarities and distinguish between $Sim(E_j.A_i, E'_j.A'_i)$ and $Sim(E_j.A_i, E'_j.A'_i)$. Hence, we use the following similarity measure:

$$Sim(E_1.A_1, E_2.A_2) = \frac{|E_1.A_1 \cap E_2.A_2|}{|E_2.A_2|} \times H(E.A_2 \ni x)$$

The first factor measures the percentage of terms in $E_2.A_2$ that are also in $E_1.A_1$. This factor alone does not constitute a good measure, as it is significantly depending on the general sizes of attributes A_1 and A_2. For example, if A_1 and A_2 can only contain two particular terms (e.g. *true* and *false*), the measure will be very high anyway ($\geq 50\%$), even if A_1 and A_2 are independent. Therefore, we normalize it by the second factor, the entropy of $E.A_1 \ni x$, which measures the amount of information of knowing that a term x belongs to attribute A_2. This entropy can be statistically estimated using the values in all A_2 attributes we know.

3.1.4 Extension to Relationship Correlations

So far, our approach only deals with attribute correlations. To involve relationship correlations, we only need to know how to measure the similarity between two separate relationships on two entities, namely $Sim(E_j.R_i, E'_j.R'_i)$. As attributes are binary relations between entities and values, we consider two attributes as being similar, if they contain similar values. In contrast, relationships are binary relations between entities (sub-entities) and entities (ob-entities), so we say that two relationships are similar if they are associated to similar ob-entities. To assess whether two ob-entities are similar, we measure the similarity between the attributes on the two ob-entities. This results in the following measure:

$$Sim(E_1.R_1, E_2.R_2) = \frac{|E_1.R_1 \cap E_2.R_2|}{|E_2.R_2|} \times H(E.R_2 \ni x)$$

where
$$E_1.R_1 = \{x | x \in E'_1.A \wedge E'_1 \in E_1.R_1\}$$
$$E_2.R_2 = \{x | x \in E'_2.A \wedge E'_2 \in E_2.R_2\}$$

It aggregates the terms in all attributes on the ob-entities, and uses them to represent the terms of the relationship, such that the similarity between relationships can be computed in the same way as that between attributes. In practice, we can selectively use the attributes that are most representative of the ob-entities.

3.2 Quantifying Schema Correlations

The DSCD algorithm generates a weight for each duplicate candidate and for each schema correlation candidate. While the weights of schema correlations indicate our belief in their correctness, they are not yet the conditional probabilities $P(A|A')$ we need to use in query relaxation. Those probabilities can only be estimated by studying the individual duplicates. We choose a set of duplicates that are most likely to be correct, and use them as samples to assess the conditional probability between two attributes or relationships.

When choosing sample duplicates, we need to find an appropriate tradeoff between quantity and quality. If we take too many duplicates, we might include a lot of false duplicates. If we take too few, the sample size is too small to give unbiased estimates. The method to find good threshold will be presented in the experimental section. With the sample duplicates, we use the maximum likelihood model to estimate the conditional probability between two attributes. It results in

$$P(A|A') = \frac{\sum_{(E_i,E_j)} \frac{|E_i.A \cap E_j.A'|}{|E_j.A'|}}{|(E_i,E_j)|}$$

where each (E_i, E_j) is a pair of duplicates. We use a similar equation to estimate the conditional probability between two relationships:

$$P(R|R') = \frac{\sum_{(E_i,E_j)} \frac{|E_i.R \cap^T E_j.R'|}{|E_j.R'|}}{|(E_i,E_j)|}$$

where $E_i.R \cap^T E_j.R'$ denotes the common ob-entities between $E_i.R$ and $E_j.R'$ and the duplicated ob-entities in $E_i.R \cup E_j.R'$ with weights larger than T. We estimate $P(A|A')$ only when the DSCD algorithm gives a significant weight to the correlation between A and A', because only correlated attributes or relationships will be used in query relaxation.

3.3 Implementation Issues

When performing the DSCD algorithm to detect duplicates and schema correlations, we need first to obtain the candidates for duplicates and the candidates for schema correlations. Given n entities, there exist n^2 possible pairs of duplicates. Given m attributes, there exist m^2 possible attribute correlations. If we take all of them as candidates for duplicate and schema correlation, runtimes become infeasible. Therefore it is necessary to prune some candidates that are very improbable to be correct. For pruning duplicate candidates, we can transform each entity to a text fragment by concatenating all its attributes and employ IR techniques (such as cosine similarity) to preselect those pairs of entities that are similar enough to be real duplicates. Since the DSCD algorithm works sufficiently well with a limited number of duplicates, we do not need to feed all possible duplicates to it. When assessing the conditional probabilities, we can use the schema correlations vector C to find more duplicates. To prune the candidates of correlated attributes or relationships, we can resort to various schema matching techniques, which utilize schema description and data types to eliminate the pairs of attributes or relationships that are impossible to be correlated.

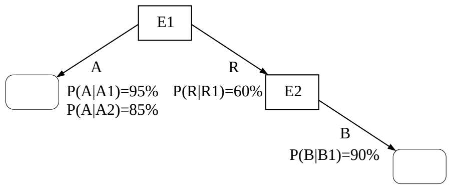

Figure 2: Query Example – Q_0

Thus, our procedure of identifying and quantifying correlations in malleable schema consists of 3 steps.

1. **Preparation:** Find all pairs of attributes that are possibly correlated; select some duplicate candidates.
2. **Verification:** Perform the DSCD algorithm to find the authentic duplicates and schema correlations.
3. **Quantification:** Use the schema correlations vector C to identify more duplicates and use the duplicates to quantify the schema correlations.

Several methods could be used to further optimize the performance of the DSCD algorithm, such as applying a sigmoid function to vectors C and D to amplify the reinforcement effect. We omit the details in this paper.

4. QUERY PROCESSING

With the correlations in a malleable schema, we can properly relax a user query and rank results according to their probabilities of relevance.

4.1 Query Relaxation Planning

We assume that the entity-relationship data are stored in relational database. In contrast to ordinary queries in relational database, a query using malleable schema will be relaxed to multiple queries that are executed on different columns or tables. The major performance consideration is to find a plan that executes as less queries as possible to retrieve sufficient relevant results. The optimal plan is to execute relaxed queries in a sequence based on the expected precisions of their result sets. For example, figure 2 shows a query $Q_0 = \{E1|E1.A \ni x \wedge E1.R \ni E2 \wedge E2.B \ni y\}$, which searches for an entity $E1$ which has attribute A and relationship R to an entity $E2$ that contains attribute B. Attribute A is correlated to attributes $A1$, $A2$, with probabilities $P(A|A1) = 95\%$, $P(A|A2) = 85\%$. Attribute B is correlated to $B1$, with probability $P(A|A1) = 90\%$. Relationship R is correlated to $R1$ with probability $P(R|R1) = 60\%$. With these correlations, Q_0 can be relaxed to queries like $Q_1 = \{E1|E1.A1 \ni x \wedge E1.R1 \ni E2 \wedge E2.B \ni y\}$, $Q_2 = \{E1|E1.A2 \ni x \wedge E1.R \ni E2 \wedge E2.B1 \ni y\}$ and so on. Based on the query relaxation model in Section 3, we have $P(Q_0|Q_1) = P(A|A1)P(R|R1) = 95\% \times 60\% = 57\%$, and $P(Q_0|Q_2) = P(A|A2)P(B|B1) = 85\% \times 90\% = 76.5\%$. It means that the results of Q_2 are more precise than the results of Q_1, so that we should return the results of Q_2 prior to the results of Q_1.

If a query contains n attributes and relationships, and each of them is correlated to m other attributes or relationships, then the query relaxation will end up with m^n relaxed queries. Sometimes, it is infeasible to evaluate all queries. In practice, we try to evaluate the relaxed queries in the order of their precisions until we obtain more than k results or the

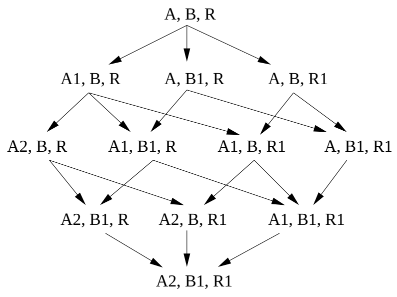

Figure 3: Query Relaxation Graph

```
func query($Q$, $k$)
   initialize $Processed$ list;
   initialize $Ready$ list;
   initialize $Result$ list;
   add $Q$ to $Ready$;
   while $Ready$ is not empty and $|Result| < k$, do
      pick $Q_i$ from $Ready$ with largest $P(Q|Q_i)$;
      add the results of exec($Q_i$) to $Result$;
      add $Q_i$ to $Processed$;
      foreach $Q_j$ that is a child of $Q_i$, do
         if $Q_j$'s parents are all in $Processed$
            add $Q_j$ to $Ready$;
      end do;
   end do;
   return $Result$;
end func
```

Figure 4: Query Relaxation Algorithm

user is satisfied and stops the processing. To achieve this, we exploit the relationship between the relaxed queries.

Parent-child relation between relaxed queries: Given a query $Q = \{E|A_1 \ni a_1 \wedge A_2 \ni a_2 \wedge ... \wedge A_k \ni a_k\}$. Let $A_i^1, A_i^2, ..., A_i^k$ denote the set of attributes/relationships that are correlated to A_i, and $\forall u < v : P(A_i|A_i^u) > P(A_i|A_i^v)$. We say that A_i^{j+1} is the child of A_i^j w.r.t. Q. (A_i^1 is the child of A_i.) Let $Q1$ and $Q2$ be two relaxed queries of Q. We say that $Q1$ is a parent of $Q2$ iff we can turn $Q1$ into $Q2$ by substituting an attribute/relationship in $Q1$ with its child attribute/relationship. □

For example, given Q_0 in Figure 2, $Q_4 = \{E1|E1.A1 \ni x \wedge E1.R \ni E2 \wedge E2.B \ni y\}$ is a child of Q_0, and $Q_5 = \{E1|E1.A2 \ni x \wedge E1.R \ni E2 \wedge E2.B \ni y\}$ and $Q_6 = \{E1|E1.A1 \ni x \wedge E1.R1 \ni E2 \wedge E2.B \ni y\}$ are children of Q_4. If we use edges to connect each relaxed query of Q_0 to its child queries, we end up with a partial order graph in Figure 3.

Proposition 1: Given a query Q and two relaxed queries $Q1$ and $Q2$, if $Q1$ is a parent of $Q2$ then $P(Q|Q1) > P(Q|Q2)$. □

Theorem 2 indicates that a relaxed query always yields better precision than its child queries, so that it should always be evaluated prior to its child queries. Thus, query relaxation using malleable schemas can be performed through the algorithm in Figure 4. The algorithm guarantees that the returned top k results are from those queries that yield the best precision. It also minimizes the number of relaxed queries to be evaluated. In [4], the authors proposed a similar algorithm for relaxing queries over multiple relaxation paths in ontologies.

4.2 Query Execution

With normal relational indexes, each relaxed query can be

```
//$Cache$ is emptied at the start of query()
func exec($Q$)
   foreach selection $\sigma$ in $Q$, do
      if $\sigma$'s results are not in $Cache$
         execute $\sigma$;
         store $\sigma$'s results to $Cache$;
   end do
   search in $Cache$ for the temp results that
   need the least joins to complete $Q$;
   execute the joins;
   store the results of the joins to $Cache$;
   return the final results;
end func
```

Figure 5: Query Execution Algorithm

executed efficiently. However, further optimization can be achieved by caching temporary results of each relaxed query in order they can be reused by subsequent relaxed queries.

The execution of a relaxed query comprises a number of selection operations and a number of join operations. For example, query $Q' = \{E|E.A \ni x \wedge E.B \ni y \wedge E.C \ni z\}$ can be implemented in the following way:

$$\left[\sigma_{A\ni x}(E)\right] \bowtie \left[\sigma_{B\ni y}(E)\right] \bowtie \left[\sigma_{C\ni z}(E)\right]$$

As another example, the Q_0 in Figure 2 can be implemented in the following:

$$\left[\sigma_{A\ni x}(E1)\right] \bowtie_{E1.R\ni E2} \left[\sigma_{B\ni y}(E2)\right]$$

The temporary results of the selections and joins can be cached and reused. For example, if the above query Q' can be relaxed to $Q'_1 = \{E|E.A1 \ni x \wedge E.B \ni y \wedge E.C \ni z\}$, then the results of $\left[\sigma_{B\ni y}(E)\right] \bowtie \left[\sigma_{C\ni z}(E)\right]$ can be reused by Q'_1. Therefore, our algorithm of query execution maintains a cache to store the temporary selection and join results. Figure 5 presents a greedy algorithm for query execution.

5. EXPERIMENTS

This section presents our experimental results, which demonstrate that our query relaxation scheme is able to accurately detect correlations in malleable schemas and effectively perform query relaxation in a fully automatic way. Section 5.1 introduces the datasets we used in our experiments. Section 5.2 presents the study on detecting duplicates and schema correlations. Section 5.3 shows the effectiveness of query relaxation. Section 5.4 demonstrates the efficiency of the query relaxation scheme.

5.1 Datasets and Experiment Setup

Being a new concept, malleable schemas have not yet been adopted in real world applications. Hence, it is difficult to find adequate datasets using authentic malleable schemas. In order to conduct our experiments, we constructed a dataset by combining real world data from different sources.

Our dataset is a combination of information about movies crawled from www.imdb.com and DVD/video items crawled from www.amazon.com. Though both sources describe similar data they organize and describe their data in different ways. Our crawl from the IMDB dataset contains information on 850,000 movies and TV series, and the Amazon.com dataset contains information on 115,000 DVDs and VHS videos. The IMDB data were joined into a single table containing 32 attributes, and the Amazon data were joined into a table containing 28 attributes. The attributes used to describe the movies and DVDs are very different, but of course

	Amazon	IMDB	confidence	P(A|I)
1	Title	title	0.701	0.619
2	Actors	actors	0.655	0.587
3	Directors	directors	0.642	0.753
4	Languages	languages	0.382	0.711
5	Edit∼Review	keywords	0.132	0.086
6	Directors	producers	0.102	0.072
7	Title	akatitles	0.090	0.097
8	ReleaseDate	year	0.081	0.098
9	Directors	writers	0.080	0.173
10	Actors	misc	0.072	0.047
11	Edit∼Review	plots	0.067	0.076
12	Actors	writers	0.061	0.098
13	Directors	proddesigners	0.059	0.002
14	Directors	cinematographers	0.054	0.023
15	Title	movielinks	0.049	0.050
16	Edit∼Review	taglines	0.046	0.056
17	Actors	producers	0.046	0.072
18	Actors	directors	0.043	0.094
19	Audi∼Rating	certificates	0.042	0.136
20	Actors	composers	0.042	0.056

Table 3: (Amazon, IMDB) Correlations

	IMDB	Amazon	confidence	P(I|A)
1	title	Title	1.000	0.923
2	actors	Actors	0.794	0.720
3	directors	Directors	0.666	0.792
4	akatitles	Title	0.380	0.303
5	languages	Languages	0.348	0.621
6	distributors	Publisher	0.264	0.335
7	distributors	Manufacturer	0.264	0.335
8	distributors	Label	0.264	0.335
9	distributors	Studio	0.264	0.335
10	movielinks	Title	0.262	0.296
11	producers	Directors	0.152	0.186
12	writers	Directors	0.104	0.259
13	year	ReleaseDate	0.081	0.098
14	technical	AspectRatio	0.073	0.108
15	actors	Creators	0.067	0.063
16	actors	Directors	0.066	0.135
17	proddesigners	Directors	0.059	0.003
18	certificates	Audi∼Rating	0.058	0.220
19	cinematographers	Directors	0.058	0.029
20	plots	Edit∼Review	0.036	0.052

Table 4: (IMDB, Amazon) Correlations

closely correlated, so that it adequately simulates the scenarios that use malleable schemas. As there is a big overlap between movies and DVDs, the dataset offers sufficient duplicates for evaluating our query relaxation scheme.

We implemented our system in Java with SDK 5. We conducted all the experiments on a PC with a 2.7GHz CPU and 1GB RAM. The DBMS we used is MySQL 4.1.11.

5.2 Performance in Detecting Duplicates and Schema Correlations

5.2.1 *The First Run*

Following the 3 steps introduced in Section 3.3, we ran our system to automatically detect and quantify the correlations between the IMDB schema and the Amazon schema.

Preparation: We pre-selected possible schema correlations by assuming that no correlation should exist between attributes of different types. This left us with 59 candidates for attribute correlations. To pre-select the duplicate candidates, we randomly picked up 1000 IMDB entities, concatenated the attributes of each entity and compare them with the Amazon entities using a TF-IDF cosine similarity. Then, we selected the 100 most similar pairs as duplicate candidates. To accelerate the process, we only used the most important and representative attributes which satisfied the following general criteria: (1) typed as text; (2) limited length ($\leq$ 100Bytes); (3) high selectivity ($\geq$ 0.05); (4) frequently used (more than 30% are not null). Besides the 100 duplicate candidates, we also included 100 non-duplicate candidates (random pairs) as baseline.

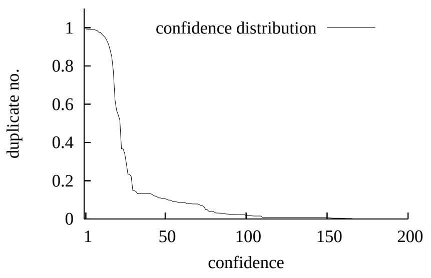

Figure 6: Confidence Distribution over Duplicates

Verification: We ran our DSCD algorithm on the pre-selected schema correlations and duplicates/non-duplicate candidates. The algorithm computed a confidence value for each schema correlation indicating the extent of belief that the match is correct. We ranked all correlations based on the confidences, and list the top 20 correlations for both directions in Tables 3 and 4 respectively. Please note that the confidence assigned to correlation (*Amazon.attribute, IMDB.attribute*) is different from that assigned to (*IMDB. attribute, Amazon.attribute*). The former shows how much the system believes that an Amazon attribute contains an IMDB attribute, and the latter the other way around.

We can see from Tables 3 and 4 that (*Title, title*), (*Actors, actors*), (*Directors, directors*) and (*Languages, languages*) are the most obvious correlations. This is consistent with the results of a manual inspection where we can actually find those attributes to be semantically equivalent. It is interesting to see that the algorithm also recognizes the direction of correlations. For example, it gave more confidence to (*IMDB.title, Amazon.Title*) than to (*Amazon.Title, IMDB.title*), as most IMDB titles contain the release years of movies while the Amazon titles do not. Apart from the most obvious correlations, our system found other interesting correlations as well. For example, the *directors* in Amazon is correlated to the *writers* and the *producers* in IMDB, as it is very common that the director of a movie also acts as writer and as one of the producers. Moreover, there exist correlations between the *EditorialReview* in Amazon and the *keywords*, *plots*, and *taglines* in IMDB, because these attributes are all related to summaries of movies. A semantically meaningful query relaxation should thus be performed on these best matching attributes.

Besides schema correlations, the algorithm verified the pre-selected duplicate candidates, too. Figure 6 shows the distribution of the confidences the algorithm gave to the total of 200 duplicate candidates and non-duplicate candidates. We can see that out of the 100 pre-selected duplicate candidates only around 22 are likely to be true (with confidence larger than 50%). We manually assessed the 22 pairs of entities and found that they are indeed all real duplicates. We can also see that the DSCD algorithm was able to distinguish the true duplicates and the false duplicates quite clearly, as the confidence values assigned to true duplicates and those for false duplicates differ significantly.

Quantification: In this step we estimate the probabilities that will be used for query relaxation, namely $P($*Amazon. attribute*$|$*IMDB.attribute*$)$ and $P($*IMDB.attribute*$|$ *Amazon.*

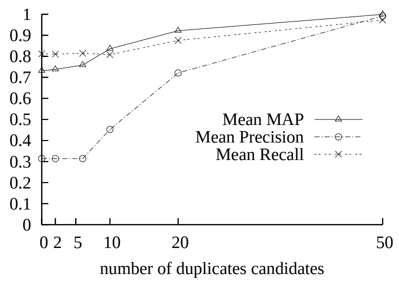

Figure 7: Sensitivity to Preselected Duplicates

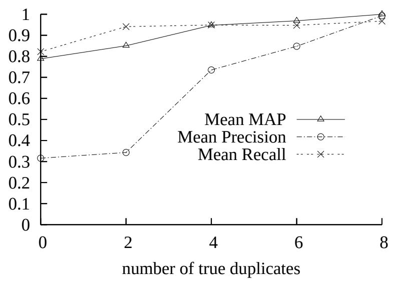

Figure 8: Sensitivity to True Duplicates

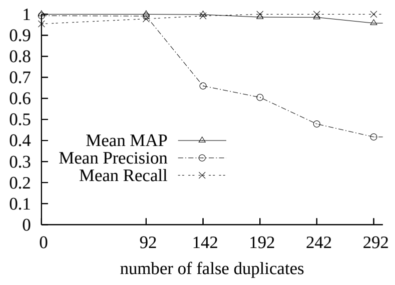

Figure 9: Sensitivity to False Duplicates

attribute). We used the K-Mean algorithm to automatically cluster the duplicate candidates into true duplicates and false duplicates based on their confidence values, and set the median of the two clusters as a threshold for duplicate detection. Then, we used the schema correlations discovered previously to find 200 more duplicates that have confidence larger than the threshold, and finally applied the functions in Section 3.3 on these duplicates to calculate the requested probabilities. The results are presented in Tables 3 and 4. They show that although confidences and probabilities are often similar, they are not completely consistent. For example, the *Languages vs. languages* are given confidence 0.3, but the corresponding probability is bigger than 0.6. Since both attributes have limited possible values, it is very likely that they are similar on real duplicates, even if they are not significantly correlated.

5.2.2 *Result Quality of the DSCD Algorithm*

After the first run, we conducted a set of experiments to evaluate the result quality of the DSCD algorithm, namely how precisely it can detect the schema correlations and duplicates. The precision of such detection determines the overall performance of our query relaxation scheme. To make such an evaluation, we used the the schema correlations (and their confidence values) discovered by the algorithm to detect additional duplicates in the database. The precision of the detection measures the accuracy of our algorithm in finding both schema correlations and duplicates.

In order to obtain objective results, we divided the 100,000 Amazon items into two sets, each containing 50,000 items. The first set was used as a training set to perform the DSCD algorithm. Then the discovered schema correlations were applied to the second set (test-set) to measure the accuracy in detecting duplicates. In particular, we used the preselection criteria of the first run to find 800 duplicate candidates from the Amazon test set and the IMDB dataset. Then we manually inspected the 800 pairs of items and picked out all pairs that are real duplicates . This resulted in 153 real duplicates. After that, we randomly selected 1000 pairs of entities from the Amazon test set and the IMDB dataset and in another manual inspection picked out all pairs that are not duplicates, which ended up with 919 non-duplicates. Finally, the 153 real duplicates and the 919 non-duplicates constituted the final test-set for our evaluation.

We used two metrics for our evaluation. The first metric was the mean average precision (MAP), which has been widely used in IR. It measures how accurately our system can order the pairs of entities in the test-set according to their likelihood of actually being duplicates. While this measure is widely applicable, it still does not clearly show whether the system can find an appropriate threshold to clearly distinguish between real duplicates and non-duplicates. Therefore, we also used classical precision and recall to measure how accurately our system can make such distinction. After performing the DSCD algorithm in the training set, we again used the K-Mean algorithm to cluster the duplicate candidates into real duplicates and non-duplicates based on their confidence values, where we used $K = 2$ and set the initial centroids to the max and min confidences respectively. Afterwards we used the central point of the two clusters as the threshold to distinguish between true duplicates and false duplicates in the test-set.

In the first set of experiments, we used the same criteria as in the first run to pre-select a certain number of duplicate candidates (varying from 0 to 50) from the Amazon training set and the IMDB dataset, and mixed them with 150 random non-duplicate candidates. We applied the DSCD algorithm on these duplicate and non-duplicate candidates, and then used the output schema correlations to detect the duplicates in the test-set. Figure 7 shows the performance of the duplicate detection. When there was no duplicate candidates, the DSCD algorithm was only able to detect correlations like *Language vs. language* and *ReleaseDate vs. year* which contain attributes with small value ranges. Hence, it was not really able to effectively find any duplicates. Though it shows a MAP of 75% the precision is only 30%, which is very bad given that there are many actual duplicates in the test-set. However, the accuracy of the DSCD algorithm increases very fast as the number of duplicate candidates grows. As shown in Figure 7, with 25 duplicate candidates, the MAP grows close to 1 and the precision and recall grows close to 80%. With 50 duplicate candidates, the precision and recall are already 95%, which means that our algorithm can almost perfectly detect all duplicates. According to the results of the first run, around 20% of the pre-selected duplicate candidates are true duplicates. This indicates that the DSCD algorithm can indeed accurately detect correlations between the Amazon and IMDB schemas already with a small number of duplicates.

In the second set of experiments, we intended to study the effects of true duplicates to the results of the DSCD algorithm. We used the same criteria as in the first run to pre-select a certain number of duplicate candidates, and manually changed the number of the true duplicates inside, such that the number of true duplicates ranged from 0 to 10 and the total number of duplicate candidates remain at 50.

Then we conducted the DSCD algorithm on the 50 duplicate candidates and another 150 non-duplicate candidates, and again used the results to detect duplicates in the test-set. The performance of the detection is shown in Figure 8. As we can see from the results, the true duplicates play an important role in the DSCD algorithm. When there were no true duplicates in the 50 pre-selected duplicate candidates, the outcome of the DSCD algorithm was hardly able to detect duplicates in the test-set (a precision of 0.3). When the number of true duplicates increased to 8, even though they were still few, the precision quickly increased close to 1.

The performance of the algorithm is also influenced by the false evidence in the pre-selected duplicate candidates. In the last set of experiments, we limited number of true duplicates in the pre-selected duplicate candidates to 8, and manually increased the duplicates candidates which were not true but had high pre-selection scores. As shown in Figure 9, both MAP and precision-recall in the final duplicate detection start to drop when the number of the false duplicates increases to a certain point (between 92 and 142). This means that when there are too many false duplicates, they will dominate the effect of true duplicates and lead the system to believe a set of wrong schema correlations. This actually happens to our human belief system too. Fortunately, our algorithm is still very tolerant to the false evidences. As we can see from Figure 9, only when there are more than 92 false duplicates (10 times more than the true duplicates), it started to take effects. In practice, false duplicates can also be relatively decreased by raising the criteria of pre-selection.

5.2.3 Schema Correlation Quantification

We also conducted a set of experiments to study how accurately our system can quantify the schema correlations. We used the schema correlations and the threshold obtained by the DSCD algorithm to find duplicates from 50 pre-selected candidates from the Amazon training set and the IMDB dataset. Then, we used the functions in Section 3.3 to estimate the probabilities based on those duplicates. We varied the number of duplicate candidates and repeated the process for several times. Figure 10 shows the correlation-coefficients between the estimated probabilities and the probabilities we obtained from the 153 real duplicates in the test-set. As expected, when there are more duplicate candidate, our system can more precisely identify duplicates and thus obtain better estimation of the probabilities.

5.3 Effectiveness of Query Relaxation

To test our query relaxation scheme, we used the correlations discovered in the first run and issued a set of queries posed to the original Amazon scheme to be relaxed to attributes of the IMDB database. We observed how the queries were relaxed and results were retrieved.

For example, we searched for the movie "Titanic (1997)" which was very popular ten years ago. As there are a lot of movies whose title contains "Titanic", we used the director "James Cameron" as an additional selection criterion, and the query managed to retrieve the unique answer "Titanic (1997)" from the Amazon data. Then, we issued the same query to the IMDB data. Using the schema correlations detected in the first run, our system relaxed the query to a number of relaxed queries, which are shown in Table 5, as well as the corresponding results retrieved by each relaxed

Original Query in Amazon	*results (Title)*
Title ∋ "Titanic" Directors ∋ "Cameron"	Titanic

Relax query in IMDB	*results (title)*
title ∋ "Titanic" directors ∋ "Cameron"	Titanic
title ∋ "Titanic" writers ∋ "Cameron"	Titanic
akatitles ∋ "Titanic" directors ∋ "Cameron"	Titanic Ghosts of the Abyss
title ∋ "Titanic" producers ∋ "Cameron"	Titanic Last Mysteries of Titanic Titanic Explorer
movielinks ∋ "Titanic" director ∋ "Cameron"	Titanic Ghosts of the Abyss
akatitles ∋ "Titanic" writers ∋ "Cameron"	Titanic
movielinks ∋ "Titanic" writers ∋ "Cameron"	Titanic
akatitles ∋ "Titanic" producers ∋ "Cameron"	Titanic Ghosts of the Abyss Titanic Explorer

Table 5: Query Relaxation Case One

Original query in Amazon	*results (Title)*
Edit∼Review ∋ "Chinese" Edit∼Review ∋ "Disney"	Mulan

Relax query in IMDB	*results (title)*
keywords ∋ "Chinese" keywords ∋ "Disney"	American Dragon
keywords ∋ "Chinese" plots ∋ "Disney"	Mulan
plots ∋ "Chinese" keywords ∋ "Disney"	Aladdin
plots ∋ "Chinese" plots ∋ "Disney"	*null*
keywords ∋ "Chinese" taglines ∋ "Disney"	*null*
taglines ∋ "Chinese" keywords ∋ "Disney"	*null*

Table 6: Query Relaxation Case Two

query. We can see that the *Title* and *Directors* in Amazon was first extended to the *title* and *directors* in IMDB, as they have the strongest correlations. Then attribute *directors* was relaxed to *writer* and *producer*, and attribute *title* to *akatitle* and *movielinks*. Based on the query relaxation algorithm in Section 4.1, the relaxed queries were executed in the sequence based on their probabilities of getting correct results. We can see that all the relaxed queries retrieved the correct result "Titanic (1997)". Thus, even though if the attributes *title* and *directors* were missing in IMDB, we can still find the movie by using the correlated attributes. As expected, the query relaxation also introduced some incorrect results, such as "Ghosts of the Abyss" and "Last Mysteries of Titanic", which are relevant to "Titanic" and "Cameron" but not the exact answer we wanted.

As another example, we looked for the famous Disney cartoon "Mulan (1998)", which was adapted from a Chinese legend. Assume we had forgotten the title of the movie and only remembered that it is a Disney work about a Chinese story. Thus, we had to search by the *EditorialReview* attribute using keywords "Disney" and "Chinese". The result we got from Amazon was exactly the movie "Mulan (1998)". The process and results of performing the same query on IMDB are shown in Table 6. We can see that the attribute *EditorialReview* in Amazon was relaxed to the corresponding attributes *keywords*, *plots* and *taglines* in IMDB and

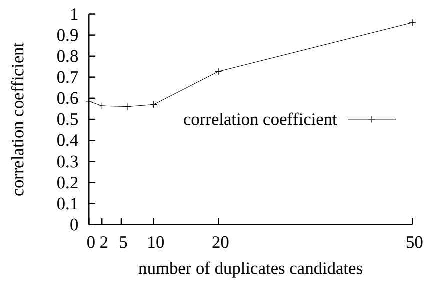

Figure 10: Accuracy in Correlation Quantification

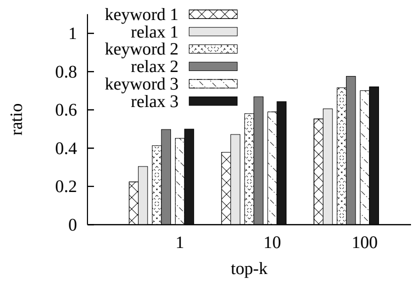

Figure 11: Chance of Finding Intended Item in Top-k Results

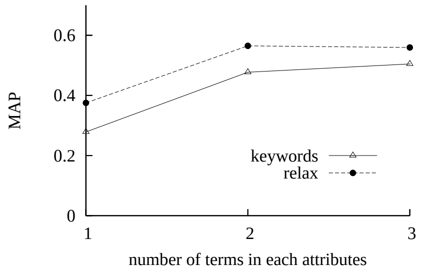

Figure 12: Result Accuracy v.s. Number of Terms

retrieved three results. Among the results, we still found "Mulan (1998)".

To further evaluate the effectiveness of query relaxation, we compared it against keywords search. We selected the most frequently used three attributes in Amazon, namely "Title", "Actors" and "Directors", and created query templates using arbitrary combinations of these attributes. For example, $\{E|\text{Title}\ni x\}$ and $\{E|\text{Actors}\ni x\wedge\text{Director}\ni y\}$ are two query templates. Thus, we ended up with 7 query templates. We used these 7 templates on the Amazon items in the 153 manually evaluated real duplicates to create queries. For each of the created Amazon queries, we conducted it on the IMDB data to see if it can effectively retrieve the corresponding duplicate in IMDB. One way to conduct the query is to use our query relaxation scheme. The other way is to combine all the terms in the Amazon query to form a keywords search query, and conduct it on all the attributes of the IMDB table.

In our experiments, we used the 7 templates and the 153 Amazon items to automatically generate 3000 queries. Each query was created by filling every query attribute with 1 to 3 randomly selected terms from the corresponding attribute of an Amazon item. We use "keyword k" to denote the queries that have k terms in each query attribute and are conducted as keywords search. We use "relax k" to denote the queries being conducted as query relaxation. We compare the results of query relaxation and keywords search in Figure 11 and Figure 12. Figure 11 shows the fractions of queries that can retrieve the intended items in the top 1, top 10 and top-100 results respectively. We can see that query relaxation outperforms keywords search in each type of queries. Figure 12 compares the mean average precisions of query relaxation and keywords search with varying number of terms in each query attribute. It shows that query relaxation clearly outperforms keywords search. We can see from both figures that the precision of query results increases with the number of terms in each query attribute. This is because increasing the number of terms will increase the selectivity of queries, so as to reduce false positive results.

We found that query relaxation does not outperform keywords search in every individual query. The main reason is that query relaxation sometimes returns empty result, as it requires that each term in the query should appear in some attributes. In contrast, keyword search will always return non-empty results, as it is based on cosine similarity. Therefore, we believe that the performance of query relaxation could be further improved by softening its conditions. This will be investigated in our future research.

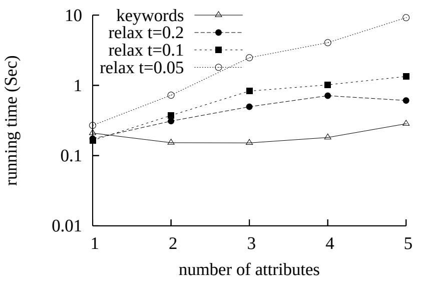

Figure 13: Query Running Time

5.4 Efficiency of Query Relaxation

We did some initial experiments to evaluate the efficiency of our query relaxation scheme. We automatically generated 500 Amazon queries using the Amazon items. Each query contained 1 to 5 attributes. Similar to previous experiments, we performed the 500 Amazon queries on the IMDB data using both query relaxation and keywords search. We limited the number of schema correlations that are used in query relaxation, by setting a threshold to their confidence. The thresholds we used were 0.2, 0.1 and 0.05 respectively. Figure 13 shows the running times of various types of queries. As expected, the running time of query relaxation can be increased by either decreasing the threshold or increasing the attributes in each query, because both will increase the number of relaxed queries to be executed. In particular, the running time grows exponentially with the number of attributes. Nevertheless, the exhibited performance was completely acceptable. With threshold larger than 0.1, query relaxation can already return very satisfactory results, and most queries can be completed within 1 second. When the threshold is larger than 0.05, all queries can be completed in 10 seconds. Moreover, as the relaxed queries are executed progressively, users can see early results in a very short time.

6. RELATED WORK

Extensive research have been done on automatical schema matching [27, 10, 24, 18]. As stated earlier, although their results could help in discovering the correlated elements in a malleable schema, they do not give semantically accurate quantifications of those correlations. To the best of our knowledge, such quantification can only be achieved by studying the co-occurrences of the correlated schema elements, which are embodied by the duplicates in real data. In [5], the authors presented the idea of using duplicates to discover schema matches. However, they did not address methods of using duplicates to quantify the schema matches so that they can be used in query. Moreover, they did not explore how to use discovered schema matches to reinforce duplication detection. Another interesting work is GLUE [11], a system that employs machine learning techniques to find mappings between different ontologies. Similar to our strategy, it uses data instances to statistically assess the correlations between concepts. However, its mechanism only applies to categorical information, and cannot be used to quantify the correlations between attributes and relationships in malleable schemas. In [9], the authors proposed to use content similarity between attributes for automatically detecting foreign key relationships. Their approaches cannot be used to quantify schema correlations, because they are not able to differentiate attributes that are conceptually different but similar in contents.

Duplicate detection in databases [14] is mainly used in data cleaning. The existing techniques usually assume a common schema and attempt to find tuples which are slightly different but refer to the same real world concept. When data schemas are different, they normally require a data transformation step [1] to integrate the data into a common schema before performing detection. In contrast, our approach in this paper takes advantage of the association between duplicates and schema matches to discover both of them simultaneously.

Query relaxation was also an important research topic in cooperative database systems [16, 8, 21], which are designed to automatically relax user queries when the selection criteria are too restrictive to retrieve enough results. Such relaxation is usually based on user preferences and values, and it is seldom applied to schemas. Recently, these techniques have been extended to relax XML queries [2, 22] using the hierarchical structures of XML data. In [4], the authors proposed a framework for relaxing user requests over ontologies. In contrast to those proposals, our work is not concerned about user preferences but the ambiguity in malleable schemas.

Recently there has been a lot of research on approximately querying XML data [15, 7, 3, 25, 23]. These techniques mainly rely on the tree structures (which does not exist in malleable schema) in XML. To relax user queries, they gradually move the selection criteria from bottom of the tree to the root. However, they normally ignore the possible correlations outside of the tree hierarchy. In this paper, we proposed a method to automatically discover the correlations within a data schema and use them for query relaxation.

7. CONCLUSION

In this paper, we presented a query relaxation scheme that enables effective search using malleable schema. We introduced a probabilistic model that allows us to perform query relaxation in a reasonably simple way. We proposed the DSCD algorithm which utilizes the duplicates in data to effectively detect and quantify the correlations within a malleable schema. We discussed the key issues in processing the query relaxation. We conducted extensive experimental study using real dataset to evaluate the performance of our system. Much investigation still needs to be done, in order for malleable schema to be used by real world applications. The future research could be focused on the storage management, the query interface and a flexible mechanism for updating malleable schemas.

8. REFERENCES

[1] Special issue on data transformations. *IEEE Data Eng. Bull.*, 22.

[2] S. Amer-Yahia, S. Cho, and D. Srivastava. Tree pattern relaxation. In *EDBT*, pages 496–513, 2002.

[3] S. Amer-Yahia, L. V. S. Lakshmanan, and S. Pandit. Flexpath: Flexible structure and full-text querying for xml. In *SIGMOD*, pages 83–94, 2004.

[4] W.-T. Balke and M. Wagner. Through different eyes: assessing multiple conceptual views for querying web services. In *WWW*, pages 196–205, 2004.

[5] A. Bilke and F. Naumann. Schema matching using duplicates. In *ICDE*, pages 69–80, 2005.

[6] A. Z. Broder and A. C. Ciccolo. Towards the next generation of enterprise search technology. *IBM Systems Journal*, 43(3):451–454, 2004.

[7] D. Carmel, Y. S. Maarek, M. Mandelbrod, Y. Mass, and A. Soffer. Searching xml documents via xml fragments. In *SIGIR*, pages 151–158, 2003.

[8] W. W. Chu, H. Yang, K. Chiang, M. Minock, G. Chow, and C. Larson. Cobase: A scalable and extensible cooperative information system. *Journal of Intelligent Information Systems*, 6(2/3):223–259, 1996.

[9] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining database structure; or, how to build a data quality browser. In *SIGMOD*, pages 240–251, 2002.

[10] A. Doan, P. Domingos, and A. Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *SIGMOD Conference*, 2001.

[11] A. Doan, J. Madhavan, P. Domingos, and A. Y. Halevy. Learning to map between ontologies on the semantic web. In *WWW*, pages 662–673, 2002.

[12] X. Dong and A. Y. Halevy. Malleable schemas: A preliminary report. In *WebDB*, pages 139–144, 2005.

[13] X. Dong and A. Y. Halevy. A platform for personal information management and integration. In *CIDR*, pages 119–130, 2005.

[14] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *accepted by The IEEE Transations on knowledge and Data Engineering (TKDE)*, Jan 2007.

[15] N. Fuhr and K. Grosjohann. XIRQL: A query language for information retrieval in XML documents. In *Research and Development in Information Retrieval*, pages 172–180, 2001.

[16] P. Godfrey. Minimization in cooperative response to failing database queries. *International Journal of Cooperative Information Systems*, 6(2):95–149, 1997.

[17] A. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10:270V294, 2001.

[18] B. He and K. C.-C. Chang. Statistical schema matching across web query interfaces. In *Sigmod*, 2003.

[19] D. Karger, K. Bakshi, D. Huynh, D. Quan, and V. Sinha. Haystack: A customizable general-purpose information management tool for end users of semistructured. In *CIDR*, 2003.

[20] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[21] N. Koudas, C. Li, A. K. H. Tung, and R. Vernica. Relaxing join and selection queries. In *VLDB*, pages 199–210, 2006.

[22] D. Lee. Query relaxation for xml model. *PhD thesis*, 2002. University of California.

[23] Y. Li, C. Yu, and H. V. Jagadish. Schema free xquery. In *VLDB*, 2004.

[24] J. Madhavan, P. A. Bernstein, A. Doan, and A. Halevy. Corpus-based schema matching. In *ICDE*, pages 57–68, 2005.

[25] N. Polyzotis, M. N. Garofalakis, and Y. E. Ioannidis. Approximate xml query answers. In *SIGMOD Conference*, pages 263–274, 2004.

[26] J. M. Ponte and W. B. Croft. A Language modeling Approach to Information Retrieval. 1998.

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

[28] G. Salton, A. Wong, and C. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.