DogmatiX Tracks down Duplicates in XML

Melanie Weis
mweis@informatik.hu-berlin.de

Felix Naumann
naumann@informatik.hu-berlin.de

Humboldt-Universität zu Berlin
Unter den Linden 6
10099 Berlin, Germany

ABSTRACT

Duplicate detection is the problem of detecting different entries in a data source representing the same real-world entity. While research abounds in the realm of duplicate detection in relational data, there is yet little work for duplicates in other, more complex data models, such as XML. In this paper, we present a generalized framework for duplicate detection, dividing the problem into three components: *candidate definition* defining which objects are to be compared, *duplicate definition* defining when two duplicate candidates are in fact duplicates, and *duplicate detection* specifying how to efficiently find those duplicates.

Using this framework, we propose an XML duplicate detection method, DogmatiX, which compares XML elements based not only on their direct data values, but also on the similarity of their parents, children, structure, etc. We propose heuristics to determine which of these to choose, as well as a similarity measure specifically geared towards the XML data model. An evaluation of our algorithm using several heuristics validates our approach.

1. XML DUPLICATE DETECTION

Duplicate detection is the problem of determining that different representations of entities in a data source actually represent the same real-world entity. The most prominent application area for duplicate detection is customer relationship management (CRM), where multiple entries of the same customer can result in multiple mailings to the same person, incorrect aggregation of sales to a certain customer, etc. Other application areas include bioinformatics, catalog integration, and in general any domain where independently collected data is integrated.

The problem has been addressed extensively for relational data stored in tables. However, more and more of today's data is represented in non-relational form. In particular, XML is increasingly popular, especially for data published on the Web and data exchanged between organizations. Conventional methods do not trivially adapt, so there is a need for methods to detect duplicate objects in nested XML data. XML data is semi-structured and is organized hierarchically. This complicates the object identification task, compared to relational data that is flat and usually well-structured. We face two problems: object definition and structural diversity.

Object definition refers to the problem of defining which data values actually describe an object, i.e., which values to consider when comparing two objects. Methods for relational duplicate detection assume that each tuple represents an object and all attribute values describe that object. Sufficiently similar data values of two tuples imply that they are duplicates. This is not the case in XML: It is not clear whether a child element represents part of the description of an element (as does a relational attribute), or if it represents a related object (as does a relationship with another table). For instance, an XML element `<Artist>` has elements `<AID>`, `<name>`, and some `<CD>` elements as direct children, and every `<CD>` nests `<CDID>` and `<title>`. Among all these nested XML elements, it is not clear how to distinguish actual objects (i.e., artists and CDs) from elements merely describing these objects. In this paper we present a means to explicitly specify which XML elements are objects, which XML elements describe the objects, and we present some heuristics to find them automatically.

Structural diversity, the second problem, addresses the fact that unlike tuples, XML elements describing the same kind of object are not necessarily equally structured. These structural differences are due to either different representations of same objects (e.g., persons may be represented as managers or employees), or differences allowed by the schema, e.g., multiplicities of elements (e.g., persons having no, one, or multiple phone numbers). Whereas different structures due to the first reason may be reconciled through schema matching techniques, the second reason is in the nature of semi-structured data and defeats conventional duplicate detection approaches for relations.

This paper makes three contributions:

1. **Duplicate Detection Framework**: We present a general duplicate detection framework, in which all necessary information for an algorithm can be provided independently of the data model (relational, XML, etc.). It allows for individual definitions to specify what objects to compare, their respective descriptions, and what makes them duplicates. The definitions can be provided offline, and are processed by a duplicate detection algorithm at runtime.

SIGMOD 2005 June 14-16, 2005, Baltimore, Maryland, USA.

2. **DogmatiX algorithm**: We introduce an algorithm, DogmatiX, where Duplicate objects get matched in XML. It specializes our framework and successfully overcomes the problems of object definition and structural diversity inherent to XML.

3. **Domain-independence**: We enable domain-independency for DogmatiX by providing heuristics that choose an object's description and by providing a domain-independent similarity measure for XML data.

The remainder of this paper is organized as follows: In Section 2, we present a generalized framework for object identification. Section 3 introduces the DogmatiX algorithm for XML duplicate detection implemented within the framework. Domain-independent heuristics relevant to the algorithm are presented in Section 4, followed by our domain-independent similarity measure in Section 5. We experimentally evaluate the effectiveness of the algorithm in Section 6 using various real-world and synthetic data sets. Section 7 describes related work, and the paper ends with a conclusion and outlook on future research in Section 8.

2. FRAMEWORK

In this section, we define a general framework for duplicate detection. To form an abstraction from any particular data model, we distinguish objects and elements. Objects are present in the real-world, while elements are present in a certain data model. Different elements and different kinds of elements can all represent the same real-world object. We perform duplicate detection among elements describing the same type of real-world object. Thus, we speak of the general problem of object identification. Our framework is flexible enough to cover a wide range of existing algorithms, and new methods can easily be included. The framework is represented in Fig. 1.

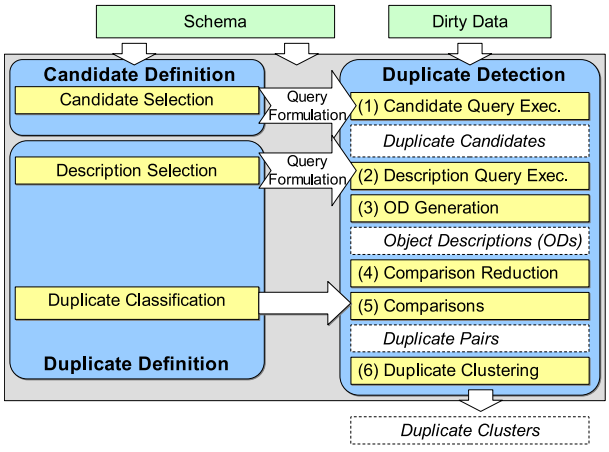

Figure 1: Object Identification Framework

Briefly speaking, the framework consists of three main components.

1. *Candidate Definition*: Defines what objects in the data source should be compared.

2. *Duplicate Definition*: Defines when two duplicate candidates are duplicates.

3. *Duplicate Detection*: Defines how we search duplicates within duplicate candidates.

Components 1 and 2 specify knowledge and definitions that tools and experts can provide offline during system setup. They serve as input to online *duplicate detection*, which works on the actual data.

The duplicate candidates component is used to specify which elements described in the schema should be compared with each other. For the actual *duplicate detection*, the *duplicate candidate selection* is translated into an executable query during *query formulation*. This query, once executed, produces sets of objects called the *duplicate candidates*. Objects within each set are potential duplicates of one another and need to be compared.

Every object is identified by a certain amount of information, i.e., typed data values. This *description* is not necessarily all information about an object in a data source (e.g., all attributes of a relation), nor is it necessarily constrained by the information provided in the data source (some external knowledge may be available). Using the *description*, we associate information with objects. This association is instantiated in the form of so called *object descriptions* (or *OD*s) during duplicate detection, by first formulating and executing the necessary query, and then transforming the result to the *OD* representation.

To us, duplicate detection takes the form of classification of pairs of *OD*s, as defined by *duplicate classification*. E.g, a pair of duplicate candidates is classified as either 'duplicates' or 'non-duplicates'. At runtime, *comparison reduction* reduces the number of pairwise element comparisons for efficiency reasons. Once all pairs of duplicates have been detected through *pairwise comparisons*, the resulting set of *duplicate pairs* can be further processed to obtain a set of *duplicate clusters*, where all objects in a same cluster are considered duplicates.

We now presents the three components in more detail.

2.1 Candidate Definition

The goal of candidate definition is to define which objects are relevant for object identification and thus need to be compared. It is based on three observations. (i) A data source may store information about various types of real-world objects, not all of which need to be considered for duplicate detection. (ii) Among the elements relevant to object identification, some may represent the same type of real-world object, just represented differently. These should be compared with each other. (iii) On the other hand, it makes no sense to compare objects of different real-world type, as they cannot be duplicates of each other. These observations yield the following definition, which formally describes duplicate candidates as a set of objects of same real-world type and can be extracted from the entire data set by selection and projection operations.

Let S be a schema containing schema elements $s_1, \ldots, s_n$. Further, let T be a real-world type describing real-world objects. In many scenarios a real world type is represented by many different schema elements. For instance, the real-world type `motion-picture` can be represented by schema elements `movie` and `film`. We assume a mapping M that associates element types to real-world types.

DEFINITION 1 (DUPLICATE CANDIDATES). *Let* $S^T := \{s_1, ..., s_k\}$ *be the set of schema elements describing the same type* T *of real-world object according to* M*. Further, let* $O_i^T = \{o_{i1}, ..., o_{in}\}$ *be the set of all instances of the schema element* s_i*. Then we define the* duplicate candidates *of type* T *as* $\Omega^T := \bigcup_{1 \leq i \leq k} O_i^T$*.*

Example 1: Consider a relational schema S including relations `Movie`, `Film`, and `Actor`. `Movie` and `Film` were possibly obtained by integrating two data sources and are different representations of the same type of real-world objects. Each relation may still contain duplicates in itself and there may be further duplicates between the two, so all tuples within `Movie` and `Film` should be compared with each other. We arbitrarily label the combined real-world type T `motion-pic`. `Actor` tuples need not be compared with either `Movie` or `Film` tuples but might contain duplicates within itself. These decisions result in $S^{\text{Actor}} = \{\texttt{Actor}\}$ and $S^{\text{motion-pic}} = \{\texttt{Movie}, \texttt{Film}\}$. The candidate instance Ω^{Actor} is the set of all tuples in the `Actor` relation, whereas $\Omega^{\text{motion-pic}}$ includes all tuples in the `Movie` relation, as well as all tuples in the `Film` relation. Note again, that these candidates might have different structure but represent objects of the same real-world type. □

Candidate selection includes the specification of S^T as a set of schema elements. The set of duplicate candidates Ω^T is generated during duplicate detection as discussed in Section 2.3).

2.2 Duplicate Definition

Towards the overall goal of identifying duplicate objects, it is essential to define what characterizes duplicates. We characterize them in the duplicate definition component by (i) their description, and (ii) a classifier for pairs of objects using a similarity measure.

Definition of Descriptions. It is often the case that not all information of an object is useable or useful for object identification. For instance, a `CD` object might be effectively described using information about artist, track list, publisher, etc. However, the textual review of a CD is usually not a useful indicator. In principle, one could choose any data item as part of the description of an object.

DEFINITION 2 (DESCRIPTION). *For every candidate duplicate* $o_i \in \Omega^T$ *we define its* description ID_i *as a set of data instances from the data source. This set is specified by selections and projections in relation to* o_i*.*

In practice, an object's description comprises sibling, child, or parent data, such as attribute values of a tuple, or children of an XML element.

We represent an object's description in a special data structure, called *object description* (OD).

DEFINITION 3 (OBJECT DESCRIPTION). *An object description* OD *is a relation with schema* $OD(value, name)$*. The attribute* value *describes an instance of some information and* name *identifies the type of information by a name.*

The 2-tuples within an OD are referred to as OD tuples. We believe that this OD schema can be used by a wide range of object identification algorithms. Nevertheless, the framework supports an easy replacement of the OD schema.

Example 2: Let us consider the an XML Schema describing the tree structure of Fig. 2. A corresponding XML document contains 3 movies, their data being represented in Tab. 1.

We assume that the description of a `movie` object is its `title`, the `year`, and their actors' `name`. The corresponding OD instances are shown in Tab. 2. □

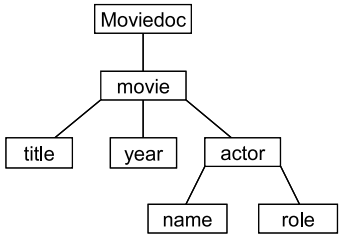

Figure 2: Sample XML Schema

id	**title**	**year**	**actor/name**	**actor/role**
1	The Matrix	1999	Keanu Reeves L.Fishburne	Neo Morpheus
2	Matrix	1999	Keanu Reeves	The One
3	Signs	2002	Mel Gibson	Graham Hess

Table 1: Sample XML data

id	**OD**
1	{(`The Matrix`, title), (`1999`, year), (`Keanu Reeves`, actor/name), (`L. Fishburne`, actor/name)}
2	{(`Matrix`, title), (`1999`, year), (`Keanu Reeves`, actor/name)}
3	{(`Signs`, title), (`2002`, year), (`Mel Gibson`, actor/name)}

Table 2: Examples for object descriptions

Duplicate Classification. Using the ODs of objects, duplicate classification classifies every pair of candidates (o_i, o_j), where both $o_i, o_j \in \Omega_C^T$ into some class $C_i \in \Gamma = \{C_0, C_1, ..., C_n\}$, according to a classifier $\delta(o_i, o_j)$. In our framework, the class C_0 is the class dedicated to pairs of non-duplicates.

In practice, the duplicate detection problem uses two or three classes, i.e., C_1 being "o_i and oj are duplicates", C_0 translating "o_i and o_j are no duplicates", and possibly C_2 that contains all pairs where "o_i and o_j may be duplicates". Typical examples for classification methods are thresholded similarity measures among objects, or rule based decisions. This framework provides a means to add new classifiers to the framework. For every classifier, classes and the corresponding classification methods can be added.

Example 3: Assume a classifier that classifies pairs of candidates (o_i, o_j) into C_1 when o_i and o_j are considered duplicates, and into C_0 otherwise. The classifier considers two candidates to be duplicates if at least half of the OD tuples in OD_i match OD tuples in OD_j, and vice versa. Using this classifier, movie *1* and movie *2* inTab. 2 are considered duplicates because the OD tuples (`1999`, year), and `Keanu Reeves`, actor) occuring in both movies represent half of the OD tuples for movie 1 and 2/3 of the OD tuples for movie 2. On the other hand, movie 3 has no duplicate because it does not share any OD with either movie 1 or movie 2. □

2.3 Duplicate Detection

The duplicate detection component specifies the algorithm that actually performs object identification, using the information provided offline in the candidate definition and duplicate definition components. To prepare the XML data for duplicate detection, it undergoes three transformations, which can be expressed as queries: the first extracts the relevant candidates, the second selects for each candidate the relevant data, and the third flattens their structure to the OD representation. The remaining three steps perform the actual detection of duplicates, based on the OD representation. The different steps the algorithm performs are the subject of this section.

Step 1: Candidate Query Formulation and Execution. The first step of duplicate detection is to obtain duplicate candidates specified by S^T in the candidate definition component. To this end, *query formulation* generates a candidate query Q_C that selects all instances of a schema element $s_i \in S^T$. The result of *candidate query execution* is the set of duplicate candidates Ω^T. The data structure for representing this instance can be any suited data structure.

Step 2: Description Query Formulation and Execution. The description of an object $o_i \in \Omega^T$ can be expressed as a query Q_D in the query language appropriate to the data model at hand. In its current state, our framework automatically derives XQueries from a description specification—the support of other query languages is under development. After execution, the schema of the query result is a subset (projection of schema elements selected by the description definition) of the original schema and still needs to be transformed to the OD-schema (value-attribute-pairs). This is the goal of OD generation.

Step 3: *OD* Generation. Since an OD is nothing other than a relation, it can be obtained by a mapping from the result of the description query formulation to the OD schema $OD(value, name)$. The result of OD generation is a set $\{OD_1, \ldots, OD_n\}$, where OD_i is the OD (the set of OD tuples) that describes object $o_i \in \Omega^T$.

All above steps include the formulation and executions of queries. Indeed, we have three queries, namely the candidate query Q_C, the description query Q_D, and the query of the mapping used for OD Generation. Clearly, they are not independent of each other, and in practice the queries may be combined to increase efficiency.

Step 4: Comparison Reduction. Once the OD instance is generated, all necessary information about duplicate candidates is available, so we could start with comparisons of object pairs. However, for large data sets, the number of pairwise comparisons is computationally prohibitive, so an efficient algorithm should include a method to reduce the number of comparisons. Consequently, the comparison reduction provides a pruning method to efficiently prune pairs.

Definition 4 (Pruning Method). *Let $\phi(o_i, o_j)$ be a pruning method on pairs of candidates o_i and o_j. $\phi(o_i, o_j)$ is a classifier with two classes that signify "pair (o_i, o_j) pruned" and "pair (o_i, o_j) not pruned", respectively.*

Possible methods for reducing the number of pairwise comparisons are filtering and clustering. A filter reduces the set of candidate duplicates by pruning pairs of candidates that provably cannot be duplicates in Ω_C^T. In the case of clustering, candidates that are likely to be duplicates are grouped, and only candidates within a group are compared with each other. Hence, all pairs consisting of candidates in different clusters are pruned.

Step 5: Comparisons. During this step, the actual object identification is performed through pairwise comparisons. Pairs of ODs are compared and classified according to the duplicate classifier δ. In general, the result of the comparison step is a set of object pairs $\{(o_i, o_j) | (o_i \in OD_i) \wedge (o_j \in OD_j)\}$ for each class $C_k \in \Gamma$. In practice, pairs are instantiated only for those classes that are input to further processing. For example, for the three classes C_1 ="duplicates" , C_2 ="possible duplicates", and C_0 ="non-duplicates", only C_1 and C_2 are instantiated. C_1 is required for data cleaning, whereas C_2 is subject to revision by a domain expert.

Step 6: Duplicate Clustering. The relationship "is-duplicate-of" is transitive. For instance, if o_1 is duplicate of o_2, and o_2 duplicate of o_3, then through transitivity o_1 is duplicate of o_3. So, if a class C_k represents pairs of duplicates, the pairs can be combined to duplicate clusters through transitivity.

Once object identification has been performed according to the duplicate detection algorithm, duplicate representations of the same real-world entity are detected and object identification is complete. The resulting *identified data* may be input to many applications, such as data fusion methods or ETL tools. We now specialize this framework for detecting duplicates in XML data.

3. DOGMATIX

As pointed out in Section 1, object identification in XML data bears additional challenges compared to object identification in relational data. The two main problems are that (i) the definition of objects and their description is not clear, and (ii) structural heterogeneity of XML elements representing the same real world type. In this section, we specialize the concepts of our framework to present the DogmatiX algorithm for object identification in XML. This algorithm takes an XML document, its XML Schema S, and a file describing a mapping M of element XPaths to a real world type T as input. The type mapping format is (name of the real-world type, set of schema elements). DogmatiX is rendered domain-independent in its description selection by using specialized heuristics, which we present in Section 4. It is also domain-independent in its duplicate classification using a domain-independent similarity measure presented in Section 5. Thus, the only remaining part of the framework that relies on expert input is the candidate selection. That is, the framework must be fed with the XPaths of objects to identify. In future work, we intend to explore methods to determine candidates automatically, e.g., by searching for primary element types and finding related types through schema matching. Before we start describing DogmatiX in more detail, an introductory example shows the basic idea.

3.1 Example

We input to our algorithm an XML document containing the 3 movies of Tab. 1. It conforms to the schema S depicted in Fig. 2. The mapping M of schema elements in S to real world types is shown in Tab. 3. Note that in this example, every real world type is represented by exactly one element in S, but of course, there can be more. Users interested in detecting duplicates in MOVIE objects simply choose MOVIE from the list of real-world types in M. Then, the candidate selection $S^{MOVIE} = \{\$doc/moviedoc/movie\}$.

For the remaining steps, DogmatiX requires a choice of a heuristic for description selection and the specification of similarity thresholds used by the duplicate classifier. These parameters need to be set manually in the current implementation, but we will explore how to make them self configuring in the future.

Real-world type	element xpaths
MOVIE	{$doc/moviedoc/movie}
TITLE	{$doc/moviedoc/movie/title}
YEAR	{$doc/moviedoc/movie/year}
ACTOR	{$doc/moviedoc/movie/actor}
ACTORNAME	{$doc/moviedoc/movie/actor/name}
ACTORROLE	{$doc/mociedoc/movie/actor/role}

Table 3: Mapping

Once duplicate detection has been performed, DogmatiX outputs the XML document shown in Fig. 3 (using the same description selection and similarity measure as in the examples provided through Section 2). For every cluster of duplicate objects, a `dupcluster` element is generated and identified by a unique object identifier `oid`. The duplicate elements within a cluster are identified by their XPaths.

```
<duplicates>
  <dupcluster>
    <oid>1</oid>
    <xpaths>
     { $doc/moviedoc/movie[1],
       $doc/moviedoc/movie[2] }
    </xpaths>
  </dupcluster>
</duplicates>
```

Figure 3: Description Selection in XML

3.2 Duplicate Definition

Given a definition of candidate duplicates as a selection of real-world types in the mapping M, DogmatiX defines description selection and duplicate classification by specializing the framework's duplicate definition component in a domain-independent way.

Description Selection. DogmatiX uses domain-independent heuristics to define the description query Q_D. These heuristics exploit the XML Schema S, e.g., the tree structure, data types, and element cardinalities as we show in Section 4. They are generally defined as follows.

DEFINITION 5 (HEURISTIC FOR DESCR. SELECTION). *Given an XML Schema S, a heuristic h determines a selection σ_i as description for every element $s_i \in S^T$. Selection σ_i is a set of XPaths in S relative to s_i.*

For instance, in the scenario of Fig. 2, a heuristic returning title and year as description for a movie results in $\sigma = \{./title, ./year\}$.

Duplicate Classification. Pairs of candidates are classified into one of the classes C_0 and C_1 denoting non-duplicates and duplicates, respectively. DogmatiX employs a classifier based on a thresholded approach, using the domain-independent similarity measure described in Section 5.

DEFINITION 6 (XML DUPLICATE CLASSIFIER). *Let $sim(o_i, o_j)$ be a similarity measure for XML objects o_i and o_j, and let θ_{cand} be a threshold value. Then, we classify the pair of objects o_i and o_j into C_0 (non-duplicates) and C_1 (duplicates) using the classifier*

$$\delta(o_i, o_j) = \begin{cases} C_1 & \text{if } sim(o_i, o_j) > \theta_{cand} \\ C_0 & \text{otherwise} \end{cases} \quad (1)$$

3.3 Query Formulation

The XML query formulation component takes as input the set of XPaths σ_i and returns an XQuery the result of which is the description of a candidate duplicate as XML. We are currently developing a graphical tool that automatically composes XQueries consisting of selections and projections, specified by users over the graphical tree representation of an XML Schema S (e.g., through selecting specific tree elements). By automating the projection of schema elements using heuristics, we can readily use the tool's query composition algorithm as query formulation component.

3.4 Duplicate Detection

After the candidate duplicate specification and duplicate definition phases, we perform candidate query formulation and execution (Step 1) and have in hand the objects that are to be compared. These definitions can be provided offline, and are prerequisites to the duplicate detection phase, during which the actual comparisons are performed (it is therefore considered an online phase). The next paragraphs follow the remaining steps of the duplicate detection component in the framework.

Steps 2 and 3: Description Query Execution and OD Generation. Our XML object identification approach starts online duplicate detection with the description query execution. The result of this XQuery is the *description* of each candidate. Next, OD generation transforms each description into the OD representation: For every XML element in the description, an OD tuple of the form $<$ `text`, `xpath` $>$ is generated. Here, `text` denotes the string representation of the text node of the XML element, and `xpath` is the string representation of the absolute XPath of the element in the XML document. Towards efficient computation of the next steps we chose a graph representation to associate ODs and their contained OD tuples.

Step 4: Comparison Reduction. To reduce the number of pairwise comparisons, we use an element filter $f(o_i)$ that is defined as an upper bound of our similarity measure, i.e.,

$$sim(o_i, o_j) \leq f(o_i) \qquad \forall j \quad (2)$$

As a reminder, two elements are considered duplicates if $sim(o_i, o_j) > \theta_{cand}$. If $f(o_i) \leq \theta_{cand}$, it follows from (2) that $sim(o_i, o_j) \leq f(o_i) \leq \theta_{cand}$ for any $o_j \in \Omega_C^T$. Thus, without expensively calculating any similarity for o_i, we can conclude that o_i has no duplicates whatsoever and can remove these duplicate candidates from Ω_C^T. Hence, we filter not only individual pairs of candidates, but entire sets of pairs in a single step, namely all pairs involving o_i. We postpone the definition of the filter $f(o_i)$ to follow the discussion of the similarity measure in Sec. 5, as they are both based on similar distance measures.

Step 5: Pairwise Comparisons. Those duplicate candidates not pruned by the object filter are compared pairwisely, using our XML duplicate classifier and similarity measure (Sec. 5).

Step 6: Duplicate Clustering. When pairs of objects are detected as duplicates, we reflect this in our graph data

structure. Finally, the transitive closure is easily computed and results in the final duplicate clusters.

4. HEURISTICS TO SELECT DESCRIPTIONS

In this section, we present heuristics that conform to Definition 5. As a reminder, we require the heuristic to determine a candidate's description by determining the selection σ_{id} of XPaths domain-independently. We propose two basic heuristics, namely r-distant ancestors and r-distant descendants. These heuristics are further refined by several conditions. We also show how heuristics may be combined.

4.1 Heuristics

The intuition behind the heuristics is that the "farther" information lies from the considered element e_0, the less related it is to e_0, as illustrated in Section 1. E.g, in the XML structure of Fig. 2, `title` and `year` are strong in describing a movie (few movies have the same title and appeared the same year), whereas an actor's `name` is less descriptive of a single movie. The observation can be verified for a wide range of schemas. Therefore, we use proximity to e_0 as heuristic for determining e_0's description. More specifically, we define proximity to e_0 in two ways that exploit the tree structure of the XML tree on both the descendants and ancestor axes.

The first, referred to as r-distant considers as description all elements whose depth in the XML Schema does not differ more than radius r from e_0's depth.

HEURISTIC 1. ***r-distant ancestors*** *h_{ra}: Let a_i be the i-th ancestor of e_0, e.g, a_1 is e_0's parent element, a_2 is its grand-parent, and so on. Further, let $r_a > 0$ be the radius of ancestors to consider. Then, a description of e_0 can be defined by the r-distant ancestors heuristic h_{ra} by $\sigma_{id} = \{xpath(a_i) | 1 \leq i \leq r_a\}$.*

HEURISTIC 2. ***r-distant descendants*** *h_{rd}: Let E_i be the set of e_0's descendants at depth i from e_0, e.g., E_1 are e_0's direct children, E_2 are its grand children, and so on. Further, let $r_d > 0$ be the radius of descendants to consider. Then, a description of e_0 can be defined by the r-distant descendants heuristic h_{rd} by $\sigma_{id} = \{xpath(e_j) \mid e_j \in E_i,\ 1 \leq i \leq r\}$.*

The second way of selecting descriptions, named k-closest, considers the next k elements following e_0 in breadth-first order. Note that we do not consider k-closest ancestors, as it is the same as r-distant when $k = r$.

HEURISTIC 3. ***k-closest descendants*** *h_{kd}: Let e_i be the i-th element in breadth-first order in the subtree rooted at e_0. Further, let k be the number of elements to consider. Then, a description of e_0 can be defined by the k-closest descendants heuristic h_{kd} by $\sigma_{id} = \{xpath(e_i) | 1 \leq i \leq r\}$.*

Figure 4 clarifies the heuristics. In the bottom figure, `movidoc` is selected by the r-distant ancestors heuristic, for $r_a = 1$. All direct children are selected by the r-distant descendants heuristic, when $r_d = 1$. The figure at the top shows the selection for the k-closest heuristic, for $k = 5$.

The advantage of the r-distant descendants heuristic h_{rd} over the k-closest descendants heuristic h_{kd} is that we are sure that all descendants within radius r are considered,

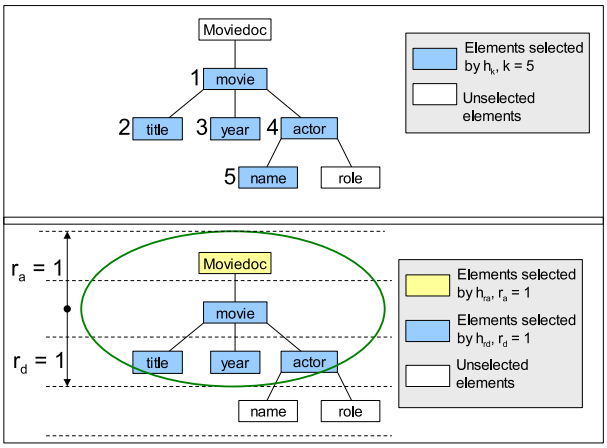

Figure 4: Heuristics k-closest and r-distance

and none is preferred over the other just because of the ordering. The advantage becomes clear if we consider an `xs:any` construct in a schema. There, it is not clear which element appears. Arbitrarily selecting the element defined first is hardly justified. On the other hand, h_{kd} avoids an explosion of elements to consider (in the schema, the number of instances may still explode), unlike h_{rd}, where the size of E_i is not limited.

Clearly, a sensible choice of r_a, r_d, and k is essential for the algorithm's effectiveness. Indeed, for small values of r and k, the resulting ODs do not contain enough information for comparisons. On the other hand, if too many elements are selected, chances are that information not relevant to comparisons is considered. Both cases may affect effectiveness dramatically. The correct choice of r and k is also essential for efficiency, because less selected elements result in less OD tuples to compare.

4.2 Additional Conditions

The XML Schema gives us additional information that can be used to refine the selection of descriptions. In this section, we discuss how we use content models, data types, and cardinality of relationships between elements (1:N, 1:1, optional).

Content Model: An XML element can have one of three content models, namely simple, complex, and mixed. Only simple and mixed content models allow a text node for an element, complex elements can only nest further elements. Without a text node, the resulting OD tuple's value is empty. Consequently, it is not similar to any other OD tuple, however, it should not be considered contradictory as it contains no data. This leads us to Condition 1.

CONDITION 1. ***Content Model:*** *The content model condition c_{cm} signifies that only elements with non-empty text node are considered for the selection of descriptions, i.e., elements of simple or mixed content model.*

Data Type: Due to the fact that the similarity measure uses string similarity, it is not accurate on other data types. Therefore, we introduce the following condition.

CONDITION 2. ***String Data Type:*** *The data type condition c_{sdt} signifies that only elements of string data type are considered for the selection of descriptions.*

Cardinality: If an element e with parent p is declared as mandatory (`minOccurs = 1`, `key`, `ID`, `nillable = false`), this signifies that p cannot exist without e. We see this as a tighter relation between p and e, compared to their relation when e is optional. Therefore, on the descendants axis of p, the description includes elements that are mandatory to p. Furthermore, it is guaranteed to be available for all comparisons. For the ancestor axis, we also consider only those elements that are tightly related to e, i.e. those for which e is mandatory.

CONDITION 3. ***Mandatory Elements**: The data type condition c_{me} signifies that only mandatory elements to e_0 are considered for the selection of e_0's description.*

If maxOccurs of a child equals 1 (which is the default), we have a 1:1 relationship between parent and child. For measuring similarity, such a relationship is more helpful than a 1:N relationship: In a 1:N relationship possible children may be missing, or elements describing the same object may have a different subset of children. In a 1:1 relationship, there is only one possible child, which does not pose the above problems. Additionally, we can be certain that the number of instances does not explode.

CONDITION 4. ***Singleton Elements**: The data type condition c_{se} signifies that only elements with a 1:1 relation with e_0 are considered for the selection of e_0's description.*

Of course, other sensible conditions are conceivable, but we restricted ourselves to the few mentioned above.

4.3 Combining Heuristics and Conditions

The heuristics and conditions presented so far can be combined in multiple ways. In this section, we discuss how they are combined. We have two heuristics, r-distant and k-best, separately applicable to ancestors and descendants. Possible combinations of heuristics are AND and OR.

COMBINATION 1. ***Combination of heuristics**: Let σ_1 and σ_2 denote the selections of heuristics h_1 and h_2, respectively. Then, we define (i) the AND combination of h_1 and h_2 as: $h_1 \wedge_h h_2 = \sigma_1 \cap \sigma_2$, and (ii) the OR combination of h_1 and h_2 as: $h_1 \vee_h h_2 = \sigma_1 \cup \sigma_2$.*

Similarly to the combination of heuristics, conditions can be combined with logical AND and OR operations.

COMBINATION 2. ***Combination of conditions**: Let c_1 and c_2 denote two conditions. Then, we denote (i) the logical AND combination of c_1 and c_2 as $c_1 \wedge_c c_2$, and (ii) the logical OR combination of c_1 and c_2 as $c_1 \vee_c c_2$.*

Last, heuristics and combinations are combined together so that combinations refine the selections performed by heuristics.

COMBINATION 3. ***Combination of heuristics with conditions**: Let h be a heuristic (can be a combination of several heuristics) with $\sigma_h = \{xpath_1, ..., xpath_n\}$, and c a condition (can be a combination of condition). The combination of h and c, denoted $h[c]$, is a new selection σ'_h defined as: $\sigma'_h = \{xpath(e_i) | xpath(e_i) \in \sigma_h \wedge e_i \text{ satisfies } c\}$.*

As an example, we want to consider all direct children of string data type and having text nodes, which corresponds to conditions c_{sdt} and c_{cm}, respectively. Moreover, only mandatory ancestors are considered by applying c_{ma} on the ancestor axis. We denote this combination of heuristics and conditions as: $h_{ra}[c_{ma}] \vee_h h_{rd}[c_{sdt} \wedge_c c_{cm}]$.

The evaluation of different heuristic and condition combinations is part of the Experiments described in Section 6.

5. DOMAIN-INDEPENDENT SIMILARITY

To classify pairs of objects as duplicates, we choose a thresholded similarity measure approach. In this section, we present a new similarity measure for XML duplicate detection specifically geared for our purposes. We consider the following conditions important for XML duplicate detection.

1. OD tuples having different real-world type (according to mapping M) between two objects should not be compared, and are considered as *incomparable data*. For instance, let us consider the OD tuples $OD_1 = \{(\texttt{The Matrix}, \text{title}), (\texttt{great!}, \text{review})\}$ and $OD_2 = \{\texttt{Matrix}, \text{title}), (\texttt{500}, \text{sold-number})\}$ describing movies. Whereas `title`s are comparable, `review` and `sold-number` are not comparable, and they cannot be used to conclude on the similarity of the two movies.

2. Data similarity between comparable elements should be considered in addition to data equality, so that typographical errors are compensated. E.g., the two movie titles `The Matrix` and `Matrix` are similar enough to be considered the same titles.

3. The identifying power of a piece of information should be considered. E.g., the fact that two movies appear in the same year is not as strong an indicator that they are duplicates as if they have the same title.

4. Comparable but contradictory data should reduce similarity, whereas non-specified (missing) data should not. E.g., the fact that two movie XML elements have several different artists is an indicator that they do not represent the same movie in the real-world. On the other hand, the fact that one movie is missing some actors should not be penalized by our similarity measure.

In the remainder of this section, we show how we meet above requirements in our similarity measure, that is defined together with an upper bound, used for filtering in comparison reduction.

5.1 Similarity Measure

In the following, we define our domain-independent similarity measure $sim(o_i, o_j)$ that is used to classify the pair of objects o_i and o_j as duplicates or non-duplicates. The ODs of these objects are denoted OD_i and OD_j, respectively.

Comparable OD Tuples: We do not wish to compare OD tuples that represent different kinds of information, because they cannot contribute to the similarity of two objects. We use the input M to lookup comparable XPaths and thus determine which OD tuples are comparable between OD_i and OD_j. By not considering incomparable data when measuring similarity, we meet the first condition.

Similar OD Tuples: To satisfy condition (2), we have to determine which OD tuples are similar between two ODs. Formally, for every possible pair $(odt_i, odt_j) \in OD_i \times OD_j$ of OD tuples, we determine a distance measure.

DEFINITION 7 (OD TUPLE DISTANCE.). *Let* $odt_i = (v_i, n_i) \in OD_i$, $odt_j = (v_j, n_j) \in OD_j$. *Further, let* $ned(s_i, s_j)$ *be the edit distance between two strings* s_i *and* s_j *normalized by the maximum of the two strings' length. Then,*

$$odtDist(odt_i, odt_j) = \begin{cases} 1 & \text{if } n_i \text{ and } n_j \\ & \text{are not comparable} \\ ned(v_i, v_j) & \text{otherwise} \end{cases} \quad (3)$$

Two OD tuples are considered similar when their *odtDist* is below a given threshold θ_{tuple}. The set of all similar OD tuples between two ODs is

$$ODT^T_{\approx}(OD_i, OD_j) = \{(odt_i, odt_j) \mid odtDist(odt_i, odt_j) < \theta_{tuple}\} \quad (4)$$

Note that we need to compare all OD tuples of OD_i with all comparable OD tuples of $OD_j, j \neq i$ by calculating their edit distance. Edit distance is a very expensive operation and needs to be avoided when possible. In [18], we introduced a simple combination of upper and lower edit distance bounds to substantially reduce the number of pairwise comparisons.

Data Relevance: In our similarity measure, we weigh the relevance of terms (condition (3)) using a variation of the inverse document frequency (IDF) [2], called $softIDF$. Generally, if D is the complete set of objects in a document, and n the number of objects in which a term k occurs, then we define the IDF of k as $IDF = \log(\frac{|D|}{n})$. In our context, $D = \Omega^T$, and a term k is an OD tuple t. To account for the fact that we do not only consider exact matches of information between two ODs, but also similar matches, we extend the definition of IDF to consider pairs (odt_1, odt_2) of similar terms. This is achieved by defining n as the number of objects in which odt_1 or odt_2 occurs.

DEFINITION 8. $softIDF$ *and* $setSoftIDF$*: Let* (odt_i, odt_j) *be a pair of OD tuples, such that* $odtDist(odt_i, odt_j) < \theta_{tuple}$. *With* $O_{odt_i} = \{OD_i \mid odt_i \in OD_i\}$ *and* $O_{odt_j} = \{OD_j \mid odt_j \in OD_j\}$, *the* $softIDF$ *of* (odt_i, odt_j) *is defined as*

$$softIDF((odt_i, odt_j)) := \log(\frac{\mid \Omega^T \mid}{\mid O_{odt_i} \cup O_{odt_j} \mid}) \quad (5)$$

The $softIDF$ *of a set* $S_{tuple} \subseteq S$ *is*

$$setSoftIDF(S_{tuple}) := \sum_{(odt_i, odt_j) \in S_{tuple}} softIDF(odt_i, odt_j) \quad (6)$$

Non-specified vs. contradictory OD Tuples: Our goal is to distinguish non-specified and contradictory data (Condition (4)) in the sense that the former does not negatively influence the similarity of two objects, whereas the latter reduces the similarity. E.g., the fact that two Movie XML elements have several different actors is an indicator that they do not represent the same movie in the real-world. On the other hand, the fact that one Movie is missing some actors should not be penalized by our similarity measure. We say an OD tuple $odt_i \in OD_i$ is contradictory to an OD tuple $odt_j \in OD_j$ if they (i) represent the same kind of object according to mapping M, but (ii) are not similar according to $odtDist(odt_i, odt_j)$, and (iii) are not considered contradictory to other OD tuples. Whereas the reasons for the first two conditions are intuitive, the third needs clarification, provided by the following example. Consider two countries $\texttt{country}_i$ and $\texttt{country}_j$ that respectively nest the cities (`New York`, `Los Angeles`, `Miami`) and (`Miami`, `Boston`). Clearly, they share one city, namely `Miami`. On the other hand, they differ in three cities. However, the contradictory data at most includes one city, because the lists of cities are not exhaustive. `Boston` is contradictory to either `Los Angeles` or `New York`, but not both. Either `Los Angeles` or `New York` may have been not-specified in the list of cities.

Formally, due to the possibly different cardinalities of XML elements reflected in OD_i and OD_j, not necessarily all OD tuples are contradictory to another, and the remaining OD tuples are considered non-specified data. Pairs of contradictory OD tuples are selected according to highest *odtDist*. E.g., for the possible contradictory pairs (Boston, Los Angeles) and (Boston, New York) with respective *odtDist* $8/11 \approx 0.72$ and $7/8 = 0.875$ we choose the latter because it has highest distance. All pairs of contradictory OD tuples between OD_i and OD_j are summarized in the set

$$ODT^T_{\neq}(OD_i, OD_j) = \{(odt_i, odt_j) | odt_i \text{ and } odt_j \text{ are contradictory}\} \quad (7)$$

Similarity Measure: Using the previous definitions and considerations, we define our similarity measure $sim(OD_i, OD_j)$.

$$sim = \frac{setSoftIDF(ODT^T_{\approx})}{setSoftIDF(ODT^T_{\neq}) + setSoftIDF(ODT^T_{\approx})} \quad (8)$$

We omitted the parameters (OD_i, OD_j) for sim, $ODT^T_{\approx}$, and $ODT^T_{\neq}$ for clarity. Intuitively, sim measures the relevance of similar data between two objects, relative to the relevance of their difference.

Note that our similarity measure considers both reasons of structural heterogeneity and leverages their impact on similarity. The first one, different schemas, is solved by considering the mapping of schema elements to real world types, and the second is captured by the fact that we distinguish unspecified vs. contradictory data. In future work, we will explore how to adapt tree edit distance to consider these issues as well, so that we can use it as similarity measure for duplicate detection.

5.2 Object Filter

As announced in Section 3, we now define the object filter used for comparison reduction. It is an upper bound to the similarity measure $sim(OD_i, OD_j)$ and is defined as follows:

$$S_{shared} = \bigcup_{i \neq j} ODT^T_{\approx}(OD_i, OD_j),$$

$$S_{unique} = \bigcap_{i \neq j} ODT^T_{\neq}(OD_i, OD_j)$$

$$f(OD_i) = \frac{setSoftIDF(S_{shared})}{setSoftIDF(S_{unique}) + setSoftIDF(S_{shared})} \quad (9)$$

Intuitively, $f(OD_i)$ measures the amount of information OD_i shares with any other OD_j, compared to the amount of information unique to OD_i. The cost of computing $f()$ is comparable to the cost of calculating $sim()$. However, $f()$ only needs to be calculated once for every element, whereas

$sim()$ has to be computed for every pair of elements. Therefore, $f()$ is a suitable filter for reducing the number of pairwise element comparisons. Experiments in Section 6 show its effectiveness.

6. EXPERIMENTS

We evaluate our domain-independent approach for object identification under two aspects, namely (i) the effectiveness of our similarity measure, depending on heuristics, and (ii) the effectiveness of our object filter. We considered two typical scenarios in which duplicates occur: In the first scenario, we search for duplicates in a single XML document. There, errors are mainly due to typos and missing data. Because the XML Schema and XML data have been created together, it is unlikely that duplicates have significantly different structure or that the data is represented in considerably different ways. This is not the case in our second scenario, where we consider two data sources that are to be merged in the context of data integration. There, we search for duplicates among different sources that are differently structured and that represent data differently. There, duplicates are due to synonyms and contradictory data, in addition to the usual typos and missing data.

We expect our algorithm to be very effective in the first scenario: Edit distance should compensate typos, and our similarity measure is specifically designed to identify duplicates despite missing data. On the other hand, synonyms, although having the same meaning, are recognized as contradictory data and the similarity decreases. They are more difficult to detect without additional knowledge, such as a thesaurus or a dictionary. Thus, we expect the second scenario to yield poorer results.

In the remainder of this section, we present the different data sets and setups for our experiments. Then, we evaluate the effectiveness of the similarity measure and finally evaluate the object filter.

6.1 Data Sets and Setup

We use three different data sets.

- Dataset 1: 500 non-duplicate CD objects extracted from the FreeDB dataset[1] + 500 artificially generated duplicates (1 for each CD).
- Dataset 2: 500 non-duplicate Movies extracted from IMDB [2] + the same 500 movies from Film-Dienst[3].
- Dataset 3: 10,000 CDs randomly extracted from FreeDB.

The 500 artificially duplicated CD objects in Dataset 1 were generated automatically with an XML Dirty Data Generator[4]. The parameters (i) percentage of duplicates, (ii) percentage of typographical errors, (iii) percentage of missing data, and (iv) percentage of synonymous (but contradictory) data were set to 100%, 20%, 10%, and 8%, respectively. Hence, Dataset 1 represents the scenario where mostly uniformly structured objects are duplicated by typos and missing data. On the other hand, Dataset 2 represents our second scenario, where duplicate objects have very different structures and representations as they come from two distinct data sources. Hence, the difference between comparable data of duplicates is mainly due to synonyms and contradictory information. For example, data from IMDB is in English, whereas data from FILMDIENST is in German, and date formats are different. Note that we did not apply any data scrubbing before performing experiments. Dataset 3 is used to show the effectiveness of our algorithm on a larger amount of real-world data.

[1] http://www.freedb.de/
[2] http://www.imdb.com
[3] http://film-dienst.kim-info.de/
[4] http://www.informatik.hu-berlin.de/mac/dirtyxml/

Experiment	Heuristic
exp1	h
exp2	$h[c_{sdt}]$
exp3	$h[c_{me}]$
exp4	$h[c_{se}]$
exp5	$h[c_{sdt} \wedge c_{me}]$
exp6	$h[c_{sdt} \wedge c_{se}]$
exp7	$h[c_{me} \wedge c_{se}]$
exp8	$h[c_{sdt} \wedge c_{se} \wedge c_{me}]$

Table 4: Combinations of conditions

r	k	Elements in OD (object description)
1	1	disc/did (string, ME, SE)
	2	disc/artist (string, ME, not SE)
	3	disc/title (string, ME, not SE)
	4	disc/genre (string, not ME, SE)
	5	disc/year (date, ME, SE)
	6	disc/cdextra (string, not ME, not SE)
	7	disc/tracks (complex, ME, SE)
2	8	disc/tracks/title (string, ME, not SE)

Table 5: Elements in Dataset 1

6.2 Effectiveness of Similarity Measure

We evaluate the effectiveness of our approach using the combinations of heuristics and conditions shown in Tab. 4, where h signifies one of the three heuristics specified individually in the experiments.

The first series of experiments uses Dataset 1 (errors mainly due to typographical errors and missing data). In Tab. 5 we see which XML Schema elements become part of an object's description for increasing r and k of h_{rd} and h_k, respectively. Next to the elements' XPaths we see their data type, whether they are mandatory elements (ME) or not, and whether they are singleton elements (SE) or not. We apply exp1 to exp8 using h_k as heuristic, varying k from 1 to 8, with $\theta_{tuple} = 0.15$ and $\theta_{cand} = 0.55$. Note that the experiments for $k = 7$ and $k = 8$ are the same as if we applied the r-distance heuristic for $r = 1$ and $r = 2$, respectively. We evaluate the effectiveness by both recall and precision. The results are summarized in Fig. 5.

We see that Experiments 1,2,3, and 5 form one group. Their recall and precision curves have similar behavior for $r = 1$ ($k \in [1, 7]$). That is, we observe an increase between $k = 1$ and $k = 3$ and stability for values $3 \leq k \leq 7$, which can be explained as follows: For $k = 1$, we only have the disc-id (`did`) available for comparisons. These IDs apparently have been generated automatically, and most IDs do not differ by more than one character. Hence, they are falsely recognized as similar according to $odtDist()$, which explains the relatively low precision. Recall is high because all duplicate pairs have a similar `did`, except for those whose `did` was affected by introduced errors. Recall and precision increase from $k = 1$ to $k = 3$ because the added elements (`artist` and `title`) have high distinguishing power (according to

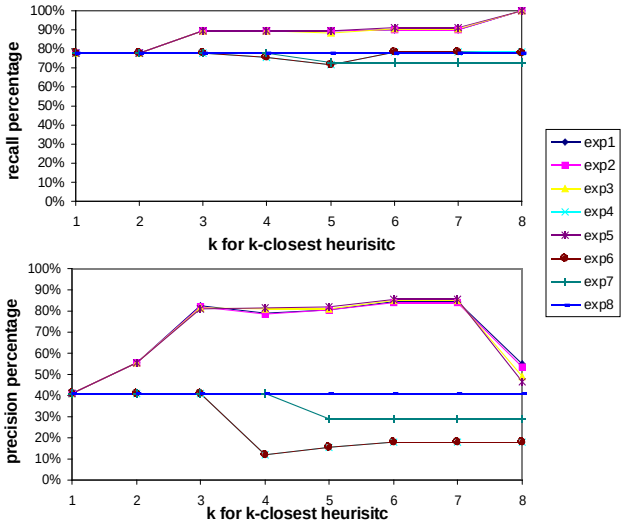

Figure 5: Effectiveness Evaluation on Dataset 1

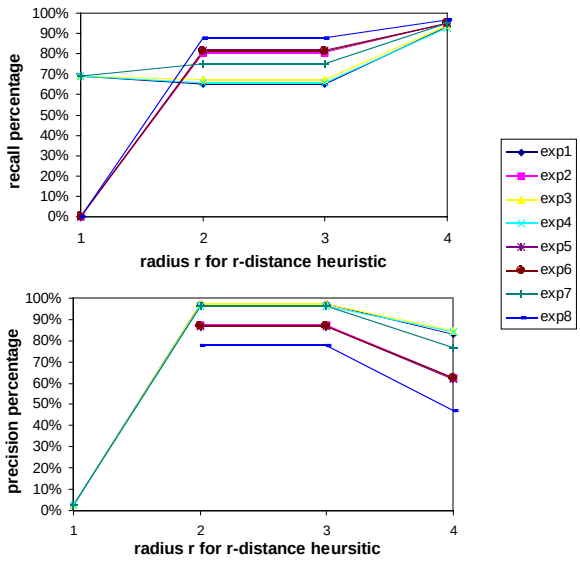

Figure 6: Effectiveness Evaluation on Dataset 2

IDF); and the more such information is available, the less impact errors have on similarity. The elements added with even higher k either have low distinguishing power (`genre`, `year`, `cd-extra`) or are not considered because they do not have a text node (`tracks`). Consequently, recall and precision are stable for $3 \leq k \leq 7$. By adding track-titles, recall increases and all duplicates are found. However, precision drastically drops: All duplicates are found because track titles include a lot of information, and errors no longer have an impact. However, due to dummy titles ("Track 1") for non-specified titles in approximately 20% of all CDs, the similarity of non-duplicates increases. The behavior of the remaining experiments can be explained as follows: exp8 only considers `did` for any k, hence, recall and precision are constant. Exp7 drops in both recall and precision at $k = 5$ because `year`, with small distinguishing power is added. For the same reason, exp4 and exp6 drop at $k = 3$, where `genre` is added.

From this series of experiments, we see that small descriptions are too sensitive to errors, and hence do not yield high precision and recall values. On the other hand, if too much data is added to the description, non-duplicates become more similar, because it is more likely that information matches. We further observe that the larger k and especially r, the smaller is the identifying power of added data. This validates the proximity assumption underlying these heuristics and emphasizes that good heuristics are valuable.

We make similar observations for the second scenario when duplicates are mainly due to structural differences (Dataset 2). We apply h_{rd} with the eight conditions of Table 4, $\theta_{tuple} = 0.15$, and $\theta_{cand} = 0.55$. Comparable elements between the two different data sources available for different values of r are summarized in Table 6. Recall and precision are represented for the r-distant heuristic in Fig. 6.

We observe that the similarity measure performs well and its effectiveness is highest when neither too few nor too much information is selected. It is the task of heuristics to select that information, and it is not clear which heuristic to choose. Due to the small size of the schemas of Dataset 1 and Dataset 2, we cannot give generally applicable advice for choosing heuristics.

On Dataset 3, we are only able to measure precision of the result, because we did not (yet) pairwisely compare the 10.000 elements by hand. For exp1 (heuristic h_k with $k = 6$) we found 252 pairs of duplicates, from which 27 pairs were exact duplicates. Figure 7 shows the precision of the result, for θ_{cand} between 0.55 and 1. We observe that precision increases with increasing θ_{cand}, hence, trivially the similarity measure is proportional to the similarity of the elements. At $\theta_{cand} = 0.85$ precision reaches 100%. At this point 36 duplicates are detected.

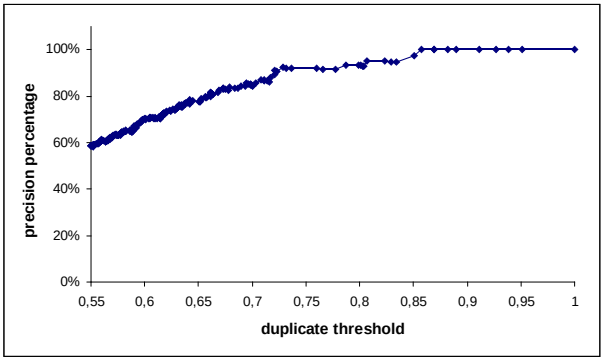

Figure 7: Effectiveness on Dataset 3

6.3 Effectiveness of the Object Filter

To evaluate the effectiveness of the object filter f defined in Section 5.2, we use the original 500 CDs from Dataset 1 and vary the percentage of artificially generated duplicates from 0% to 90%. For instance, at 0% duplicates, the data consists of 500 non-duplicates, at 50% duplicates, we have generated 250 duplicates, so we have 250 duplicate pairs and 250 singletons. In Fig. 8, we plot the recall and precision of the filter for exp1 and $k = 6$. Recall is measured as the number of correctly pruned candidates divided by the number of

r	Elements in OD (object description) from IMDB	Elements in OD (object description) from FILMDIENST
1	movie/year(date, ME, not SE)	movie/year(date, ME, SE)
2	movie/title (string, ME, SE)	movie/movie-title/title (string, ME, SE)
	movie/aka-title/title (string, optional, not singleton)	
	movie/genre(string, not ME, not SE)	movie/genres/genre(string, not ME, not SE)
	movie/release-date/date(date, ME, SE)	movie/premiere(date, not ME, SE)
3	-	-
4	movie/people/actors/actor/name (string, ME, SE)	movie/people/person/firstname + lastname (string, ME, SE)
	movie/people/actresses/actress/name (string, ME, SE)	
	movie/people/producers/producer/name (string, ME, SE)	

Table 6: Elements in Dataset 2

non-duplicate candidates in the XML documents. Precision is defined as the number of correctly pruned candidates divided by the total number of pruned candidates. As we see, both the filter's recall and precision are high (above 70%) for any percentage of duplicates. So we can conclude that the filter is effective in pruning candidates that do not have a duplicate.

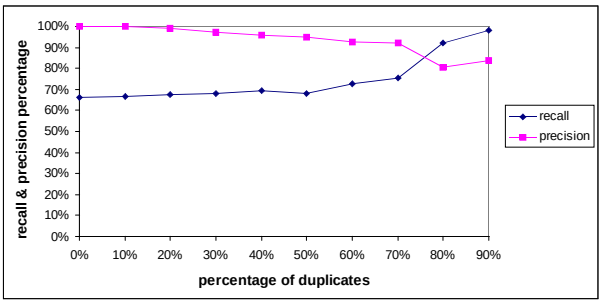

Figure 8: Effectiveness of Object Filter

7. RELATED WORK

Object identification has received much attention in the relational world, and first efforts exist in identifying duplicate objects in hierarchical and XML data. In this section, we give a brief overview of past research in that area.

7.1 Object Identification for Relational Data

Object identification has received much attention in the database and other communities [5, 15, 16]. In particular, research has concentrated on efficiently and effectively finding duplicate records in relational data. Research in relational duplicate detection can be classified into two areas: domain-dependent and domain-independent. The first assumes a certain domain, such as the domain of address-data, and the help of human experts to calibrate algorithms. For instance, in the sorted neighborhood method duplicate records within a table are detected by first sorting the records according to a user-specified key [7]. For XML data, even defining the sorting key by hand is not at all straightforward. Examples where domain-dependent solutions have been applied include census data sets [19], medical data [8], and genealogical data [14].

In [12], the authors describe a domain-independent and more efficient version of the sorted neighborhood method. While the same problem of finding a sorting key remains, we have adopted a graph structure similar to the one suggested in [12] and plan to include the notion of prime representatives in future work. Other domain-independent solutions include [9, 11]. Other approaches include learning similarity measures [3, 17].

The problem of object identification in relational data has also been positioned in a framework in [13]. In this framework, object identification is divided into three steps, namely (i) conversion to obtain comparable and descriptive information about objects, (ii) comparison of objects, and (iii) classification of pairs as matches (duplicates), non-matches, and matches to some degree. These steps can be found in our framework as well. However, we set an additional focus on *candidate definition* and the distinction between definitions independent of the kind of data source, and the algorithm. There are also data cleaning frameworks, such as AJAX [5], that include object identification. AJAX separates the logical and physical levels of the process that separate quality and performance of a data cleansing system. This is similar to our separation of offline definitions and online algorithm. However, the focus of AJAX is data cleansing, so the details of object identification are not described in detail. In a sense, we consider our framework a zoom-in of their matching operator on both logical and physical level.

7.2 Object Identification for Hierarchical and XML Data

To the best of our knowledge, the closest approach for detecting duplicates in hierarchical data is the DELPHI project, from which our work is inspired [1]. DELPHI identifies duplicates in hierarchically organized tables of a data warehouse. Duplicates in the outer-most dimension are discovered first and help duplicate detection in the children table and finally in the fact table. Therein, the authors focus on a single branch of the hierarchy, and do not consider the cases where one table may have several children tables. Our approach and DELPHI further differ in the similarity measure. In DELPHI, the authors choose a non-symmetrical measure (i.e., 'A is duplicate of B' does not imply that 'B is duplicate of A'), which determines the degree of *containment* of one element within another. As a consequence, the *difference* of the two elements is not reflected in the result. Our similarity measure, is both symmetrical and takes into account not only similarity but also differences of the compared elements.

There is other work on identifying similar XML data. However, most do not consider the accuracy (e.g., in terms of recall and precision) of their similarity join. Rather, the authors concentrate on fast execution of the algorithm: In [6], the focus is on the efficient incorporation of tree edit distance in a framework performing approximate XML joins. The authors present upper and lower bounds for the tree edit distance, which are used as filters to avoid expensive tree edit distance computations. They further introduce a

sampling method to effectively reduce the amount of data examined during the join operation. Another solution has been proposed in [10]. They present various filtering techniques for structural and for content-based information in tree-structured data. The filters are integrated in a so-called filter-refinement architecture. Its goal is to reduce the number of complex and time consuming distance calculations in the query process. They also focus on efficiency and effectiveness of their filtering techniques, but not on the effectiveness of the actual duplicate detection. The only approach we are aware of that considers recall and precision of their XML similarity joins is [4]. They present four different strategies to define the similarity function using the vector space model. An experimental comparison of our work with their results is under way.

8. CONCLUSION AND OUTLOOK

In this paper, we presented a generalized framework for object identification and an XML-specific specialization. The framework consist of three main components: (i) the candidate definition that specifies what objects to compare, (ii) the duplicate definition defining what information is part of a candidate's description and when two candidates are duplicates. These definitions are processed by (iii) the duplicate detection component. Duplicate detection is divided into six steps. The first three prepare the data for comparisons, whereas the remaining three perform the actual duplicate detection by pruning, comparing pairs of candidates, and clustering pairs of duplicates.

Further, we presented an algorithm specifically geared towards solving the problems of object definition and structural heterogeneity inherent to XML data. The DogmatiX algorithm for XML duplicate detection uses heuristics to determine candidate descriptions domain-independently. Furthermore, it uses a domain-independent similarity measure tailored to hierarchical and semi-structured XML data.

Through experimental evaluation of the effectiveness of the similarity measure, we have seen that it yields good results in terms of recall and precision when adequate heuristics are chosen. But clearly the choice of the best heuristic is not trivial, and future investigation will include automating the choice of a good heuristic by exploiting the XML Schema and statistics about the data. Additional experiments in this paper consider the effectiveness of the object filter, used to reduce the number of pairwise candidate comparisons. They showed that the candidate filter is effective in pruning candidates that are not duplicates of any other candidate.

We intend to further validate our similarity measure by comparing its effectiveness to other similarity measures when applied to XML. Preliminary experiments have shown that our similarity measure performs better than other approaches for data from heterogeneous data sources, but we still need to validate this observation. Furthermore, we intend to explore the automation of the selection of candidates, so that no domain-knowledge whatsoever is required. Another very important aspect in object identification only marginally considered yet, is the efficiency of the approach.

Acknowledgment. This research was supported by the German Research Society (DFG grant no. NA 432).

9. REFERENCES

[1] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *International Conference on Very Large Databases*, Hong Kong, China, 2002.

[2] R. A. Baeza-Yates and B. A. Ribeiro-Neto. Modern information retrieval. *ACM Press / Addison-Wesley*, 1999.

[3] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *International Conference on Knowledge Discovery and Data Mining*, Washington, DC, 2003.

[4] J. C. Carvalho and A. S. da Silva. Finding similar identities among objects from multiple web sources. In *CIKM-2003 Workshop on Web Information and Data Management*, pages 90–93, New Orleans, Louisiana, USA, 2003.

[5] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C. Saita. Declarative data cleaning: Language, model, and algorithms. In *International Conference on Very Large Databases*, pages 371–380, Rome, Italy, 2001.

[6] S. Guha, H. V. Jagadish, N. Koudas, D. Srivastava, and T. Yu. Approximate XML joins. In *International Conference on Management of Data*, pages 287–298, Madison, Wisconsin, USA, 2002.

[7] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *International Conference on Management of Data*, pages 127–138, San Jose, CA, May 1995.

[8] M. A. Jaro. Probabilistic linkage of large public health data files. *Statistics in Medicine*, 14(5–7):491–498, 1995.

[9] L. Jin, C. Li, and S. Mehrotra. Efficient record linkage in large data sets. In *International Conference on Database Systems for Advanced Applications*, Kyoto, Japan, 2003.

[10] K. Kailing, H.-P. Kriegel, S. Schnauer, and T. Seidel. Efficient similarity search for hierarchical data in large databases. In *International Conference on Extending Database Technology*, pages 676–693, Heraclion, Crete, 2004.

[11] E.-P. Lim, J. Srivastava, S. Prabhakar, and J. Richardson. Entity identification in database integration. In *International Conference on Data Engineering*, pages 294–301, April 1993.

[12] A. E. Monge and C. P. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *SIGMOD-1997 Workshop on Research Issues on Data Mining and Knowledge Discovery*, pages 23–29, Tuscon, AZ, May 1997.

[13] M. Neiling and S. Jurk. The object identification framework. In *KDD03 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, Washington DC, 2003.

[14] D. Quass and P. Starkey. Record linkage for genealogical databases. In *KDD-2003 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, pages 40–42, Washington, DC, 2003.

[15] E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin, Volume 23, pages 3-13*, 2000.

[16] V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *International Conference on Very Large Databases*, pages 381–390, Rome, Italy, 2001.

[17] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *International Conference on Knowledge Discovery and Data Mining*, Edmonton, Alberta, 2002.

[18] M. Weis and F. Naumann. Duplicate detection in XML. In *SIGMOD-2004 Workshop on Information Quality in Information Systems*, pages 10–19, Paris, France, 2004.

[19] W. E. Winkler. Advanced methods for record linkage. Technical report, Statistical Research Division, U.S. Census Bureau, Washington, DC, 1994.