Efficient and Effective Duplicate Detection in Hierarchical Data

Luís Leitão, Pável Calado, Melanie Herschel

Efficient and Effective Duplicate Detection in Hierarchical Data

Luís Leitão, Pável Calado, Melanie Herschel

Abstract—Although there is a long line of work on identifying duplicates in relational data, only a few solutions focus on duplicate detection in more complex hierarchical structures, like XML data. In this paper, we present a novel method for XML duplicate detection, called XMLDup. XMLDup uses a Bayesian network to determine the probability of two XML elements being duplicates, considering not only the information within the elements, but also the way that information is structured. In addition, to improve the efficiency of the network evaluation, a novel pruning strategy, capable of significant gains over the unoptimized version of the algorithm, is presented. Through experiments, we show that our algorithm is able to achieve high precision and recall scores in several datasets. XMLDup is also able to outperform another state of the art duplicate detection solution, both in terms of efficiency and of effectiveness.

Index Terms—duplicate detection, record linkage, entity resolution, XML, Bayesian networks, data cleaning, optimization

1 Introduction

Electronic data plays a central role in numerous business processes, applications, and decisions. As a consequence, assuring its quality is essential. Data quality, however, can be compromised by many different types of errors, which can have various origins [1]. In this paper, we focus on a specific type of error, namely *fuzzy duplicates*, or *duplicates* for short. Duplicates are multiple representations of the same real-world object (e.g., a person) that differ from each other because, for example, one representation stores an outdated address.

What makes duplicate detection a non-trivial task is the fact that duplicates are not *exactly equal*, often due to errors in the data. Consequently, we cannot use common comparison algorithms that detect exact duplicates. Instead, we have to compare all object representations, using a possibly complex matching strategy, to decide if they refer to the same real-world object or not.

Due to its highly practical relevance in data cleaning and data integration scenarios, duplicate detection has been studied extensively for relational data stored in a single table [2]. In this case, the detection strategy typically consists in comparing pairs of tuples (each tuple representing an object) by computing a similarity score based on their attribute values. Then, two tuples are classified as duplicates if their similarity is above a predefined threshold. However, this narrow view often neglects other available related information as, for instance, the fact that data stored in a relational table relates to data in other tables through foreign keys. The opportunity of considering such relations during pairwise comparisons has recently been realized and new algorithms have been proposed [3], [4]. Among these, several focus on the special case of detecting duplicates in hierarchical and semi-structured data, most notably, on *XML data* [5], [6], [7], [8].

Methods devised for duplicate detection in a single relation do not directly apply to XML data, due to the differences between the two data models [5]. For example, instances of a same object type may have a different structure at the instance level, whereas tuples within relations always have the same structure. But, more importantly, the hierarchical relationships in XML provide useful additional information that helps improve both the runtime and the quality of duplicate detection. We illustrate this fact based on the following example that we will use throughout the paper.

Consider the two XML elements depicted as trees in Figure 1. Both represent person objects and are labeled

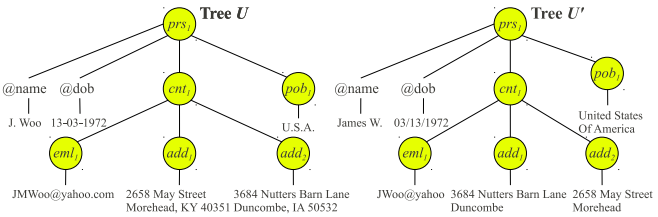


Fig. 1. Two XML elements that represent the same person. Nodes are labeled by their XML tag name and an index for future reference.

prs. These elements have two attributes, namely the date of birth (dob) and $name$. They nest further XML elements representing place of birth (pob) and contacts (cnt). A contact consists of several addresses (add) and an email (eml), represented as children XML elements of cnt. Leaf elements have a text node which stores the actual data. For instance, dob has a text node containing the string "13-03-1972" as its value.

In this example, the goal of duplicate detection is

- *Luís Leitão and Pável Calado are with IST/INESC-ID, Portugal. E-mails: luis.leitao@ist.utl.pt and pavel.calado@ist.utl.pt*
- *Melanie Herschel is with the University of Tübingen, Germany. E-mail: melanie.herschel@uni-tuebingen.de*

to detect that both persons are duplicates, despite the differences in the data. To do this, we can compare the corresponding leaf node values of both objects. In this work, we propose that the hierarchical organization of XML data helps in detecting duplicate prs elements, since descendant elements (e.g., eml or add) can be detected to be similar, which increases the similarity of the ancestors, and so on in a top-down fashion.

Contributions. In this paper, we first present a probabilistic duplicate detection algorithm for hierarchical data called XMLDup. This algorithm considers both the similarity of attribute contents and the relative importance of descendant elements, with respect to the overall similarity score. The algorithm presented here extends our previous work [6], [9] by (i) significantly improving efficiency and (ii) showing a more extensive set of experiments.

Our contributions, especially compared to our previous work, can be summarized as follows: (i) we address the issue of efficiency of our initial solution [6] by introducing a novel pruning algorithm and studying how the order in which nodes are processed affects runtime. A major result is that XMLDup now outperforms DogmatiX [5], a previously more efficient state of the art algorithm for XML duplicate detection [9]; (ii) we describe how to increase efficiency when a slight drop in recall, i.e., in the number of identified duplicates, is acceptable. This process can be manually tuned or performed automatically, using known duplicate objects from other databases; and (iii) we provide a more extensive evaluation of our algorithms than in our previous work. More specifically, we demonstrate the effectiveness of our algorithm on a larger number of data sets, from different domains than those used in [6], [9]. Also, we extensively evaluate efficiency.

Unlike previous works, such as [10], where the goal is to reduce the number of pairwise comparisons performed, here we are concerned with how to efficiently perform each pairwise comparison. In fact, our approach could be combined with such solutions to achieve an even more effective result.

Structure. This paper is organized as follows. Section 2 presents related work. Section 3 summarizes our baseline algorithm, first presented in [6]. Our strategies to accelerate this algorithm are then presented in Section 4. We perform an experimental evaluation of these techniques over artificial and real world data, and discuss the results in Section 5. Finally, in Section 6 we conclude and present suggestions for future work.

2 Related Work

In this section, we survey the state of the art for duplicate detection in hierarchical data, which is the focus of this paper. For a more complete discussion of related work, we refer readers to the book by Naumman and Herschel [2], which includes duplicate detection in a single relation, tree data, and graph data.

Among studies that deal with hierarchical data, we mainly find works focusing on the XML data model. The only exception is [3], which focuses on hierarchical tables in a data warehouse. Early work in XML duplicate detection was mostly concerned with the efficient implementation of XML join operations. A pioneering approach was presented by Guha et al. [11], who suggested an algorithm to perform approximate joins in XML databases. However, their main concern was on how to efficiently join two sets of similar elements, and not on how accurate the joining process was. Thus, they focused on an efficient implementation of a tree edit distance, which could later be applied in an XML join algorithm.

The concern with accuracy was later approached by Carvalho and Silva, in [12]. Although not specifically focused on XML, their work proposes a solution to the problem of integrating tree-structured data extracted from the Web. Two *object representations*, e.g., two hierarchical representations of person elements, are compared by transforming each into a vector of terms and using a variation of the cosine measure to evaluate their similarity [13]. The hierarchical structure of object representations is mostly ignored, and a linear combination of weighted similarities is used to account for the relative importance of the different fields within the vectors. The authors show that this simple strategy manages to achieve high precision values in a collection of scientific publications. Nevertheless, and because of its more general nature, their approach does not take advantage of the useful features existing in XML databases, such as the element structure or tag semantics.

Only more recently has research been performed with the specific goal of discovering duplicate object representations in XML databases [5], [6], [8], [10]. These works differ from previous approaches since they were specifically designed to exploit the distinctive characteristics of XML object representations: their structure, textual content, and the semantics implicit in the XML labels. We briefly describe the main features of these methods here, and refer readers to [9] for a detailed theoretical and experimental comparison of these approaches.

The DogmatiX framework aims at both efficiency and effectiveness in duplicate detection [5] . The framework consists of three main steps: *candidate definition*, *duplicate definition*, and *duplicate detection*. Whereas the first two provide the definitions necessary for duplicate detection (i.e., the set of object representations to compare and the duplicate classifier to use), the third component includes the actual algorithm, an extension to XML data of the work of Ananthakrishna et al. [3].

The XMLDup system first proposed in [6] uses a Bayesian Network model for XML duplicate detection. Its approach is the basis for the algorithms proposed in this paper, and is further described in Section 3.

Milano et al. propose a distance measure between two XML object representations that is defined based on the concept of *overlays* [8]. An overlay between two XML

trees U and V is a mapping between their nodes, such that a node $u \in U$, is mapped to a single node $v \in V$ if, and only if, they have the same path from the root. This measure is then used to perform a pairwise comparison between all candidates. If the distance measure determines that two XML candidates are closer than a given threshold, the pair is classified as a duplicate.

Finally, SXNM (Sorted XML Neighborhood Method) [10] is a duplicate detection method that adapts the relational sorted neighborhood approach (SNM) [14] to XML data. Like the original SNM, the idea is to avoid performing useless comparisons between objects by grouping together those that are more likely to be similar.

3 A Bayesian Network for Duplicate Detection

We now present the XMLDup approach to XML duplicate detection. We first present how to construct a Bayesian Network model (BN) for duplicate detection, and then show how this model is used to compute the similarity between XML object representations. Given this similarity, we classify two XML objects as duplicates if it is above a given threshold. This section essentially summarizes the work discussed in more detail in [6].

Throughout our work, we assume a schema mapping step has preceded duplicate detection, so that all XML elements we compare comply to the same schema. We note that the process of schema mapping is by itself complex and, for our algorithms to be effective, its result must first be validated to ensure a high quality mapping. This issue, however, is outside the scope of this paper.

3.1 Bayesian Network Construction

Bayesian Networks provide a concise specification of a joint probability distribution. They can be seen as a directed acyclic graph, where the nodes represent random variables and the edges represent dependencies between those variables. We first outline how the Bayesian Network for XML duplicate detection is constructed. Afterwards, we explain how probabilities are computed in order to decide if two objects are in fact duplicates. For a more detailed description of Bayesian Networks and their applications, the reader is referred to [15].

3.1.1 BN Structure for Duplicate Detection

Our approach for XML duplicate detection is centered around one basic assumption: *The fact that two XML nodes are duplicates depends only on the fact that their values are duplicates and that their children nodes are duplicates*. Thus, we say that two XML trees are *duplicates* if their root nodes are duplicates.

To illustrate this idea, consider the goal of detecting that both persons represented in Figure 1 are duplicates. This means that the two person objects, represented by nodes tagged prs, are duplicates depending on whether or not their children nodes (tagged pob and cnt) and their values for attributes $name$ and dob are duplicates. Furthermore, the nodes tagged pob are duplicates depending on whether or not their values are duplicates, and the nodes tagged cnt are duplicates depending on whether or not their children nodes (tagged eml and add) are duplicates. This process goes on recursively until the leaf nodes are reached. If we consider trees U and U' of Figure 1, this process can be represented by the Bayesian Network of Figure 2, as explained in the following.

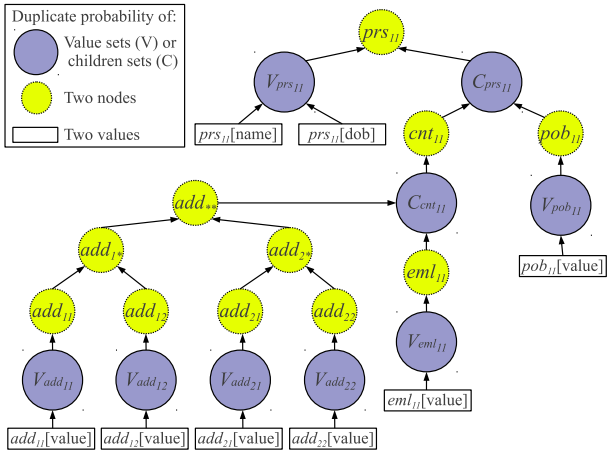


Fig. 2. BN to compute the similarity of the trees in Figure 1.

Let us first consider the XML nodes tagged prs. As illustrated in Figure 2, the BN will have a node labeled prs_{11} representing the possibility of node prs_1 in the XML tree U being a duplicate of node prs_1 in the XML tree U'. Node prs_{11} is assigned a binary random variable. This variable takes the value 1 (*active*) to represent the fact that the XML prs nodes in trees U and U' are duplicates. It takes the value 0 (*inactive*) to represent the fact that the nodes are not duplicates.

In accord with our assumption, the probability of the two XML nodes being duplicates depends on (i) whether or not their values are duplicates, and (ii) whether or not their children are duplicates. Thus, node prs_{11} in the BN has two parent nodes, as shown in Figure 2. Node $V_{prs_{11}}$ represents the possibility of the values in the prs nodes being duplicates. Node $C_{prs_{11}}$ represents the possibility of the children of the prs nodes being duplicates. As before, a binary random variable, that can be active or inactive, is assigned to these nodes, representing the fact that the values and children nodes are duplicates or non-duplicates, respectively.

We assume that the probability of the XML node values being duplicates depends on each attribute independently. This is represented in the network by adding new nodes for the attributes as parents of node $V_{prs_{11}}$, represented as rectangles in Figure 2. In this case, these new nodes represent the possibility of the $name$ values in the prs nodes being duplicates and of the dob values in the prs nodes being duplicates.

Similarly, the probability of the children of the prs nodes being duplicates depends on the probability of each pair of children nodes being duplicates. Thus, two more nodes are added as parents of node $C_{prs_{11}}$: node pob_{11} represents the possibility of node pob_1 in tree U being a duplicate of the node pob_1 in tree U'; node cnt_{11} represents the possibility of node cnt_1 in tree U being a duplicate of node cnt_1 in tree U'.

We can now repeat the whole process for these two nodes. However, a slightly different procedure is taken when representing multiple nodes of the same type, as is the case for the XML nodes labeled add. In this case, we wish to compare the full set of nodes, instead of each node independently. Thus, we say that the set of add nodes being duplicate depends on each add node in tree U being a duplicate of *any* add node in tree U'. This is represented by nodes add_{**}, add_{1*}, and add_{2*} in the BN of Figure 2.

Finally, each add_{ij} node represents the possibility that node add_i in tree U is a duplicate of node add_j in tree U'. Since the add nodes have no children, their probability of being duplicates only depends on their values. Thus, each node add_{ij} in the network has only one parent node $V_{add_{ij}}$ which has one parent representing the possibility of both XML nodes, add_i and add_j, having duplicate values. A more detailed explanation of the BN construction algorithm, including its pseudocode, can be found in [6].

Note that our approach is not symmetrical, i.e. the network obtained for U and U' is not always the same as that obtained for U' and U. However, it would be difficult to satisfy this feature without a significant decrease in efficiency, while we would not expect a high increase in effectiveness.

3.1.2 Computing the Probabilities

As we have seen, we assign a binary random variable to each node, which takes the value 1 to represent the fact that the corresponding data in trees U and U' are duplicates, and the value 0 to represent the opposite. Thus, to decide if two XML trees are duplicates, the algorithm has to compute the probability of the root nodes being duplicates. In our example, this corresponds to computing $P(prs_{11} = 1)$, which can be interpreted as a similarity value between the two XML elements. To obtain this probability, the algorithm propagates the prior probabilities associated to the BN leaf nodes, which will set the intermediate node probabilities, until the root probability is found.

In the following, we explain how these probabilities can be defined. For simplicity, we will use the notation $P(x)$ to mean P($X = 1$).

Prior probabilities. In the network of Figure 2, we need to define the prior probabilities of values being duplicates in the context of their parent XML node, i.e., $P(prs_{11}[name])$, $P(prs_{11}[dob])$, $P(pob_{11}[value])$, $P(eml_{11}[value])$, and $P(add_{ij}[value])$. These probabilities can be defined based on a similarity function $sim(\cdot)$ between the values, normalized to fit between 0 and 1. However, it is sometimes not possible, or not efficient, to measure the similarity between two attribute values. In this case, we define the probability as a small constant k_a, named the *default probability*, representing the possibility of any two values being duplicates before we observe them.

Thus, we define $P(t_{ij}[a]) = \mathrm{sim}(V_i[a], V_j[a])$ if the similarity was measured, and $P(t_{ij}[a]) = k_a$ if otherwise, where $V_i[a]$ is the value of attribute a of the i-th node with tag t in the XML tree. For instance, for the $name$ attribute in the prs nodes, we can define $\mathrm{sim}(n_1, n_2) = 1 - \mathrm{ned}(n_1, n_2)$, where $\mathrm{ned}(\cdot)$ is the normalized edit distance between the strings. The default probability k_a can be derived from the distribution of attribute values in the database, or simply be set to a small number.

Conditional probabilities. We further need to define the following four types of conditional probabilities:

Conditional probability 1 (CP1): The probability of the values of the nodes being duplicates, given that each individual pair of values contains duplicates. Intuitively, (i) if all attribute values are duplicates, we consider the XML node values as duplicates; (ii) if none of the attribute values are duplicates, we consider the XML node values as non-duplicates; (iii) if some of the attribute values are duplicates, we determine that the probability of the XML nodes being duplicates equals a given value, w_a. This value represents the importance of the corresponding attribute a in determining if the nodes are duplicates.

Conditional probability 2 (CP2): The probability of the children nodes being duplicates, given that each individual pair of children are duplicates. Intuitively, it makes sense to say that two nodes are duplicates only if all of their child nodes are also duplicates. However, it may be the case that the XML tree is incomplete, or contains erroneous information. Thus, we relax this assumption and state that the more child nodes in both trees are duplicates, the higher the probability that the parent nodes are duplicates.

Conditional probability 3 (CP3): The probability of two nodes being duplicates given that their values and their children are duplicates. Essentially, we consider the nodes as duplicates if both their values and their children are duplicates.

Conditional probability 4 (CP4): The probability of a set of nodes of the same type being duplicates given that each pair of individual nodes in the set are duplicates. We start by defining the probability $P(t_{**}|t_{1*}, t_{2*}, \dots)$ of the set of nodes being duplicates given that each of its nodes is a duplicate. As before, we assume that the more nodes are duplicates, the higher the probability that the whole set of nodes is a duplicate. We can then define the probability $P(t_{i*}|t_{i1}, t_{i2}, \dots)$, which reflects the fact that a node in an XML tree is a duplicate if it is a duplicate of at least one node of the same type in the other XML tree.

The formulas for these probabilities and their correspondence to our running example are summarized in

CP	Formula	Example
1	$P(V_{t_{ij}}\|t_{ij}[a_1],\ldots,t_{ij}[a_n]) = \sum_{1\leq k\leq n\|t_{ij}[a_k]=1} w_{a_k}$ subject to $\sum_{1\leq k\leq n} w_{a_k} = 1$	$P(V_{prs_{11}}\|prs_{11}[name], prs_{11}[dob])$ $P(V_{pob_{11}}\|pob_{11}[value])$ $P(V_{eml_{11}}\|eml_{11}[value])$ $P(V_{add_{ij}}\|add_{ij}[value])$
2	$P(C_{t_{ij}}\|t^1_{ij}, t^2_{ij},\ldots,t^n_{ij}) = \frac{1}{n}\times\sum_{k=1}^{n} t^k_{ij}$	$P(C_{prs_{11}}\|pob_{11}, cnt_{11})$ $P(C_{cnt_{11}}\|eml_{11}, add_{**})$
3	$P(t_{ij}\|V_{t_{ij}}, C_{t_{ij}}) = \begin{cases} 1 & \text{iff } V_{t_{ij}} = C_{t_{ij}} = 1 \\ 0 & \text{otherwise} \end{cases}$	$P(prs_{11}\|V_{prs_{11}}, C_{prs_{11}})$ $P(pob_{11}\|V_{pob_{11}})$ $P(cnt_{11}\|C_{cnt_{11}})$ $P(eml_{11}\|V_{eml_{11}})$ $P(add_{ij}\|V_{add_{ij}})$
4	$P(t_{**}\|t_{1*}, t_{2*},\ldots,t_{n*}) = \frac{1}{n}\sum_{k=1}^{n} t_{k*}$ $P(t_{i*}\|t_{i1}, t_{i2},\ldots,t_{in}) = \begin{cases} 1 & \text{iff } \exists_j \mid t_{ij} = 1 \\ 0 & \text{otherwise} \end{cases}$	$P(add_{**}\|add_{1*}, add_{2*})$ $P(add_{i*}\|add_{i1})$

TABLE 1
Formulas for conditional probabilities and their applicability to our example of Figure 2.

Table 1.

Final probability. Once all prior and conditional probabilities are defined, the BN can be used to compute the probability of two XML trees being duplicates, i.e. $P(t_{11})$, where t is the tag for the root node of both trees. This can be achieved by any probability propagation algorithm, such as those described in [15]. In our example, we will have that:

$$P(prs_{11}) = \Big(w_{name}P(prs_{11}[name]) + w_{dob}P(prs_{11}[dob])\Big) \times \Big(P(pob_{11}[value]) + \big(1 - \prod_{i=1}^{2}(1 - P(add_{1i}[value]))\big) + 1 - \prod_{i=1}^{2}(1 - P(add_{2i}[value])) + 2P(eml_{11}[value])\Big) \times \frac{1}{4}\Big) \times \frac{1}{2} \quad (1)$$

We should note that our solution only compares pairs of objects and, thus, we do not apply the transitive closure. In the case where more than two XML elements represent the same real-world object, the transitive closure over identified duplicate pairs could be applied in a post-processing step.

4 Accelerating the BN Evaluation

To compute the final probability described in Section 3.1.2, one needs to analyze the whole network and calculate the probabilities for every node. This process, which has a complexity of $O(n \times n')$, where n and n' are the number of nodes in each XML tree being compared, can be time consuming, especially if we are dealing with a large network. However, when performing duplicate detection, we are usually interested only in objects whose duplicate probability is above a given threshold. This allows us to optimize the network evaluation process. In this section we propose a novel strategy to reduce the time spent on the BN evaluation.

4.1 Network Pruning

In order to improve the BN evaluation time, we propose a lossless pruning strategy. This strategy is lossless in the sense that no duplicate objects are lost. Only object pairs incapable of reaching a given duplicate probability threshold are discarded.

As stated before, network evaluation is performed by doing a propagation of the prior probabilities, in a bottom up fashion, until reaching the topmost node. The prior probabilities are obtained by applying a similarity measure to the pair of values represented by the content of the leaf nodes. Computing such similarities is the most expensive operation in the network evaluation, and in the duplicate detection process in general. Therefore, the idea behind our pruning proposal lies in avoiding the calculation of prior probabilities, unless they are strictly necessary.

The strategy follows the premise that, before comparing two objects, all the similarities are assumed to be 1 (i.e., the maximum possible score). The idea is to, at every step of the process, maintain an upper-bound on the final probability value. At each step, whenever a new similarity is computed, the final probability is estimated taking into consideration the already known similarities and the unknown similarities that we assume to be 1. When we verify that the network root node probability can no longer achieve a score higher than the defined duplicate threshold, the object pair is discarded and, thus, the remaining calculations are avoided. The process is presented in detail in Algorithm 1.

The algorithm takes as input a node N from the BN and a user defined threshold T. It starts by gathering a list of all the parent nodes of N and assuming that their duplicate probability score is 1 (lines 1 and 2). It then proceeds to compute the actual probability value of each of the parents of N (lines 4–16).

If a given parent node n is a value node (line 6), its probability score is simply the similarity of the values it represents. If, on the other hand, n also has parent nodes, its probability score depends on the scope of its

Algorithm 1 $NetworkPruning(N, T)$

Require: The node N, for which we intend to compute the probability score; threshold value T, below which the XML nodes are considered non-duplicates
Ensure: Duplicate probability of the XML nodes represented by N
1: $L \leftarrow getParentNodes(N)$ {Get the ordered list of parents}
2: $parentScore[n] \leftarrow 1, \forall n \in L$ {Maximum probability of each parent node}
3: $currentScore \leftarrow 0$
4: **for** each node n in L **do** {Compute the duplicate probability}
5: **if** n is a value node **then**
6: $score \leftarrow getSimilarityScore(n)$ {For value nodes, compute the similarities}
7: **else**
8: $newThreshold \leftarrow getNewThreshold(T, parentScore)$
9: $score \leftarrow NetworkPruning(n, newThreshold)$
10: **end if**
11: $parentScore[n] \leftarrow score$
12: $currentScore \leftarrow computeProbability(parentScore)$
13: **if** $currentScore < T$ **then**
14: $End\ network\ evaluation$
15: **end if**
16: **end for**
17: **return** $currentScore$

own parents, which we compute recursively (line 9). However, the algorithm should now be called with a different threshold value, that depends on the equation used to combine the probabilities for node N (line 8). Using the example from Figure 2, assume our initial threshold was $T = 0.8$ and that we wish to compute $P(prs_{11})$. Assume also that the probability score of $V_{prs_{11}}$ was already computed as 0.9. To compute the probability score of prs_{11}, we would require $C_{prs_{11}}$ to have a probability such that

$$\sum_{V_{prs_{11}}, C_{prs_{11}}} P(prs_{11}|V_{prs_{11}}, C_{prs_{11}})P(V_{prs_{11}})P(C_{prs_{11}}) > 0.8$$

If we were using the formula for CP3 (see Table 1), this would be equivalent to $P(V_{prs_{11}})P(C_{prs_{11}}) > 0.8$, which, in Algorithm 1 corresponds to $parentScore[V_{prs_{11}}] \times parentScore[C_{prs_{11}}] > 0.8$. Thus, our new threshold would be $0.8/0.9 = 0.89$.

Once the score for node n is computed, the algorithm checks if the total score for N is still above T, and decides whether to continue computing or to stop the network evaluation (lines 13–15). Function $computeProbability$ consists in applying one of the probability equations presented in Section 3.1.2.

We now prove the correctness of the pruning method and discuss its complexity. We start by showing that the evaluation of each individual (non-leaf) node is not affected by the fact that some of its parent nodes may not be evaluated. We then show that the evaluation of the parent nodes of a given node will also return the expected value. For succinctness, in the following, we will denote $P(X = 1)$ as $P(x)$ and recall that $P(X = 1) = 1 - P(X = 0)$.

Theorem 1 *Given a (non-leaf) node N of the duplicate detection Bayesian Network, and a threshold value T, the $networkPruning$ algorithm computes exactly the same probability score as the non-pruned version of the procedure, if $s \geq T$, where s is the score computed.*

Proof: Let N be a node in the BN and let $N_1, \ldots, N_k$ be the parent nodes of N. The final score to be computed by Algorithm 1 is:

$$s = P(n) = \sum_{N_1,\ldots,N_k} P(n|N_1, \ldots, N_k) \times P(N_1) \times \cdots \times P(N_k) \quad (2)$$

where $P(n|N_1, \ldots, N_k)$ is one of the conditional probabilities defined in Table 1. Let F_i be the value computed by Algorithm 1 when only the first i parent nodes have been evaluated (i.e., when we assume that $P(n_j) = 1, \forall j > i$). We want to show that if, for any i, $F_i < T$, then $s < T$ and the algorithm can safely stop. This can be shown for each of the conditional probabilities in Table 1. Due to space constraints, we will limit our proof to conditional probability CP2.

By replacing the conditional probability in Equation (2) with CP2, and through some algebraic manipulation, we get that

$$s = \frac{P(n_1) + \cdots + P(n_k)}{k} \quad (3)$$

In this case, we have that

$$F_i = \frac{\sum_{j=1}^{i} P(n_j) + (k - i)}{k} \quad (4)$$

Note that $s = F_k$. Since, for all i, $P(n_i) \leq 1$, it is clear that $F_1 \geq F_2 \geq F_3 \geq \cdots \geq F_{k-1} \geq F_k$. Thus, if any $F_i < T$, then $s < T$ and the algorithm can stop. Otherwise, the processing will continue until F_k is computed. □

Theorem 2 *Given a (non-leaf) node N of the duplicate detection Bayesian Network, and a threshold value T, the recursive call to evaluate a parent node N_i of N will return the same value as if the network rooted at N_i was fully evaluated, or stop only if $P(n) < T$.*

Proof: Assume that the recursive call will be applied to node N_i. We need to find a new threshold T' for $P(n_i)$ such that, if $P(n_i) < T'$, then $P(n) < T$. As before, we will limit our demonstration to conditional probabilities CP2, although the same can be shown for the remaining probabilities.

From Theorem 1, we know that the network evaluation should stop if $F_i < T$. This yields:

$$P(n_i) < kT - \sum_{j=1}^{i-1} P(n_j) - (k - i) \quad (5)$$

Thus, evaluation should stop when the inequality in Equation (5) holds. This means that, to evaluate the subnetwork rooted at node N_i, we can safely use the value $T' = kT - \sum_{j=1}^{i-1} P(n_j) - (k - i)$ as the new threshold in the recursive call of line 9. □

Regarding the complexity of the algorithm, since we cannot guarantee that all objects in the dataset will contain very different information (which would cause many nodes to be pruned out of the BN evaluation) the

number of comparison remains quadratic, in the worst case. In addition, the algorithm also incurs a small overhead, due to the computation of the cutoff thresholds. However, in a real world scenario, where we expect to have only a marginal amount of very similar attributes, the number of comparisons can be substantially reduced. The experiments presented in Section 5.3.2, over artificial and real world data, confirm this perspective, showing that the duplicate detection with the pruning optimization was able to perform up to 4.5 times faster than the unoptimized version of the algorithm.

4.2 The Effect of Node Order on Pruning

Our proposed strategy works by estimating the highest achievable score after computing the probability of each single node. Thus, by choosing the appropriate order by which to evaluate the nodes, we can assure that the algorithm makes the minimal number of calculations, before deciding if a pair of objects is to be discarded.

The rationale supporting this idea derives from the fact that every object contains elements that hold more distinctive information than others. Consider, for example, the objects in Figure 1. Surely, it is more unlikely to have two persons with the same name than having two persons born on the same date. Thus, we can expect lower similarity scores on names than on dates of birth. For this reason, if we first evaluate the name value, we will discover sooner if the object can be discarded since, as a consequence of the lower similarity scores, the cut-off threshold will be reached sooner.

Since we do not have a way of knowing the probability scores in advance, we need to devise a heuristic for sorting nodes that, hopefully, approximates the ideal node order. In this work, we propose three such heuristics: sorting by *depth*, by *average string size*, and by *distinctiveness*. Each of these heuristics corresponds to a different way of ordering the nodes in the set L, obtained in line 1 of Algorithm 1, as explained in the following.

Using the depth of a node follows the intuition that, in a hierarchic structure like an XML document, the most important information is usually stored in nodes that are placed closer to the root, while nodes with less distinctive power are stored in deeper levels. Therefore, by ordering the nodes according to the depth of the branch to which they belong, we cause the more distinctive nodes to be evaluated first. This may, of course, not always be true, but it was shown to be accurate on our tested datasets.

Using the size of the node consists in first evaluating nodes whose values have a smaller average string length. The idea is simply to perform the cheaper comparisons first, expecting that non-duplicate nodes to be discarded before the longer strings have to be compared. We should note that, since shorter strings are more likely to be similar than larger strings, this node ordering could delay the reaching of the cut-off threshold. However, we expect the cheaper string comparisons to compensate for the increment in the number of compared strings.

The final heuristic consists in sorting the nodes according to their distinctiveness. We define the distinctiveness of a node n as $\log(A/d_n)$, where A is the total number of objects containing node n and d_n is the number of distinct values of node n. This measure follows the idea that nodes with a high number of distinct values are less likely to be similar and thus should be evaluated first. We note that the overhead introduced by the precomputation of these heuristics is negligible.

Each one of these heuristics provides a way of prioritizing the evaluation of some nodes in order to conclude as soon as possible if further network evaluation is needed. They can be used individually or in combination. In Section 5.3.2, we show, through experiments, that this is in fact true and that network evaluation can be considerably accelerated, spending about 60% less time than the unoptimized version.

4.3 Varying the Pruning Factor

As stated in Section 4.1, before evaluation, every node is assumed to have a duplicate probability of 1. We call this assumed probability the *pruning factor*. We now propose that, by lowering the pruning factor, we can evaluate the network even faster, although at the cost of not detecting a small number of duplicate objects. The idea behind this new strategy is explained as follows.

As shown in Theorem 1, having a pruning factor equal to 1 guarantees that the duplicate probability estimated for a given node is always above the true node probability. Therefore, no duplicate pair of objects is ever lost. By lowering the pruning factor, we lose this guarantee. Thus, a pair of objects may be prematurely discarded, even if they are true duplicates. However, with a lower pruning factor, we also know that all probability estimates will be lower. This will cause the defined duplicate threshold to be reached sooner and, consequently, the network evaluation to stop sooner. Thus, fewer similarity calculations will be performed.

Based on this knowledge, our strategy relies on the assumption that, by slightly lowering the pruning factor, we can achieve high gains in performance, while loosing only a very small amount of true duplicates. This assumption is empirically shown to be correct, as we demonstrate in Section 5.3.2.

4.4 Automatic Pruning Factor Selection

There are still two ways in which we can further exploit the advantages of using a pruning factor lower than 1. First, since different attributes in an XML object have different characteristics, they could also have different pruning factors. Second, fine-tuning one, or several, pruning factors manually can be a complex task, especially if the user has little knowledge of the database, thus we should be able to compute all pruning factors automatically. In this section we describe a method that automatically determines which pruning factor to use for each attribute, in order to optimize efficiency, while minimizing the loss in effectiveness.

4.4.1 Mapping Attributes to Pruning Factors

Consider, for example an attribute representing the gender of a person. Naturally, we expect it to have many similar values, since it can only be "male" or "female". It would, therefore, be advisable to assign it a high pruning factor (close to 1). On the other hand, if we are dealing with an attribute whose contents are more diverse, and thus less likely to be similar, such as person names, a lower pruning factor can be applied. In sum, for a given attribute, the higher the likelihood of its values being similar, the lower the pruning factor should be.

The question is: how do we determine the likelihood of the value being similar without actually computing all the similarities? To achieve this, we have adopted a strategy similar to the one proposed in [16], which we describe in the following.

Let $\mathcal{A}$ be the set of all attributes in the schema of the XML objects in the dataset. Each attribute $a \in \mathcal{A}$ is represented as a vector of statistical features. We expect these features to provide information useful in determining the pruning factor. Our goal is to find a mapping function $M : \mathcal{A} \rightarrow [0, 1]$, which takes an attribute a as input and returns the corresponding pruning factor.

To represent the attributes we use 12 different features. These are grouped according to the characteristics they try to capture.

Group1: Uniqueness features. Features that capture the diversity of attribute values within a dataset. The group contains the features distinctiveness, entropy of distinctiveness, harmonic mean of distinctiveness, standard deviation of distinctiveness, and the diversity index of the attributes content [17].

Group2: Format features. Features that provide information about the type of contents in attributes values. These are the ratio of attributes that contain numeric values, alphabetical values, and both.

Group3: Content length features. Since we use a string edit distance to compare attribute values, and given that the outcome of this measure is strongly related to the size of the strings, this group contains the features average string size and entropy of the string sizes.

Group4: Absence features. This group contains only one feature that measures how many objects are missing the given attribute. Attributes that are missing in many objects should probably be taken less into account.

Group5: Occurrence features. This group contains only one feature that measures the number of occurrences of an attribute per object. Attributes that appear many different times (e.g., actor names in a movie) should be treated differently from singular attributes (e.g., the title of a movie).

Once the features are defined, finding function M can be seen as a regression problem. In this work, we have solved it using Support Vector Regression (SVR) [18].

4.4.2 Learning the Mapping Function

To learn the function M through regression, we need to provide a set of examples. In our case, the examples would be sets of attributes, together with their ideal pruning factors. Since, in a real case scenario, we do not expect to have such data available, we need to find an approximate solution. We now describe our approach to this problem.

Assume that we have available a set $\mathcal{D}$ of one or more XML databases, where some duplicate objects were previously identified. These databases can be different from the one where we are performing duplicate detection and do not even need to be on the same domain. Given $\mathcal{D}$, a straightforward way of finding the appropriate pruning factors would be to (i) try every possible set of pruning factor values, (ii) for each set, measure the accuracy of our duplicate detection method, and (iii) choose the set that yielded the best results. This process would result in a set of attributes and pruning factors that could be used to learn the function $\mathcal{M}$.

Doing such an exhaustive search is, however, unfeasible. We therefore do an approximate search using the method of *simulated annealing* (SA) [19]. SA is an algorithm intended to determine the maximum (or minimum) value of a function with several independent variables, under a fraction of the time needed to find an absolute maximum (or minimum) in a large search space. This function is called the *objective function* and its results depend on the possible variable configurations, or *states*. SA consists in shifting among candidate states in order to approximate the *objective function* to a global optimum, using an *acceptance function* to decide if the new state should be accepted.

In our approach, the objective function to minimize is the number of comparisons performed and the states are the sets of pruning factor values. To determine if the algorithm should change state, we use the following variation of the commonly used acceptance function:

$$P(p_s, p_n, t) = \begin{cases} 0 & \text{if recall} < R \\ 1 & \text{if } p_n > p_s \\ e^{\left(\frac{p_n - p_s}{t}\right)} & \text{otherwise} \end{cases} \quad (6)$$

where p_s is the objective function value for the current state, p_n is the objective function value for the new state, t is the fraction of iterations the algorithm still needs to execute, and R is a minimum threshold for the recall achieved. Equation (6) assures that a new set of pruning factor values is accepted only if the loss in recall is not too high. In our experiments, we defined R as 90% of the recall achieved by the lossless pruning strategy.

We used the common acceptance function of SA, with a slight difference: new states are accepted only if the loss in recall is less then 10%, when compared to the algorithm with a pruning factor of 1.

In sum, we perform a search to find the near-optimal pruning factors for the set of attributes in the databases of $\mathcal{D}$, we use these pruning factors to learn the function $\mathcal{M}$ through regression, and then use $\mathcal{M}$ to compute the pruning factors for the database where we are performing duplicate detection.

4.4.3 Pruning Factors for Inner Nodes

The process so far enables us to find different pruning factors for each attribute. However, to evaluate the Bayesian network, we need to apply a pruning factor in every node, and not just the attribute nodes. We solve this problem using a simple strategy.

While performing a similarity calculation, we determine the pruning factors for each attribute (leaf) node. Since pruning factors can be seen as upper bounds on the probability of each node being active, we can propagate these values bottom-up along the network, as if they were the actual probability values. The value computed at each inner node will then be used as its pruning factor.

This process requires us to evaluate the network twice for each similarity calculation. However, this is a much cheaper evaluation, since we are not computing similarities, but only propagating values. In practice, experiments presented in Section 5.3.2 show that, by automatically selecting the pruning factors, we were able to improve efficiency over the lossless strategy with small losses in recall, without the need of any manual tuning.

5 Experiments on Duplicate Detection

In this section we present an evaluation of the XMLDup algorithm described in the previous sections. We evaluate the algorithm both in terms of effectiveness and efficiency. First, we evaluate effectiveness by comparing it to a state of the art duplicate detection system, called DogmatiX [5], that proved to be the most competitive so far [9]. We then evaluate the efficiency of XMLDup when using our proposed pruning optimization, node ordering heuristics, varying the pruning factor, and automatically selecting the most adequate pruning factors. The experiments are concluded with a discussion of the results.

Testing the impact of data quality on duplicate detection is important to confirm the effectiveness of a given algorithm. In previous work we have shown that XMLDup manages to cope with errors like typos or duplicate erroneous elements without any significant degradation of the results and even performs effectively when dealing with reasonable amounts of missing data [6], [9]. Due to space constraints, those experiments will not be repeated here.

5.1 Datasets

Our tests were performed using seven different datasets, representing five different data domains. The first three datasets, *Country*, *CD* and *IMDB*, consist of XML objects taken from a real database and artificially polluted by inserting duplicate data and different types of errors, such as typographical errors, missing data, and duplicate erroneous data [6]. The remaining four datasets, *Cora*, *IMDB+FilmDienst*, another dataset containing CD records (which we are going to refer to as *CD 2*), and *Restaurant*, are composed exclusively of real world data, containing naturally occurring duplicates.

The datasets vary in size from 9763 objects (*CD 2*) through 864 objects (*Restaurant*). All datasets contained objects nested in a hierarchy of up to 3 levels, except for the *Restaurant* dataset, which contained 4 levels. For each dataset, attributes that did not contain information useful for duplicate detection (such as movie rating or year, which can be common to many different objects) were discarded. Before each test, we enumerate the actual attributes used. In all experiments, we used a duplicate threshold of 0.5 for all datasets, except for *CD 2* and *Cora*, where we used 0.4 and 0.3, respectively. Additionally, DogmatiX uses a threshold to identify similar object descriptors [5], which was set to 0.15.

The *Cora*, *Country*, *IMDB*, *IMDB+FilmDienst* and both *CD* datasets are available at the Hasso Plattner Institute website[1]. The *Restaurant* dataset was obtained from the University of Texas Machine Learning Research Group website[2].

5.2 Experimental setup

Experiments were performed to compare the effectiveness and efficiency of the tested algorithms. To assess effectiveness, we applied the commonly used *precision*, *recall*, and *r-precision* measures [13]. Precision measures the percentage of correctly identified duplicates, over the total set of objects determined as duplicates by the system. Recall measures the percentage of duplicates correctly identified by the system, over the total set of duplicate objects. R-precision measures the precision at cut-off R, when R is the number of duplicates in the dataset. To assess efficiency, we measured the runtime and number of comparisons of the duplicate detection process. To perform SVR, we used the SVMLight implementation [20], with a linear function kernel.

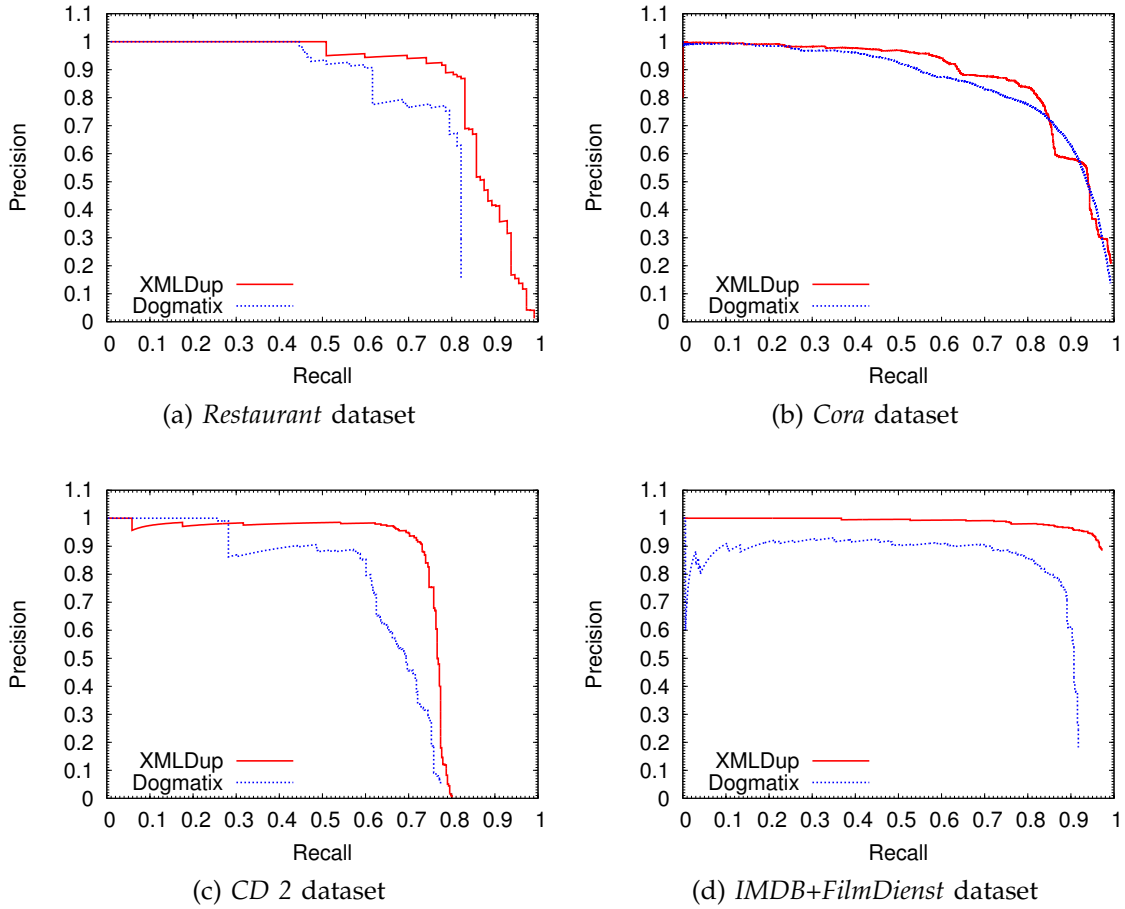
To compare the object data, we considered all the attribute values as textual strings, using the formula $\mathrm{sim}(V_1, V_2) = 1 - \frac{\mathrm{ed}(V_1,V_2)}{\max(|V_1|,|V_2|)}$ as the similarity measure, where $\mathrm{ed}(V_1, V_2)$ is the string edit distance between values V_1 and V_2, and $|V_i|$ is the length (number of characters) of string V_i. To construct the Bayesian network we used the probabilities defined in Section 3.

The experimental evaluation was performed on an Intel two core CPU at 2.53GHz and 4GB of RAM, having a 64-bit Ubuntu as its operating system. Both tested algorithms were fully implemented in Java, using the DOM API to process the XML objects.

5.3 Results

We now present the experimental results for the XMLDup algorithm, both in terms of effectiveness and efficiency.

1. http://www.hpi.uni-potsdam.de/naumann/projekte/repeatability/
2. http://www.cs.utexas.edu/users/ml/riddle/data.html



(a) *Restaurant* dataset (b) *Cora* dataset

(c) *CD 2* dataset (d) *IMDB+FilmDienst* dataset

Fig. 3. Comparison results for XMLDup and DogmatiX representing precision and recall values for the *Restaurant* dataset (a), *Cora* dataset (b), *CD 2* dataset (c), and *IMDB+FilmDienst* dataset (d).

5.3.1 Effectiveness Evaluation

To evaluate effectiveness, we present a comparison of XMLDup to the DogmatiX duplicate detection system. We chose DogmatiX because, although it does not use the object structure to compute similarities, it was shown to be the most competitive in previous tests [9].

In this experiment, only the real world datasets were used. The attributes used for the *Restaurant* dataset were *name*, *address*, *phone* and other four attributes representing geographic coordinates. For the *Cora* dataset we used the attributes *author*, *title* and *venue name*. For the *CD 2* dataset, the attributes *artist*, *disc title* and *track title* were used. For the *IMDB+FilmDienst* dataset we used the attributes *title*, *aka-title* and *cast name*. Figure 3 shows the precision/recall results obtained for each experiment.

The performed tests reveal a pattern in the behavior of both systems. For the *Restaurant*, *Cora* and *CD 2* datasets, both XMLDup and DogmatiX present similar curves of near 100% of precision up to 45%, 40% and 28% of recall, respectively. After that point, the DogmatiX curve drops, while we can observe higher levels of precision for XMLDup. This shows a higher resilience in the XMLDup scoring method, which avoids false positives with high scores.

When using the *IMDB+FilmDienst* dataset, DogmatiX presents a different behavior, placing some false positives on an early stage. This shows some issues in the scoring method, which has some difficulties in separating duplicate and non-duplicate pairs on the scoring scale, leading to the alternate presence of true positives and false positives. This phenomenon is observed sooner than before because of the presence of remakes and sequels of the same movie.

After the initial fall DogmatiX is still capable of a slight recovery, yet maintaining its curve below XMLDup. Consistent to what was shown before, the XMLDup scoring method maintains its ability of grouping together true positives in bigger blocks, reaching precision scores above 90% until about 70% of recall, in the worst case.

In the next section we test our algorithm for efficiency, also comparing its performance results to those obtained by DogmatiX.

5.3.2 Efficiency Evaluation

To evaluate efficiency, we performed experiments using XMLDup with our proposed pruning algorithm. First, using the artificially polluted datasets, we determined the best node sorting strategy, as explained in Section 4.2. Second, using this sorting strategy, we applied XMLDup on real world datasets and studied its runtime efficiency.

Datasets	Country		CD		IMDB	
Sorting Strategy	*Time*	*Comparisons*	*Time*	*Comparisons*	*Time*	*Comparisons*
Unsorted	00:00:22	8079097	00:00:43	7228014	00:47:08	603480020
Depth	00:00:09 (-59.1%)	1589229 (-80.3%)	00:00:43 (0.0%)	7228014 (0.0%)	00:50:50 (+7.9%)	215104693 (-64.4%)
Dstnct	00:00:23 (+4.5%)	8465624 (+4.8%)	00:03:17 (+358.1%)	51914696 (+618.2%)	01:34:12 (+99.9%)	600065280 (-0.6%)
AvgSS	00:00:20 (-9.1%)	7090107 (-12.2%)	00:02:12 (+207.0%)	32709561 (+352.5%)	00:44:10 (-6.3%)	536555956 (-11.1%)
Depth+Dstnct	00:00:10 (-54.5%)	1715275 (-78.8%)	00:00:43 (0.0%)	7225379 (0.0%)	01:16:24 (+62.1%)	212822177 (-64.7%)
Depth+AvgSS	00:00:09 (-59.1%)	1629926 (-79.8%)	00:00:43 (0.0%)	7228014 (0.0%)	00:50:38 (+7.4%)	213975085 (-64.5%)

TABLE 2
Performance in time and number of comparisons using the pruning method on artificial data, with nodes ordered by depth (**Depth**), distinctiveness (**Dstnct**), average string size (**AvgSS**) and by combined strategies (**Depth+Dstnct** and **Depth+AvgSS**). Time is shown in the format hh:mm:ss.

Next, we tested several variations of the pruning factor and evaluated their actual impact on recall. Furthermore, we fine tuned the pruning factor to confirm that minor reductions can significantly accelerate the duplicate detection process, with a minor impact in the system effectiveness, as explained in Section. 4.3. The achieved results are also compared with the results obtained by DogmatiX.

Note that DogmatiX also employs a strategy to accelerate its duplicate detection process. It consists of applying a filtering function to eliminate candidates that share few information with all other candidates. Discarding these candidates avoids comparisons involving their content. Since the filtering function is an upper-bound of the similarity measure, the algorithm guarantees that no pair above the defined duplicate similarity threshold is discarded. Finally, we test how our procedure to automatically discover the pruning factor for each attribute, described in Section 4.4, works when compared to the lossless pruning factor strategy.

Impact of Node Sorting Strategies

We start by studying the impact that the heuristics proposed in Section 4.2 for sorting nodes have on the efficiency of XMLDup, when employing the pruning strategy. Table 2 shows the time spent by the algorithm when performing duplicate detection and the number of string comparisons, for each of the sorting strategies. Additionally we tested two hybrid strategies that combine the heuristics we defined, namely *Depth+Dstnct* (elements are first sorted by depth and then by distinctiveness) and *Depth+AvgSS* (elements are first sorted by depth and then by the average string size). Between parentheses, we show the improvement obtained over the baseline, i.e., *Unsorted*.

Tests were performed using all attributes. Although the use of attributes with less relevant information can reduce the system's effectiveness we believe that it can provide a more accurate insight on how node sorting influences efficiency.

As we can see, the order by which elements appear in the object structure has a major impact on efficiency. In fact, with the exception of the *CD* dataset, the unsorted structure is always outperformed by almost all of the strategies applied. In the best case (sorting by depth on the *Country* dataset), the performance enhancement reduces runtime by about 59% and the number of comparisons by about 80%. In fact, the more consistent results were obtained when sorting by depth, thus confirming our thesis that it is useful to look at elements closer to the root as those with higher distinctive power. We note that, for the *CD* dataset, the original structure is already near optimal. Thus, it presents better results than most sorting strategies.

For the *Country* and the *IMDB* datasets, sorting by average string size is also able to outperform the unsorted structure. It is interesting to see that, in the *IMDB* dataset, despite the higher number of string comparisons, the algorithm is able to perform faster than some of the remaining strategies. This can be explained by the trade-off between the number of strings compared and the string size. In fact, we notice that, in *IMDB*, the attributes with high distinctive power contain longer strings. Additionally, some of these attributes present multiple occurrences of the same type. Nevertheless, though comparing less distinctive attributes with smaller strings first requires more comparisons to be made, it is often worth avoiding the comparison of longer strings and the additional complexity of considering multiple children of the same type.

From the three individual sorting approaches, sorting by distinctiveness presented the worst results. Due to typographical errors, there are many strings in the dataset that differ only by a few characters. This artificially increases the distinctiveness values, making this measure inappropriate to choose the best attributes. Using a more accurate distinctiveness measure, e.g., one that considers strings within a certain edit distance, could be useful and should be tested in the future.

Both combined approaches presented run times equal or below the unsorted structure in almost all cases. This shows that different sorting strategies combined are capable of exploiting the different characteristics of the attributes. The exception is the *IMDB* dataset. In *IMDB*, many long string attributes are located close to the root. This explains why, even though the number

	pf=0.7		pf=0.6		pf=0.5		pf=0.4	
Dataset	*Recall*	*R-P*	*Recall*	*R-P*	*Recall*	*R-P*	*Recall*	*R-P*
Restaurant	99	83	99	83	98	83	94	92
Cora	99	82	99	82	99	82	98	82
CD 2	80	75	80	75	80	76	80	74
IMDB+FD	97	95	96	95	91	93	88	88

TABLE 3
Recall and r-precision (*R-P*) achieved for each tested pruning factor.

of comparisons is lower, the total time spent evaluating similarities is higher.

Overall, just by sorting the object structure, we were able to perform duplicate detection up to 2.4 times faster, with up to 80% less string comparisons. The high amplitude results obtained by the different kinds of sorting, specially when testing the *CD* dataset, strongly suggest that an optimal element order can produce a highly significant performance improvement. In contrast, poor element sorting can severely compromise the algorithm's efficiency. In the following experiments, we use the *Depth+AvgSS* sorting strategy, because of its best overall results.

Varying the pruning factor

Another set of experiments was performed to evaluate the improvements achieved by our pruning strategy. We applied the duplicate detection process to all real world datasets, with and without our network pruning strategy. When using the pruning strategy, we tested pruning factor values of 1 (lossless pruning), 0.7, 0.6, 0.5, and 0.4. For values below 0.4 no significant gains in time were achieved and, in some cases, there was a high loss in precision and recall. For values between 1 and 0.7 recall and precision did not change, in all datasets.

Table 3 shows the maximum recall achieved for each tested pruning factor, along with the r-precision score. Additionally, Table 4 presents the corresponding time performance values. For completeness, we include a comparison with the DogmatiX system. The gains achieved are shown between parentheses.

In the *Restaurant* dataset we observe that, although the improvement in efficiency between pruning factors is not very large, mainly because of the small dataset size, we are still able to reduce the runtime by 6 seconds when using the lowest pruning factor, with a decrease of 5% in recall. Moreover, XMLDup was able to perform the duplicate detection process in 72.2% less time than DogmatiX. Notice that the r-precision increases by 9% when compared with the lossless version (pf=1). This is explained by the fact that some false positives that would appear before are now being discarded.

In the *Cora* dataset, like before, when we decrease the pruning factor to 0.4, recall drops only by 1 point. Nevertheless, runtime drops to about 2 minutes, a 84.6% drop in relation to DogmatiX. For a pruning factor of 0.5, the algorithm runs in about 20 seconds less than the lossless strategy and presents no impact both in recall and r-precision.

In the *CD 2* dataset, with a pruning factor of 0.4, we were able to perform the process about 2.6 times faster, with no loss in recall, and with only 1% loss in r-precision. Also, we observe that XMLDup became about 5.3 times more efficient than DogmatiX.

Results in the *IMDB+FilmDienst* dataset are consistent to what was previously observed. In this case, a maximum of 9% in recall is lost when we decrease from 1 to 0.4. For a pruning factor of 1, which represents no false dismissals, our approach performs about 66% less efficiently than DogmatiX, although being 1.4 times faster than XMLDup without pruning. However, when we use a pruning factor of 0.7 the algorithm shows an improvement in efficiency of 54.5% over DogmatiX, baring no loss in r-precision and recall. The algorithm was further capable of accelerating the process 9.3 times, when compared to the lossless strategy, losing only 1% of recall.

One can notice that, differently from what happens for the remaining experiments, when the non-pruned strategy is used, XMLDup is slower than DogmatiX on both the *CD 2* and the *IMDB+FilmDienst* datasets. This can be explained by the presence of multiple occurrences of attributes of the same type (e.g. as tracks and movie cast members), which increases the number of comparisons performed. When the pruning strategy is employed, comparisons that result from this situation are avoided and, thus, the observed runtimes decrease drastically.

We also note that the recall values presented for the pruning factor of 0.6 in Table 3 never drop below the recall values achieved by the DogmatiX system. Furthermore, for a pruning factor of 0.4, XMLDup loses at most 4% of recall when compared to DogmatiX, but achieves an r-precision from 4% (*Cora*) through 24% (*IMDB+FilmDienst*) higher for all datasets, except *CD 2*, where it looses by 9%.

Automatically Selecting The Pruning Factor

The last set of experiments was devised to verify how our automatic pruning factor selection would behave when compared to the lossless strategy. To assess the improvements in performance achieved by this method we started by learning the near-optimal pruning factors for each attribute as described in Section 4.4.2. To that end, we allowed the pruning factors to assume values between 0 and 1, with a 0.1 step. The model was optimized by using all databases except the one being tested to generate the examples.

To learn the regression model (Section 4.4.1) we tested several combinations of feature groups, in order to infer which ones were able to build the most reliable classifier. Table 5 shows the results obtained in terms of total recall and number of comparisons. As a baseline, we show the lossless strategy with pruning optimization ($pf = 1$) and a lossy strategy with the best pruning factor manually

Dataset	DogmatiX	XMLDup(np)	XMLDup(pf=1)	XMLDup(pf=0.7)	XMLDup(pf=0.6)	XMLDup(pf=0.5)	XMLDup(pf=0.4)
Restaurant	00:00:18	00:00:11 (-39.9%)	00:00:07 (-61.1%)	00:00:06 (-66.7%)	00:00:05 (-72.2%)	00:00:05 (-72.2%)	00:00:05 (-72.2%)
Cora	00:13:23	00:02:42 (-79.8%)	00:02:41 (-80.0%)	00:02:31 (-81.2%)	00:02:27 (-81.7%)	00:02:22 (-82.3%)	00:02:04 (-84.6%)
CD 2	04:17:48	09:35:17 (+123.2%)	02:07:17 (-50.6%)	00:58:35 (-77.3%)	00:54:08 (-79.0%)	00:49:51 (-80.7%)	00:48:32 (-81.2%)
IMDB+FD	00:14:06	00:33:10 (+135.2%)	00:23:26 (+66.2%)	00:06:25 (-54.5%)	00:02:32 (-82.0%)	00:02:19 (-83.6%)	00:02:18 (-83.7%)

TABLE 4
Performance achieved using the pruning method on real data. Time is shown in the format hh:mm:ss. XMLDup(np) refers to the non-pruned version.

selected (*best pf*). To choose the best pruning factor, we started with a value of 1 and performed decrements of 0.1, until the number of comparisons was the closets possible, but not lower, to the number of comparisons performed by the best feature combination (G3,G1). Due to space limitations, we show only all combinations of two feature groups that include Group 3. Group 3 was chosen because it yielded the best performance when used alone.

As Table 5 shows, the majority of feature combinations was able to significantly improve efficiency in the real world datasets, by reducing the number of comparisons without suffering a high decrease in recall. In fact, all combinations of features were able to maintain the recall values within 90% of the recall achieved by the lossless strategies.

In the artificial datasets, however, results were inferior. This is particularly evident in the *CD* dataset, which presented an absolute loss of about 30% in recall. This can be attributed to lack of training data. Besides being smaller than the real datasets, only three were available, against the larger four real world datasets. Our regression model was, thus, unable to capture the relation between attribute features and pruning factor values.

Finally, with the exception of Group 2, all the groups combined with Group 3 presented fairly good results, which shows the importance of the complementary information provided by the different features. Interestingly when using all features results were equally good. However, more comparisons where performed in general. Thus, there is a need to fine-tune the process of feature selection before applying our method.

5.4 Discussion

When compared against the DogmatiX state-of-the-art algorithm [5], XMLDup showed a consistent capacity to maintain higher precision scores until later recall values. These results were achieved in four different real world datasets, which shows that our algorithm can be effective in several contexts of real world scenarios.

Efficiency tests revealed that our network pruning strategy can benefit from considering nodes in a particular order, given by a predefined sorting heuristic. Indeed, in our experiments, we observed performance gains up to about 60% over the default, unsorted case. From the applied strategies, sorting by depth produced the best overall results for the tested datasets and can be employed with success in every type of object representation, providing that it has a somewhat nested structure. Sorting by distinctiveness and by average string size can be useful, for instance, in more flat datasets. Furthermore, combining the sorting by depth and the sorting by average string size heuristics revealed to be the most effective approach on the datasets we considered.

We can also conclude that our proposed lossless pruning strategy can achieve a very significant gain in efficiency, being able to accelerate the runtime by about 4 times. Even with lossy pruning factors we were able to maintain high precision scores, only suffering from a recall reduction of 9%, in the worst case. This indicates that this strategy can be very practical in some application contexts, such as object ranking [21], [22], where often the detection of only a certain amount of duplicates is acceptable.

When compared with DogmatiX, we observed that, using the lossless pruning factor XMLDup presented an improvement in performance up to 80%. When we used a pruning factor of 0.4, the worst case presented an improvement of about 72% and a loss of less than 10% of the duplicates.

The automatic pruning factor optimization provided consistent improvements in performance. Although we observed a higher loss of recall for the artificial datasets, the same was not observed in the real datasets. In addition, in all cases, the number of comparisons was always lower. Thus, when there is little knowledge of the database being processed, or when manually tuning the pruning factor is not viable, our method is appropriate.

Finally, we point out that some of the values in this section are different from previous results presented in [9] for the same datasets. This occurs because different configurations were used. Namely, a new parameter was used to force similarity measures to assume value 0 if they are below a given threshold, which varied between 0.6 and 0.7 in our experiments. This feature is particularly useful when applied to occurrences of multiple nodes of the same type (see CP4 in Table 1), since it avoids the similarity values to grow too fast, when comparing a large number of attributes.

6 Conclusions

In this paper we presented a novel method for XML duplicate detection called XMLDup. Our algorithm uses a Bayesian Network to determine the probability of two XML objects being duplicates. The Bayesian Network

	Restaurants		Cora		CD 2		IMDB+FD		Country		CD		IMDB	
Baseline	*Recall*	*Cmp*	*Recall*	*Cmp*	*Recall*	*Cmp*	*Recall*	*Cmp*	*Recall*	*Cmp*	*Recall*	*Cmp*	*Recall*	*Cmp*
pf=1	99	1158551	99	71881813	80	1289763000	97	391739723	99	2050762	99	7097755	100	39368351
best pf	98	396576	98	60414848	74	109242306	97	211046654	99	1451553	99	3893111	100	27418945
Features	-	-	-	-	-	-	-	-	-	-	-	-	-	-
All features	96	373793	95	54984780	77	200780264	94	140495570	87	1533476	62	2769574	81	14766592
G3	95	373530	93	54274778	80	1102304184	93	104299301	56	814902	68	3776193	88	26619244
G3, G1	96	373693	93	37132454	73	108137466	94	144201738	85	1449369	68	3120126	82	17749878
G3, G2	96	373925	93	53235078	80	1075517847	92	78335656	48	659794	59	3116117	88	26764834
G3, G4	95	373546	93	36482838	76	168803322	94	133667869	87	1515536	63	2939026	83	18695713
G3, G5	95	373524	94	54923936	77	176531664	95	163946325	83	1421253	70	3251755	80	13627625

TABLE 5
Recall scores and comparisons (*Cmp*) achieved by combining feature groups.

model is composed from the structure of the objects being compared, thus all probabilities are computed considering not only the information the objects contain, but also the way such information is structured. XMLDup requires little user intervention, since the user only needs to provide the attributes to be considered, their respective default probability parameter, and a similarity threshold. However, the model is also very flexible, allowing the use of different similarity measures and different ways of combining probabilities.

To improve the runtime efficiency of XMLDup, a network pruning strategy is also presented. This strategy can be applied in two ways. A lossless approach, with no impact on the accuracy of the final result, and a lossy approach, which slightly reduces recall. Furthermore, the second approach can be performed automatically, without needing user intervention. Both strategies produce significant gains in efficiency over the unoptimized version of the algorithm.

Experiments performed on both artificial and real world data showed that our algorithm is able to achieve high precision and recall scores in several contexts. When compared to another state-of-the-art XML duplicate detection algorithm, XMLDup consistently showed better results regarding both efficiency and effectiveness.

The success demonstrated in experimental results leaves motivation for future work. Among other tasks we intend to extend the BN model construction algorithm to compare XML objects with different structures and apply machine learning methods to derive the conditional probabilities and network structure, based on the existing data.

REFERENCES

[1] E. Rahm and H. H. Do, "Data cleaning: Problems and current approaches," *IEEE Data Engineering Bulletin*, vol. 23, pp. 3–13, 2000.

[2] F. Naumann and M. Herschel, *An Introduction to Duplicate Detection*. Morgan and Claypool, 2010.

[3] R. Ananthakrishna, S. Chaudhuri, and V. Ganti, "Eliminating fuzzy duplicates in data warehouses," in *Conference on Very Large Databases (VLDB)*, Hong Kong, China, 2002, pp. 586–597.

[4] D. V. Kalashnikov and S. Mehrotra, "Domain-independent data cleaning via analysis of entity-relationship graph." *ACM Transactions on Database Systems (TODS)*, vol. 31, no. 2, pp. 716–767, 2006.

[5] M. Weis and F. Naumann, "Dogmatix tracks down duplicates in XML," in *Conference on the Management of Data (SIGMOD)*, Baltimore, MD, 2005, pp. 431–442.

[6] L. Leitão, P. Calado, and M. Weis, "Structure-based inference of XML similarity for fuzzy duplicate detection," in *Proceedings of the 16th ACM international conference on Information and knowledge management*, 2007, pp. 293–302.

[7] A. M. Kade and C. A. Heuser, "Matching XML documents in highly dynamic applications," in *ACM Symposium on Document Engineering (DocEng)*, 2008, pp. 191–198.

[8] D. Milano, M. Scannapieco, and T. Catarci, "Structure aware XML object identification," in *VLDB Workshop on Clean Databases (CleanDB)*, Seoul, Korea, 2006.

[9] P. Calado, M. Herschel, and L. Leitão, "An overview of XML duplicate detection algorithms," *Soft Computing in XML Data Management, Studies in Fuzziness and Soft Computing*, vol. 255, 2010.

[10] S. Puhlmann, M. Weis, and F. Naumann, "XML duplicate detection using sorted neigborhoods," in *Conference on Extending Database Technology (EDBT)*, Munich, Germany, 2006, pp. 773–791.

[11] S. Guha, H. V. Jagadish, N. Koudas, D. Srivastava, and T. Yu, "Approximate XML joins," in *Conference on the Management of Data (SIGMOD)*, 2002.

[12] J. C. P. Carvalho and A. S. da Silva, "Finding similar identities among objects from multiple web sources," in *CIKM Workshop on Web Information and Data Management (WIDM)*, New Orleans, Louisiana, USA, 2003, pp. 90–93.

[13] R. A. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[14] M. A. Hernández and S. J. Stolfo, "The merge/purge problem for large databases," in *Conference on the Management of Data (SIGMOD)*, San Jose, CA, 1995, pp. 127–138.

[15] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of plausible inference*, 2nd ed. Morgan Kaufmann Publishers, 1988.

[16] L. Leitão and P. Calado, "Duplicate detection through structure optimization," in *Proceedings of the 20th ACM international conference on Information and knowledge management*, 2011, pp. 443–452.

[17] E. H. Simpson, "Measurement of diversity," *Nature*, vol. 163, p. 688, 1949.

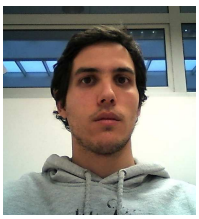
[18] H. Drucker, C. J. Burges, L. Kaufman, A. Smola, and V. Vapnik, "Support vector regression machines," *Advances in Neural Information Processing Systems (NIPS)*, vol. 9, pp. 155–161, 1996.

[19] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.

[20] T. Joachims, *Making large-scale support vector machine learning practical*. Cambridge, MA, USA: MIT Press, 1999, pp. 169–184.

[21] Z. Nie, Y. Zhang, J.-R. Wen, and W.-Y. Ma, "Object-level ranking: bringing order to web objects," in *International conference on World Wide Web (WWW)*, 2005, pp. 567–574.

[22] L. Chen, L. Zhang, F. Jing, K.-F. Deng, and W.-Y. Ma, "Ranking web objects from multiple communities," in *Proceedings of the 15th ACM international conference on Information and knowledge management*, 2006, pp. 377–386.



Luís Leitão studied Computer Engineering at the Superior Technical Institute of the Technical University of Lisbon. He received his MS degree in 2007, after finishing his master thesis on duplicate detection in XML databases. After working for the industry in between degrees, he started in 2008 his PhD studies at the Superior Technical Institute in Lisbon, Portugal.

Pável Calado received a degree in Computer Engineering from the Superior Technical Institute of the Technical University of Lisbon. In 2000 he received an MS degree in Computer Science from the Federal University of Minas Gerais (UFMG), where he also obtained his PhD in 2004. He is currently an assistant professor at the Superior Technical Institute and a researcher at INESC-ID, in Lisbon.

Melanie Herschel graduated in Information Technology from the University of Cooperative Education Stuttgart in 2003. She performed research on duplicate detection at the Humboldt-University of Berlin and at the Hasso Plattner Institute in Potsdam. She completed her PhD thesis in 2007 and from 2008 to 2009 worked at the IBM Almaden Research Center. Since 2009, she is a post-doctoral researcher at the University of Tubingen, Germany.