Relationship-Based Duplicate Detection

Melanie Weis and Felix Naumann
Institut für Informatik, Humboldt-Universität zu Berlin
Unter den Linden 6, D-10099 Berlin, Germany
{mweis,naumann}@informatik.hu-berlin.de

Abstract

Recent work both in the relational and the XML world have shown that the efficacy and efficiency of duplicate detection is enhanced by regarding relationships between ancestors and descendants. We present a novel comparison strategy that uses relationships but disposes of the strict bottom-up and top-down approaches proposed for hierarchical data. Instead, pairs of objects at any level of the hierarchy are compared in an order that depends on their relationships: Objects with many dependants influence many other duplicity-decisions and thus it should be decided early if they are duplicates themselves. We apply this ordering strategy to two algorithms. RECONA allows to re-examine an object if its influencing neighbors turn out to be duplicates. Here ordering reduces the number of such re-comparisons. ADAMA is more efficient by not allowing any re-comparison. Here the order minimizes the number of mistakes made.

1 Introduction

Duplicate detection is the problem of determining that different representations of entities actually represent the same real-world object. Duplicate detection is a necessary task in data cleansing [9, 16] and is relevant for data integration [6], personal information management [7], and many other areas. The problem has been studied extensively for data stored in a single relational table with sufficient attributes to make sensible comparisons. However, much data comes in more complex structures, so conventional approaches cannot be applied. For instance, within XML data, XML elements may lack any text. However, the hierarchical relationships with other elements potentially provide enough information for meaningful comparisons. The problem of XML duplicate detection is particularly tackling in applications like catalog integration or online data cleansing.

In this paper, we present a duplicate detection approach for XML data, which, unlike the common top-down and bottom-up approaches, performs well in presence of all kinds of relationships between entities, i.e., 1:1, 1:n, and n:m. The basic idea has been previously outlined in a poster [20]. Basically, we consider an object to depend on another object if the latter helps finding duplicates of the first. For example, actors help find duplicates in movies, so movies *depend* on actors, and actors *influence* movies. Due to mutual dependencies that can occur, detecting duplicates of one XML element helps find duplicates of the other, and vice versa. Therefore, algorithms such as [7] use dependencies to increase effectiveness by performing pairwise comparisons more than once. The focus of this paper lies on the efficient exploitation of dependencies between entities. The contributions are summarized as follows.

Comparison order. We propose a comparison order that reduces the number of necessary re-comparisons, while obtaining the same, optimal result as any other order. It is based on the estimated number of re-comparisons of neighboring elements that become necessary if the pair indeed were a duplicate.

Algorithm RECONA. The *re-examining algorithm* (RECONA), allows a pairwise comparison to be performed more than once, and the proposed comparison order reduces the number of re-comparisons. Experiments show that we save up to 90% of re-comparisons, compared to other orders, hence it is suited to increase efficiency.

Algorithm ADAMA. The second algorithm, ADAMA, does not allow comparisons to be performed more than once, so it is essential to perform the initial comparisons in an order that misses few re-comparisons and thus obtain good effectiveness. In experiments our order yields up to 10% higher f-measure than other orders.

Early classification is an extension to the algorithms, which computes an upper and a lower bound for the similarity measure and can thereby classify a pair as duplicate or non-duplicate without expensively computing the actual similarity. This extension saves up to 35% of pairwise similarity computations.

Constraint enforcement is an extension, which prunes comparisons by introducing constraints. More specifically, it prunes comparisons between elements that do not share a common related element, a valid constraint in 1:1 and 1:n relationships, thereby restoring a top-down approach where appropriate. Applied to real-world data up to 99% of pairwise comparisons are pruned.

We restrict the discussion to XML data, but the proposed algorithms apply to other types of data stored in complex schemas, such as normalized relational data, as well.

The paper is organized as follows: Related work is discussed in Sec. 2. In Sec. 3, we provide preliminaries, including definitions, the data structure we use and a motivating example. We show that the order of comparisons is relevant and define a suited comparison order in Sec. 4. This order is applied to ADAMA and RECONA in Sec. 5. Next, we present the two extensions in Sec. 6. In Sec. 7, we evaluate our approaches before we conclude in Sec. 8.

2 Related Work

The problem of identifying multiple representation of a same real-world object, originally defined by Newcombe et al. [14], was first formalized by Fellegi and Sunter [8]. Since then, the problem has been addressed in a large body of work. Ironically, it appears under numerous names itself, such as record linkage [21], merge/purge [10], object matching [9], object consolidation [5], and reference reconciliation [7] to name just a few.

Broadly speaking, research in duplicate detection falls into two categories: techniques to improve effectiveness and techniques to improve efficiency. Research on the former problem is concerned with improving precision and recall, for instance by developing sophisticated similarity measures. Examples are [7] and [18], where the relationships among objects are used to improve the quality of the results. Research on the second problem assumes a given similarity measure and develops algorithms that try to avoid having to apply the measure to all pairs of objects. An example is the sorted neighborhood method, which trades off effectiveness for higher efficiency by comparing only objects within a certain window [10]. The main contribution of this article falls into this second category, i.e., we assume a given similarity function and find an efficient order of comparison. In a first algorithm, this order is used to obtain the same effectiveness (same results) as any order, but faster; in a second variant we further enhance efficiency but at the price of missing some duplicates. Both our extensions further reduce the number of comparisons.

DATA MODEL	DETECTION APPROACH		
	Learning	**Clustering**	**Iterative**
Graph	Singla05[17] (Q)	Chen05[5](Q) Lee05[12](Q,T) Bhatt.05[2](Q,T)	Dong05[7](Q) **RECONA/ADAMA(T)**
Tree			Anant.02[1](Q,T) Weis04[18](Q,T) Puhlmann06[15](T)
Table	Bilenko03[3] (Q) Doan03[6] (Q)		Hernandez95[10](T) Monge97[13](T) Chaud.05[4](Q, T) Jin03[11](T)

Table 1: Summary of duplicate detection approaches (T: focus on efficiency, Q: focus on effectiveness)

In Tab. 1, we summarize several duplicate detection methods including those presented in this paper, classifying them along the two dimensions *data model* and *detection approach*. For the data model we distinguish (i) data in a single relation, without multi-valued attributes, (ii) hierarchical data such as the hierarchical organization of data warehouse tables or XML data, and (iii) data represented as a graph, e.g., XML with keyrefs or personal information management (PIM) data [7]. The second dimension discerns between three approaches that have been used to perform duplicate detection: (i) machine learning, where models and similarity measures are learned, (ii) the use of clustering algorithms, and (iii) iterative algorithms, which iterate over the data to detect pairs of duplicates, which in turn are aggregated to clusters in a post-processing step, e.g., using transitive closure. Tab. 1 also shows whether an article focuses on efficiency (T), effectiveness (Q), or both. We do not discuss all approaches in detail but we refer readers to [22] for a broad survey. Instead, we limit the discussion to the approaches in [1] and [15], which—like this paper—iteratively exploit relationships while having efficiency in mind. We also discuss [7], which proposes an effective iterative duplicate detection algorithm in graph data.

Using a *top-down approach* DELPHI [1] regards not only the immediate attributes of objects, but also their children and parents in a complex warehouse schema. For instance, in the three sample `<movie>` elements in Fig. 1 one may detect that all three are duplicates despite their different titles, using descendant actors to compare movies, thereby increasing effectiveness. Within the top-down approach, efficiency is improved by limiting subsequent comparisons in descendants to descendants that have same or duplicate ancestors.

```
<movie>          <movie>          <movie>
  <title>          <title>          <title>
    Troy             Troja            The Illiad Project
  </title>         </title>         </title>
  <actors>         <set>
    <name>           <actor>          <actor>
      Brad Pitt        Brad Pit         Prad Pitt
    </name>          </actor>         </actor>
    <name>           <actor>
      Eric Bana        Erik Bana
    </name>          </actor>
                     <actor>          <actor>
                       Brian Cox        Brian Cox
                     </actor>         </actor>
  </actors>        </set>
</movie>         </movie>         </movie>
```

Figure 1: Sample XML elements

This pruning technique is based on the assumption that two different movies do not share actors, a valid assumption if a 1:n relationship between movies and their actors held. However, actors usually star in several movies, that is, the entities movie and actor are in an m:n relationship. In this case, when detecting duplicates in actors, actors starring in different movies should also be compared. SXNM [15] is a *bottom-up approach* that first detects duplicates at leaf level and prunes comparisons in ancestors if their children sets are not significantly similar. SXNM is efficient by using a sliding window approach (initially proposed in [10]) over comparable elements on the same hierarchy level. The bottom-up approach can thereby efficiently detect duplicates among actors of different movies. However, the pruning of comparisons in parents is efficient only if a 1:n relationship holds between parent and child. The work presented in this paper overcomes the limitation of a strict bottom-up or top-down approach.

Dong et al. perform duplicate detection in the PIM domain by using relationships to propagate similarities from one duplicate classification to another [7]. The main focus of their approach is the increase of effectiveness by using relationships. In contrast, we concentrate on increasing efficiency by using relationships. Before describing our approach in detail we give some definitions and present an example of our approach.

3 Preliminaries

3.1 Definitions

In our approach it is important to distinguish "candidates" and "object descriptions", which define what entities are subject to comparisons and what data to use for these comparisons, respectively. These definitions are used to define "influencing and dependent elements" of a given candidate. Afterwards, we formalize the types of relationships between entities. Last, we provide a template for a similarity measure that is used to classify pairs of candidates as duplicates or non-duplicates.

CANDIDATES AND OBJECT DESCRIPTIONS. In an XML document every XML element is considered to represent an object, and every object can (but is not required to) be subject to duplicate detection, in which case it is called a *candidate*. To define a set of candidates, we specify a set of XPaths $P = \{p_1, p_2, ..., p_k\}$. Every path expression $p_i, 1 \leq i \leq k$ evaluates to a sequence of elements E_i. The set of candidates is then defined as $\bigcup_{1 \leq i \leq k} E_i$. Every candidate is assigned an *object description (OD)*, which comprises all information considered during comparisons. We define ODs using queries parameterized by the candidate. Details are beyond the scope of this paper, for which we refer readers to [19].

As a simple example, consider the XML elements of Fig. 1. We decide that movies, titles, and actors (represented by `<actor>` and `<name>` elements) are candidates. As Tab. 2 shows, movies are described by their title and their descendants representing actors. Duplicate titles, in turn, can be recognized by the movie they are subelement of and by their text node. Duplicate actors are determined by their text nodes alone.

Candidate	OD
`movie`	`./title`, `.//actor union .//name`
`movie/title`	`../movie`, */text()*
`movie//actor, movie//name`	*/text()*

Table 2: Sample OD definition

INFLUENCING AND DEPENDENT ELEMENTS. Two candidates are compared by comparing their ODs, so we say that the candidates are influenced by the elements in their ODs. Formally, we define influencing elements as follows.

Definition 1 *Let c be a candidate instance and let $od(c)$ be the set of description instances of c. Then, the set $I(c)$ of* influencing elements of c *is $I(c) := od(c)$.*

We define dependent elements similarly.

Definition 2 *Let c be a candidate instance; let $od(c)$ be the set of description instances of c. Then, the set $D(c)$ of* dependent elements of c *is $D(c) := \{c' | c' \neq c \wedge c \in od(c'))\}$.*

ENTITY RELATIONSHIPS. Up to now, we have provided only an intuitive definition of 1:n and m:n relationships between entities. In this section, we give a more concise definition.

Definition 3 *Two entities t and t' are in a* 1:n relationship *if for a given instance of t there can be 1 to n influencing instances of t' and a particular instance of t' influences exactly one instance of t.*

For example, `<movie>` and `<title>` are in a 1:n relationship because a movie can have multiple alternative titles, and the same title element can belong only to one movie. Opposed to that, `<movie>` and `<actor>` are in a m:n relationship, as defined below.

Definition 4 *Two entities t and t' are in an* m:n relationship *if for a given instance of t there can be 1 to n influencing instances of t' and if a particular instance of t' influences 1 to m instances of t.*

THRESHOLDED SIMILARITY MEASURE. To classify pairs of candidates as duplicates or non-duplicates, we use a thresholded similarity approach. The details of the actual similarity function are not relevant in

this paper; here we provide a general template for the similarity measure that will be useful later on when defining the early classification in Sec. 6. In our experiments, we use the similarity measure defined in [19] that complies to this template.

First, we define the set $N_{inf}^{\approx}$ of duplicate influencing neighbors of a pair of candidates (c, c') as

$$N_{inf}^{\approx}(c, c') := \{(n_1, n_2) | n_1 \in I(c) \wedge n_2 \in \cup I(c') \wedge n_1, n_2 \text{ duplicates}\}$$

The set $N_{inf}^{\neq}$ of non-duplicate influencing neighbors is

$$N_{inf}^{\neq}(c, c') := \{(n_1, \perp) | n_1 \in I(c) \wedge n_1 \text{ has no duplicates in } I(c')\} \cup \{(\perp, n_2) | n_2 \in I(c') \wedge n_2 \text{ has no duplicates in } I(c)\}$$

We further introduce a weight function $w(S)$ with S being a set of pairs (n_i, n_i') that captures the relevance of a pair of elements (n, n') when comparing two candidates c, c', $n \in I(c), n' \in I(c')$. This weight function has two properties:

$$w(S) = \sum_{(n_i, n_i') \in S} w(\{(n_i, n_i')\}) \tag{1}$$

$$w(\{(n, \perp)\}) + w(\{(\perp, n')\}) \geq w(\{(n, n')\}) \tag{2}$$

where $\perp$ signifies a null value, because n does not have a duplicate in this case. A simple example for such a weight function is $w(S) = |S|$.

The final similarity measure has the following template:

$$sim(c, c') = \frac{w(N_{inf}^{\approx}(c, c'))}{w(N_{inf}^{\neq}(c, c')) + w(N_{inf}^{\approx}(c, c'))} \tag{3}$$

Note that in defining $N_{inf}^{\approx}$ and $N_{inf}{\neq}$, we assume that we already know that n_1 and n_2 are duplicates or non-duplicates. Although we do not know these sets initially, the similarity measure is defined to increase as this knowledge is gained during comparisons, so we can classify two candidates as duplicates after their influencing neighbors have been classified.

The similarity measure obtains a result between 0 and 1. Given a user defined similarity threshold θ, if $sim(v, v') > \theta$, v and v' are duplicates, otherwise they are classified as non-duplicates.

3.2 The Data Graph

To support our algorithms and comparisons between objects we use an internal graph representation—the *data graph*. The data graph comprises three components, namely element vertices, text vertices, and dependency edges. More specifically, an element vertex v_e is created for every candidate. For every text node in an OD, we create a text vertex v_t. We represent influence and dependence between elements as directed edges in the graph. More specifically, let v and v' be two object vertices. We add an edge $e_d = (v, v')$ directed from v to v' if $v' \in I(v)$[1].

In the XML data of Fig. 1 it is obvious for human experts that all three movie elements represent a single movie (Troy) and there are three distinct actors (Brad Pitt, Eric Bana, and Brian Cox). For ease of presentation, we identify movies, titles and actors using m_i, t_i, and a_i, respectively. To mark duplicates, we use $'$, e.g., m_1, m_1' signifies that the movies are duplicates. Figure 2 depicts the data graph, assuming the OD definitions of Tab. 2. When considering the movie $m1$, the set of influencing neighbors $I(m1) = \{t1, a1, a2\}$, and the set of dependent neighbors $D(m1) = \{t1\}$.

[1] $I(v)$ denotes the set of influencing vertices of v, analogously to $I(c)$ that denotes the set of influencing elements of c. $D(v)$ is defined similarly.

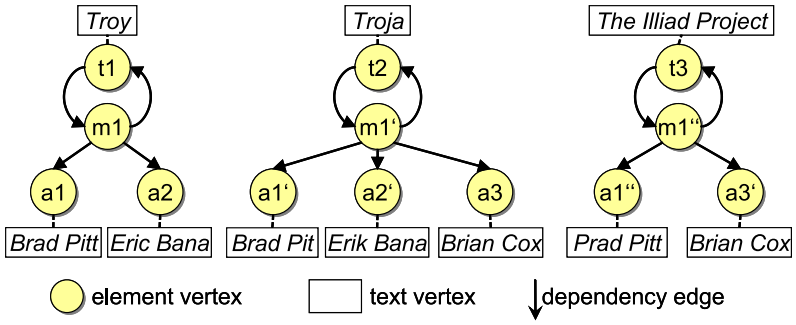


Figure 2: Sample data graph

3.3 Example

To give readers an intuition of the discussion that follows in the subsequent sections, we illustrate the benefit of re-comparisons and show the importance of comparison order on our example.

Assume that for pairwise comparisons, we decide to consider pairs of candidates in the order $\{(m1, m1'), (m1, m1''), (m1', m1''), (t1, t2), (t1, t3), (t2, t3), (a1, a2), ...\}$. When comparing $m1$ and $m1'$, they appear to have no related object in common because actors and titles have not yet been compared, so we conclude for now that they are not duplicates. The same is true for all other comparisons between movies and between titles. Continuing along the list we start to compare actors and find duplicates $(a1, a1')$, $(a1, a1'')$, $(a1', a1'')$, $(a2, a2')$, and $(a3, a3')$. Knowing that movies depend on their actors, the similarity of movies potentially increases due to shared actors. So we compare movies again and find that they are duplicates. As titles in turn are related to movies, we compare titles again, but do not find further duplicates. The point is that by re-comparing movies after duplicates in related objects have been detected, we were able to find duplicates where we could not before; hence, re-comparing pairs can increase effectiveness. In Sec. 5, we present two algorithms that are aware of the fact that due to cyclic dependencies between entities it is useful to classify pairs of objects more than once.

It is easy to see that if we had started by comparing actors, we would have saved re-classifying movies a second time. Clearly, the order in which comparisons are performed influences the number of necessary re-comparisons. Next we present a comparison order that reduces the number of necessary re-comparisons.

4 Comparison Order

The order is obtained by computing a rank $r(v, v')$ for every candidate pair (v, v'). Intuitively, $r(v, v')$ estimates for every pair of candidates v and v' the *number of re-comparisons* necessary if the similarity was calculated at the current processing state. We can reduce the total number of re-comparisons by comparing candidates in ascending order of r.

Calculation of r takes into account both how often a pair (v, v') is re-classified, estimated by $n_{self}(v, v')$, and how many comparisons of other pairs are triggered by pair (v, v') if v and v' were classified as duplicates, estimated by $n_{trig}(v, v')$.

ESTIMATING OWN RE-COMPARISONS ($\mathbf{n_{self}}$): The integer $n_{self}(v, v')$ is an upper bound to the number of times that the pair (v, v') is re-classified. Intuitively, the comparison of pairs with a high n_{self} value should be performed as late as possible, at best after all influencing neighbors have been correctly classified. The maximum number of re-comparisons of (v, v') occurs when every neighboring pair $(n_1, n_2), n_1\, n_2 \in I(v) \cup I(v')$ is a duplicate and triggers the re-comparison of (v, v').

Using additional knowledge about pairs that are already known to be duplicates and hence do not trigger further re-comparisons, the worst case is reduced to the case where any influencing neighbor pair (n_1, n_2) that has not been classified as duplicate is a duplicate and triggers a re-comparison of (v, v'). Following that reasoning we define n_{self}:

Definition 5 *Let v and v' be two object vertices and let $I(v)$ and $I(v')$ be their respective sets of influencing neighbors. Further, let $I^{\approx}_{(v,v')} = \{(n_1, n_2) | n_1, n_2 \in I(v) \cup I(v') \wedge (n_1, n_2)$ classified $\}$. Then*

$$n_{self}(v, v') := (|I(v)| - |I^{\approx}_{(v,v')}|) * (|I(v')| - |I^{\approx}_{(v,v')}|) \quad (4)$$

By classified we mean that the pair is never re-classified, either because it is a duplicate or because of other constraints. Further details are provided in Sec. 5 where we apply the order to two algorithms.

ESTIMATING TRIGGERED RE-COMPARISONS ($\mathbf{n_{trig}}$): We define n_{trig} in the same spirit as n_{self}. When comparing two object vertices v and v', the worst case is that all dependent neighbors not yet classified as duplicates need to be re-classified as the similarity of their influencing pair (v, v') increases. Pairs with a high n_{trig} value should be classified as late as possible, because they trigger many comparisons and re-comparisons.

Definition 6 *Let v and v' be two vertices, and let $D(v)$ and $D(v')$ be their respective sets of dependent neighbors. Further, let $D^{\approx}_{(v,v')} = \{(n_1, n_2) | n_1, n_2 \in D(v) \cup D(v') \wedge (n_1, n_2)$ classified $\}$. Then*

$$n_{trig}(v, v') := (|D(v)| - |D^{\approx}_{(v,v')}|) * (|D(v')| - |D^{\approx}_{(v,v')}|) \quad (5)$$

ESTIMATING RANK ($\mathbf{r(v, v')}$): We use the two estimates above to calculate the estimate of the total number of re-comparisons $r(v, v')$ as

$$r(v, v') := n_{self}(v, v') + n_{trig}(v, v') * n_{self}(v, v') \quad (6)$$

The intuition behind this is that each time a pair (v,v') may be reclassified, it may trigger n_trig re-comparisons.

Applying this estimate r to the pairs of candidates used in our example, we obtain the initial order shown in Tab. 3. Note that in general, pairs are not necessarily sorted by type.

pair	h
$(a1, a2)$	$0 + 1 * 0 = 0$
$(a1, a1')$	$0 + 1 * 0 = 0$
...	...
$(t1, t2)$	$1 + 1 * 1 = 2$
...	...
$(m1, m1'')$	$9 + 9 * 1 = 18$
$(m1, m1')$	12 + 12*1 = 24
$(m1', m1'')$	12 + 12*1 = 24

Table 3: Example for initial order

5 Comparison Algorithms

We present two algorithms, RECONA and ADAMA, which use the ascending order of r and exploit relationships between objects for the purpose of duplicate detection. RECONA re-computes similarities between two objects whenever there is a potential increase in similarity due to classified duplicates in their influencing sets, until the result converges. In the case of ADAMA, re-comparisons are altogether avoided to further increase efficiency.

In choosing the ascending order of r as comparisons order, we basically want to compare objects first that have few dependent objects. In that case, if two objects are detected to be duplicates, the ripple effect to neighbors is low. Thus, a carefully chosen order helps to avoid re-comparisons in RECONA, improving efficiency. For ADAMA a carefully chosen order improves effectiveness, because comparison of objects that are highly dependent on the status of their neighbors are postponed until the status of their neighbors has been determined.

Please note that in the following, text vertices are not mentioned explicitly when discussing the algorithms. Of course, whenever we have text available for a comparison, we use it by computing its similarity using edit distance in addition to dependencies, the main focus of our discussion.

5.1 RECONA: Allowing Re-comparisons

RECONA is based on the observation that detecting duplicates to an object may affect similarity and duplicate classification on other objects. It is an exact algorithm for solving the duplicate detection problem, in that it guarantees to find *all* duplicates as defined by the similarity measure and the ODs.

The algorithm has two phases (Listing 1). The initialization phase (lines 2-10) defines a priority queue *OPEN*, which contains all pairs of candidates. The priority order of *OPEN* is the ascending order of r. *DUPS* is a set of candidate pairs and is used to keep track of found duplicates to avoid unnecessary recomparisons, and to compute the rank r where the set of classified neighbor pairs (see Equations 4 and 5) is the set of neighbor pairs that are in $DUPS$. *CLOSED* is the set of possibly re-classified pairs, i.e., a set of pairs that have been classified as duplicates. Only these pairs can be added to *OPEN* again. For duplicate classification, we specify a similarity threshold θ. During the initialization phase, we also initialize the data graph G by reading the XML data.

The comparison phase (l.10-19) compares pairs of candidates using a similarity measure complying to the template of Sec. 3. As a reminder, to classify pairs of candidates as duplicates, we use a thresholded similarity approach, i.e., if $sim(v, v') > \theta$ they are classified as duplicates. For recomparisons, it is important to note that $sim(v, v')$ is a complex operation that increases with increasing co-occurrence in the influencing neighbor sets $I(v)$ and $I(v')$. Detected duplicates are added to *DUPS* (l.16) to ensure that duplicates are never revoked in subsequent comparisons. Consequently, similarity can only increase when re-calculated, and the algorithm always terminates. Non-duplicate pairs of dependent neighbors of v and v' are fed back into *OPEN* (l.17) according to the procedure $updateOpen(v, v')$ described in Listing 2, because their similarity may yet increase.

```
1  procedure ReconA()
2   G: data Graph;
3   OPEN:priority queue of candidate pairs
4       ordered in ascending order of r;
5   DUPS: set of duplicate pairs;
6   CLOSED: set of possibly re-classified pairs;
7   θ: similarity threshold;
8   Initialize G;
9   Add all candidate pairs to OPEN;
10  while OPEN not empty do
11  begin
12      (v_i, v_j) ← OPEN.popFirst();
13      sim = sim(v_i, v_j);
14      if sim > θ then
15      begin
16          DUPS := DUPS ∪ {(v_i, v_j)};
17          updateOpenReconA(v_i, v_j);
18      end
19  end
```

Listing 1: RECONA Algorithm

```
1  procedure updateOpenReconA(Vertex v,Vertex v')
2    D(v, v') = {(n_1, n_2)|n_1 ∈ D(v) ∧ n_2 ∈ D(v') ∧ n_1 ≠ n_2};
3    forall (n_1, n_2) ∈ D(v, v') do
4        if (n_1, n_2) not ∈ DUPS then
5        begin
6            r_update := r(n_1, n_2);
7            if (n_1, n_2) ∈ OPEN then
8                OPEN.updateRank((n_1,n 2), r_update);
9            else if (n_1, n_2) ∈ CLOSED then
10               OPEN.push((n_1, n_2), r_update);
11       end
```

Listing 2: Updating OPEN in RECONA

The procedure $updateOpenReconA(v, v')$ first determines the set of dependent neighbor pairs of v and v' (l.2). Pairs in *DUPS* are not added back to *OPEN*, because they are already known to be duplicates. If a pair is not in *DUPS*, we distinguish two cases. In the first case, the potentially added pair (n_1, n_2) is already in *OPEN*, because it has not been classified yet, and we merely update its position in the priority queue according to the newly calculated rank. This is required because the value of $r(n_1, n_2)$ depends on duplicates among $I(n_1)$ and $I(n_2)$, and the neighbor pair (v, v') has just been classified as duplicate. In case (n_1, n_2) is neither in *OPEN*, nor in *DUPS*, (n_1, n_2) is pushed into *OPEN*.

The complexity of RECONA is N in the best case, N being the size of *OPEN* at initialization. Note that it does not necessarily contain all possible pairs if a filtering technique is used prior to duplicate detection. The purpose of *CLOSED* is to avoid adding pairs to OPEN that were filtered before initialization. The average and worst case are $O(N^2)$.

Let us examine how RECONA behaves using r to sort *OPEN*. Using the initial order of Tab. 3, we first pop $(a1, a2)$ from *OPEN*. The pair is classified as non-duplicate. Next, we classify $(a1, a1')$, find that they are duplicates, and add the pair to *DUPS*. Now, we must add all pairs (n_1, n_2) of depending neighbors to *OPEN*, if they are not in *DUPS* already. To this end, we reestimate $r(n_1, n_2)$ and add (n_1, n_2) with a new rank $r(n_1, n_2)$ back to *OPEN* as defined in updateOpenReconA(). After comparing all actors and titles, we obtain, without any re-comparison yet,

$$\begin{aligned} \textit{OPEN} = \quad & \{(m1, m1'), r = 2), ((m1', m1''), r = 2), \\ & ((m1, m1''), r = 4)\} \\ \textit{DUPS} = \quad & \{(a1, a1'), (a1, a1''), (a1', a1''), (a2, a2'), (a3, a3')\} \end{aligned}$$

After this, we find movies to be duplicates and consequently have to reclassify their titles. We see that the number of re-comparisons using r is less than the number of re-comparisons necessary when using the arbitrary order of the example in Sec. 3.3, thus, duplicate detection is performed more efficiently while achieving the same, optimal effectiveness.

Note that re-estimating r resulted in reversing the relative order of $(m1, m1'')$ to the other movie pairs. Whereas in Tab. 3, $(m1, m1'')$ preceded $(m1, m1')$ and $(m1', m1'')$, it is now last in *OPEN*. Of course, reestimating r is costly, so we also examine in the experiments an order called r-static, which is independent of duplicates found.

5.2 ADAMA: No Re-comparisons

ADAMA is defined similarly to RECONA using the order of r, but with an important difference: We do not add candidate pairs to *OPEN* again once they have been classified, regardless of whether they were classified as duplicates or non-duplicates. Thus, no pairwise comparison is performed more than once (Listing 3).

```
1  procedure AdamA()
2      G, OPEN, t, sim, DUPS as in ReconA;
3      NONDUPS: set of non-duplicate pairs;
4      Initialize G;
5      Add all candidate pairs to OPEN;
6      while OPEN not empty do
7      begin
8          (v_i, v_j) ← OPEN.popFirst();
9          sim = sim(v_i, v_j);
10         if sim > θ then
11             updateOpenAdamA(v_i, v_j);
12         else
13             NONDUPS := NONDUPS ∪ {(v_1, v_j)};
14     end
```

Listing 3: ADAMA Algorithm

Note that in the initialization phase we declare another set of candidates, namely *NONDUPS*, which contains all pairs classified as non-duplicates. It is used to prevent recomparisons and to compute the rank r where the set of classified neighbor pairs is now defined as the neighbor pais not in $DUPS$ and not in $OPEN$. For ADAMA, the order in which *OPEN* is initially sorted is important to increase the duplicates found without re-comparisons. If we choose the poor order used in Sec. 3.3, we would find only duplicates among actors, because we do not reconsider movies. On the other hand, if we take the better order actors, titles, and then movies, we find all duplicates without re-computing any similarity. We see that when a pair's similarity is below the threshold θ, it is added to *NONDUPS* and is never reconsidered for re-comparisons as defined in *updateOpenAdamA()* (see Listing 4). Within *updateOpenAdamA()* we update only ranks of pairs that are still in *OPEN*. Clearly, by not allowing re-comparisons, the complexity of ADAMA is N.

```
1 procedure updateOpenAdamA(Vertex v, Vertex v')
2   D(v,v') = {(n1,n2)|n1 ∈ D(v) ∧ n2 ∈ D(v') ∧ n1 ≠ n2};
3   forall (n1,n2) ∈ D(v,v') do
4       if (n1,n2) not ∈ DUPS ∪ NONDUPS then
5       begin
6           r_update := r(n1,n2);
7           if (n1,n2) ∈ OPEN then
8              OPEN.updateRank((n1,n 2),r_update;
9       end
```

Listing 4: Updating OPEN in ADAMA

6 Extensions

Up to this point we have introduced two algorithms that perform pairwise comparisons between candidate duplicates. As mentioned earlier, filtering techniques can be used to reduce the size of the initial priority queue. In this section, we present two techniques to prune comparisons during the comparison phase of our algorithms to further increase efficiency by (i) avoiding computations of the complex similarity measure using early classification and (ii) avoiding pairwise comparisons using constraint enforcement.

6.1 Early Classification

Early classification calculates an upper and a lower bound to the similarity of two elements to classify pairs as duplicates and non-duplicates without actually computing the similarity measure.

In defining $N^{\approx}_{inf}$ and $N^{\neq}_{inf}$ for the similarity measure in Sec. 3 we assume that we already know for any influencing neighbors n_1 and n_2 whether they are duplicates or non-duplicates. In the above algorithms, we do not have that information at initialization, because n_1 and n_2 have to be compared first. However, we can approximate these sets during the comparison phase. Indeed, when coming to classify v and v', we have already performed classifications, and we can determine a set $S_{match}(v,v') \subseteq N^{\approx}_{inf}(v,v')$ of already classified influencing neighbor pairs that are duplicates, as well as a set $S_{nonmatch(v,v')} \subseteq N^{\neq}_{inf}(v,v')$. As comparisons proceed, S_{match} approaches the final $S_{\approx}$ and $S_{nonmatch}$ approaches $S_{\neq}$. We further know the remaining comparisons summarized in the set $S_{unclassified}$. Using these sets, we define an upper bound $f_u(v,v')$ and a lower bound $f_l(v,v')$ to $sim(v,v')$:

$$f_u = \frac{w(S_{match}) + w(S_{unclassified})}{w(S_{nonmatch}) + w(S_{match})} \tag{7}$$

$$f_l = \frac{w(S_{match})}{w(S_{nonmatch}) + w(S_{match}) + w(S_{unclassified})} \tag{8}$$

If $f_u \leq \theta$ we can classify the pair of objects as non-duplicates without further comparing influencing elements. If $f_l > \theta$ we can classify the pair of objects as duplicates without further comparing influencing elements. Early classification is an exact pruning technique with respect to any similarity measure complying to Equation 3, because the final classification is the same as the one obtained without pruning.

We now sketch a proof for the correctness of the upper and lower bound to sim. We know that $S_{match}(v,v') \subseteq N^{=}_{inf}(v,v')$, hence $w(S_{match}) \leq w(N^{=}_{inf})$. Furthermore, $S_{nonmatch} \subseteq N^{\neq}_{inf}$, so it is also true that $w(S_{nonmatch}) \leq N^{\neq}_{inf}$. According to the first property defined in Sec. 3.1, we also know that any pair that is in $N^{=}_{inf}$ but not in S_{match} results in a higher weight of $S_{unclassified}$ or $S_{unmatched}$, depending on where it is. However, it cannot be in both. Consequently,

$$\begin{array}{lll} & w(S_{match}) + w(S_{unclassified}) & \geq w(N^{=}_{inf}) \\ \text{and} & w(S_{nonmatch}) + w(S_{match}) & \leq w(N^{=}_{inf}) + w(N^{\neq}_{inf}) \\ \Rightarrow & f_u \geq sim & \end{array}$$

Similarly,

$$\begin{array}{rrl} & w(S_{unclassified}) & \\ & +w(S_{nonmatch}) & \\ & +w(S_{match}) & \geq w(N^{=}_{inf}) + w(N^{\neq}_{inf}) \\ \text{and} & w(S_{match}) & \leq w(N^{=}_{inf}) \\ \Rightarrow & f_l & \leq sim \end{array}$$

Implementing the early classification extension is straightforward. We do not have to compute bounds for similarities at initialization, because $S_{match} = \{\}$ and $S_{nonmatch} = \{\}$. Hence, we compute the range and check if we can perform early classification only when we consider adding a pair to *OPEN* again during the comparison phase. As we have seen earlier, adding a candidate pair (v, v') back to *OPEN* can either mean to update its rank $r(v, v')$ if (v, v') is already in *OPEN*, and if not, adding it again. Because an update of r can occur in RECONA and ADAMA, the filters can be applied to both algorithms. However, it is interesting to note that filtering using the upper bound is not applicable to RECONA. Indeed, $S_{nonmatch}$ is always empty because we are never sure that a pair is not re-classified, hence $S_{unclassified}$ contains all neighbors not in S_{match}. Except at initialization where $S_{match} = \{\}$ and f_u is thereby undefined, we note that $f_u \geq 1$ at any time, and therefore never filters any pair. In the case of ADAMA, $S_{nonmatch}$ is not always empty. If the filter f_u classifies a pair as non-duplicate, we add it to *NONDUPS*. For both algorithms, if a pair is classified as duplicate according to f_l, we have to add its depending neighbors to *OPEN* as well. Also note that the filter may have an influence on comparison order because as duplicates get classified their dependent neighbors' rank changes.

6.2 Constraint enforcement

We have seen in Sec. 3 that for 1:1 and 1:N relationships, it is true that there is exactly one possible child for a same parent or there is a set of children that can belong to only a single parent. In this scenario, efficiency can be greatly improved using a top-down approach[1]. To reestablish a top-down approach when applicable, we introduce an additional constraint in the data graph as a new type of edge.

Definition 7 *A* dependency edge e *directed from a vertex* s *to a vertex* t *is marked as* mandatory edge *if its source* s *requires that its target* t *is equal or similar between candidate duplicates.*

In our implementation we define a function *boolean isCandidate(v,v')*, which is called before every potential insertion into *OPEN*, i.e., at initialization and during comparisons. We first check whether any out-edge of v or v' is a mandatory edge. If there are mandatory out-edges, we verify that they share all targets. If they do, $isCandidate(v, v')$ returns true, otherwise false. If it returns false, the pair is not added to *OPEN*. In our experimental evaluation, we show the savings in terms of comparisons are very high.

7 Experiments

In this section, we evaluate the performance of our order compared to other orders in terms of efficiency (by reducing the number of pairwise comparisons in RECONA) and in terms of effectiveness (by missing few duplicates in ADAMA). Furthermore, we compare RECONA and ADAMA with each other in these two dimensions. Finally, we evaluate early classification and constraint enforcement.

7.1 Three Data Sets

7.1.1 Artificial Data with Interdependencies

We generate artificial data, which has the advantage of permitting us to vary several parameters that we can take into account for our experiments on re-comparisons. More specifically, we evaluate the order defined by rank r, ADAMA, and RECONA as well as early classification on artificial data and we vary the interdependence that exists between candidates. Furthermore, we can easily measure recall and precision because we know all true duplicates in the data.

We generate an artificial data graph consisting of two types of elements, namely <a> and <m>. Each <a> depends on its text node containing an actor name (extracted from a set of actor names from IMDB known to be non-duplicates), and a variable number of <m> elements that represent movies. Similarly, each <m> depends on its text node, generated to contain a movie title (again extracted from a set of titles from IMDB known to be non-duplicates), and a variable number of <a> elements. To generate dependencies between <a> and <m> elements, we use two parameters: (i) the interval i, which defines how many <m> elements have at least one <a> by the formula number of <m>$/i$, and (ii) the connection degree apm (short for "actors per movie"), defined as the average number of <a> elements influencing an <m> selected by i. Relationship edges between <m> and <a> are added in both directions, so that <m> depend on <a> and vice versa. To each of the clean vertices, we create a duplicate that can contain typographical errors or contradictory data, i.e., data not recognized as similar by our text similarity measure. The errors are not mutually exclusive. In the example shown in Fig. 3, we have $i = 1$ and $apm = 2$, meaning that every <m> has two associated <a>. In the generated duplicates, we have two out of eight text nodes with typographical errors, i.e., typos were introduced with 25% probability. The same is true for contradictory data.

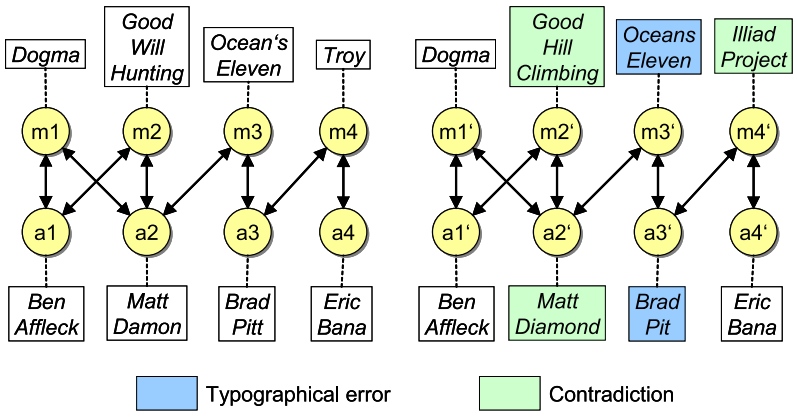


Figure 3: Sample artificial data graph

We generate two data sets using the graph generator. The first one, called *small data graph (SG)* consists of 400 vertices with non-empty text node from which 100 are clean actors, 100 are clean movies, and each is duplicated once. The *larger data graph (LG)* consists of 250 clean actors, 250 clean movies, and one duplicate of each, i.e., 1,000 vertices, none of which has an empty text node. Relationships between <a> and <m> nodes are varied using i and apm. In the experiments shown, typographical errors and contradictory data are introduced with a 20% probability. Note that using these data sets, both RECONA and ADAMA perform at least 39,800 (249,500) pairwise comparisons using SG (LG).

7.1.2 Real-World Data with Hierarchical dependencies

When re-comparisons are not the primary concern of the experiments, we can use real-word data that contains few interdependencies. The constraint enforcement extension, that allows us to enforce a top-down approach when applicable, can be applied on these data sets. For this purpose, we reuse two real-world data that we previously used in [19].

The first data set comes from the CD domain[2] and is referred to as CD. Dependencies (on schema level) are shown in Fig. 4(a). We extracted 1272 entities and verified manually that they contain no duplicates. We then added a duplicate to every entity, giving us a total number of 2544 entities to compare. Duplicates were artificially added to the clean CD data sample using a dirty XML data generator. This way, we can easily measure recall and precision.

[2] http://www.freedb.org/

We also use a data set from the movie domain, dubbed MOVIE, which exclusively consists of real-world data. It was obtained by integrating data from two movie data sources[3], and duplicates are due to two representations of a movie, each from one of the sources. The data set we consider here consists of 2140 entities and 8796 dependencies. The dependencies of the data set are depicted in Fig. 4(b) at schema level.

In both real-world data sets, we have some mutual dependencies, e.g., between movies and actors. However, due to the small size of the data set, a movie has 1:n actors but an actor is rarely associated with more than one movie. Hence, the number of potential re-comparisons is negligible. As for the other mutual dependencies, they can be defined because we have a 1:1 relationship between the elements, e.g., a track list exists for one CD only and a CD has only one track list, which again results in negligible number of recomparisons. Consequently, a top-down approach is suited to increase efficiency, and mandatory edges are added as depicted in Fig. 4(again at schema level). We assume that the domain-expert is capable of placing mandatory edges at appropriate positions in the OD schema.

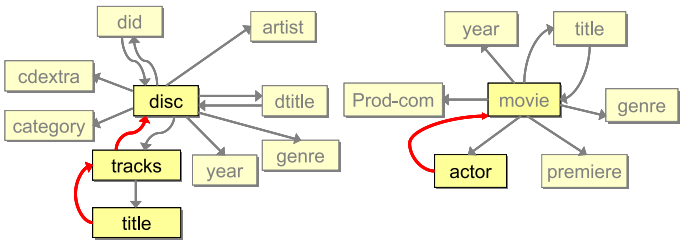


Figure 4: Relationships in MOVIE and CD

7.1.3 Real-world Data with Interdependencies

As real-world data with interdependencies, we used the hand-labeled Cora dataset, which is provided by McCallum and has previously been used by [7, 17] and others. The XML version of the data we created from the downloadable file[4] contains 1643 `<publication>` elements, which nest `<venue>`, `<title>`, and `<author>` elements. Venues in turn nest `<name>` elements. The elements refer to 174 distinct publications, i.e, a publication is cited 9.4 times on average. In addition to the readily labeled duplicate publications, we labeled roughly 30% of duplicate venues, as well. The venues being nested under publications, only a single publication influences a venue. We did not add foreign keys to relate a venue to other publications.

The schema of the data graph and the dependencies used throughout our experiments are shown in Fig. 5. We justify the choice of dependency edges as follows. Having labels for publications and venues only, we consider publications and venues to be the candidates of interest, whose interdependence is considered by having mutual dependency. The other entities are only considered to help find duplicates in publications. Once a duplicate has been detected in publications, we are not interested in finding more duplicates among titles and authors, because they will not trigger any other re-comparison of publications. Consequently, we do not add dependency edges back from title (author) to publication.

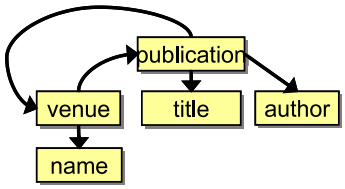


Figure 5: Relationships in CORA

[3] `http://www.imdb.com` and `http://film-dienst.kim-info.de/`
[4] http://www.cs.umass.edu/m̃ccallum/data/cora-refs.tar.gz

7.2 Competing Orders

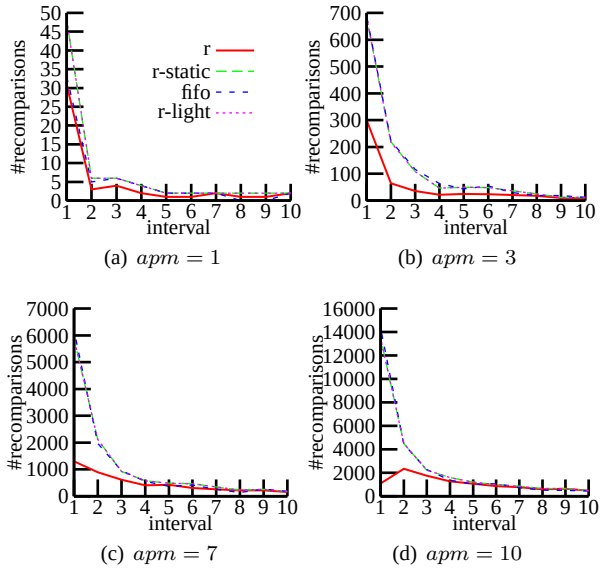
The first series of experiments aims at showing the influence of order in both RECONA and ADAMA.

In addition to the order defined by rank r, denoted `r`, we use the three additional orders of Tab. 4. The order `r-static` is defined similarly to `r`, but does not take into account previously detected duplicates. Hence, there is no need to update it when calling *updateOpen()* (both for ADAMA and RECONA). It basically keeps the order that `r` gives at initialization. `r-light` is defined as the order obtained when using the ascending order of $r_{light} = |D(v)| * |D(v')|$. It thus can be considered a static version of n_{trig} (or n_{self}, because the graph generates $|I(v)| = D(v)$). It is considered a small simplification of `r-static`. Finally, `fifo` simply takes comparable pairs (meaning of same schema type) in the order they appear in the input data. This is a fairly random order.

Order	Description
`r`	the order obtained using rank r
`r-static`	order r at initialization
`r-light`	order defined by $\|D(v)\| * \|D(v')\|$
`fifo`	first-in-first-out

Table 4: Comparison Orders

ORDER AND RE-COMPARISONS FOR RECONA. For RECONA we evaluate the number of re-comparisons that are performed using the different orders described in Tab. 4. Results for data set *SG* are shown in Fig. 6 for connection degrees of $apm = 1$, $apm = 3$, $apm = 7$, and $apm = 10$ ((a) through (d)).



(a) $apm = 1$ (b) $apm = 3$

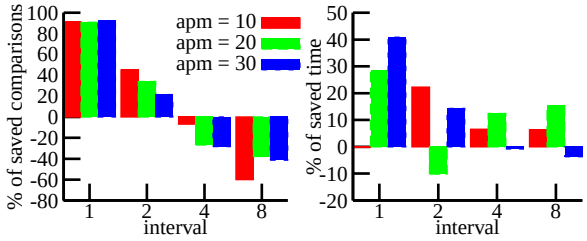
(c) $apm = 7$ (d) $apm = 10$

Figure 6: Recomparisons using RECONA

The graphs show the number of re-comparisons necessary for a given connection degree and a varying interval i. We observe that for large values of i, the number of re-comparisons is generally low but increases with increasing connection degree, meaning with decreasing i (as more `<m>` get actors) and increasing apm. This is easily explained by the fact that the higher i, the less mutual dependencies we have, so the less re-comparisons are possible. Similarly, the higher the connection degree, the higher the number of re-comparisons necessary when duplicates are detected. More interestingly, we also observe in Fig. 6 that order `r` performs significantly less re-comparisons than other orders when apm and i represent a significant degree of connectivity. Indeed, for low apm (a), we do not observe a significant difference, however as apm increases ((b) through (d)) the benefit of `r` over other orders increases for $i \leq 4$. This shows that `r` is a well suited order to reduce the number of recomparisons for RECONA. In this scenario, we also observe that

`r-static` does not provide a significantly better order than `r-light`. This can be explained by the fact that `r-static` orders candidate pairs similarly to `r-light`, because the in-degree and the out-degree of every vertex is generated to be equal. Hence, `r-static` is essentially (`r-light`)^2.

The same behavior can be observed on *LG*, as shown in Fig. 7. Again we observe that for small intervals, `r` performs better than `fifo`. At $i = 1$, the difference of 90% in the number of comparisons signifies that `r` performs only 10,667 re-comparisons instead of 115,260 (for $apm = 20$). However, the decrease in benefit of `r` over `fifo` is very steep, at intervals as small as 4, we already observe that `fifo` is better than `r`. Put in perspective, the -60% for 10 apm at $i = 8$ represent 530 re-comparisons only. For such small variations in the number of re-comparisons, we do not observe a significant difference in processing time, as Fig. 7(b) shows. For instance, the time difference for $i = 8$ is less than a second for 20 apm, in favor of `r`, opposed to the difference in re-comparisons where `fifo` performed better. On the other hand, for 30 apm and $i = 1$, we save 34 seconds by using `r`.



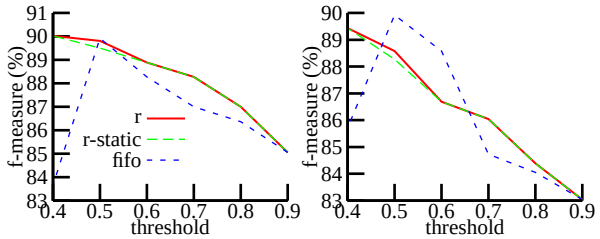
(a) Comparison difference (b) Time difference

Figure 7: Comparing `r` and `fifo` on *LG*

From this set of experiment we conclude that (i) the order defined by rank r outperforms other orders when the entities strongly depend on each other, which was simulated by using small intervals and large connection degrees, and (ii) when entities do not strongly depend on each other, the difference in terms of saved comparisons and hence efficiency is not significant.

ORDER AND EFFECTIVENESS FOR ADAMA. We reuse the same graph generator and orders as for RECONA and perform experiments on both *SG* and *LG*.

For *SG*, we have measured the f-measure for the different orders and intervals varying between 1 and 10 for a connection degrees of 10 actors per movie. We observe that `r` obtains highest f-measure in roughly 80% of all intervals tested for $1 \leq i \leq 4$. The interval between $i = 1$ and $i = 4$ is the interval where the difference in the number of missed re-comparisons using ADAMA is highest (e.g., Fig. 6(d) shows the difference for $\theta = 0.7$). As representative results, we show the graphs for $i = 1$ and $i = 4$ in Fig. 9. They show the f-measure obtained using ADAMA and different orders for different similarity thresholds((a) and (c)), and the corresponding difference of f-measure between order `r` and order `r-static`, as well as between `r` and `fifo`((b) and (d)).

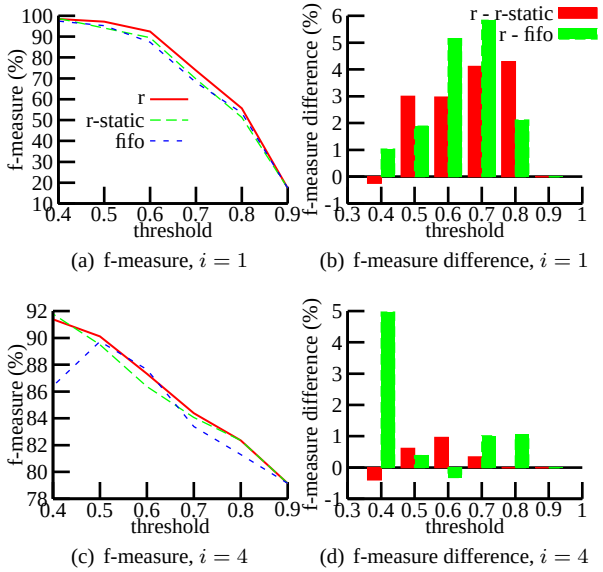


(a) f-measure, $i = 6$ (b) f-measure, $i = 7$

Figure 8: ADAMA Effectiveness (large i)

For $i > 4$ the orders miss roughly the same amount of re-comparisons. Representative results are shown in Fig. 8 for $i = 6$ (a) and $i = 7$ (b). We observe that the order `fifo` starts behaving indeterminably and is

sometimes better, sometimes worse that the other orders. We are currently investigating on the reason for this irregular behavior, and want to find out what makes fifo better in some cases. Opposed to that random behavior, we observe that `r` and `r-static` degrade gracefully.



(a) f-measure, $i = 1$ (b) f-measure difference, $i = 1$

(c) f-measure, $i = 4$ (d) f-measure difference, $i = 4$

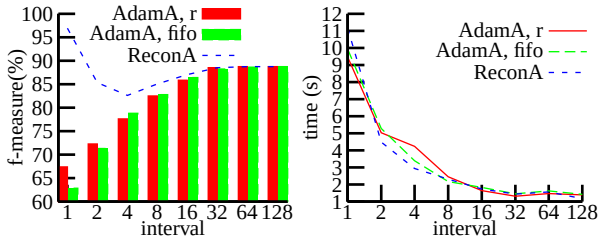
Figure 9: ADAMA Effectiveness (small i)

Throughout our experiments, we observed that using `r` as order obtains better f-measure in most cases where the number of missed re-comparisons when using ADAMA is significant. However, the benefit in f-measure obtained using `r` in ADAMA is less than the benefit in efficiency when using `r` in RECONA. For example, in Fig. 7, we see that for $apm = 20$ and $i = 1$, 91% of re-comparisons have been saved using `r` at threshold $\theta = 0.7$. In all experiments of effectiveness for ADAMA, we in fact never observed a benefit in effectiveness of using `r` over `fifo` larger than 10%. This shows that in the additional recomparisons other orders perform compared to r, there are not proportionally more duplicates.

COMPARISON OF ADAMA AND RECONA. We now evaluate ADAMA against RECONA in terms of effectiveness and efficiency. We expect that RECONA is more effective than ADAMA because it allows re-comparisons. On the other hand, the recomparisons represent costly comparisons and we expect ADAMA to be faster than RECONA. We again use *LG* and vary the connection degree defined by apm and the interval between movies having actors. In terms of effectiveness, RECONA obtains the same recall and precision for different orders (at different cost, though), which is higher or equal to the precision and recall obtained using ADAMA with any order. This is shown in Fig. 10(a), where the f-measure is plotted for orders `r` and `fifo` applied to ADAMA as well as for RECONA in function of the interval. The chosen similarity threshold is 0.7, and apm = 10.

We observe that the difference in f-measure between RECONA and ADAMA using `r` or `fifo` behaves similarly to the number of re-comparisons that have been missed. That is, for increasing interval, the difference between the result of ADAMA and RECONA decreases, just like the number of re-comparisons performed by RECONA and missed by ADAMA decreases with increasing interval.

Clearly, we obtain better effectiveness by using recomparisons. In Fig. 10(b), we show the time needed for RECONA and ADAMA using the same order r. We observe that the time is similar, meaning that a significant overhead due to re-comparisons for RECONA cannot be observed. This is explained by the fact that the number of re-comparisons performed by RECONA is a very small fraction of the number of pairwise comparisons performed by both RECONA and ADAMA. Indeed, to compare the 1,000 vertices in the graph, we have to perform 249,500 comparisons, so for example the 3,151 re-comparisons additionally performed by RECONA at $i = 1$ represent only 1% of additional comparisons.

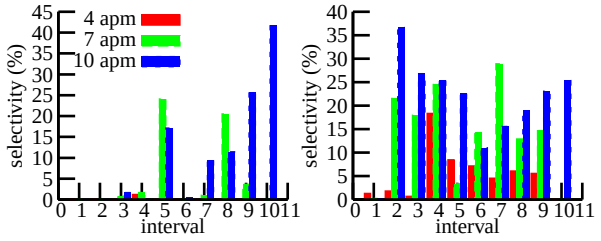


(a) f-measure comparison (b) time comparison

Figure 10: Comparison of ADAMA and RECONA

7.3 Early Classification

We evaluate our early classification technique on *SG*. In Fig. 11, we see the selectivity of the early classification extension obtained using the extension on RECONA (a) and ADAMA (b) with $apm = 10$, $\theta = 0.7$ and varying intervals. The selectivity is defined as the ratio of early classifications (that would otherwise have updated *OPEN*) to the total number of updates. Note that for RECONA, only the lower bound applies.



(a) RECONA (b) ADAMA

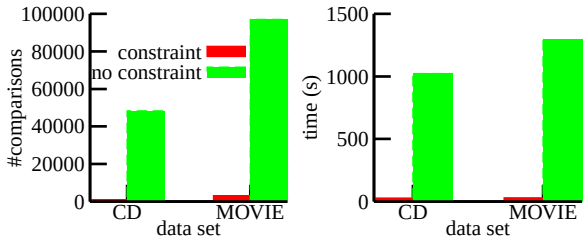
Figure 11: Early Classification

We observe that for increasing apm and decreasing interval (i.e., with increasing interdependence), the lower bound becomes less selective and soon reaches 0% selectivity. This has two reasons. First, the more apm, the less significant a duplicate in the set of descriptions gets. For example, at $apm = 1$, assuming a unit weight for any dependence, we have $f_l = 1/2$ when the actors are not known to be duplicates. (we consider the text node to be similar, hence, the only difference is the actor). After having identified that the actors match, $f_l = 1$, which represents an increase of 0.5 per duplicate found in influencing actors. For $apm = 2$, $f_l = 1/3$ and increases to $2/3$ when the actors are classified as duplicates, i.e., and increase of only $1/3$. Since f_l is still below the similarity threshold of 0.7, we would have to find the other pair of actors to be duplicates and to re-classify the movies a second time to be able to classify them as duplicates using f_l. This however, is avoided by the second reason: by using the order `r`, we avoid re-comparisons so the comparison of the above pair is postponed until the actors are classified. Hence, premature classification using f_l gets no chance to consider a pair before the classifier itself does it.

Let us now consider how early classification performs for ADAMA. In this case, we can use both the upper and the lower bound. We see that the selectivity of early classification on ADAMA increases with increasing apm and reaches selectivities of up to 35%. On this small data set, we did not observe significant differences in processing time (neither better nor worse) again explained by the fact that `r` provides a good ordering that generates few updates in *OPEN*.

7.4 Constraint Enforcement

The principle of increasing efficiency and possibly effectiveness has already been shown elsewhere[1, 18], here, we want to focus on the fact that the extension of constraint enforcement can enforce top-down and the ensuing efficiency gains when applicable. This is clearly shown in Fig. 12.



(a) Comparisons (b) Time

Figure 12: Effects of Constraint Enforcement

7.5 The CORA Dataset

We use the Cora data set to evaluate our approach against previous approaches. Due to the large number of element pairs, it is impractical to compare them all. We use a blocking technique to drastically reduce the number of pairwise comparisons to 1.9% of all possible comparisons, without pruning a significant number of true duplicates. The remaining pairs are compared both using RECONA and ADAMA, and order r. RECONA performs 34983 recomparisons, which represent 3.5% of comparisons performed by ADAMA.

Approach	Runtime	Recall Publication	Precision Publication	F-measure Publication	Recall Venue	Precision Venue	F-measure Venue
ADAMA	N	83.3 %	89.3 %	**86.2 %**	**16.8 %**	100 %	28.8 %
RECONA	$O(N^2)$	83.3 %	89.3 %	**86.2 %**	**88.7 %**	72.6 %	79.8 %
Standard [17]	N	89.7 %	85.3 %	**86.9 %**	**36.0 %**	75.4 %	48.6 %
Combined Learning [17]	Polynomial	86.1 %	87.1 %	**85.8 %**	**85.7 %**	88.7 %	86.5 %
Standard [7]	N	91.3 %	98.5 %	**94.8 %**	**36.9 %**	98.2 %	52.9 %
DEPGRAPH [7]	$O(N^2)$	92.4 %	98.5 %	**95.4 %**	**71.4 %**	83.7 %	77.1 %

Table 5: Comparison of Results on Cora dataset

In Tab. 5, we summarize the results of RECONA and ADAMA and results on the cora dataset reported in [17] and [7], two approaches that use relationships to increase effectiveness. Within these papers, the authors also compared their results to the standard approach that does not use relationships, which we name *Standard*. ADAMA performs the same number of comparisons as Standard, but takes care in choosing a good order for comparisons by using r. Bold results in Tab. 5 are part of the discussion below.

PUBLICATIONS. We observe that ADAMA and RECONA obtain the same result in terms of effectiveness when considering publications. That is, by choosing a suited comparison order, we are able to be as effective as more complex algorithms (runtime $> N$). The results are comparable to those in [17]. The results in [7] are better for both the standard approach and the approach using relationships. The standard approach already being better indicates that the similarity measure used is better than ours, because the comparisons that are performed are the same.

VENUES. Effectiveness for venues, measured on the 30% of venues we annotated is low compared to the effectiveness of other approaches that use the standard technique. This may be due to (i) a poor similarity measure, or (ii) the fact that we did not link a venue to all publications, but only to the publication it is nested in. That is, venues whose only publication has not been identified as duplicate are unlikely to be classified as a duplicate. By increasing the number of publications per venue, it is more likely that publications are shared, which increases their similarity. We suspect that the second reason is the cause, because the recall for venues obtained by RECONA outperforms the recall for venues of other approaches allowing re-comparisons. In that case, venues are simply re-compared as their related publication is classified as duplicate. The re-comparisons performed by RECONA only represent an overhead of 3.5 % compared to ADAMA, hence, efficiency is not compromised. These promising results will be verified on the complete data set in the near future.

8 Conclusion

In this paper we presented a novel duplicate detection approach for XML data, which, unlike the common top-down and bottom-up approaches, performs well in presence of all kinds of relationships between entities, i.e., 1:1, 1:n, and m:n. The comparison strategy we presented considers pairwise comparisons in ascending order of a rank, which estimates how many pairs must be reconsidered if the original pair was classified at the current processing state. We applied this strategy to two algorithms: RECONA uses the order to classify only few pairs more than once, whereas ADAMA does not perform re-comparisons and uses the order to miss few re-comparisons and therefore potentially few duplicates. We further presented two extensions to the algorithms: Early classification reduces the number of complex similarity computations by computing an upper and lower bound to the similarity measure that can be used as filters. Constraint-enforcement is used to benefit from a top-down strategy when applicable, by enforcing that influencing elements are required to be duplicates in order for a candidate pair to be duplicates.

Experiments showed that the order obtained using ascending r is very effective in reducing the number of re-comparisons for RECONA, given a high interdependency between entities. For low interdependency there are only few possible re-comparisons, so the difference between the orders is not significant for efficiency. When applied to ADAMA we observed that the order using r performs slightly better in terms of recall and precision than other orders for high interdependency. However, the benefit in effectiveness is less than the benefit of saved re-comparisons in RECONA, which shows that additional re-comparisons induced by other orders do not find proportionally more duplicates. When comparing RECONA and ADAMA we observe that the more re-comparisons are missed by ADAMA, the larger the difference is between the f-measure achieved by RECONAẆhen evaluating our extensions we observed that the selectivity of early classification is moderate, because it applies only to re-comparisons, which do not occur frequently when using order r. As for constraint enforcement, we showed that we can significantly improve efficiency by enforcing top-down when appropriate. On the real-world Cora dataset, we observed that our solution is competitive with other, possibly more complex solutions, and that re-comparisons do not jeopardize efficiency while greatly improving effectiveness.

Future work will include further validation both on the Cora dataset and a large real-world data set for which we are currently manually (and tediously) determining all duplicates. In order to apply our algorithms on this data, we will also address scalability issues. We will also further investigate on increasing effectiveness, which was not the goal of this research. An interesting research issue we plan to tackle is how to resolve the conflicts between nested XML duplicates to obtain clean XML data.

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

References

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

[2] I. Bhattacharya and L. Getoor. Relational clustering for multi-type entity resolution. *Workshop on Multi-Relational Data Mining (MRDM)*, 2005.

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

[4] S. Chaudhuri, V. Ganti, and R. Motwani. Robust identification of fuzzy duplicates. In *Proceedings of the International Conference on Data Engineering (ICDE)*, Tokyo, Japan, 2005.

[5] Z. Chen, D. V. Kalashnikov, and S. Mehrotra. Exploiting relationships for object consolidation. In *SIGMOD-2005 Workshop on Information Quality in Information Systems*, Baltimore, MD, 2005.

[6] A. Doan, Y. Lu, Y. Lee, and J. Han. Object matching for information integration: A profiler-based approach. *IEEE Intelligent Systems, pages 54-59*, 2003.

[7] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *International Conference on the Management of Data (SIGMOD)*, Baltimore, MD, 2005.

[8] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 1969.

[9] 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 (VLDB)*, pages 371–380, Rome, Italy, 2001.

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

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

[12] D. Lee, B.-W. On, J. Kang, and S. Park. Effective and scalable solutions for mixed and split citation problems in digital libraries. In *SIGMOD-2005 Workshop on Information Quality in Information Systems*, Baltimore, MD, 2005.

[13] 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.

[14] H. Newcombe, J. Kennedy, S. Axford, and A. James. Automatic linkage of vital records. *Science 130 (1959) no. 3381*, pages 954–959, 1959.

[15] S. Puhlmann, M. Weis, and F. Naumann. XML duplicate detection using sorted neigborhoods. *International Conference on Extending Database Technology (EDBT)*, 2006.

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

[17] P. Singla and P. Domingos. Object identification with attribute-mediated dependences. In *European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, Porto, Portugal, 2005.

[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] M. Weis and F. Naumann. Dogmatix tracks down duplicates in XML. In *International Conference on the Management of Data (SIGMOD)*, Baltimore, MD, 2005.

[20] M. Weis and F. Naumann. Detecting duplicates in complex XML data. In *International Conference on Data Engineering (ICDE)*, Atlanta, Georgia, 2006.

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

[22] W. E. Winkler. The state of record linkage and current research problems. Technical report, U. S. Bureau of the Census, 1999.