VLDB Journal manuscript No.
(will be inserted by the editor)

Creating Probabilistic Databases from Duplicated Data

Oktie Hassanzadeh · Renée J. Miller

Received: 14 September 2008 / Revised: 1 April 2009 / Accepted: 26 June 2009

Abstract A major source of uncertainty in databases is the presence of duplicate items, i.e., records that refer to the same real world entity. However, accurate deduplication is a difficult task and imperfect data cleaning may result in loss of valuable information. A reasonable alternative approach is to keep duplicates when the correct cleaning strategy is not certain, and utilize an efficient probabilistic query answering technique to return query results along with probabilities of each answer being correct. In this paper, we present a flexible modular framework for scalably creating a probabilistic database out of a dirty relation of duplicated data and overview the challenges raised in utilizing this framework for large relations of string data. We study the problem of associating probabilities with duplicates that are detected using state-of-the-art scalable approximate join methods. We argue that standard thresholding techniques are not sufficiently robust for this task, and propose new clustering algorithms suitable for inferring duplicates and their associated probabilities. We show that the inferred probabilities accurately reflect the error in duplicate records.

1 Introduction

The presence of duplicates is a major concern for the quality of data in large databases. To detect duplicates, *entity resolution* also known as *duplication detection* or *record linkage* is used as a part of the data cleaning process to identify records that potentially refer to the same entity. Numerous deduplication techniques exist to normalize data and remove erroneous records [42]. However, in many real world applications accurately merging duplicate records and fully eliminating erroneous duplicates is still a very human-labor intensive process. Furthermore, full deduplication may result in the loss of valuable information.

An alternative approach is to keep all the data and introduce a notion of *uncertainty* for records that have been determined to potentially refer to the same entity. Such data would naturally be inconsistent, containing sets of duplicate records. Various methodologies exist with different characteristics for managing uncertainty and inconsistency in data [2,3,15,22,51]. A large amount of previous work addresses the problem of efficient query evaluation on probabilistic databases in which it is assumed that meaningful probability values are assigned to the data in advance. Given these probabilities, a query can return answers together with a probability of the answer being correct, or alternatively return the top-k most likely answers. For such approaches to work over duplicate data, the record probabilities must accurately reflect the error in the data.

To illustrate this problem, consider the dirty relations of Figure 1. To assign probabilities, we must first understand which records are potential duplicates. For large data sets, a number of scalable approximate join algorithm exist which return pairs of similar records and their similarity scores (e.g., [4,8,38]). Given the result of an approximate join, we can group records into sets of potential duplicates using a number of techniques. The most simple technique is to group all records whose similarity is within some threshold value. We show that using simple thresholding with such techniques to determine groups (clusters) of duplicates often results in poor accuracy. This is to be expected as

Work supported in part by NSERC.

Department of Computer Science
University of Toronto
E-mail: {oktie,miller}@cs.toronto.edu

Company

tid	name	emp#	hq	cid	prob
t_1	Altera Corporation	6K	NY	$c1$	0.267
t_2	Altersa Corporation	5K	New York	$c1$	0.247
t_3	lAtera Croporation	5K	New York	$c1$	0.224
t_4	Altera Corporation	6K	NY, NY	$c1$	0.262
t_5	ALTEL Corporatio	2K	Albany, NY	$c2$	0.214
t_6	ALLTEL Corporation	3K	Albany	$c2$	0.208
t_7	ALLTLE Corporation	3K	Albany	$c2$	0.192
t_8	Alterel Coporation	5K	NY	$c2$	0.184
t_9	ALTEL Corporation	2K	Albany, NY	$c2$	0.202

Product

pid	product	tidFk	cidFk	cid	prob
p_1	MaxLink 300	t_1	$c1$	$c3$	0.350
p_2	MaxLink 300	t_8	$c2$	$c3$	0.350
p_3	MaxLnk 300	t_4	$c1$	$c3$	0.300
p_4	SmartConnect	t_6	$c2$	$c4$	1.0

Price

rid	product	price	cid	prob
r_1	MaxLink 300	\$285	$c5$	0.8
r_2	MaxLink 300	\$100	$c5$	0.2

Fig. 1 A sample dirty database with **Company**, **Product** and **Price** relations.

thresholding does not take into account the characteristics of the data or the duplicate detection task. To overcome this, we consider existing and new scalable clustering algorithms that are designed to produce high quality clusterings even when the number of clusters is unknown.

In Figure 1, the clustering is indicated by the cluster identifier in the `cid` attribute. Records that share a cluster identifier are potential duplicates. Once a clustering is determined, we consider how to generate probabilities. For our uncertainty model, we adopt the model of Andritsos et al. [2] and Dalvi and Suciu [22] called *disjoint-independent databases*. In this model, tuples within a cluster (potential duplicates) are mutually disjoint. Tuples in different clusters are independent. This reflects the intuition that errors are introduced for different (real-world) entities independently. So, the probability that t_1 (from Cluster c_1) is in the clean (deduplicated) database is independent of the probability of t_8 (from Cluster c_2) being in the clean database. An important motivation for our choice of uncertainty model is that efficient query answering techniques are known for large classes of queries over such databases, which is important since keeping duplicate information is only worthwhile if it can be queried and used effectively in decision making. As further motivation, the probabilistic databases we create, can be used as input to query evaluation techniques which model clustering uncertainty (that is the uncertainty introduced by the clustering process itself) [10]. We elaborate on this in Section 2.4.

We also consider clustering techniques that produce overlapping clusters. In this approach, records that have been assigned to multiple clusters are no longer independent. Such probabilistic databases require more complex query processing techniques which might be supported by the lineage mechanisms of systems like Trio [51] or world-set semantics of MayBMS [3].

To assign probabilities within a cluster, we follow the common wisdom in uncertain data management which has noted that record probabilities are mostly internal to the system and useful primarily for ranking answers [24, 43]. Hence, in this work, we do not consider different probability distributions within clusters, but focus instead on assigning confidence scores that accurately reflect the error in the records. That is, among a set of duplicate records, a record with less error should have a lower probability than a record containing more error.

1.1 Outline and Contributions

In this paper, we propose a flexible modular framework for scalably creating a probabilistic database out of a dirty relation of duplicated data (Section 2). This framework consists of three separate components. The input to the first component is a base relation R and the output of the third component is a probabilistic relation. Our framework complements and extends some existing entity resolution and approximate join algorithms, permitting their results to be used in a principled way within a probabilistic database management system. We study this framework for the case of string data, where the input relation consists of duplicated string records and no additional information exists or is usable to enhance the deduplication process. This in fact is the case in many real world problems.

For each component of our framework, we briefly overview the state-of-the art (Sections 2.1-2.3) to further describe the characteristics of our framework in comparison with other deduplication techniques. We also present a detailed discussion of query evaluation over probabilistic databases focusing on how the probabilistic databases we create can be used. We justify the scalability and adaptability of our framework and the need for thorough evaluation of the performance of each component. We perform this evaluation using a methodology (summarized in Section 2.5) heavily based on existing evaluation methods.

We present an overview of several string similarity measures used in state-of-the-art similarity join techniques and benchmark the accuracy of these measures

in our framework (Section 3). Unlike previous comparisons, we focus on measures useful for duplicate detection [33]. Given pairs of similar records, we present several clustering algorithms for string data suitable for our framework (Section 4).

We address the problem of assigning probabilities (confidence scores) to records within each cluster that naturally reflect the relative error in the record (Section 5). We present several algorithms based on a variety of well-performing similarity measures for strings, and an algorithm using the information bottleneck method [2,47] which assigns probabilities based on the relative information content of records within a cluster.

An important characteristic of our framework is its modularity with components that are reusable for other cleaning tasks. Hence, we thoroughly benchmark each component individually to evaluate the effectiveness of different techniques in terms of both accuracy and running time. For each component, we present a summary of the results of extensive experiments which used many datasets with different characteristics to ensure that our framework is robust. We used several existing and some novel measures of accuracy in our evaluations. We also present an end-to-end evaluation of the components when used together for creating a probabilistic database.

2 Framework

Figure 2 shows the components of our framework. The input to this framework is a base relation R and the output is a probabilistic relation. In this work, we focus on creating a framework using scalable algorithms that do not rely on a specific structure in the input relation R. There are duplicate detection algorithms that can take advantage of other types of input such as co-citation or co-occurrence information [13]. Such information may be available in bibliographic co-citation data or in social networks. However, we do not consider these specialized algorithms as such information is often not present in the data.

An important characteristic of our framework is its modularity. This makes our framework adaptable to other data cleaning tasks. As new deduplication techniques are developed, they may replace one or both of our first two components. Moreover, if the input relation contains additional information that can be used to enhance the accuracy of deduplication, these different methods may be used. Furthermore, by dividing the system into three separate modules, we are able to evaluate the performance of each module individually.

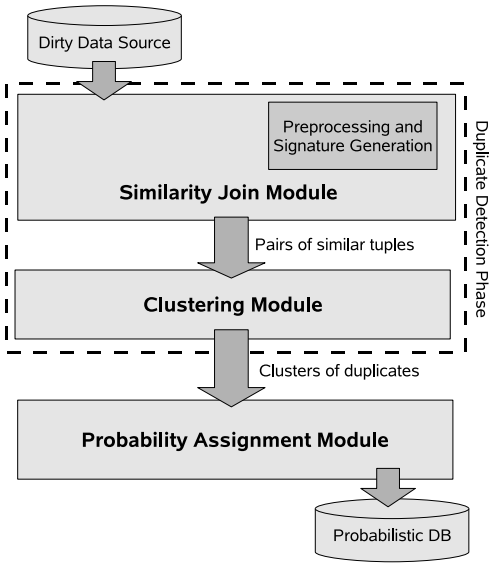


Fig. 2 Components of the framework

2.1 Similarity Join

The first component of the system is the similarity join module. The input to this module is a relation $R = \{r_i : 1 \leq i \leq N\}$, and the output is a set of pairs $(r_i, r_j) \in R \times R$ where r_i and r_j $(i < j)$ are *similar* and a similarity score for each pair. In existing join approaches, two records are considered similar when their similarity score based on a similarity function $sim()$ is above a threshold θ. Many join methods typically model records as strings. We denote by $\mathbf{r}$ the set of *q-grams* (sequences of q consecutive characters of a string) in r. For example, for t='db lab', $\mathbf{t}$={' d','db', 'b ' ,' l','la', 'ab', 'b '} for tokenization using 2-grams[1]. In certain cases, a *weight* may be associated with each token.

Similarity join methods use a variety of different similarity measures for string data [20,31]. Recently, there has been an increasing interest in using measures from the information retrieval field [4,16,28,31,45]. In [31], several such similarity measures are introduced and benchmarked for approximate selection where the goal is to sort the tuples in a relation based on their similarity with a query string. The extension of approximate selection to approximate join is not considered. Furthermore, the effect of threshold values on accuracy for approximate joins is also not considered. To fill in this gap, we show that the performance of the similarity predicates in a similarity join is slightly different (than in selection) mainly due to the effect of choosing a single threshold for matching all the tuples as opposed to

[1] Strings are first padded with whitespaces at the beginning and the end, then all whitespaces are replaced with $q-1$ occurrences of special unused symbol (e.g., a \$).

ranking the tuples and choosing a different threshold for each selection query.

Our work is motivated by the recent advancements that have made similarity join algorithms highly scalable. *Signature-based* approaches (e.g., [4,16,45]) address the efficiency and scalability of similarity joins over large datasets. Many techniques are proposed for set-similarity join, which can be used along with qgrams for the purpose of (string) similarity joins, and are mostly based on the idea of creating signatures for sets (strings) to reduce the search space. Some signature generation schemes are derived from dimensionality reduction. One efficient approach uses the idea of *Locality Sensitive Hashing* [36] in order to hash similar sets into the same values with high probability and therefore provides an approximate solution. Arasu et al. [4] proposed algorithms specifically for set-similarity joins that are exact and outperform previous approximation methods in their framework, although parameters of the algorithms require extensive tuning. More recent work [8] proposes algorithms based on novel indexing and optimization strategies that do not rely on approximation or extensive parameter tuning and outperform previous state-of-the-art approaches. One advantage of our approach is that all these techniques can be applied to make this first component of the framework scalable.

2.2 Clustering

The clustering module outputs a set of clusters of records $c_1,\ldots,c_k$ where records in each cluster are highly similar and records in different clusters are more dissimilar. Most of the data clustering algorithms assume that clusters are disjoint, i.e., $c_i \cap c_j = \emptyset$ for all $i, j \in 1 \ldots k$. We will also present algorithms for a model in which clusters are not disjoint, i.e., records may be present in two or more clusters. This makes sense for the duplication detection problem where it may be impossible to allocate a record with certainty to a single cluster. Record t_8 in the database of Figure 1 is an example of such a record where there may be uncertainty as to whether t_8 belongs to cluster c_2 or c_1.

Given our framework, we consider clustering techniques that do not require as input the number of clusters. There is a large body of work on clustering, including the use of clustering for information retrieval [6,34] and record linkage [25,35,39,41]. We consider existing and new techniques that do not require input parameters such as the number of clusters. In this sense, our motivation is similar to the use of generative models and unsupervised clustering in entity resolution [14]. Notably however, we are dealing with large datasets and scalability is an important goal. Moreover, as noted earlier, our evaluation is based on the assumption that structural or co-occurrence information does not exist or such information cannot effectively be used to enhance deduplication. The only input to our clustering component is the result of a similarity join, i.e., the similar pairs and the similarity scores between them. Our algorithms will generally be linear in this input, with the exception that some techniques will require sorting of this input.

Therefore, we do not consider relational clustering algorithms or any of the new generative clustering models. Notably algorithms like Latent Dirichlet Allocation (LDA) [14] are not scalable at present. For example, one recent promising application of LDA to entity resolution requires hours of computation on relatively small data sets of less than 10,000 entities [12].

The majority of existing clustering algorithm that do not require the number of clusters as input [7,17, 27,50] do not meet the requirements of our framework. Specifically, they may require another parameter to be set by the user and/or they may be computationally expensive and far from practical. There are other clustering algorithms that produce non-disjoint clusters, like Fuzzy C-Means [11], but like K-Means they require the number of clusters. We refer the reader to [25] and references therein for details of numerous clustering algorithms used for duplicate detection. A thorough experimental comparison of diverse clustering algorithms from the Information Retrieval, Machine Learning, and Data Management literature can be found elsewhere [32]. These include the disjoint algorithms presented in this paper (Section 4), as well as algorithms not considered here, like correlation clustering and its optimizations [1,23] that were shown to not perform well (or not better than those we consider) for the duplicate detection task.

2.3 Creating a Probabilistic Database

The final component of our framework creates a probabilistic database. Managing uncertainty and inconsistency has been an active research topic for a long time. Various methodologies exist with different characteristics that handle uncertainty and inconsistency in a variety of applications [2,3,15,22,51]. A large amount of previous work addresses the problem of efficient query evaluation on databases in which it is assumed that a probability value is assigned to each record in the database beforehand. The vast majority of approaches do not address the problem of creating probabilistic databases. A common assumption is that the probabilities reflect the reliability of the data source, for example, based on the reliability of the device (e.g. RFID sen-

sor) that generates the data or statistical information about the reliability of a web data source. The `Price` relation in Figure 1 is an example of such database, where it is assumed that there is an existing knowledge about the reliability of the data sources that provide the prices.

Andritsos et al. [2] propose a method for creating a probabilistic database for duplicated categorical data. In categorical data, the similarity between two attribute values is either 0 (if the values are different) or 1 (if the values are the same). They first cluster the relation using a scalable algorithm based on the Agglomerative Information Bottleneck [47], and then assign a probability to each record within a cluster that represents the probability of that record being in the clean database. However, they do not evaluate the accuracy of the probabilities assigned. The Andritsos et al. [2] work creates a database with row-level uncertainty (probabilities are associated with records). Gupta and Sarawagi [29] present a method for creating a probabilistic database with both row- and column-level uncertainty from statistical models of structure extraction. In structure extraction, unlike duplicate detection, there is uncertainty about not only the correctness/existence of a record, but also the correctness of attribute values within each record.

Dalvi and Suciu [21] propose an online approach for generating the probabilities in which the SQL queries are allowed to have approximate equality predicates that are replaced at execution time by a user defined `MATCH()` operator. Accurate and efficient implementation of a `MATCH()` operator is not a trivial task as partly shown in this paper.

2.4 Query Evaluation

An important motivation for our work is the increased value that can be found from effectively modeling duplicates and their uncertainty. To realize this value, we must be able to query and use the database we create. Consider again our example of Figure 1. It may be possible to normalize (or standardize) the names of companies and their location by, for example, choosing one common convention for representing cities. However, in other attributes there may be true disagreement on what the real value should be. For the first company (Altera), we do not know how many employees (emp#) it has. By keeping all values and using some of the query answering techniques described in this subsection, we can still give users meaningful answers to queries. For example, if we want to find small companies (with less than 1000 employees), we know not to return Altera. If we want to know the total number of employees in New York, we can again use our assigned probabilities to give probabilities for the possible answers to this query.

In this subsection, we briefly discuss several query processing techniques suitable for probabilistic databases generated by our framework. We begin with a recent proposal for modeling and querying possible repairs in duplicate detection. We then discuss two other proposals that have considered efficient query evaluation on the specific probabilistic database we create (that is disjoint-independent databases). We then consider techniques for top-k query evaluation on probabilistic databases along with a new proposal for using probabilistic information (of the type we can create) to help in data cleaning. Finally, we describe a simple extension to our framework to create databases with attribute-level uncertainty.

2.4.1 Querying Repairs of Duplicate Data

Beskales et al. [10] present an uncertainty model for representing the possible clusterings generated by any fixed parametrized clustering algorithm, as well as efficient techniques for query evaluation over this model. Any probabilistic database generated by our framework can be viewed as a *duplication repair* in this model. Their approach provides a way of modeling clustering uncertainty on top of our probabilistic databases. Hence, their queries use both the probabilities we assign and in addition account for possible uncertainty in the clustering itself (e.g., uncertainty in the assignment of tuples to clusters). Their model is based on the notion of U-Clean Relations. A U-Clean relation R^c of an unclean relation R is defined as a set of c-records. A c-record is a representative record of a cluster along with two additional attributes C and P. The attribute C of a c-record is the set of record identifiers in R that are clustered together to form the c-record, and the attribute P is the parameter settings of the clustering algorithm A that leads to the generation of the cluster C. In Beskales et al. [10], possible parameter values are represented using a continuous random variable τ, and P is an interval for τ that results in C. Here, we consider possible parameter values as a discrete random variable θ, and P as the set of thresholds θ used for the similarity join component that results in the cluster C. Let θ^l denote the lower bound and θ^u denote the upper bound for the threshold. For many string similarity joins, $\theta^l = 0$ and $\theta^u = 1$. Applying A to the unclean relation R with parameter θ, generates a possible clustering of R, denoted by $A(R, \theta)$.

Consider again the Company relation in the dirty database of Figure 1. Assume that any threshold less than or equal to 0.3 results in one clusters $\{ t_1, t_2, t_3, t_4,$

Company^c

ID	···	C	P
CP1	···	{ t_1,t_2,t_3,t_4,t_5,t_6,t_7,t_8 }	{ 0.2,0.3 }
CP2	···	{ t_1,t_2,t_3,t_4,t_8 }	{ 0.4 }
CP3	···	{ t_5,t_6,t_7 }	{ 0.4 }
CP4	···	{ t_1,t_2,t_3,t_4 }	{ 0.5,0.6,0.7 }
CP5	···	{ t_4,t_5,t_6,t_7 }	{ 0.5,0.6,0.7 }

Fig. 3 U-Clean relation created from the Company relation in the dirty database of Figure 1

t_5, t_6, t_7, t_8 }, threshold $\theta = 0.4$ results in two clusters { t_1, t_2, t_3, t_4, t_8 } and { t_5, t_6, t_7 }, and any threshold above 0.4 and below 0.7 results in two clusters { t_1, t_2, t_3, t_4 } and { t_4, t_5, t_6, t_7 }. Figure 3 shows the C and P attributes of the corresponding U-Clean relation.

The set of all clusterings χ is defined as { $A(R, \theta)$: $\theta \in \{\theta^l, \cdots, \theta^u\}$ }. Let function $f_\theta(t)$ be the probability that t is the suitable parameter setting. The probability of a specific clustering $X \in \chi$, denoted $Pr(X)$, is derived as follows:

$$Pr(X) = \sum_{t=\theta^l}^{\theta^u} f_\theta(t) \cdot h(t, X) \quad (1)$$

where $h(t, X) = 1$ if $A(R, t) = X$, and 0 otherwise.

In our framework, the function $f_\theta(t)$ can be derived by manual inspection of a possibly small subset of the clustering results, and calculating (and normalizing) the quality measures presented in Section 4.3 over a subset of the data using different thresholds. Efficient algorithms are proposed in [10] for evaluation of Selection, Projection, Join (SPJ) and aggregation queries. Moreover, an extension of this model is presented in which the uncertainty in merging the clusters (or choosing the representative record for each cluster) is also considered. Our probability assignment component (Section 5) can be used to generate such U-Clean relations.

2.4.2 Clean answers over Duplicated Data

This approach, presented by Andritsos et al. [2], requires a probabilistic database with row-level uncertainty, where probabilities are assigned in a way such that for each $i \in 1 \ldots k$, $\sum_{t \in C_i} prob(t) = 1$. Such a database is referred to as a *dirty database*. The probability values reflect the probability of the record being the best representation of the real entity. Even if the database does not contain a completely clean record, such a database can be used for accurate query answering.

Consider the example dirty database in Figure 1. This database consists of three dirty relations: `Company` with original schema `Company(tid, name, emp#, hq)`, `Product` with original schema `Product(pid, product, tidFk)` and `Price` with original schema `Price(tid, product, price)`. Two new attributes are introduced in all the relations: `cid` for the identifier of the clustering produced by the clustering component, and `prob` for the tuple probabilities. In relation `Product`, a new attribute `cidFk` is introduced for the identifier of the company referenced by `Product.tidFk`. The values of this attribute are updated using a process called *identifier propagation* which runs after the clustering phase and adds references to the cluster identifiers of the tuples in all the relations that refer to those tuples.

A *candidate database* D^{cd} for the dirty database D is defined as a subset of D that for every cluster c_i of a relation in D, there is exactly one tuple t from c_i such that t is in D^{cd}. Candidate databases are related to the notion of *possible worlds*, which has been used to give semantics to probabilistic databases. Notice, however, that the definition of candidate database imposes specific conditions on the tuple probabilities: the tuples within a cluster must be exclusive events, in the sense that exactly one tuple of each cluster appears in the clean database, and the probabilities of tuples from different clusters are independent. For the example database in Figure 1 without the relation `Price`, the candidate databases are:

$D_1^{cd} = \{t_1, t_5, p_1, p_4\}$ $D_2^{cd} = \{t_2, t_5, p_1, p_4\}$ $D_3^{cd} = \{t_3, t_5, p_1, p_4\}$
$D_4^{cd} = \{t_4, t_5, p_1, p_4\}$ $D_5^{cd} = \{t_1, t_6, p_1, p_4\}$ $D_6^{cd} = \{t_2, t_6, p_1, p_4\}$
$D_7^{cd} = \{t_3, t_6, p_1, p_4\}$ $D_8^{cd} = \{t_4, t_6, p_1, p_4\}$ $D_9^{cd} = \{t_1, t_7, p_1, p_4\}$
$D_{10}^{cd} = \{t_2, t_7, p_1, p_4\}$ $D_{11}^{cd} = \{t_3, t_7, p_1, p_4\}$ $D_{12}^{cd} = \{t_4, t_7, p_1, p_4\}$
$D_{13}^{cd} = \{t_1, t_8, p_1, p_4\}$ $D_{14}^{cd} = \{t_2, t_8, p_1, p_4\}$ $D_{15}^{cd} = \{t_3, t_8, p_1, p_4\}$
$D_{16}^{cd} = \{t_4, t_8, p_1, p_4\}$ $D_{17}^{cd} = \{t_1, t_9, p_1, p_4\}$ $D_{18}^{cd} = \{t_2, t_9, p_1, p_4\}$
$D_{19}^{cd} = \{t_3, t_9, p_1, p_4\}$ $D_{20}^{cd} = \{t_4, t_9, p_1, p_4\}$ $D_{21}^{cd} = \{t_1, t_5, p_2, p_4\}$
$D_{22}^{cd} = \{t_2, t_5, p_2, p_4\}$ $D_{23}^{cd} = \{t_3, t_5, p_2, p_4\}$ $D_{24}^{cd} = \{t_4, t_5, p_2, p_4\}$
$D_{25}^{cd} = \{t_1, t_6, p_2, p_4\}$ $D_{26}^{cd} = \{t_2, t_6, p_2, p_4\}$ $D_{27}^{cd} = \{t_3, t_6, p_2, p_4\}$
$D_{28}^{cd} = \{t_4, t_6, p_2, p_4\}$ $D_{29}^{cd} = \{t_1, t_7, p_2, p_4\}$ $D_{30}^{cd} = \{t_2, t_7, p_2, p_4\}$
$D_{31}^{cd} = \{t_3, t_7, p_2, p_4\}$ $D_{32}^{cd} = \{t_4, t_7, p_2, p_4\}$ $D_{33}^{cd} = \{t_1, t_8, p_2, p_4\}$
$D_{34}^{cd} = \{t_2, t_8, p_2, p_4\}$ $D_{35}^{cd} = \{t_3, t_8, p_2, p_4\}$ $D_{36}^{cd} = \{t_4, t_8, p_2, p_4\}$
$D_{37}^{cd} = \{t_1, t_9, p_2, p_4\}$ $D_{38}^{cd} = \{t_2, t_9, p_2, p_4\}$ $D_{39}^{cd} = \{t_3, t_9, p_2, p_4\}$
$D_{40}^{cd} = \{t_4, t_9, p_2, p_4\}$ $D_{41}^{cd} = \{t_1, t_5, p_3, p_4\}$ $D_{42}^{cd} = \{t_2, t_5, p_3, p_4\}$
$D_{43}^{cd} = \{t_3, t_5, p_3, p_4\}$ $D_{44}^{cd} = \{t_4, t_5, p_3, p_4\}$ $D_{45}^{cd} = \{t_1, t_6, p_3, p_4\}$
$D_{46}^{cd} = \{t_2, t_6, p_3, p_4\}$ $D_{47}^{cd} = \{t_3, t_6, p_3, p_4\}$ $D_{48}^{cd} = \{t_4, t_6, p_3, p_4\}$
$D_{49}^{cd} = \{t_1, t_7, p_3, p_4\}$ $D_{50}^{cd} = \{t_2, t_7, p_3, p_4\}$ $D_{51}^{cd} = \{t_3, t_7, p_3, p_4\}$
$D_{52}^{cd} = \{t_4, t_7, p_3, p_4\}$ $D_{53}^{cd} = \{t_1, t_8, p_3, p_4\}$ $D_{54}^{cd} = \{t_2, t_8, p_3, p_4\}$
$D_{55}^{cd} = \{t_3, t_8, p_3, p_4\}$ $D_{56}^{cd} = \{t_4, t_8, p_3, p_4\}$ $D_{57}^{cd} = \{t_1, t_9, p_3, p_4\}$
$D_{58}^{cd} = \{t_2, t_9, p_3, p_4\}$ $D_{59}^{cd} = \{t_3, t_9, p_3, p_4\}$ $D_{60}^{cd} = \{t_4, t_9, p_3, p_4\}$

Clearly, not all the candidate databases are equally likely to be clean. This is modeled with a probability distribution, which assigns to each candidate database a probability of being clean. Since the number of candidate databases may be huge (exponential in the worst case), the distribution is not given by extension. Instead, probabilities of each tuple are used to calculate

the probability of a candidate database being the clean one. Since tuples are chosen independently, the probability of each candidate database can be obtained as the product of the probability of each of its tuples: $Pr(D^{cd}) = \prod_{t \in D^{cd}} prob(t)$.

Although the clean database is not known, a query can be evaluated by being applied to the candidate databases. Intuitively, a result is more likely to be in the answer if it is obtained from candidates with higher probability of being clean. A *clean answer* to a query q is therefore defined as a tuple t such that there exists a candidate database D^{cd} such that $t \in q(D^{cd})$. The probability of t is: $p = \sum_{D^{cd}: t \in q(D^{cd})} Pr(D^{cd})$.

The clean answers to a query can be obtained directly from the definition if we assume that the query can be evaluated for each candidate database. However, this is an unrealistic assumption due to the potentially huge number of candidate databases. Andritsos et al. [2] propose a solution to this problem by rewriting the SQL queries to queries that can be applied directly on the dirty database in order to obtain the clean answers along with their probabilities. The following two examples illustrate this approach.

Example 1 *Consider a query q_1 for the dirty database in Figure 1 that retrieves all the companies that have at least 5K employees.*

Company cluster c_1 has more than 5K employees in all the candidate databases and therefore is a clean answer with probability 1. The cluster c_2, however, has at least 5K employees only in the candidate databases that include tuple t_8. The probability of this candidate database is 0.184. *The following re-written query returns the clean answers along with their probability values.*

```
select cid, sum(prob)
from company
where emp# >= 5K
group by cid
```

The previous example focuses on a query with just one relation. However, as shown in the next example, the rewriting strategy can be extended to queries involving foreign key joins.

Example 2 *Consider a query q_2 for the dirty database in Figure 1 that selects the products and the companies for those companies that have at most $5K$ employees.*

The product cluster c_4 associated with company cluster c_2 appears in every candidate database and the employee count of c_2 is always at most $5K$. Therefore (c_4, c_2) has probability 1 of being a clean answer. The query answer (c_3, c_2) appears only in the result of applying the query $q2$ to the candidate databases that include tuples t_8 and p_2 (D^{cd}_{33}, D^{cd}_{34}, D^{cd}_{35} and D^{cd}_{36}), and sum of their probabilities is 0.064. *(c_3, c_1) does not appear in any of the candidate databases and therefore is not a clean answer (i.e., has probability zero). It is easy to see that the clean answers can be obtained by the following rewriting of the query.*

```
select p.cid, p.cidFk, sum(p.prob * c.prob)
from company c, product p
where p.cidFk = c.cid
      and c.emp# <= 5K
group by p.cid, c.cid
```

The above rewriting strategy works only for a certain class of queries. Let q be a Select-Project-Join (SPJ) query. The identifier of a relation is defined as the attribute containing the cluster id (which identifies the tuples which are duplications). The join graph G of q is defined as a directed graph such that the vertices of G are the relations used in q and there is an arc from R_i to R_j if a non-identifier attribute of R_i is equated with the identifier attribute of R_j. Andritsos et al. [2] define an SPJ query q with join graph G as a *rewritable query* if: 1) all the joins involve the identifier of at least one relation 2) G is a tree 3) a relation appears in the `from` clause at most once, and 4) the identifier of the relation at the root of G appears in the `select` clause. These conditions rule out, for example, joins that do not involve an identifier attribute and queries that are cyclic or contain self joins.

Dalvi and Suciu [22] present a theoretical study of the problem of query evaluation over dirty databases (also known as *disjoint independent databases*). They present a dichotomy for the complexity of query evaluation for queries without self-joins: evaluating every query is either PTIME or #P-hard. #P-hard queries are called hard queries and are in one of the following forms (the underlined attributes are the keys of the relations):

- $h_1 = R(\underline{x}), S(\underline{x}, y), T(\underline{y})$
- $h_2 = R(\underline{x}, y), \cdots, R_k(\underline{x}, y), S(\underline{y})$
- $h_3 = R(\underline{x}, y), \cdots, R_k(\underline{x}, y), S_1(\underline{x, y}), \cdots, S_m(x, \underline{y})$

The hardness of any conjunctive query without self-joins follows from a reduction from one of these three queries. Any query that is not hard (#P-hard) is referred to as *safe* and can be evaluated in PTIME.

2.4.3 Top-k Query Evaluation

The problem of evaluating top-k query results on probabilistic databases has been studied in previous work [43, 49, 48]. Different types of top-k queries are possible for uncertain data. Consider the following queries over the dirty database of Figure 1:

- Find the location of the headquarters of companies that have a product selling for more than \$300, return only the top k locations (ranked according to their probabilities).
- Find the top k most expensive products.
- Find the companies that have the k most expensive products (ranking based on the price in all the possible worlds).

Here again, these queries can be answered by materializing all the candidate databases, obtaining answers for each candidate database and aggregating the probabilities of identical answers, which could be prohibitively expensive because of the huge number of candidate databases. For evaluation of the first query, the fact that the user is interested only in the top 3 most probable answers can be used to make the query evaluation more efficient. Ré et al. [43] present an approach for generating the top-k probable query answers using Monte-Carlo simulation. In this approach, the top k answers of a SQL query (according to their probabilities) are returned, and their probabilities are approximated only to the extent needed to compute their ranking. Although the probabilities are approximate, the answers are guaranteed to be the correct k highest ranked answers. The queries considered in this work are of the following form:

```
TOP k
SELECT B̄, agg1(A1), agg2(A2), ···
FROM R̄
WHERE C
GROUP BY B̄
```

The aggregate operators can be `sum`, `count`, `min` and `max`; `avg` is not supported.

The other type of top-k query requires finding the top k tuples according to their `price` values (or some other scoring function). The second and third queries above are examples of such queries. Soliman et al. [48, 49] present a single framework for processing both score and uncertainty leveraging current DBMS storage and query processing capabilities. Their work is based on an uncertainty model that includes generation rules, which are arbitrary logical formulas that determine the valid worlds. Tuples that are not correlated using generation rules are independent. Such a model is particularly useful for duplicate detection. The disjointness (mutual exclusion) of tuples within clusters that can be expressed using generation rules. In addition, two clusters can share a single tuple with a generation rule that states that the shared tuple cannot be present in both clusters. Therefore, for the example database in Figure 1 where there may be uncertainty as to whether t_8 belongs to cluster c_2 or c_1, it is possible to include a new tuple t'_8 in cluster c_1 which has the same values as tuple t_8, using the generation rule $(t_8 \oplus t'_8)$ which means that both tuples cannot be present in a single candidate database. This model makes it possible to use the non-disjoint clustering algorithms we propose in this paper.

2.4.4 Cleaning with Quality Guarantees

Another interesting application of uncertain data management for duplicate detection is cleaning the data in order to increase the quality of certain query results. Cheng et al. [19] recently proposed a framework for this purpose. In their work, they present the PWS-quality metric, which is a universal measure that quantifies the level of ambiguity of query answers under the possible worlds semantics. They provide efficient methods for evaluating this measure for two classes of queries:

- Non-rank-based queries, where a tuple's qualification probability is independent of the existence of other tuples. For example, range queries, i.e., queries that return a set of tuples having an attribute value that is in a certain range.
- Rank-based queries, where a tuple's qualification probability depends on the existence of other tuples, such as `MAX` query which is the main focus of the techniques in this framework.

Using the PWS-quality, a set of uncertain objects in the database can be chosen to be cleaned by the user, in order to achieve the best improvement in the quality of query answers.

2.4.5 Attribute-level Uncertainty

We have limited our discussions so far to databases with tuple-level (row-level) uncertainty. It is also possible to use the probability assignment methods we present in this paper to create databases with attribute-level (column-level) uncertainty. This can be done easily by applying our techniques to each attribute individually (essentially applying our techniques to a *column-store* version of the database). Figure 4 shows such a database for the sample dirty relations in Figure 1. Relations with attribute-level uncertainty can either be transformed to several relations with tuple-level uncertainty and be used along with one of the query evaluation techniques described in this section, or they can be stored and queried in more efficient frameworks designed for efficient handling of attribute-level uncertainty [3, 46].

2.5 Evaluation Framework

To generate datasets for our experiments, we use an enhanced version of the UIS database generator which

Company

cid	name	emp#	hq
$c1$	Altera Corporation {0.430} Altersa Corporation {0.297} lAtera Croporation {0.273}	5K {0.5} 6K {0.5}	New York {0.50} NY {0.25} NY, NY {0.25}
$c2$	ALTEL Corporatio {0.310} ALLTEL Corporation {0.254} ALLTLE Corporation {0.234} Alterel Coporation {0.202}	2K {0.4} 3K {0.4} 5K {0.2}	Albany, NY {0.4} Albany {0.4} NY {0.2}

Product

cid	product	cidFk
$c3$	MaxLink 300 {0.5} MaxLnk 300 {0.5}	$c1$ {0.66} $c2$ {0.33}
$c4$	SmartConnect {1.0}	$c2$ {1.0}

Price

rid	product	price
$c5$	MaxLink 300 {1.0}	\$285 {0.8} \$100 {0.2}

Fig. 4 The sample dirty database of Figure 1 with attribute-level uncertainty.

has been effectively used in the past to evaluate duplicate detection algorithms and has been made publicly available [31,35]. We follow a relatively standard methodology of using the data generator to inject different types and percentages of errors to a clean database of string attributes. The erroneous records made from each clean record are put in a single cluster (which we use as ground truth) in order to be able to measure quality (precision and recall) of the similarity join and clustering modules. The generator permits the creation of data sets of varying sizes, error types and distributions, thus is a very flexible tool for our evaluation. The kind of typographical errors injected by the data generator are based on studies on common types of errors present in string data in real databases [37]. Therefore the synthetic datasets resemble real dirty databases, but allow thorough evaluation of the results based on robust quality measures.

Our data generator provides the following parameters to control the error injected in the data:

- the size of the dataset to be generated
- the fraction of clean records to be utilized to generate erroneous duplicates
- *distribution of duplicates*: the number of duplicates generated for a clean record can follow a uniform, Zipfian or Poisson distribution.
- *percentage of erroneous duplicates*: the fraction of duplicate records in which errors are injected by the data generator.
- *extent of error in each erroneous record*: the percentage of characters that will be selected for injecting character edit error (character insertion, deletion, replacement or swap) in each record selected for error injection.
- *token swap error*: the percentage of word pairs that will be swapped in each record that is selected for error injection.

We use two different clean sources of data: a data set consisting of *company names* and a data set consisting of titles from *DBLP*. Statistical details for the two datasets are shown in Table 1. Note that we can generate reasonably large datasets out of these clean sources. For the company names dataset, we also inject domain specific *abbreviation errors*, e.g., replacing `Inc.` with `Incorporated` and vice versa. We describe the characteristics of the specific datasets generated for evaluating each component (parameters used to create datasets) in the related sections.

Table 1 Statistics of clean datasets

dataset	#rec.	Avg. rec. length	#words/rec.
Company Names	2139	21.03	2.92
DBLP Titles	10425	33.55	4.53

3 Similarity Join Module

There are a large number of similarity functions for string data. The choice of the similarity function highly depends on the characteristics of the datasets. In what follows, we briefly describe the similarity measures that are suitable for our framework. Since one of our main goals in this work is scalability, we only consider those similarity measures that could have efficient implementation. Our contribution in this section is benchmarking accuracy of these measures in order to choose the measure with highest performance for this framework.[2]

3.1 Similarity Measures

The similarity measures that fit in our framework are those based on q-grams created out of strings along with a similarity measure that has been shown to be effective in previous work. The measures discussed here share one or both of the following properties.

- High scalability: There are various techniques proposed in the literature as described in Section 2.1 for enhancing the performance of the similarity join operation using q-grams along with these measures.
- High accuracy: Previous work has shown that these measures perform better or equally well in terms

[2] A presentation of the evaluation was given at the International Workshop on Quality in Databases [33].

of accuracy when compared with other string similarity measures. Specifically, these measures have shown good accuracy in name-matching tasks [20] or in approximate selection [31]. We include these measures to compare their accuracy to the scalable measures. The results of our experiments show that some highly scalable measures outperform other highly accurate but non-scalable measures in terms of accuracy on the approximate join task.

3.1.1 Edit Similarity

Edit-distance is widely used as the measure of choice in many similarity join techniques. Specifically, previous work [28] has shown how to use q-grams for an efficient implementation of this measure in a declarative framework. Recent work on enhancing performance of similarity join has also proposed techniques for scalable implementation of this measure [4,38].

Edit distance between two string records r_1 and r_2 is defined as the transformation cost of r_1 to r_2, $tc(r_1, r_2)$, which is equal to the minimum cost of edit operations applied to r_1 to transform it to r_2. Edit operations include character *insert* (inserting a new character in r_1 to transform it into r_2, *delete* (deleting a character from r_1 for the transformation) and *substitute* (substitute a character in r_1 with a new character for the transformation) [30]. The edit similarity is defined as:

$$sim_{edit}(r_1, r_2) = 1 - \frac{tc(r_1, r_2)}{\max\{|r_1|, |r_2|\}} \tag{2}$$

There is a cost associated with each edit operation. There are several cost models proposed for edit operations for this measure. The most commonly used measure called Levenshtein edit distance, which we will refer to as edit distance in this paper, uses unit cost for all the operations.

3.1.2 Jaccard and Weighted Jaccard

Jaccard similarity is the fraction of tokens in r_1 and r_2 that are present in both. Weighted Jaccard similarity is the weighted version of Jaccard similarity, i.e.,

$$sim_{WJaccard}(r_1, r_2) = \frac{\sum_{t \in \mathbf{r}_1 \cap \mathbf{r}_2} w_R(t)}{\sum_{t \in \mathbf{r}_1 \cup \mathbf{r}_2} w_R(t)} \tag{3}$$

where $w_R(t)$ is a weight function that reflects the commonality of the token t in the relation R. We choose a slightly modified form of the Inverse Document Frequency (IDF) weights based on the Robertson/Sparck-Jones (RSJ) weights for the tokens which was shown to be effective in our experiments and in previous work [31]:

$$w_R(t) = log\left(\frac{N - n_t + 0.5}{n_t + 0.5}\right) \tag{4}$$

where N is the number of tuples in the base relation R and n_t is the number of tuples in R containing the token t.

3.1.3 Measures from IR

A well-studied problem in information retrieval is the problem of given a query and a collection of documents, return the most *relevant* documents to the query. In the measures for this problem, records are treated as documents and q-grams are seen as words (tokens) of the documents. Therefore, the same techniques for finding relevant documents to a query can be used to return *similar* records to a query string. In the rest of this subsection, we present three measures that have been shown to have higher performance for the approximate selection problem [31]. Note that IR models may be asymmetric, but we are able to still use them since we are using *self-joins* for duplicate detection.

Cosine w/tf-idf The *tf-idf cosine* similarity is a well established measure in the IR community which leverages the vector space model. This measure determines the closeness of the input strings r_1 and r_2 by first transforming the strings into unit vectors and then measuring the angle between their corresponding vectors. The *cosine* similarity with tf-idf weights is given by:

$$sim_{Cosine}(r_1, r_2) = \sum_{t \in \mathbf{r}_1 \cap \mathbf{r}_2} w_{r_1}(t) \cdot w_{r_2}(t) \tag{5}$$

where $w_{r_1}(t)$ and $w_{r_2}(t)$ are the normalized *tf-idf* weights for each common token in r_1 and r_2 respectively. The normalized *tf-idf* weight of token t in a given string record r is defined as follows:

$$w_r(t) = \frac{w'_r(t)}{\sqrt{\sum_{t' \in \mathbf{r}} w'_r(t')^2}} \quad , \quad w'_r(t) = tf_r(t) \cdot idf(t)$$

where $tf_r(t)$ is the term frequency of token t within string r and $idf(t)$ is the inverse document frequency with respect to the entire relation R.

BM25 The BM25 similarity score for a query r_1 and a string record r_2 is defined as follows:

$$sim_{BM25}(r_1, r_2) = \sum_{t \in \mathbf{r}_1 \cap \mathbf{r}_2} \hat{w}_{r_1}(t) \cdot w_{r_2}(t) \tag{6}$$

where:

$$\hat{w}_{r_1}(t) = \frac{(k_3+1)\cdot tf_{r_1}(t)}{k_3+tf_{r_1}(t)}$$
$$w_{r_2}(t) = w_R^{(1)}(t)\frac{(k_1+1)\cdot tf_{r_2}(t)}{K(r_2)+tf_{r_2}(t)}$$
$$w_R^{(1)}(t) = \log\left(\frac{N-n_t+0.5}{n_t+0.5}\right)$$
$$K(r) = k_1\left((1-b)+b\frac{|r|}{avg_{rl}}\right)$$

and $tf_r(t)$ is the frequency of the token t in string record r, $|r|$ is the number of tokens in r, avg_{rl} is the average number of tokens per record, N is the number of records in the relation R, n_t is the number of records containing the token t and k_1, k_3 and b are set of independent parameters.

Hidden Markov Model The approximate string matching could be modeled by a discrete Hidden Markov process which has been shown to have better performance than Cosine w/tf-idf in the IR literature [40] and high accuracy and low running time for approximate selection [31]. This particular Markov model consists of only two states where the first state models the tokens that are specific to one particular "String" and the second state models the tokens in "General English", i.e., tokens that are common in many records. A complete description of the model and possible extensions are presented elsewhere [31,40].

The HMM similarity function accepts two string records r_1 and r_2 and returns the probability of generating r_1 given r_2 is a similar record:

$$sim_{HMM}(r_1, r_2) = \prod_{t\in\mathbf{r_1}}(a_0P(t|GE) + a_1P(t|r_2)) \quad (7)$$

where a_0 and $a_1 = 1 - a_0$ are the transition states probabilities of the Markov model and $P(t|GE)$ and $P(t|r_2)$ is given by:

$$P(t|r_2) = \frac{\text{number of times } t \text{ appears in } r_2}{|\mathbf{r}_2|}$$
$$P(t|GE) = \frac{\sum_{r\in R}\text{number of times } t \text{ appears in } r}{\sum_{r\in R}|\mathbf{r}|}$$

3.1.4 Hybrid Measures

The implementation of these measures involves two similarity functions, one that compares the strings by comparing their word tokens and another similarity function which is more suitable for short strings and is used for comparison of the word tokens.

GES The generalized edit similarity (GES) which is a modified version of *fuzzy match similarity* [16], takes two strings r_1 and r_2, tokenizes the strings into a set of words and assigns a weight $w(t)$ to each token. GES defines the similarity between the two given strings as a minimum transformation cost required to convert string r_1 to r_2 and is given by:

$$sim_{GES}(r_1, r_2) = 1 - \min\left(\frac{tc(r_1, r_2)}{wt(r_1)}, 1.0\right) \quad (8)$$

where $wt(r_1)$ is the sum of weights of all tokens in r_1 and $tc(r_1, r_2)$ is the minimum cost of a sequence of the following transformation operations:

- *token insertion:* inserting a token t in r_1 with cost $w(t).c_{ins}$ where c_{ins} is the insertion factor constant and is in the range between 0 and 1. In our experiments, $c_{ins} = 1$.
- *token deletion:* deleting a token t from r_1 with cost $w(t)$.
- *token replacement:* replacing a token t_1 by t_2 in r_1 with cost $(1 - sim_{edit}(t_1, t_2)) \cdot w(t)$ where sim_{edit} is the edit-distance between t_1 and t_2.

SoftTFIDF SoftTFIDF is another hybrid measure proposed by Cohen et al. [20], which relies on the normalized *tf-idf* weight of word tokens and can work with any arbitrary similarity function to find the similarity between word tokens. In this measure, the similarity score is defined as follows:

$$sim_{SoftTFIDF}(r_1, r_2) = \sum_{t_1\in C(\theta,r_1,r_2)} w(t_1, r_1)\cdot w(\arg\max_{t_2\in r_2}(sim(t_1, t_2)), r_2)\cdot \max_{t_2\in r_2}(sim(t_1, t_2)) \quad (9)$$

where $w(t, r)$ is the normalized *tf-idf* weight of word token t in record r and $C(\theta, r_1, r_2)$ returns a set of tokens $t_1 \in r_1$ such that for $t_2 \in r_2$ we have $sim(t_1, t_2) > \theta$ for some similarity function $sim()$ suitable for comparing word strings. In our experiments $sim(t_1, t_2)$ is the Jaro-Winkler similarity as suggested by Cohen et al. [20].

3.2 Evaluation

We only evaluate the *accuracy* of the similarity measures, since there has been several studies on the scalability of these measures, but little work studying the accuracy of the join operation. The accuracy is known to be dataset-dependent and there is no common framework for evaluation and comparison of accuracy of different similarity measures and techniques. This makes comparing their accuracy a difficult task. Nevertheless, we argue that it is possible to evaluate relative performance of different measures for approximate joins by using datasets containing different types of well-known

Table 2 Datasets used for the results in this paper

Group	Name	Percentage of			
		Erroneous Duplicates	Errors in Duplicates	Token Swap	Abbr. Error
Dirty	D1	90	30	20	50
	D2	50	30	20	50
Medium Error	M1	30	30	20	50
	M2	10	30	20	50
	M3	90	10	20	50
	M4	50	10	20	50
Low Error	L1	30	10	20	50
	L2	10	10	20	50
Single Error	AB	50	0	0	50
	TS	50	0	20	0
	EDL	50	10	0	0
	EDM	50	20	0	0
	EDH	50	30	0	0

quality problems such as typing errors and differences in notations and abbreviations.

Datasets In order to evaluate the effectiveness of different similarity measures described in this section, we use the same datasets used in an evaluation of approximate selection [31]. As described in Section 2.5, the errors in these datasets include commonly occurring typing mistakes (edit errors, character insertion, deletion, replacement and swap), token swap and abbreviation errors (e.g., replacing `Inc.` with `Incorporated` and vice versa). For the results presented in this section, the datasets are generated by the data generator out of the clean company names dataset described in Table 1. The errors in the datasets have a uniform distribution. For each dataset, on average 5000 dirty records are created out of 500 clean records. We have also run experiments on datasets generated using different parameters. For example, we generated data using a Zipfian distribution, and we also used data from the other clean source in Table 1 (DBLP titles). We also created larger datasets. For these other datasets, the accuracy trends remain the same. Table 2 shows the description of all the datasets used for the results in this paper. We used 8 different datasets with mixed types of errors (edit errors, token swap and abbreviation replacement). Moreover, we used 5 datasets with only a single type of error (3 levels of edit errors, token swap or abbreviation replacement errors) to measure the effect of each type of error individually.

Measures We use well-known measures from IR, namely precision, recall, and F_1, for different values of the threshold to evaluate the accuracy of the similarity join operation. We perform a self-join on the input table using a similarity measure with a fixed threshold θ. *Precision (Pr)* is defined as the percentage of *duplicate* records among the records that have a similarity score above the threshold θ. In our datasets, *duplicate* records are marked with the same cluster ID as described above. *Recall (Re)* is the ratio of the number of *duplicate* records that have similarity score above the threshold θ to the total number of *duplicate* records. Therefore, a join that returns all the pairs of records in the two input tables as output has low (near zero) precision and recall of 1. A join that returns an empty answer has precision 1 and zero recall. The F_1 measure is the harmonic mean of precision and recall, i.e., $F_1 = \frac{2 \times Pr \times Re}{Pr + Re}$ We measure precision, recall, and F_1 for different values of the similarity threshold θ. For comparison of different similarity measures, we use the maximum F_1 score across different thresholds.

Settings For the measures based on q-grams, we set $q = 2$ since it yields the best accuracy in our experiments for all these measures. We use the same parameters for BM25 and HMM score formula that were suggested elsewhere [40,44] [31].

Results Appendix A contains the full precision-recall curves for all the measures described above. The results of our experiments show that the "dirtiness" of the input data greatly affects the value of the threshold that results in the most accurate join. For all the measures, a lower value of the threshold is needed as the degree of error in the data increases. For example, Weighted Jaccard achieves the best F_1 score over the dirty group of datasets with threshold 0.3, while it achieves the best F_1 for the low-error datasets at threshold 0.55. BM25 and HMM are less sensitive and the best value of the threshold varies from 0.25 for dirty datasets to 0.3 for low-error datasets. We will discuss later how the degree of error in the data affects the choice of the most accurate measure.

Effect of types of errors: Figure 5 shows the maximum F_1 score for different values of the threshold for different measures on datasets containing only edit-errors (the `EDL, EDM` and `EDH` datasets). These figures show that weighted Jaccard and Cosine have the highest accuracy followed by Jaccard, and edit similarity on the low-error dataset `EDL`. By increasing the amount of edit error in each record, HMM performs as well as weighted Jaccard, although Jaccard, edit similarity, and GES perform much worse on high edit error datasets. Considering the fact that edit-similarity is mainly proposed for capturing edit errors, this shows the effectiveness of weighted Jaccard and its robustness with varying amount of edit errors. Figure 6 shows the effect of token swap and abbreviation errors on the accuracy of different measures. This experiment indicates that edit similarity is not capable of modeling such errors. HMM, BM25 and Jaccard also are less capable

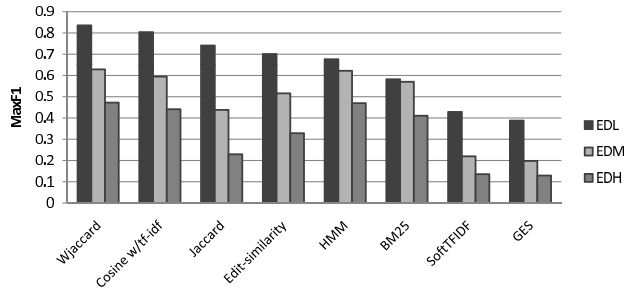


Fig. 5 Maximum F_1 score for different measures on datasets with only edit errors

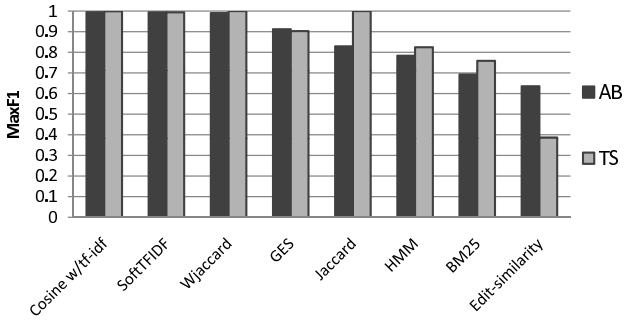


Fig. 6 Maximum F_1 score for different measures on datasets with only token swap and abbr. errors

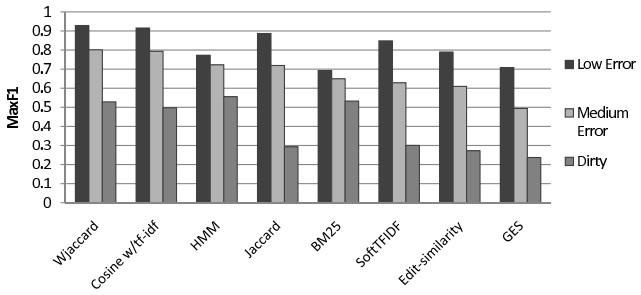


Fig. 7 Maximum F_1 score for different measures on dirty, medium and low-error group of datasets

of modeling abbreviation errors than cosine with tf-idf, SoftTFIDF and weighted Jaccard.

Comparison of measures: Figures 7 shows the maximum F_1 score for different values of the threshold for different measures on dirty, medium and low-error datasets. Here, we have aggregated the results for all the dirty data sets together (respectively, the moderately dirty or medium data sets and the low-error data sets). The results show the effectiveness and robustness of weighted Jaccard and cosine in comparison with other measures. Again, HMM is among the most accurate measures when the data is extremely dirty, and has relatively low accuracy when the percentage of error in the data is low.

3.3 Our Choice of Similarity Measure

Unless specifically mentioned, we use weighted Jaccard similarity as the measure of choice for the rest of the paper due to its relatively high efficiency and accuracy compared with other measures. Note that this similarity predicate can be implemented declaratively and used as a join predicate in a standard RDBMS engine [31], or used with some of the specialized, high performance, approximate join algorithms as described in Section 2. Specifically, the *Weighted Enumeration* (WtEnum) signature generation algorithm can be used to significantly improve the running time of the join [4]. In addition, novel indexing and optimization techniques can be utilized to make the join even faster [8].

4 Clustering Module

Here, we consider algorithms for clustering records based on the output of the similarity join module. So the input to this module is a set of similar pairs of records and the output is a set of clusters of records $\mathcal{C} = \{c_1, \ldots, c_k\}$ where records in each cluster are highly similar. We present two groups of algorithms, one for creating disjoint clusters, i.e., non-overlapping clusters that partition the base relation, and the other for non-disjoint clustering, i.e., we allow a few records to be present in two or more clusters.

The scalable similarity join will eliminate large portions of the data (records without duplicates) from the clustering. Specifically, the similarity graph used in the clustering will be much smaller after using a similarity join. Of course, we want to be able to handle large amounts of error in the data, so we do also focus on clustering techniques that can still handle large data sets containing hundreds of thousands of potential duplicates. But the combination of a scalable similarity join, with a clustering technique that can handle large similarity graphs, greatly enhances the end-to-end scalability of the overall approach and permits the generation of probability values (Section 5) on very large databases.

There exists a variety of clustering algorithms in the literature each with different characteristics. However, as mentioned earlier, we are dealing with a rather different clustering problem here. First of all, we use only the output of the similarity join module for the clustering. Our goal of clustering is to create a probabilistic database and therefore we need to seek specific characteristics that fit this goal. For example, a few extra records in a cluster is preferable to a few missing records, since the few extra records will get less probability in the probability assignment component. More-

over, since the similarity join module needs a threshold for the similarity measure which is hard to choose and dataset-dependent, we seek clustering algorithms that are less sensitive to the choice of the threshold value. A comprehensive study of the performance of clustering algorithms in duplicate detection including the disjoint algorithms presented here and several more sophisticated clustering algorithms can be found elsewhere [32].

4.1 Disjoint Algorithms

In this group of algorithms, the goal is to create clusters of similar records $\mathcal{C} = \{c_1, \ldots, c_k\}$ where the value of k is unknown, $\bigcup_{c_i \in \mathcal{C}} c_i = R$ and $c_i \cap c_j = \emptyset$ for all $c_i, c_j \in \mathcal{C}$, i.e., clusters are disjoint and partition the base relation.

We can think of the source relation as a graph $G(U, V)$ in which each node $u \in U$ presents a record in the base relation and each edge $(u, v) \in V$ connects two nodes u and v having corresponding records that are similar, i.e., their similarity score based on some similarity function $sim()$ is above a specified threshold θ. Note that the graph is undirected, i.e., $(u, v) = (v, u)$. The task of clustering the relation is then clustering the nodes in the graph. In our implementation, we do not materialize the graph. In fact, all the algorithms can be efficiently implemented by a single scan of the list of similar pairs returned by the similarity join module, although some require the list to be sorted by similarity score. We only use the graph G to illustrate our techniques.

4.1.1 Algorithm1: Partitioning

In this algorithm, Partitioning (or transitive closure), we cluster the graph of records by finding the connected components in the graph and putting the records in each component in a separate cluster. This can be done by first assigning each node to a different cluster and then scanning the list of similar pairs and merging clusters of all connected nodes. Figure 8(a) shows the result of this algorithm on a sample graph. As Figure 8(a) shows, this algorithm may put many records that are not similar in the same cluster. Partitioning is a common algorithm used in early entity resolution work [35, 25], and is included as a baseline.

4.1.2 Algorithm2: CENTER

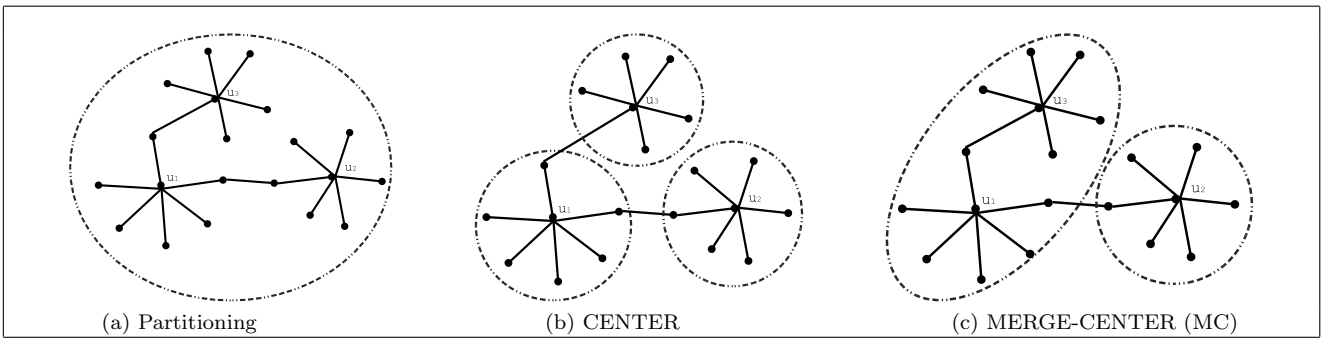
This algorithm, which we call CENTER as in [34] performs clustering by partitioning the graph of the records so that each cluster has a *center* and all records in the cluster are similar to the center. This can be performed by a single scan of the sorted list of similar pairs. The first time a node u appears in the scan, it is assigned as the center of the cluster. All the subsequent nodes v that appear in a pair (u, v) are assigned to the cluster of u and are not considered again. Figure 8(b) shows how this algorithm clusters a sample graph of records, where node u_1 is the first node in the sorted list of similar records and node u_2 appears right after all the nodes similar to u_1, and node u_3 appears after all the nodes similar to $u2$. This algorithm may result in more clusters than Partitioning since it puts into one cluster only those records that are similar to one record which is the center of the cluster.

4.1.3 Algorithm3: MERGE-CENTER

MERGE-CENTER, or MC, is similar to CENTER, but merges two clusters c_i and c_j whenever a record similar to the *center* node of c_j is already in the cluster c_i, i.e., it is similar to a node that is the center or is similar to the center (or one of the center nodes) of the cluster c_i (Note that when two clusters are merged, we do not choose a single center node in this algorithm, so each cluster can have multiple center nodes). As with CENTER, this is done using a single scan of the list of similar records, but keeping track of the records that are already in a cluster. The first time a node u appears in the scan, it is assigned as the center of the cluster. All the subsequent nodes v that appear in a pair (u, v) and are not present in any cluster, are assigned to the cluster of u, and are not selected as the center of any other cluster. Whenever a pair (u, v') is encountered such that v' is already in another cluster, all the nodes in the cluster of u (records similar to u) are merged with the cluster of v'. Figure 8(c) shows the clusters created by this algorithm assuming again that the nodes u_1, u_2 and u_3 are the first three nodes in the sorted list of similar records that are selected as the center of a cluster. As shown in the Figure, this algorithm creates fewer clusters for the sample graph than the CENTER algorithm, but more than the partitioning algorithm.

4.2 Non-Disjoint Algorithms

In this group of algorithms, we do not require $c_i \cap c_j = \emptyset$ for all $i, j \in 1 \ldots k$. For this purpose, we use the results of the similarity join module along with the similarity scores of the similar records. The idea is to have a *core* for each cluster that consists of the records that are highly similar, and *marginal* records for each cluster that are relatively less similar. The core of the clusters are created based on the results of the similarity join with similarity score above a high threshold θ_1. The marginal records are added to the clusters based on the



(a) Partitioning (b) CENTER (c) MERGE-CENTER (MC)

Fig. 8 Illustration of disjoint clustering algorithms

results of the similarity join with a threshold $\theta_2 \leq \theta_1$. Using the terminology from probabilistic record linkage [26], we can say that we put the records that *match* with the center of the cluster in its core, and records that *probably match* with the center in the marginal records of the cluster. Each record appears in the core of only one cluster, but may appear in the marginal records of more than one cluster.

4.2.1 Algorithm4: Non-disjoint Clustering

Our first non-disjoint algorithm, ND, creates a set of core clusters (in a similar way to MERGE-CENTER), then a set of records are added to each cluster which are less similar to the center of the cluster. The algorithm performs as follows. Assume that we have the list of records with similarity score above a threshold θ_2 along with their similarity score from the output of the similarity join module. The algorithm starts by scanning the list. The first time a node u appears in the scan, it is assigned as the center of the core of the cluster. All the subsequent nodes v that appear in a pair (u, v), have $sim(u, v) \geq \theta_1$, and are not present in the core of any other cluster, are assigned to the core of the cluster of u and are not selected as the center of any other cluster. Other pairs (u, v) that have $sim(u, v) \leq \theta_1$ (but have $sim(u, v) \geq \theta_2$) are added as the marginal members of the cluster. Whenever a pair (u, v') with $sim(u, v') \geq \theta_1$ is encountered such that v' is already in the core of another cluster, all the nodes in the cluster of u are merged with the cluster of v'.

4.2.2 Algorithm5: Improved Non-disjoint Clustering with Information Bottleneck Method

The ND algorithm performs well when thresholds θ_1 and θ_2 are chosen accurately. However, the choice of the thresholds highly depends on the similarity measure used in the similarity join module and the type of errors in the datasets. Therefore, it is plausible to be able to choose a low value for the lower threshold θ_2 and then enhance the accuracy of the clustering by pruning extra records from each cluster in a uniform way regardless of the value of the thresholds. Here, we adopt an approach from the information theory field called the information bottleneck in order to enhance the results of non-disjoint clustering. The idea is to prune those marginal records in clusters that are less similar to the records in the core of the clusters.

Our ND-IB algorithm is based on the Agglomerative Information Bottleneck (IB) algorithm for clustering data [47] which we briefly explain here.

Assume R is the set of records, $n = |R|$ is the number of records, T is the set of qgrams of the strings and $d = |T|$ is the total number of qgrams in all records. In the information bottleneck method for clustering data, the goal is to partition the records in R into k clusters $C = \{c_1, c_2, \ldots, c_k\}$ where each cluster $c_i \in C$ is a non-empty subset of R such that $c_i \cap c_j = \emptyset$ for all i, j. Giving equal weight to each record $r \in R$, we define $p(r) = \frac{1}{n}$. We also set the probability of a qgram t given a record $p(t|r) = \frac{idf(t)}{\sum_{t' \in r} idf(t')}$ where $idf(t)$ is the inverse document frequency of qgram t in the relation. For $c \in C$, the elements of R, T and C are related as follows:

$$p(c) = \sum_{r \in c} p(r) \tag{10}$$

$$p(t|c) = \frac{1}{p(c)} \sum_{r \in c} p(r)p(t|r) \tag{11}$$

Merging two clusters c_i and c_j is performed by setting the following parameters for the new cluster c^*:

$$p(c^*) = p(c_i) + p(c_j)$$

$$p(t|c^*) = \frac{p(c_i)}{p(c^*)} p(t|c_i) + \frac{p(c_j)}{p(c^*)} p(t|c_j) \tag{12}$$

In the IB algorithm, clustering is performed by first assuming that each record is a separate cluster and then iteratively merging the clusters $n-k$ times to reduce the number of clusters to k. In each iteration, two clusters

are chosen to be merged so that the amount of *information loss* as a result of merging the clusters is minimum. Information loss is given by the following formula [47]:

$$\delta I(c_i, c_j) = [p(c_i) + p(c_j)] \cdot D_{JS}[p(t|c_i), p(t|c_j)] \quad (13)$$

where $D_{JS}[p(t|c_i), p(t|c_j)]$ is equal to:

$$\frac{p(c_i)}{p(c^*)} D_{KL}[p(t|c_i), \bar{p}] + \frac{p(c_j)}{p(c^*)} D_{KL}[p(t|c_i), \bar{p}]$$

where:

$$\bar{p} = \frac{p(c_i)}{p(c^*)} p(t|c_i) + \frac{p(c_j)}{p(c^*)} p(t|c_j) \quad (14)$$

$$D_{KL}[p, q] = \sum_{r \in R} p(r) \log \frac{p(r)}{q(r)} \quad (15)$$

$$(16)$$

The pruning algorithm for our non-disjoint clustering performs as follows. For each cluster: 1. The records in the core of the cluster are merged using the merge operation and put in cluster c_{core}. 2. For each record r_i in the set of marginal records $M = \{r_1, \ldots, r_k\}$, the amount of information loss for merging r_i with the core cluster c_{core}, $il_i = \delta I(r_i, c_{core})$, is calculated. 3. Assume avg_{il} is the average value of il_i for $i \in 1 \ldots k$ and $stddev_{il}$ is the standard deviation. Those marginal records that have $il_i \geq avg_{il} - stddev_{il}$ are pruned from the cluster.

The intuition behind this algorithm is that by using the information in all the qgrams of the records from the core of the cluster that are identified to be duplicates (and *match*), we can identify which of the marginal records (that *probably match*) are more probably duplicates that belong to that cluster. For this, the records in the core of each cluster are merged using the merge operation (equation 12). If merging a marginal record with the core of the cluster would result in high information loss, then the record is removed from the marginal records of the cluster.

4.3 Evaluation

Datasets: The datasets used for accuracy results in this section are the same datasets described in Table 1 of Section 3.2. Most of the results presented here are for the medium error group of these datasets. In our evaluation, we note when the trends on the other groups of datasets are different than those shown in this report. Note again that we limited the size of the datasets only for our experiments on accuracy. For running time experiments, we used the data generator with DBLP titles dataset of Table 1 to generate larger datasets. In order to show that these results are not limited to the specific datasets we used here, we have made the results of our extensive experiments over various datasets (with different sizes, types and distribution of errors) publicly available at

`http://dblab.cs.toronto.edu/project/stringer/evaluation/`

Accuracy Measures: We evaluate the quality of the clustering algorithms based on several measures from the clustering literature and also measures that are suitable for evaluation of these clusterings in duplicate detection. The latter measures are taken from Hassanzadeh et al. [32]. Suppose that we have a set of k ground truth clusters $G = \{g_1, \ldots, g_k\}$ of the base relation R and let C denote a clustering of records into k' clusters $\{c_1, \ldots, c_{k'}\}$ produced by a clustering algorithm. Consider mapping f from elements of G to elements of C, such that each cluster g_i is mapped to a cluster $c_j = f(g_i)$ that has the highest percentage of common elements with g_i. We define precision, Pr_i, and recall, Re_i, for a cluster g_i, $1 \leq i \leq k$ as follows:

$$Pr_i = \frac{|f(g_i) \cap g_i|}{|f(g_i)|} \text{ and } Re_i = \frac{|f(g_i) \cap g_i|}{|g_i|} \quad (17)$$

Intuitively, Pr_i measures the accuracy with which cluster $f(g_i)$ reproduces cluster g_i, while Re_i measures the completeness with which $f(g_i)$ reproduces class g_i. We define the precision and recall of the clustering as the weighted averages of the precision and recall over all ground truth clusters. More precisely:

$$Pr = \sum_{i=1}^{k} \frac{|g_i|}{|R|} Pr_i \text{ and } Re = \sum_{i=1}^{k} \frac{|g_i|}{|R|} Re_i \quad (18)$$

Again, we also use the F_1-measure (the harmonic mean of precision and recall).

We think of precision, recall and F_1-measure as indicative values of the ability of the algorithm to reconstruct the indicated clusters in the dataset. However, since in our framework the number of clusters created by the clustering algorithm is not fixed and depends on the datasets and the thresholds used in the similarity join, we should also take into account this value in our quality measure. We use two other measures more suitable for our framework. The first, called clustering precision, CPr_i, is the ratio of the pairs of records in each cluster c_i that are in the same ground truth cluster $g_j : c_i = f(g_j)$, i.e.,

$$CPr_i = \frac{|(t, s) \in c_i \times c_i | t \neq s \wedge \exists j \in 1 \ldots k, (t, s) \in g_j \times g_j|}{\binom{|c_i|}{2}} \quad (19)$$

Clustering precision, CPr, is then the average of CPr_i for all clusters with size ≥ 2. Cpr measures the ability of the clustering algorithm to put the records

that must be in the same cluster in one cluster regardless of the number and the size of the clusters. We also need to have a measure that penalizes those algorithms that create more or fewer clusters than the ground truth number of clusters. $PCPr$ is CPr multiplied by the percentage of the extra or missing clusters in the result of clustering, i.e.,

$$PCPr = \begin{cases} \frac{k}{k'} CPr & k < k' \\ \frac{k'}{k} CPr & k \geq k' \end{cases} \quad (20)$$

Partitioning and CENTER algorithms: We measure the quality of clustering algorithms based on different thresholds of the similarity join. The table below shows the values for our medium-error datasets and thresholds that result in the best F1 measure and the best PCPr measure values. We have chosen these thresholds to show how the threshold value could affect the accuracy of the algorithms, and also justify using the PCPr measure. Similar trends can be observed for other thresholds and datasets.

	Partitioning		CENTER	
	Best PCPr	Best F_1	Best PCPr	Best F_1
PCPr	0.554	0.469	0.593	0.298
CPr	0.946	0.805	0.760	0.692
Pr	0.503	0.934	0.586	0.971
Re	0.906	0.891	0.783	0.805
F1	0.622	0.910	0.666	0.877
Cluster#	353	994	472	1305

Note that the number of clusters in the ground truth datasets is 500. The last row in the table shows the number of clusters generated by each algorithm. These results show that, precision, recall and F_1 measures cannot alone determine the best algorithm since they do not take into account the number of clusters generated. As it can be seen, the best value of F_1 measure among different thresholds is 0.910 for partitioning and 0.877 while the corresponding number of clusters are 994 and 1305 respectively. However, the best value of PCPr among different thresholds is 0.554 for partitioning and 0.593 for CENTER, with 353 and 472 clusters in the results respectively. This justifies using CPr and PCPr measures. Also note that the accuracy of these algorithms highly depend on the threshold used for the similarity join module. The results above show that the CENTER algorithm is more suitable than the partitioning algorithm for identification of the correct number of clusters.

MERGE-CENTER (MC) algorithm: The accuracy results for the MERGE-CENTER algorithm for the medium error datasets are shown below. The results are for the similarity threshold that produced the best PCPr results although the trend is the same for both algorithms with any fixed threshold. These results show that the MC algorithm results in significant improvement in all accuracy measures comparing with CENTER and Partitioning algorithms.

	Partitioning	MC	Diff.
PCPr	0.554	0.696	+25.6%
CPr	0.946	0.940	-0.1%
Pr	0.503	0.658	+30.8%
Re	0.906	0.950	+4.9%
F_1	0.622	0.776	+24.8%
Cluster #	353	459	

Non-disjoint algorithms (ND and ND-IB): We compare the results of MERGE-CENTER (MC) with our non-disjoint algorithms, ND and ND-IB, below. Adding marginal records to the clusters increases PCPr, CPr with a small drop in recall but a significant drop in the precision. Note that for our goal which is creating probabilistic databases, recall is more important than precision, since missing records can result in missing results for queries over the output probabilistic database, whereas a few extra records result in extra answers with lower probability values. For these results, we set the threshold $\theta = 0.3$ for MC, the lower threshold $\theta_2 = 0.2$ and the higher threshold $\theta_1 = 0.4$ for non-disjoint algorithms, and we use our low error datasets. We observed a similar trend using many different thresholds and other datasets. In fact non-disjoint algorithms become more effective when used on highly erroneous datasets as partly shown in Figure 9.

	MC $\theta = 0.3$	ND $\theta_1 = 0.4$, $\theta_2 = 0.2$		ND-IB $\theta_1 = 0.4$, $\theta_2 = 0.2$	
			Diff.(MC)		Diff.(ND)
PCPr	0.696	0.930	+0.234	0.924	-0.006
CPr	0.940	0.999	+0.059	0.993	-0.007
C./Rec.	1.0	3.3	+2.3	2.2	-1.07

A key benefit of using the non-disjoint algorithm with information bottleneck (IB) is that the clustering algorithm becomes less sensitive to the value of the threshold used for the similarity join. In the above results, changing the threshold for the MC algorithm to $\theta = 0.4$ results in a much higher PCPr but lower CPr score and setting $\theta = 0.2$ results in a significant drop in PCPr but higher CPr. The last row shows the average number of clusters to which each record belongs, e.g., in the non-disjoint algorithm with the threshold used for the results in this table, each record is present in 3.3 clusters on average. As it can be seen, PCPr and CPr are slightly decreased but in return, the average number of clusters for each record is significantly decreased. This results in decreasing the overhead associated with having non-disjoint clusters as well as increasing the precision of the clustering.

Effect of amount of error: In order to show the the effect of the amount of error in the datasets on the accuracy of the algorithms, we measure the CPr score of all the clustering algorithms, with threshold $\theta = 0.5$

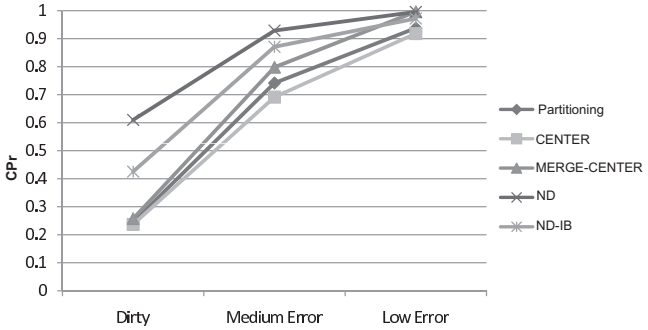


Fig. 9 CPr score of clustering algorithms for datasets with different amount of error

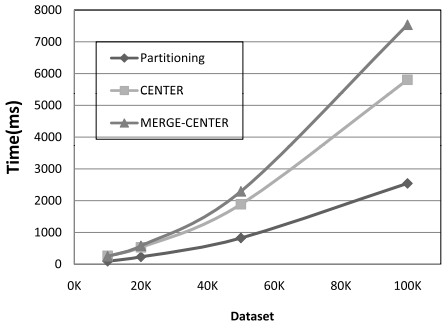


Fig. 10 Running time: disjoint algorithms

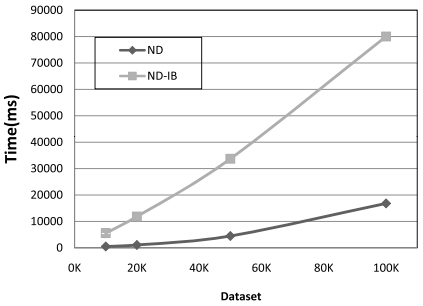


Fig. 11 Running time: non-disjoint algorithms

for disjoint algorithms and lower threshold $\theta_2 = 0.3$ and higher threshold $\theta_1 = 0.5$ for non-disjoint algorithms. Figure 9 shows the results. For all datasets, the relative performance of the algorithms remains the same. All algorithms perform better on lower error datasets. MERGE-CENTER algorithm becomes more effective on cleaner datasets comparing with Partitioning and CENTER algorithms. Non-disjoint algorithms become less effective on cleaner datasets mainly due to higher accuracy of the disjoint algorithm with the threshold used.

Performance Results: We ran our experiments using a Dell 390 Precision desktop with 2.66 GHz Intel Core2 Extreme Quad-Core Processor QX6700, 4GB of RAM running 32-bit Windows Vista. Each experiment is run multiple times to obtain statistical significance. Figures 10 and 11 show the running time of the disjoint and non-disjoint algorithms. These results are obtained from DBLP datasets of size 10K-100K records. The average percentage of erroneous duplicates is 50%, and the average percentage of errors in each duplicate record, the average amount of token swaps, and the average amount of abbreviation errors is 30%. For disjoint algorithms, a fix threshold of $\theta = 0.5$ is chosen for the similarity join and for non-disjoint algorithms lower threshold of $\theta = 0.4$ and higher threshold of $\theta = 0.6$ is chosen, although we observed a similar trend with many other threshold values. As expected, the Partitioning algorithm is the fastest in disjoint algorithms since it does not need the output of the similarity join to be sorted. CENTER and MERGE-CENTER both require the output to be sorted, and MERGE-CENTER has an extra merge operation which makes it a little slower than CENTER. The results for non-disjoint algorithms show that the overhead for the information bottleneck pruning makes the algorithm 5-10 times slower, but still reasonable for an offline process.

5 Probability Assignment Module

Assuming that the records in the base relation R are clustered using a clustering technique, the output of the probability assignment module is a probabilistic database in which each record has a probability value that reflects the error in the record. We present two classes of algorithms here. One based on the similarity score between the records in each cluster, and the other based on information theory concepts.

5.1 Algorithms

MaxSim Algorithm: In this algorithm, first a record in each cluster is chosen as the representative of a cluster and then the probability value is assigned to each record that reflects the similarity between the record and the cluster representative. This algorithm is based on the assumption that there exists a record in the cluster that is clean (has no errors) or has less errors, and that this record is the most similar record to other records in the cluster. Therefore, this record is chosen as the cluster representative and the probability of the other records being clean is proportional to their similarity score with the cluster's representative.

Figure 12 shows the generic procedure for finding probabilities in this approach. For each cluster, the record that has the maximum sum of similarity score with all

ALGORITHM MAXSIM
INPUT: A set of records R
A clustering $\mathcal{C} = \{c_1, c_2, \ldots, c_k\}$ of R
A similarity function $sim()$
1. Repeat for each cluster c_i:
2. let $rep = \arg\max_{r \in c_i} (\sum_{s \in c_i} sim(r, s))$
3. For each record t in cluster c_i:
4. $$p(t) = \frac{sim(t, rep)}{\sum_{r \in c_i} sim(r, rep)}$$

Fig. 12 MaxSim algorithm

ALGORITHM IB
INPUT: A set of records R
A clustering $\mathcal{C} = \{c_1, c_2, \ldots, c_k\}$ of R
1. Repeat for each cluster c_i:
2. Merge the records in the cluster (equation 12) to calculate $p(t|c_i)$ for each $t \in T_{c_i}$
3. For each record r in cluster c_i:
4. $$p_c(r) = \frac{p'_c(r)}{\sum_{r' \in c_i} p'_c(r')}$$
where $p'_c(r) = \frac{\sum_{t \in \mathbf{r}} p(t|c_i)}{|\mathbf{r}|}$

Fig. 13 IB algorithm

other records in the cluster (based on some similarity function $sim()$) is chosen as the cluster representative. The probability assigned to each record is basically the similarity score between the representative and the record, normalized for each cluster.

Information Bottleneck Method: Here, we present a technique for assigning probability values to records within each cluster based on the Information Bottleneck (IB) approach. While similar in spirit to the method of Andritsos et al. [2], our method is designed specifically for dirty string data. Assume again that R is the set of all records, T is the set of qgrams of the records, C is the set of all the clusters and T_{c_i} is the set of all qgrams in the records inside cluster $c_i \in C$. Giving equal weight to each record $r \in R$, we define $p(r) = \frac{1}{n}$. The probability of a qgram given a record can be set as $p(t|r) = \frac{1}{|r|}$ (equal values, as shown in the example below) or $p(t|r) = \frac{idf(t)}{\sum_{t' \in r} idf(t')}$ (based on importance of the tokens, which is our choice for the experiments). For $c \in C$, the elements of R, T and C are related by Equations 10 and 11 in Section 4. Merging two clusters c_i and c_j is performed by the merge operation using Equation 12 (Section 4).

Figure 13 shows the steps involved in this algorithm. To find a cluster representative for cluster c_i, we merge the records in the cluster using the merge operation. The result is the probability distribution $p(t|c_i)$ for all qgrams $t \in T_{c_i}$. We define the cluster representative to be $(T_{c_i}, p(t|c_i))$, i.e., the set of all the qgrams of the records in the cluster c_i along with their probability values $p(t|c_i)$. Note that a cluster representative does not necessarily consist of qgrams of a single record in that cluster. The probability value for each record in the cluster is basically the sum of the values of the probabilities $p(t|c_i)$ for the qgrams in the record r divided by the length of the record, normalized so that the probabilities of the records inside a cluster sum to 1. The intuition behind this algorithm is that by using the information from all the q-grams in the cluster, a better cluster representative can be found. This is based on the assumption that in the cluster c_i, the q-grams that belong to a "clean" record are expected to appear more in the cluster and therefore have a higher $p(t|c_i)$ value. As a result, the records containing q-grams that are frequent in the cluster (and are more likely to be clean) will have higher probability values.

Example 3 *Suppose R is a set of four strings $r1$ = "William Turner", $r2$ = "Willaim Turner", $r3$ = "Willliam Turnet" and $r4$ = "Will Turner" in a cluster. Figure 14 shows the initial $p(t|r)$ values for each record r and q-gram t, as well as the final probability distribution values for the cluster representative.*[3] *The output of the algorithm is $p_c(r1) = 0.254$, $p_c(r2) = 0.240$, $p_c(r3) = 0.233$ and $p_c(r4) = 0.272$.*

5.2 Evaluation

Measure: We evaluate the effectiveness of the probability assignment techniques by introducing a measure that shows how sorting by the assigned probability values will preserve the correct order of the error in the records. We call this measure *Order Preserving Ratio (OPR).* OPR is calculated as follows. For each cluster, we create an ordered list of records $L_{output} = (r_1, \ldots, r_k)$ sorted by the probability values assigned to the records, i.e., $p_a(r_i) \leq p_a(r_j)$ iff $i \leq j$ where $p_a(r)$ is the probability value assigned to the record r. Suppose the correct order of the records is $L_{correct}$ and the true probability value of the record r being the clean one is $p_t(r)$. We can measure the extent to which the sorted output list preserves the original order by counting the percentage of pairs (r_i, r_j) for which r_i appears before r_j in both L_{output} and $L_{correct}$, i.e.,

$$OPR_C = \frac{|(r_i, r_j)| r_i, r_j \in L_{output}, i \leq j, p_t(r_i) \leq p_t(r_j)|}{\binom{k}{2}} \quad (21)$$

Note that $\binom{k}{2}$ is the total number of pairs in L_{output}. OPR is the average of $\frac{OPR_c - 0.5}{0.5}$ over all clusters. Since

[3] We omit the initial and ending grams ' w', 't ', 'r ' to fit this on the page.

$r1$ = "William Turner", $r2$ = "Willaim Turner", $r3$ = "Willliam Turnet", $r4$ = "Will Turner"

t	'wi'	'il'	'll'	'li'	'la'	'l '	'ai'	'im'	'ia'	'am'	'm '	' T'	'Tu'	'ur'	'rn'	'ne'	'er'	'et'
$p(t\|r1)$	1/13	1/13	1/13	1/13	0	0	0	0	1/13	1/13	1/13	1/13	1/13	1/13	0	1/13	1/13	0
$p(t\|r2)$	1/13	1/13	1/13	0	1/13	0	1/13	1/13	0	0	1/13	1/13	1/13	1/13	1/13	1/13	1/13	0
$p(t\|r3)$	1/14	1/14	1/14	1/14	0	0	0	0	1/14	1/14	1/14	1/14	1/14	1/14	0	1/14	0	1/14
$p(t\|r4)$	1/10	1/10	1/10	0	0	1/10	0	0	0	0	0	1/10	1/10	1/10	1/10	1/10	1/10	0
$p(t\|rep)$	.071	.071	.071	.033	.017	.021	.017	.017	.033	.033	.050	.071	.071	.071	.071	.071	.054	.017

Fig. 14 Example IB representative calculation

0.5 is the average value of OPR_c if the records are sorted randomly, OPR shows the extent to which the ordering by probabilities is better than a random ordering.

Results: We use the same data generator to create a dataset of strings with different amounts of error within the strings, marking each string with the percentage of error in that string which allows sorting the records based on the relative amount of error and obtaining the ground truth. We ran experiments on datasets with varying sizes and degree of error made out of the company names and DBLP titles datasets (Table 1). The trends observed are similar over all datasets. We report the results for a dataset containing 1000 clusters generated out of our clean company names dataset. Table 3 shows the OPR values for this dataset for IB and MaxSim algorithm. We have tried MaxSim with different string similarity functions described in Section 3 for similarity join module, namely Weighted Jaccard (WJaccard), SoftTfIdf, Generalized Edit Similarity (GES), Hidden Markov Models (HMM), BM25 and Cosine similarity with tf-idf weights (Cosine w/tfidf). Interestingly, MaxSim produces the best results when used with Weighted Jaccard similarity, the measure of our choice for the similarity join module described in Section 3. The IB algorithm performs as well as MaxSim with the best choice of similarity function in terms of accuracy. Table 3 also shows the running time for these algorithms for a DBLP titles dataset of 20K records. The trend is similar for larger datasets and the algorithms scale linearly. The IB algorithm is also significantly faster than MaxSim with weighted Jaccard. Another advantage of IB over the MaxSim algorithm is that the cluster representatives can be stored and updated very efficiently, but for the MaxSim algorithm, when a record is added to database, the algorithm must be run again to find the new representative. This makes the IB algorithm suitable for large dynamic databases, and also for on-line calculation of the probabilities.

Table 3 OPR values/times for IB & MaxSim algs

Algorithm		OPR	Time(ms)
IB		**0.683**	**749**
MaxSim	WJaccard	**0.674**	4324
	SoftTfIdf	0.653	1280
	GES	0.490	**1249**
	HMM	0.485	3852
	BM25	0.480	4009
	Cosine w/tfidf	0.470	5397

5.3 Putting It All Together

In Section 4, we showed how the quality of the clusters is affected by the similarity measure and threshold used in the similarity join module. However, the results presented so far in this section are based on a perfect clustering as input to the probability assignment module. In this part, we will show the results of our experimental evaluation of the effect of the quality of the clusters on the quality of the probabilities. Our goal is to ensure that when creating a probabilistic database, the errors introduced in the first two modules (the clustering errors) do not compound the potential errors in our probability assignment module in a way that makes the final probabilities meaningless.

Measure: We need to slightly modify OPR to measure the quality of the probability values when the clustering is imperfect. We call this measure OPR_t. Suppose that we have a set of k ground truth clusters $G = \{g_1, \ldots, g_k\}$ of the base relation R and let C denote a clustering of records into k' clusters $\{c_1, \ldots, c_{k'}\}$ produced by a clustering algorithm. Consider mapping f from elements of C to elements of G, such that each cluster c_i is mapped to a cluster $f(c_i)$ that has the highest percentage of common elements with c_i. Here again, we create an ordered list of records $L = (r_1, \ldots, r_k)$ for each cluster $c_l \in C$, sorted by the probability values assigned to the records, i.e., $p_a(r_i) \leq p_a(r_j)$ iff $i \leq j$ where $p_a(r)$ is the probability value assigned to the record r. Let $p_t(r \in c_l)$ be the probability value of the record r being the ground truth cluster $f(c_l)$ if $t \in f(c_l)$ and zero otherwise. We can measure the extent to which the sorted output list preserves the original order in the matched ground truth cluster $f(c_l)$ by counting the percentage of pairs (r_i, r_j) for which at least one of r_i and r_j are in $f(c_l)$, $p_{t \in c}(r_i) \leq p_{t \in c}(r_j)$ and r_i appears before r_j in L, i.e.,

$$\frac{|(r_i, r_j)|r_i, r_j \in L, i \leq j, p_t(r_i \in c_l) \leq p_t(r_j \in c_l)| - e}{\binom{k}{2} - e}$$

$$e = |(r_i, r_j)|r_i, r_j \notin f(c_l)|$$

This is based on the assumption that we are indifferent about the order of the records that are not in the matched ground truth cluster. OPR_t is the average of the value calculated in the formula above over all output clusters in C.

Results: The table below shows OPR_t values for the same dataset used for the results in Table 3. The values are shown for a perfect clustering as well as clusters created by (disjoint) MERGE-CENTER algorithm performed on the output of similarity join with Weighted Jaccard similarity measure and different values of the similarity threshold, and using the IB algorithm for probability assignment. Similar trends were observed for other clustering algorithms. Moreover, the relative performance of IB and MaxSim algorithms remained the same as Table 3 and therefore we do not report OPR_t values for them.

Similarity Threshold	F1	PCPr	Cluster#	OPR_t
Perfect Clustering (No Similarity Join)	1.000	1.000	500	0.832
$\theta = 0.1$	0.008	0.004	2	0.571
$\theta = 0.2$	0.479	0.389	259	0.655
$\theta = 0.3$	0.726	0.335	934	0.713
$\theta = 0.4$	0.724	0.118	1673	0.700
$\theta = 0.5$	0.614	0.042	2370	0.625

The results above show that the quality of the clustering does affect the effectiveness of the probability assignment module. This effect is not significant when the clusters have higher accuracy. However, the quality of the probability values further decreases as the accuracy of the clustering decreases.

6 Case Study on Real Data

In this section, we report the results of applying our framework to a real world dirty data source. In order to effectively evaluate our framework, we need a dirty data source that contains several possibly dirty attributes with duplicate clusters of various sizes and characteristics. Many real world dirty data sources meet these requirements. Examples include the bibliographic data available on DBLP, CiteSeer and DBWorld, the clinical trial data available on ClinicalTrials.gov, shopping information on Yahoo! Shopping, and hotel information from Yahoo! Travel [9]. For the experiments in this section, we use the Cora dataset [39], which contains computer science research papers integrated from several sources. It has been used in several other duplicate detection projects [2,5,13,39] and we take advantage of previous labelings of the tuples into clusters. To the best of our knowledge, Cora is the only real world dirty database freely available for which the ground truth is known, and that meets the requirements for evaluation of this framework.

Table 4 Statistics of the tables in Cora dataset

dataset	#rec.	#clusters	Avg. len.	#words/rec.
`pubstr`	1,878	185	118.22	17.76
`pubtitles`	1,878	185	50.84	6.13
`pubauthors`	714	240	13.76	2.78
`pubvenues`	615	131	47.07	8.58

We use a version of Cora that is available in XML format, and transform the data into four relational tables: the `pubstr` table contains a single string attribute which is obtained by concatenation of the title, venue and author attributes, `pubtitles` which contains the titles of the publications, `pubauthors` that contains the author names, and `pubvenues` that contains the venue information including name, date and volume number. The statistics of these tables are shown in Table 4.

6.1 Similarity Join Results

Figure 15 shows the maximum F_1 score across different thresholds for all the similarity measures over the four tables. The relative performance of the similarity measures differs considerably for each of these tables. This is expected since 1) the attributes have different characteristics such as length, amount and type of errors, and 2) these tables are relatively small, and failure of an algorithm on a small subset the records can notably affect the average values of the accuracy measures. However, it can be seen that those algorithms that performed better in our experiments in Section 3, are more robust across the four tables. For example, the weighted Jaccard similarity measure performs reasonably well for all the four tables, although it is not the best measure for any of them. Note that again due to the small size of these tables, weighted measures do not perform as expected since the IDF weights over a small collection do not reasonably reflect the commonality of the tokens. We would not expect this to be the case for larger real world dirty data.

6.2 Clustering Algorithms Results

In order to compare the performance of clustering algorithms on the datasets, we again compare the maximum value of the F_1 score and PCPr that the algorithms can achieve using different thresholds. Figure 16 shows the results. All the clustering algorithms perform better on the `pubstr` and `pubtitles` tables. The reason for this is that for the `pubauthors` table, our framework's duplicate detection phase (i.e., a string similarity join along

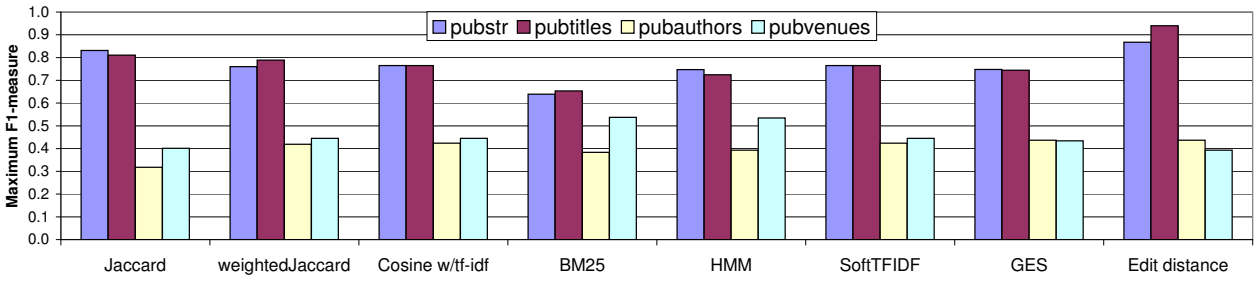


Fig. 15 Maximum F_1 score for Cora datasets

with a clustering algorithm) results in many false positives due to the existence of highly similar (or exactly equal) author names that refer to different real world entities. The same is true for the `pubvenues` table. As stated in Section 2, previous work has addressed this problem by using more complex, iterative clustering algorithms that can take advantage of additional co-occurrence information existing in the data. However, the relatively high quality of the clusters for `pubstr` table shows the effectiveness of our framework in detection of duplicate publications by a simple concatenation of all the attributes and without the use of co-occurrence information (indeed collective resolution has been developed precisely for highly ambiguous domains like author name).

If the same threshold is used for all the algorithms, CENTER produces clusters of much higher quality when used with a low threshold, while the trend for Partitioning is the opposite. MERGE-CENTER is more robust to the value of the threshold than both Partitioning and CENTER when the same threshold is used. Figure 17 shows this fact for `pubstr` table. Similar trends were observed in all other datasets.

The following table shows the effectiveness of the non-disjoint algorithms for the `pubstr` table. Again, similar trends were observed for the other tables.

	MC $\theta = 0.2$	ND $\theta_1 = 0.1$, $\theta_2 = 0.3$		ND-IB $\theta_1 = 0.1$, $\theta_2 = 0.3$	
			Diff.(MC)		Diff.(MC)
F1	0.789	0.756	-0.033	0.833	+0.043
PCPr	0.728	0.965	+0.238	0.952	+0.224
CPr	0.975	0.998	+0.022	0.984	+0.009
C./Rec.	1.0	2.7	+1.7	1.8	+0.8

6.3 Probability Assignment

The evaluation of the probability assignment algorithms for this dataset is inherently a difficult task since the ground truth is not known (only the cluster labels are known). It is hard to determine the correct ordering of the records within each cluster. However, we have performed a qualitative evaluation of the results over the output probabilistic tables using simple queries similar

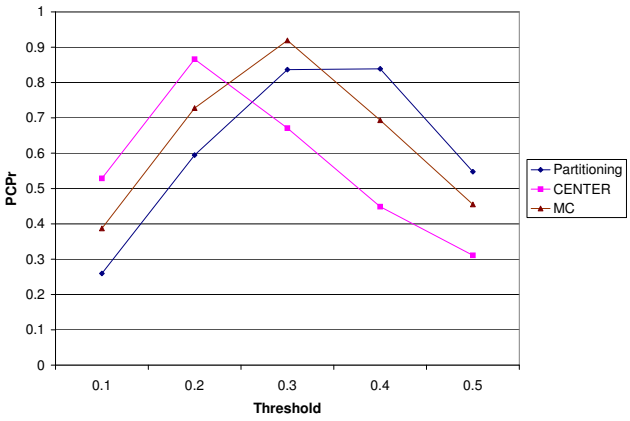
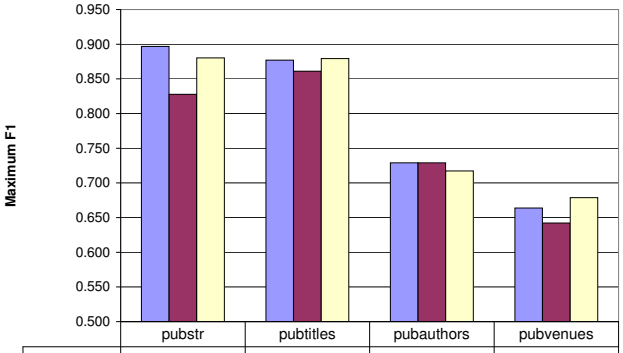


Fig. 17 PCPr score for different thresholds on `pubstr` table

to the queries in the examples of Section 2.4.2. Overall results are consistent with the results shown in Section 5. Moreover, the results clearly show the advantage of the probabilistic approach for management of duplicated data, as opposed to cleaning the data upfront. As one example, we used a query retrieving conference title, volume and other information for conferences held in 1995. Over a cleaned database (where we have kept the most probable tuple in each cluster), the query results are less informative, sometimes omitting potentially valuable information about a conference that was contained in attribute values of lower probability tuples. However, using consistent query answering techniques [2], queries over our probabilistic database can report how much collective evidence there is (among all the tuples no matter how dirty) for different values. Our sample SQL queries, along with their rewritings obtained using the approach discussed in Section 2.4.2 [2] and a subset of their results are available online at our project's web page:

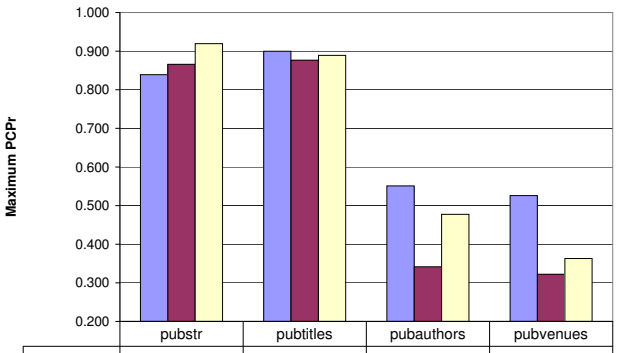
`http://dblab.cs.toronto.edu/project/stringer/evaluation/`

Also, several probabilistic tables created from synthetic and real dirty databases using different thresholds and algorithms are published on the above page. We hope that these probabilistic databases can serve as a benchmark for evaluation of probabilistic data man-



	pubstr	pubtitles	pubauthors	pubvenues
Partitioning	0.897	0.877	0.729	0.664
CENTER	0.828	0.861	0.729	0.642
MC	0.880	0.879	0.717	0.679

(a) Maximum F_1 score



	pubstr	pubtitles	pubauthors	pubvenues
Partitioning	0.839	0.900	0.551	0.526
CENTER	0.866	0.877	0.342	0.322
MC	0.919	0.889	0.477	0.363

(b) Maximum PCPr score

Fig. 16 Accuracy of clustering algorithms on Cora dataset

agement techniques in the future. Our future plan includes extending the real datasets by, for example, hand labeling a subset of the clinical trials data we have gathered in our LinkedCT[4] project. This could provide probabilistic databases for management of duplicated data in an important real-world domain.

7 Conclusion

We proposed a framework for managing potentially duplicated data that leverages existing approximate join algorithms together with probabilistic data management techniques. Our approach consists of three phases: application of a (scalable) approximate join technique to identify the similarity between pairs of records; clustering of records to identify sets of records that are potential duplicates; the assignment of a probability value to each record in the clusters that reflects the error in the record. We presented and benchmarked a set of scalable algorithms for clustering records based on their similarity scores and on their information content. We also introduced and evaluated algorithms for probability assignment.

The modularity of our framework makes it amenable for a variety of data cleaning tasks. For example, in domains where aggregate constraints for deduplication are known [18], these constraints can replace our unsupervised clustering techniques, and our probability assignment methods can still be used to create a probabilistic database for querying and analysis.

Acknowledgments. We thank Periklis Andritsos, Lise Getoor and Chen Li for their detailed reviews, insights and support of this work. We also thank Mohammad Sadoghi and George Beskales for their helpful input.

[4] `http://linkedct.org`

References

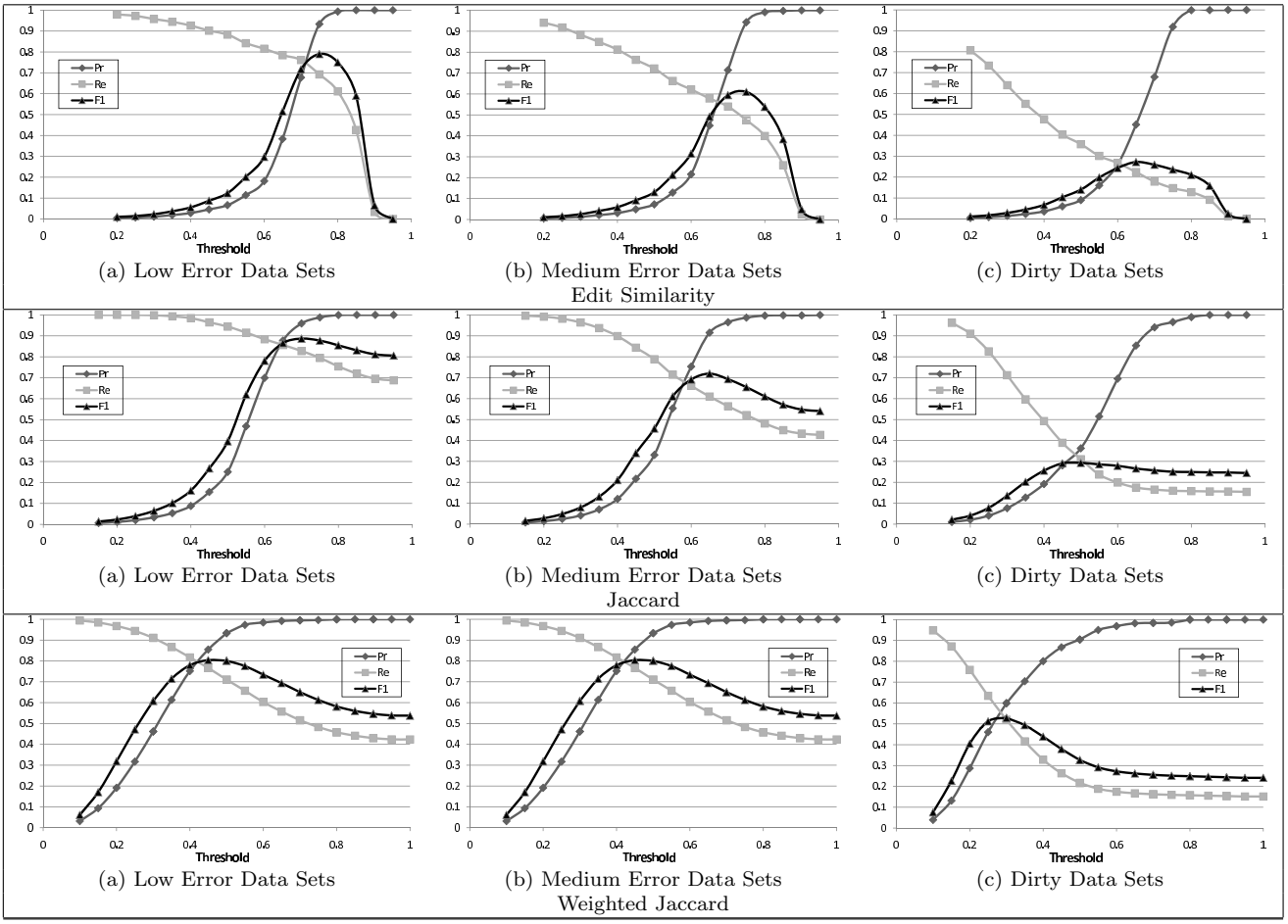
1. N. Ailon, M. Charikar, and A. Newman. Aggregating Inconsistent Information: Ranking and Clustering. In *ACM Symp. on Theory of Computing (STOC)*, pages 684–693, 2005.
2. P. Andritsos, A. Fuxman, and R. J. Miller. Clean Answers over Dirty Databases: A Probabilistic Approach. In *IEEE Proc. of the Int'l Conf. on Data Eng.*, page 30, 2006.
3. L. Antova, C. Koch, and D. Olteanu. Fast and Simple Relational Processing of Uncertain Data. In *IEEE Proc. of the Int'l Conf. on Data Eng.*, pages 983–992, 2008.
4. A. Arasu, V. Ganti, and R. Kaushik. Efficient Exact Set-Similarity Joins. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 918–929, 2006.
5. A. Arasu, C. Ré, and D. Suciu. Large-Scale Deduplication with Constraints Using Dedupalog. In *IEEE Proc. of the Int'l Conf. on Data Eng.*, pages 952–963, 2009.
6. J. A. Aslam, E. Pelekhov, and D. Rus. The Star Clustering Algorithm For Static And Dynamic Information Organization. *Journal of Graph Algorithms and Applications*, 8(1):95–129, 2004.
7. N. Bansal, A. Blum, and S. Chawla. Correlation Clustering. *Machine Learning*, 56(1-3):89–113, 2004.
8. R. J. Bayardo, Y. Ma, and R. Srikant. Scaling Up All Pairs Similarity Search. In *Int'l World Wide Web Conference (WWW)*, pages 131–140, Banff, Canada, 2007.
9. O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom. Swoosh: A Generic Approach to Entity Resolution. *The Int'l Journal on Very Large Data Bases*, 18(1):255–276, 2009.
10. G. Beskales, M. A. Soliman, I. F. Ilyas, and S. Ben-David. Modeling and Querying Possible Repairs in Duplicate Detection. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, 2009 (To Appear). Available as University of Waterloo, Tech. Report CS-2009-15, 2009.
11. J. C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Kluwer Academic Publishers, 1981.
12. I. Bhattacharya and L. Getoor. A Latent Dirichlet Model for Unsupervised Entity Resolution. In *Proc. of the SIAM International Conference on Data Mining (SDM)*, pages 47–58, Bethesda, MD, USA, 2006.

13. I. Bhattacharya and L. Getoor. Collective Entity Resolution in Relational Data. *IEEE Data Engineering Bulletin*, 29(2):4–12, 2006.
14. D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.
15. J. Boulos, N. Dalvi, B. Mandhani, S. Mathur, C. Re, and D. Suciu. MYSTIQ: A System For Finding More Answers By Using Probabilities. In *ACM SIGMOD Int'l Conf. on the Mgmt. of Data*, pages 891–893, 2005.
16. S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and Efficient Fuzzy Match for Online Data Cleaning. In *ACM SIGMOD Int'l Conf. on the Mgmt. of Data*, pages 313–324, 2003.
17. S. Chaudhuri, V. Ganti, and R. Motwani. Robust Identification of Fuzzy Duplicates. In *IEEE Proc. of the Int'l Conf. on Data Eng.*, pages 865–876, Washington, DC, USA, 2005.
18. S. Chaudhuri, A. Das Sarma, V. Ganti, and R. Kaushik. Leveraging Aggregate Constraints for Deduplication. In *ACM SIGMOD Int'l Conf. on the Mgmt. of Data*, pages 437–448, 2007.
19. R. Cheng, J. Chen, and X. Xie. Cleaning Uncertain Data with Quality Guarantees. *Proceedings of the VLDB Endowment (PVLDB)*, 1(1):722–735, 2008.
20. W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A Comparison of String Distance Metrics for Name-Matching Tasks. In *Proc. of IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03)*, pages 73–78, Acapulco, Mexico, 2003.
21. N. Dalvi and D. Suciu. Efficient Query Evaluation on Probabilistic Databases. *The Int'l Journal on Very Large Data Bases*, 16(4):523–544, 2007.
22. N. Dalvi and D. Suciu. Management of Probabilistic Data: Foundations and Challenges. In *ACM SIGMOD Int'l Conf. on the Mgmt. of Data*, pages 1–12, 2007.
23. E. D. Demaine, D. Emanuel, A. Fiat, and N. Immorlica. Correlation Clustering In General Weighted Graphs. *Theor. Comput. Sci.*, 361(2):172–187, 2006.
24. X. L. Dong, A. Y. Halevy, and C Yu. Data Integration with Uncertainty. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 687–698, 2007.
25. A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007.
26. I. P. Fellegi and A. B. Sunter. A Theory for Record Linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.
27. G. W. Flake, R. E. Tarjan, and K. Tsioutsiouliklis. Graph Clustering and Minimum Cut Trees. *Internet Mathematics*, 1(4):385–408, 2004.
28. L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate String Joins in a Database (Almost) for Free. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 491–500, 2001.
29. R. Gupta and S. Sarawagi. Creating Probabilistic Databases from Information Extraction Models. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 965–976, 2006.
30. D. Gusfield. *Algorithms on Strings, Trees, and Sequences.* Computer Science and Computational Biology. Cambridge University Press, 1997.
31. O. Hassanzadeh. Benchmarking Declarative Approximate Selection Predicates. Master's thesis, University of Toronto, February 2007.
32. O. Hassanzadeh, F. Chiang, H. C. Lee, and R. J. Miller. Framework for Evaluating Clustering Algorithms in Duplicate Detection. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, 2009.
33. O. Hassanzadeh, M. Sadoghi, and R. J. Miller. Accuracy of Approximate String Joins Using Grams. In *Proc. of the International Workshop on Quality in Databases (QDB)*, pages 11–18, Vienna, Austria, 2007.
34. T. H. Haveliwala, A. Gionis, and P. Indyk. Scalable Techniques for Clustering the Web. In *Proc. of the Int'l Workshop on the Web and Databases (WebDB)*, pages 129–134, Dallas, Texas, USA, 2000.
35. M. A. Hernández and S. J. Stolfo. Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.
36. P. Indyk, R. Motwani, P. Raghavan, and S. Vempala. Locality-Preserving Hashing in Multidimensional Spaces. In *ACM Symp. on Theory of Computing (STOC)*, pages 618–625, 1997.
37. K. Kukich. Techniques for Automatically Correcting Words in Text. *ACM Computing Surveys*, 24(4):377–439, 1992.
38. C. Li, B. Wang, and X. Yang. VGRAM: Improving Performance of Approximate Queries on String Collections Using Variable-Length Grams. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 303–314, Vienna, Austria, 2007.
39. A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proc. of the Int'l Conf. on Knowledge Discovery & Data Mining*, pages 169–178, 2000.
40. D. R. H. Miller, T. Leek, and R. M. Schwartz. A Hidden Markov Model Information Retrieval System. In *ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 214–221, 1999.
41. A. E. Monge and C. Elkan. An Efficient Domain-Independent Algorithm for Detecting Approximately Duplicate Database Records. In *Proc. of SIGMOD Workshop on Data Mining and Knowledge Discovery (DMKD)*, 1997.
42. E. Rahm and H. Hai Do. Data Cleaning: Problems and Current Approaches. *IEEE Data Engineering Bulletin*, 23(4):3–13, 2000.
43. C. Re, N. Dalvi, and D. Suciu. Efficient Top-k Query Evaluation on Probabilistic Data. In *IEEE Proc. of the Int'l Conf. on Data Eng.*, pages 886–895, 2007.
44. S. Robertson. Understanding Inverse Document Frequency: On Theoretical Arguments for IDF. *Journal of Documentation*, 60(5):503–520, 2004.
45. S. Sarawagi and A. Kirpal. Efficient Set Joins On Similarity Predicates. In *ACM SIGMOD Int'l Conf. on the Mgmt. of Data*, pages 743–754, Paris, France, 2004.
46. P. Sen, A. Deshpande, and L. Getoor. Representing Tuple and Attribute Uncertainty in Probabilistic Databases. In *ICDM Workshops*, pages 507–512, 2007.
47. N. Slonim. *The Information Bottleneck: Theory And Applications.* PhD thesis, The Hebrew University, 2003.
48. M. A. Soliman, I. F. Ilyas, and K. C. Chang. Top-k Query Processing in Uncertain Databases. In *IEEE Proc. of the Int'l Conf. on Data Eng.*, pages 896–905, 2007.
49. M. A. Soliman, I. F. Ilyas, and K. C. Chang. Probabilistic top-k and ranking-aggregate queries. *ACM Trans. on Database Sys. (TODS)*, 33(3):1–54, 2008.
50. S. van Dongen. *Graph Clustering By Flow Simulation.* PhD thesis, University of Utrecht, 2000.
51. Jennifer Widom. Trio: A System for Integrated Management of Data, Accuracy, and Lineage. In *Proc. of the Conference on Innovative Data Systems Research (CIDR)*, pages 262–276, 2005.

Appendices

A Similarity Join Evaluation: Precision/Recall Curves

Figures 18 and 19 show the precision, recall, and F_1 values for all measures described in Section 2, over the datasets we have defined with mixed types of errors. For all measures except HMM and BM25, the horizontal axis of the precision/recall graph is the value of the threshold. For HMM and BM25, the horizontal axis is the percentage of maximum value of the threshold, since these measure do not return a score between 0 and 1.



(a) Low Error Data Sets (b) Medium Error Data Sets (c) Dirty Data Sets

Edit Similarity

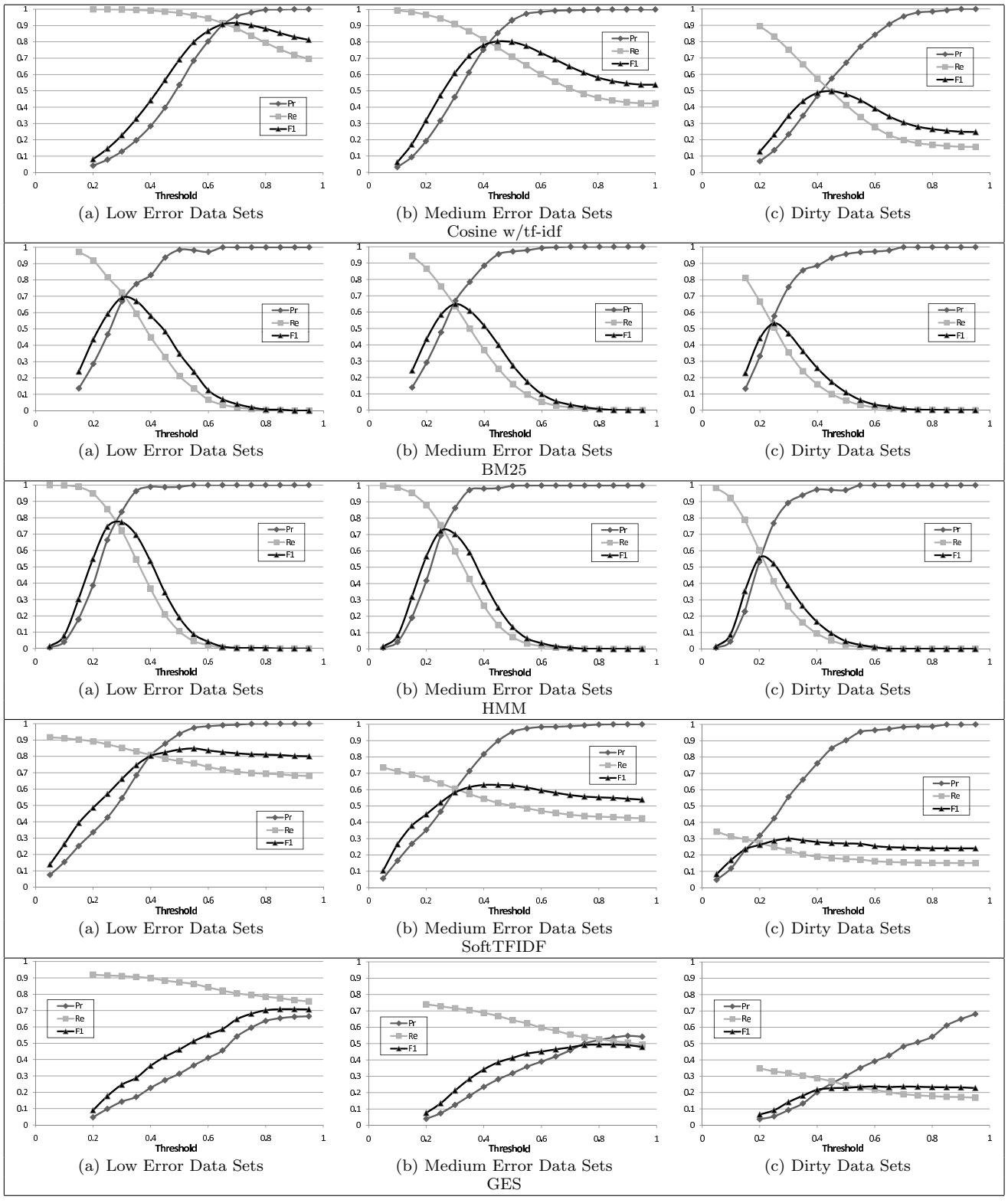
(a) Low Error Data Sets (b) Medium Error Data Sets (c) Dirty Data Sets

Jaccard

(a) Low Error Data Sets (b) Medium Error Data Sets (c) Dirty Data Sets

Weighted Jaccard

Fig. 18 Accuracy of similarity join using Edit-Similarity, Jaccard and Weighted Jaccard measures relative to the value of the threshold on different datasets



(a) Low Error Data Sets (b) Medium Error Data Sets (c) Dirty Data Sets

Cosine w/tf-idf

(a) Low Error Data Sets (b) Medium Error Data Sets (c) Dirty Data Sets

BM25

(a) Low Error Data Sets (b) Medium Error Data Sets (c) Dirty Data Sets

HMM

(a) Low Error Data Sets (b) Medium Error Data Sets (c) Dirty Data Sets

SoftTFIDF

(a) Low Error Data Sets (b) Medium Error Data Sets (c) Dirty Data Sets

GES

Fig. 19 Accuracy of similarity join using measures from IR and hybrid measures relative to the value of the threshold on different datasets