Auditing Disclosure by Relevance Ranking

Rakesh Agrawal[*]
Microsoft Search Labs
1065 La Avenida
Mountain View, CA
rakesh.agrawal@microsoft.com

Alexandre Evfimievski   Jerry Kiernan
IBM Almaden Research Center
650 Harry Rd.
San Jose, CA
{evfimi,jkiernan}@us.ibm.com

Raja Velu[*]
Yahoo! Inc.
701 First Ave.
Sunnyvale, CA
rvelu@yahoo-inc.com

ABSTRACT

Numerous widely publicized cases of theft and misuse of private information underscore the need for audit technology to identify the sources of unauthorized disclosure. We present an auditing methodology that ranks potential disclosure sources according to their proximity to the leaked records. Given a sensitive table that contains the disclosed data, our methodology prioritizes by relevance the past queries to the database that could have potentially been used to produce the sensitive table. We provide three conceptually different measures of proximity between the sensitive table and a query result. One measure is inspired by information retrieval in text processing, another is based on statistical record linkage, and the third computes the derivation probability of the sensitive table in a tree-based generative model. We also analyze the characteristics of the three measures and the corresponding ranking algorithms.

Categories and Subject Descriptors: H.2.4 [Database Management] : Systems

General Terms: Algorithms, Security

Keywords: Hippocratic database, privacy, information retrieval, record linkage, derivation probability

1. INTRODUCTION

As enterprises collect and maintain increasing amounts of personal data, individuals are exposed to greater risks of privacy breaches and identity theft. Many recent reports of personal data theft and misappropriation highlight these risks. As a result, many countries have enacted data protection laws requiring enterprises to account for the disclosure of personal data they manage [3, 5, 8, 14]. Hence, modern information systems must be able to track who has disclosed sensitive data and the circumstances of disclosure. For instance, the U.S. President's Information Technology Advisory Committee in its report on healthcare recommends that healthcare information systems must have the capability to audit who has accessed patient records [24].

We assume there is a data table called sensitive table, which is suspected to have originated from one or more queries that were run against a given database. Information on the past queries is available from a query log. Since the number of queries can be very large, our goal is to rank them so that the more likely sources of leakage can be examined by the auditor first.

We rank the queries based on the proximity of their results with the sensitive table. We provide and compare three methods of measuring proximity:

Partial Tuple Matching (PTM) This method measures the proximity of a query result to the sensitive table by considering common pieces of information (partial tuple matches) between the tuples of the two tables, while factoring in the rarity of a match at the same time. This method is inspired by the TF-IDF (term frequency - inverse document frequency) measure from the information retrieval literature [28].

Statistical Tuple Linkage (STL) This method employs statistical record matching techniques and mixture model parameter estimation via expectation maximization to find the best one-to-one match between the closest tuples in the two tables, and then evaluates the overall proximity by aggregating the scores of individual matches. This proximity measure has roots in the record linkage literature [6, 12, 30].

Derivation Probability Gain (DPG) This method, inspired by the minimum description length principle [25], evaluates proximity of the sensitive table to the query result table by computing the gain in probability for the sensitive tuples through their maximum-likelihood derivation from the query result table.

The following scenario illustrates a practical application of the proposed auditing system. Sophie, who is the privacy officer of Physicians Inc., comes across a promotion that includes a table of names of patients who have been treated and benefited from a newly introduced HIV treatment. Sophie becomes suspicious that this table might have been extracted from queries run against her company's database. There are very many queries run everyday, but fortunately they are logged along with the timestamp and other information such as who ran them. The database system also versions previous state before updating any data item to meet the need of reconstructing history as needed. Sophie can use the techniques proposed in this paper to identify

[*] The work on the algorithms presented in this paper was done while the authors were at IBM.

SIGMOD'07, June 11–14, 2007, Beijing, China.
Copyright 2007 ACM 978-1-59593-686-8/07/0006 ...$5.00.

and rank the queries that she should examine first for investigating this potential data leak.

1.1 Related Work

The problem of auditing a log of past queries and updates by means of an audit query that represents the leaked data has been studied in [1]. The suspicious queries are identified by finding past queries in the log whose results depend on the same "indispensable" data tuples as the audit query; a tuple is considered indispensable for a query if its omission makes the result of the query different. The indispensability of a tuple bears resemblance to the notion of a critical tuple in [21]. However, given some sensitive data, it is often difficult to formulate a concise audit query with near-perfect recall and precision. Moreover, the tuples in the sensitive table may have undergone a certain amount of arbitrary perturbation. Finally, the number of suspicious queries produced can be very large, necessitating an ordering based on relevance for an auditor's investigation.

Database watermarking [2] has also been proposed to track the disclosure of information. Database fingerprinting [18] can additionally identify the source of a leak by injecting different marks in different released copies of the data. Both the techniques require data to be modified to introduce a pattern and then recover the pattern in the sensitive data to establish disclosure. These techniques depend on the availability of a set of attributes that can withstand alteration without significantly degrading their value. They also require that a large portion of the pattern is carried over in the sensitive data.

As has been pointed out in [1], Oracle [23] offers a "fine-grained auditing" function where the administrator can specify that read queries should be logged if they access specified tables. This function logs various user context data along with the query issued, the time it was issued, and other system parameters such as the "system change number". Oracle also supports "flashback queries" whereby the state of the database can be reverted to the state implied by a given system change number. A logged query can then be rerun as if the database was in that state to determine what data was revealed when the query was originally run. However, there does not appear to be any automated facility to determine which queries should be audited.

Paper layout. The remainder of the paper is organized as follows. Section 2 gives the problem definition for ranking queries for the purpose of auditing. Sections 3-5 present our three methods for measuring proximity and ranking queries. Section 6 contrasts the characteristics of each method. Section 7 is an empirical study of the effectiveness of the proposed methods. Section 8 offers some concluding remarks and describes opportunities for future work.

2. AUDITING QUERY LOGS

2.1 Problem Definition and Notation

We are given a table S that contains sensitive data suspected to have been misappropriated (the *sensitive table* for short). S has schema $A_1 \times A_2 \times \ldots \times A_d$ where d is the number of attributes and A_j is the domain of the j^{th} attribute. The auditor wants to find a ranked list of the past queries to the database D that could have potentially been used to produce S. [1]

All the past queries issued over a period of time against the database D are available in a query log L. We assume, for simplicity, that the results produced by all logged queries $Q_1, \ldots, Q_n$ have the same schema as S, namely $A_1 \times A_2 \times \ldots \times A_d$ where d is the number of attributes and A_j is the domain of the j^{th} attribute.

For conciseness, we will refer to the table resulting from the execution of a query Q simply as the *query table* and abuse the notation by denoting it also as Q. We will view a table as a matrix and use lower index s_i or q_i for tuples in the i^{th} position of their corresponding tables. We will use upper index s_i^j, q_i^j to refer to the j^{th} attribute of the i^{th} tuple.

2.2 Schema Mapping and Database Updates

We stated earlier that this paper will assume that all the logged queries Q_i have the same schema as the sensitive table S. In general, the schema of the logged queries, as well as of the database itself, may differ from the schema of the sensitive table. While the problem of schema matching remains complex and needs to be fully addressed in future work, we finesse the issue by assuming that the auditor provides a one-to-one mapping query V to map attributes $A_j \in S$ to attributes of the database tables $A_j \in T_i \in D$.

The candidate set of suspicious queries $Q_1, \ldots, Q_n$ comprises of queries that have at least one table and at least one projected attribute in common with those mapped by V. If needed, we use V to rename the projected attributes of Q_i to match the schema of S. If a query table has extra attributes beyond the common schema, we omit them. If an attribute $A_j \in S$ is not projected by Q_i, we add a column of null values in its place to match S's schema.

We reuse the techniques from [1] for the organization of the query log and for recovering the state of the database at the time of each individual query. Briefly, for each table T in the database, all versions of tuples $t \in T$ are maintained in a backlog table such that the version of T at the time of any query Q_i in the query log can easily be reconstructed from its backlog table. For the purposes of this paper, we ignore schema changes that might have occurred over time.

3. PARTIAL TUPLE MATCHING

Our first method of measuring proximity between query results and tables is inspired by prior work in information retrieval [28]. In order to rank text documents by relevance to keyword searches, a document is commonly represented by a weighted vector of terms $\langle y_1, \ldots, y_N \rangle$. A non-zero value in y_k indicates that the term t_k is present in the document, and its weight represents the term's search value. The weight depends on the term frequency in the document and on the inverse frequency across all documents that use the term (TF-IDF). The smaller the number of documents having t_k, the more valuable t_k is for relevance ranking.

In the context of database auditing, the terms are tuples in the query tables and the documents are the query tables Q_1 through Q_n, while the tuples in the sensitive table S is the collection of keywords to search for. However, there are significant differences between this context and that of information retrieval:

[1] The queries may be perfectly legitimate, but their results may have subsequently been stolen or inappropriately disclosed. The exact cause of the disclosure is determined by comprehensive investigation; our goal is to provide a tool that helps to focus and prioritize the leads.

1. Term frequency in Q_i, i.e. the number of duplicate tuples, adds no value to a match between S and Q_i.
2. Document frequency, i.e. the number of tables in $\{Q_1, \ldots, Q_n\}$ having a given tuple $t \in S$, is critically important: we are looking precisely for the queries that could have contributed t to S.
3. Tuples can match partially, when only a subset of their attributes match. Even a single common value, if rare, can be a significant indication of disclosure.
4. The number of logged queries $n = \big|\{Q_1, \ldots, Q_n\}\big|$ may be very large or very small, depending on how these queries were selected.

We could address the issue of partial matches by treating attribute values as terms, rather than tuples as terms. However, if only combinations of attribute values are rare, but not the individual values, such single-attribute matching would miss important disclosure clues. To handle combinations, we enrich the "term vocabulary" by all possible *partial tuples*, with some attribute values replaced with wildcards (here denoted by "$*$"). For example, one full tuple $\langle a, b, c\rangle$ is augmented with six[2] partial ones: $\langle *, b, c\rangle$, $\langle a, *, c\rangle$, $\langle a, b, *\rangle$, $\langle a, *, *\rangle$, $\langle *, b, *\rangle$, and $\langle *, *, c\rangle$.

Definition 1. Table Q_i is said to *contain*, or *instantiate*, a partial tuple t when the wildcards in t can be instantiated with attribute values to produce a tuple $q \in Q_i$. The *frequency count* of a partial tuple t in a collection of tables $\{Q_1, \ldots, Q_n\}$, denoted by $\operatorname{freq}(t)$, is the number of the Q_i's that contain t.

If we take a table with 1000 tuples and 30 attributes and augment it with all possible partial tuples, we will have about $1000 \cdot 2^{30} \approx 10^{12}$ tuples, too many even by modern database standards. We limit this combinatorial explosion by restricting attention to the terms we search for, i.e. the partial tuples contained in S. Furthermore, for each query table Q_i we generate a single partial tuple per each tuple in S. Every Q_i is thus represented by the same number $|S|$ of partial tuples, regardless of its own size $|Q_i|$.

For each query Q_i and for each tuple $s \in S$ we find a single "representative" partial tuple t such that (1) t can be instantiated to s and to some tuple $q \in Q_i$, and (2) t has the smallest frequency count $\operatorname{freq}(t)$ across all such tuples. Condition 1 ensures that t represents common information between s and Q_i, while condition 2 picks a tuple most valuable for our search. Such tuple t can always be found among intersections $s \wedge q$ for $q \in Q_i$ defined below:

Definition 2. Let s and q be two tuples of the same schema. Their *intersection* $t = s \wedge q$ has a value at each attribute where s and q share this same value, and has wildcards at all other attributes. In other words, t is the most informative partial tuple that can be instantiated to both s and q. Example: $\langle a, b, c\rangle \wedge \langle a, b, d\rangle = \langle a, b, *\rangle$.

Tuple t that satisfies conditions 1 and 2 may not be unique; however, its frequency count is unique as a function of Q_i and s and is computed as follows:

$$\operatorname{minf}(s, Q_i) \;\stackrel{\text{def}}{=}\; \min_{q \,\in\, Q_i} \operatorname{freq}(s \wedge q).$$

Every Q_i corresponds to a multiset (bag) of exactly $|S|$ minimum frequency counts $\operatorname{minf}(s, Q_i)$, one count for each tuple $s \in S$. It is convenient to represent this multiset as a histogram: a sequence of numbers $h_1, h_2, \ldots, h_n$ where h_k is the number of tuples $s \in S$ giving the minimum frequency count of k. Denote this *frequency histogram* by $\operatorname{hist}(Q_i)$:

$$\begin{aligned} \operatorname{hist}(Q_i) \;&\stackrel{\text{def}}{=}\; (h_1, h_2, \ldots, h_n) \\ &\text{where}\quad h_k \;=\; \big|\{s \in S \mid \operatorname{minf}(s, Q_i) = k\}\big|. \end{aligned} \qquad (1)$$

Given the critical importance of document frequency counts in relevance ranking, we decided to use the above frequency histogram $\operatorname{hist}(Q_i)$ to describe the relationship between Q_i and S. We could assign a weight to each common partial tuple based on its frequency count, then aggregate the weights to compute a proximity score; but this is risky due to the high variability in the number of the Q_i's. So, we sidestep weight aggregation and simply assume that a common tuple t with lower $\operatorname{freq}(t)$ is infinitely more important than any number of tuples with higher $\operatorname{freq}(t)$. That is, frequency-1 matches between S and Q_i are infinitely more valuable than frequency-2 matches, and these are infinitely more valuable than frequency-3 matches etc. Hence, we rank the queries $\{Q_1, \ldots, Q_n\}$ in the decreasing lexicographical order of their frequency histograms:

$$\begin{aligned} (h_1, h_2, \ldots, h_n) \,>\, (h'_1, h'_2, \ldots, h'_n) \;&\stackrel{\text{def}}{\Leftrightarrow}\; \exists\, K = 1...n: \\ h_1 = h'_1 \;\;\&\ldots\&\;\; h_{K-1} = h'_{K-1} \;\;&\&\;\; h_K > h'_K. \end{aligned} \qquad (2)$$

Now the method is fully defined. We summarize it in Algorithm 1. Below is an example to illustrate this algorithm:

Example 1. Consider a schema of two attributes $A_1 \times A_2$, where A_1 has domain $\{a, b, c, \ldots\}$ and A_2 has domain $\{0, 1\}$. Let the sensitive table S and three query tables Q_1, Q_2 and Q_3 be as defined in Table 1. The frequency counts $\operatorname{freq}(t)$ for all involved partial tuples are given in Table 2. The computation of $s \wedge q$ for all tuple pairs between S and Q_i, the computation of minimum frequency counts, and the subsequent formation of histograms is given in Table 3. The ranking output is as follows: $(0_1, 3_2, 0_3) < (1_1, 1_2, 1_3) < (1_1, 2_2, 0_3) \Rightarrow Q_1 < Q_2 < Q_3$.

To obtain a numerical proximity measure from a frequency histogram in an order-preserving manner, pick some $\alpha > 0$, e.g. $\alpha = 1$, and define

$$\operatorname{prox}(Q_i, S) \;\stackrel{\text{def}}{=}\; f\big(\operatorname{hist}(Q_i)\big), \;\;\text{where} \qquad (3)$$

$$f(h_1, h_2, \ldots, h_n) \;=\; \sum_{k=1}^{n} \frac{h_k}{\alpha + h_k} \prod_{l=1}^{k-1} \frac{\alpha}{(\alpha + h_l)(\alpha + h_l + 1)}$$

Let us justify this measure by the following lemma:

Lemma 1. *In all valid settings,* $\operatorname{hist}(Q_i) > \operatorname{hist}(Q_j)$ *if and only if* $\operatorname{prox}(Q_i, S) > \operatorname{prox}(Q_j, S)$.

Proof. Denote $f_k = f(h_k, h_{k+1}, \ldots, h_n, 0, \ldots, 0)$; notice the following recursion:

$$\begin{aligned} f_{n+1} \,=\, 0; \quad f_k \;&=\; \frac{h_k}{\alpha + h_k} \,+\, \frac{\alpha \cdot f_{k+1}}{(\alpha + h_k)(\alpha + h_k + 1)} \;= \\ &=\; \frac{h_k}{\alpha + h_k} \,+\, \left(\frac{h_k + 1}{\alpha + h_k + 1} - \frac{h_k}{\alpha + h_k}\right) f_{k+1} \end{aligned} \qquad (4)$$

[2] The 7th partial tuple of $\langle a, b, c\rangle$, namely $\langle *, *, *\rangle$, is valid, but has no matching value.

Algorithm 1 : The partial tuple matching (PTM) method for ranking/measuring proximity of tables $Q_1, \ldots, Q_n$ with respect to S.

```
Require: $Q_1, \ldots, Q_n$ and $S$ are tables having the same schema $A_1 \times A_2 \times \ldots \times A_d$.
Ensure: A linear order over the set $\{Q_1, \ldots, Q_n\}$ by decreasing relevance to $S$.
For $i = 1...n$ and $\forall\,(s, q) \in S \times Q_i$, let $\mathrm{freq}(s \wedge q)$ be the number of tables in $\{Q_1, \ldots, Q_n\}$ containing an instance of partial tuple $s \wedge q$;
For $i = 1...n$ and $\forall\, s \in S$, compute minimum frequency $\mathrm{minf}(s, Q_i) = \min_{\,q \in Q_i} \mathrm{freq}(s \wedge q)$;
For $i = 1...n$, build histogram $\mathrm{hist}(Q_i) = (h_1, h_2, \ldots, h_n)$ where $h_k = \big|\{s \in S \mid \mathrm{minf}(s, Q_i) = k\}\big|$;
Sort $\{Q_1, \ldots, Q_n\}$ in the decreasing lexicographic order of their frequency histograms $\mathrm{hist}(Q_i)$;
Output this linear order and/or proximity scores (3).
```

S	Q_1	Q_2	Q_3
$\langle a, 1\rangle$	$\langle a, 0\rangle$	$\langle a, 1\rangle$	$\langle b, 1\rangle$
$\langle b, 1\rangle$	$\langle b, 0\rangle$	$\langle c, 1\rangle$	$\langle c, 0\rangle$
$\langle c, 0\rangle$	$\langle c, 0\rangle$		

Table 1: Sensitive table S and query tables Q_1, Q_2 and Q_3 for Example 1.

Tuple	freq	Q_1	Q_2	Q_3
$\langle a, 1\rangle$	1		✓	
$\langle b, 1\rangle$	1			✓
$\langle c, 0\rangle$	2	✓		✓
$\langle *, 0\rangle$	2	✓		✓
$\langle *, 1\rangle$	2		✓	✓

Tuple	freq	Q_1	Q_2	Q_3
$\langle a, *\rangle$	2	✓	✓	
$\langle b, *\rangle$	2	✓		✓
$\langle c, *\rangle$	3	✓	✓	✓
$\langle *, *\rangle$	3	✓	✓	✓

Table 2: Full and partial tuple frequency counts across queries Q_1, Q_2, Q_3 in Example 1.

	Q_1:			min	Q_2:		min	Q_3:		min
S:	$\langle a, 0\rangle$	$\langle b, 0\rangle$	$\langle c, 0\rangle$	freq	$\langle a, 1\rangle$	$\langle c, 1\rangle$	freq	$\langle b, 1\rangle$	$\langle c, 0\rangle$	freq
$\langle a, 1\rangle$	$\langle a, *\rangle$	$\langle *, *\rangle$	$\langle *, *\rangle$	2	$\langle a, 1\rangle$	$\langle *, 1\rangle$	1	$\langle *, 1\rangle$	$\langle *, *\rangle$	2
$\langle b, 1\rangle$	$\langle *, *\rangle$	$\langle b, *\rangle$	$\langle *, *\rangle$	2	$\langle *, 1\rangle$	$\langle *, 1\rangle$	2	$\langle b, 1\rangle$	$\langle *, *\rangle$	1
$\langle c, 0\rangle$	$\langle *, 0\rangle$	$\langle *, 0\rangle$	$\langle c, 0\rangle$	2	$\langle *, *\rangle$	$\langle c, *\rangle$	3	$\langle *, *\rangle$	$\langle c, 0\rangle$	2
	Histogram: $(0_1, 3_2, 0_3)$				Hist: $(1_1, 1_2, 1_3)$			Hist: $(1_1, 2_2, 0_3)$		

Table 3: The computation of frequency histograms for queries Q_1, Q_2, Q_3 in Example 1.

Assume $\mathrm{hist}(Q_i) = (h_1, h_2, \ldots, h_n) > (h'_1, h'_2, \ldots, h'_n) = \mathrm{hist}(Q_j)$ as defined in (2); then $h_k = h'_k$ for $k = 1...K-1$ and $h_K > h'_K$, implying $h_K \geqslant h'_K + 1$ since these are two integers. Denote $f'_k = f(h'_k, h'_{k+1}, \ldots, h'_n, 0, \ldots, 0)$; from (4) we have $0 \leqslant f^{(\prime)}_{K+1} < 1$ by induction, and furthermore,

$$\frac{h'_K}{\alpha + h'_K} \leqslant f'_K < \frac{h'_K + 1}{\alpha + h'_K + 1} \leqslant \frac{h_K}{\alpha + h_K} \leqslant f_K < \frac{h_K + 1}{\alpha + h_K + 1}$$

Therefore $f_K > f'_K$, and $f_1 > f'_1$ too because $h_k = h'_k$ for $k = 1...K-1$ and recursion (4) is strictly monotone with respect to f_{k+1}.

The above proves that $\mathrm{hist}(Q_i) > \mathrm{hist}(Q_j)$ implies $\mathrm{prox}(Q_i, S) > \mathrm{prox}(Q_j, S)$. Analogously, $\mathrm{hist}(Q_i) < \mathrm{hist}(Q_j)$ implies $\mathrm{prox}(Q_i, S) < \mathrm{prox}(Q_j, S)$, and "=" implies "=". Because for every pair of histograms one of these alternatives holds, the lemma is proven. □

4. STATISTICAL TUPLE LINKAGE

Record linkage [6, 12, 30] is a well-established area of statistical science, which traces its origin to the dawn of the computer era. Ever since government organizations and private businesses began collecting large volumes of records about individual people, they faced a pressing need to efficiently identify and match different records about the same person. Attribute values in such records are often missing, misspelled, have multiple variants, are approximate or even intentionally modified, exacerbating the complexity of the linkage problem. For datasets where direct key-based matching does not work, probabilistic record linkage methods were developed. Here we follow one popular method based on finite mixture models [20] and measure proximity between tables by optimally matching their records.

4.1 Statistical Tuple Linkage Framework

We have S, which is an $|S| \times d$ table with schema $A_1 \times A_2 \times \ldots \times A_d$, and Q, which is a $|Q| \times d$ table with the same schema. Assume that each tuple in S and in Q describes one entity (e. g. person) from a certain unspecified collection. We want to find pairs of tuples $\langle s_i, q_{i'}\rangle$ from $S \times Q$ that both describe the same entity.

Definition 3. For every pair of tuples $s_i \in S$ and $q_{i'} \in Q$, define a d-dimensional *comparison vector* $\gamma = \gamma(s_i, q_{i'})$ such that $\gamma^j = 1$ if the tuples match on the j^{th} attribute and 0 otherwise. If the j^{th} attribute is missing in one of the tuples, let $\gamma^j = *$:

$$\begin{aligned} \gamma(s_i, q_{i'}) \;&=\; \langle \gamma^1, \gamma^2, \ldots, \gamma^d \rangle : \\ \forall\, j = 1 \ldots d: \;\; \gamma^j \;&=\; \begin{cases} 1, & s_i^j = q_{i'}^j \\ 0, & s_i^j \neq q_{i'}^j \\ *, & \text{missing } s_i^j \text{ or } q_{i'}^j \end{cases} \end{aligned}$$

Overall, we have $|S| \cdot |Q|$ vectors $\gamma(s_i, q_{i'})$, one for each pair of tuples.

Let $\Gamma = \langle \gamma_k \rangle_{k=1}^{|S|\,|Q|}$ denote the $|S|\,|Q| \times d$ matrix of all comparison vectors. We shall define a probabilistic model that describes the distribution of these vectors. The model is centered around the notion of *true matching* between two tuples. We assume that there is an unknown function

$$\mathrm{Match}: S \times Q \;\rightarrow\; \{M, U\}, \tag{5}$$

where "M" means "tuples match" and "U" means "tuples do not match." We can also think of M and U as a partition of $S \times Q$ into two disjoint subsets formed by matching and nonmatching tuple pairs. For example, if S and Q contain tuples representing distinct individuals, a pair $s_i \in S$, $q_{i'} \in Q$ is a true match if s_i and $q_{i'}$ represent the same person. In this case at most $\min(|S|, |Q|)$ can be true matches (belong to M), the remainder of $S \times Q$ belong to U.

The record linkage process attempts to classify each tuple pair $\langle s_i, q_{i'}\rangle$ as either M or U, by observing comparison vectors $\gamma(s_i, q_{i'})$. This clarification is possible because the distribution of $\gamma(s_i, q_{i'})$ for M-labeled tuple pairs is very different from its distribution for U-labeled pairs. Let us define two sets of conditional probabilities:

$$\begin{aligned} m(\gamma) \;&=\; \mathbf{P}\,[\,\gamma(s_i, q_{i'}) \mid \langle s_i, q_{i'}\rangle \in M\,]; \\ u(\gamma) \;&=\; \mathbf{P}\,[\,\gamma(s_i, q_{i'}) \mid \langle s_i, q_{i'}\rangle \in U\,]. \end{aligned} \tag{6}$$

In other words, $m(\gamma)$ is the probability to find a comparison vector γ if indeed the tuples are in a true match, whereas

$u(\gamma)$ is the probability of observing γ when the tuples are not a true match. If $\langle s_i, q_{i'}\rangle \in M$, then the probability of $\gamma^j = 1$ for most attributes with non-missing values should be high, unless the data contains many errors. If instead $\langle s_i, q_{i'}\rangle \in U$, then the probability of an accidental attribute match depends upon the distribution of attribute values in S and Q.

A comparison vector γ that involves missing values, i. e. with $\gamma^j = *$ for some attributes, stands for the set

$$I(\gamma) \;=\; \bigl\{\gamma' \in \{0,1\}^d \;\big|\; \forall\, j{=}1...d: \;\gamma^j \neq * \;\Rightarrow\; \gamma'^{\,j} {=} \gamma^j\bigr\}$$

Accordingly, for such γ we define

$$m(\gamma) \;=\; \sum\nolimits_{\gamma' \in I(\gamma)} m(\gamma'), \quad u(\gamma) \;=\; \sum\nolimits_{\gamma' \in I(\gamma)} u(\gamma'). \quad (7)$$

Fellegi and Sunter [9] formalized the matching problem. Let us briefly describe the main elements of their work and state the fundamental theorem. Let the *comparison space* $\mathfrak{G}$ be the set of all possible realizations of γ. In our case, assume that no values are missing and set $\mathfrak{G} = \{0,1\}^d$. A (probabilistic) *matching rule* D is a mapping from $\mathfrak{G}$ to a set of three random decision probabilities

$$D(\gamma) \;=\; \bigl\langle P(\hat{M}\,|\,\gamma),\, P(\hat{?}\,|\,\gamma),\, P(\hat{U}\,|\,\gamma)\bigr\rangle$$
$$\text{such that } \; P(\hat{M}\,|\,\gamma) \,+\, P(\hat{?}\,|\,\gamma) \,+\, P(\hat{U}\,|\,\gamma) \;=\; 1$$

Here, $\hat{M}$ is the decision that there is a true match between tuples s_i and $q_{i'}$, and $\hat{U}$ is the decision that there is no true match. In practice, there will be cases where we will not be able to make such clear cut decisions, hence we allow for a "possible match" decision denoted by "$\hat{?}$". We define two types of errors:

1. Linking unmatched comparisons:

$$\mu \;=\; P(\hat{M}\,|\,U) \;=\; \sum\nolimits_{\gamma \in \mathfrak{G}} u(\gamma)\, P(\hat{M}\,|\,\gamma); \qquad (8)$$

2. Non-linking a matched comparison:

$$\lambda \;=\; P(\hat{U}\,|\,M) \;=\; \sum\nolimits_{\gamma \in \mathfrak{G}} m(\gamma)\, P(\hat{U}\,|\,\gamma). \qquad (9)$$

We write a matching rule D as $D(\mu, \lambda, \mathfrak{G})$ to explicitly note its errors $\mu(D)$ and $\lambda(D)$.

Definition 4. A matching rule $D(\mu, \lambda, \mathfrak{G})$ is said to be *optimal* among all rules satisfying (8) and (9) if

$$P(\hat{?}\,|\,D) \;\leqslant\; P(\hat{?}\,|\,D')$$

for every $D'(\mu, \lambda, \mathfrak{G})$ in this class. Intuitively, less ambiguous matching rules should be preferred to others with the same level of errors.

In order to construct the optimal rule, select two thresholds $T_\mu > T_\lambda$ and fix the pair (μ, λ) of admissible error levels such that

$$\mu \;=\; \sum\nolimits_{\frac{m(\gamma)}{u(\gamma)} \,\geqslant\, T_\mu} u(\gamma), \quad \lambda \;=\; \sum\nolimits_{\frac{m(\gamma)}{u(\gamma)} \,\leqslant\, T_\lambda} m(\gamma) \qquad (10)$$

Define a deterministic[3] matching rule $D_0(\mu, \lambda, \mathfrak{G})$ for any comparison vector γ as follows:

$$D_0(\gamma_k) \;=\; \begin{cases} \hat{M} & \text{if } T_\mu \leqslant m(\gamma)/u(\gamma), \\ \hat{?} & \text{if } T_\lambda < m(\gamma)/u(\gamma) < T_\mu, \\ \hat{U} & \text{if } m(\gamma)/u(\gamma) \leqslant T_\lambda. \end{cases} \qquad (11)$$

Theorem 1 (Fellegi, Sunter). *The matching rule $D_0(\mu, \gamma, \mathfrak{G})$ defined by (11) is the optimal matching rule on $\mathfrak{G}$ at the error levels of μ and λ.*

Proof. See [9]. □

4.2 Mixture Model and EM

As Theorem 1 demonstrates, the evaluation of $m(\gamma)/u(\gamma)$ is crucial in deciding whether or not two records truly match. But how can we compute the conditional probabilities $m(\gamma)$ and $u(\gamma)$? Their definitions (6) cannot be directly applied because no pair of records is labeled with M or U. There is no way to compute them that works in all cases; however, given certain assumptions about the data, $m(\gamma)$ and $u(\gamma)$ can be efficiently estimated. Quite commonly [4, 15, 17] the assumptions combine blocking and mixture models.

Blocking [15] consists in labeling a large fraction of $S \times Q$ pairs with U (non-match) according to some heuristic. This method substantially reduces the scope of the matching problem by eliminating pairs of tuples that are obvious non-matches. For example, a blocking strategy for census data may exclude tuple pairs that do not match on zip code, with the assumption being that two people in different zip codes cannot be the same person.

We shall assume that, after blocking, all pairs and their comparison vectors $\gamma_k \in \Gamma$ with index $k = 1 \ldots K_{\mathrm{B}}$ are left unlabeled, whereas all γ_k with index $k = K_{\mathrm{B}}{+}1 \ldots |S|\,|Q|$ are labeled with U.

For the mixture model, let us assume that the comparison vectors $\gamma_k = \gamma(s_i, q_{i'})$ are conditionally independent from each other given the M- or U-label of the pair $\langle s_i, q_{i'}\rangle$. In addition, assume that the M- and U-labels are themselves independently assigned to each pair, with probability $p \in [0,1]$ to assign an M-label and probability $1{-}p$ to assign a U-label. Then, the probability that some unlabeled pair $\langle s, q\rangle$ has a comparison vector $\hat{\gamma}$ equals

$$\begin{aligned} \mathbf{P}\,[\gamma(s,q) = \hat{\gamma}] \;&=\; p\;\mathbf{P}\,[\hat{\gamma}\,|\,M] \,+\, (1-p)\,\mathbf{P}\,[\hat{\gamma}\,|\,U] \\ &=\; p\, m(\hat{\gamma}) \,+\, (1-p)\, u(\hat{\gamma}) \end{aligned}$$

For a pair $\langle s, q\rangle$ whose label is known to be U (through blocking) the probability of both the label and vector $\hat{\gamma}$ equals just $(1-p)\,u(\hat{\gamma})$. Thus, the probability for the entire observed matrix of comparison vectors Γ and the observed U-labels assigned by blocking is given by the product[4]

$$\prod_{k=1}^{K_{\mathrm{B}}} \Bigl(p\, m(\gamma_k) \,+\, (1-p)\, u(\gamma_k)\Bigr) \cdot \prod_{k=K_{\mathrm{B}}+1}^{|S|\,|Q|} (1-p)\, u(\gamma_k) \qquad (12)$$

Now one can use maximum likelihood estimation to search for $m(\gamma)$ and $u(\gamma)$ that maximize the probability (12). This estimation is carried out through the EM algorithm [13, 7]. Before we turn to EM, let us denote by $z_k \in \{0,1\}$ a random variable such that

$$z_k = 1 \;\Longleftrightarrow\; \text{Match}\, \langle s_{i(k)}, q_{i'(k)}\rangle \;=\; M$$

In our generative model, we assume that each z_k follows Bernoulli$\,(p)$. Note that the z_k's are not known for $k = 1 \ldots K_{\mathrm{B}}$, i. e. pairs left unlabeled after blocking, and $z_k = 0$ for the blocked pairs. Recall that index k refers

[3] For a (μ, λ) not constrained by (10) the optimal rule may have to make probabilistic decisions for borderline γ.

[4] An alternative approach is when the mixture model and EM covers only the tuple pairs left unlabeled by blocking [15]. This would increase p, but could introduce bias.

to a tuple pair $\langle s_{i(k)}, q_{i'(k)} \rangle$ in product $S \times Q$, while index j on top of γ_k^j denotes a coordinate of γ_k for attribute A_j.

Given a joint distribution $\mathbf{P}[X, Z \,|\, \Theta]$ with an observed random vector X, a hidden random vector Z and a parameter vector Θ, the EM algorithm is an iterative procedure to find parameters Θ^* where the marginal distribution $\mathbf{P}[X \,|\, \Theta] = \sum_Z \mathbf{P}[X, Z \,|\, \Theta]$ achieves a local maximum. This algorithm is often used to estimate parameters of mixture models [19]. The iteration step of the algorithm is given by the following formula:

$$\Theta_{n+1} \;=\; \underset{\Theta}{\operatorname{argmax}} \underset{Z \,\sim\, \mathbf{P}[Z \,|\, X, \Theta_n]}{\mathrm{E}} \log \mathbf{P}[X, Z \,|\, \Theta] \tag{13}$$

In our case, X includes the observed comparison matrix Γ and the blocking U-labels $\langle z_k \rangle_{k=K_\mathrm{B}+1}^{|S|\,|Q|}$ while the hidden labels are $Z = \langle z_k \rangle_{k=1}^{K_\mathrm{B}}$, and we want to estimate probabilities $\Theta = \langle p, m(\gamma), u(\gamma) \rangle_{\gamma \in \Gamma}$. The joint distribution of both X and Z equals the product

$$\mathbf{P}[X, Z \,|\, \Theta] \;=\; \prod_{k=1}^{|S|\,|Q|} \big(p\, m(\gamma_k)\big)^{z_k} \, \big((1-p)\, u(\gamma_k)\big)\big)^{1-z_k}$$

The logarithm of this expression is linear with respect to the z_k's, making it easy to take the expectation:

$$\underset{Z \,\sim\, \mathbf{P}[Z \,|\, X, \Theta_n]}{\mathrm{E}} \log \mathbf{P}[X, Z \,|\, \Theta] \;=\; \sum_{k=1}^{|S|\,|Q|} \bar{z}_k \, \log\big(p\, m(\gamma_k)\big) \tag{14}$$

$$+ \sum_{k=1}^{|S|\,|Q|} (1 - \bar{z}_k) \, \log\big((1-p)\, u(\gamma_k)\big)\big), \;\text{ where } \bar{z}_k = \underset{Z \,\sim\, \mathbf{P}[Z \,|\, X, \Theta_n]}{\mathrm{E}} z_k.$$

Computation of the expectations $\bar{z}_k$ for non-blocked pairs is the "E-step" of the EM algorithm, and the subsequent recomputation of next-iteration parameters $\hat{p}$, $\hat{m}(\gamma_k)$, $\hat{u}(\gamma_k)$ to maximize (14) is the "M-step." Denote the n^{th} iteration parameters by p_n, $m_n(\gamma_k)$, $u_n(\gamma_k)$; then the E-step is given by the Bayes formula as follows:

$$\begin{aligned} \bar{z}_k \;&=\; \mathbf{P}[z_k = 1 \,|\, \gamma_k] \;=\; \mathbf{P}[M \,|\, \gamma_k] \;= \\ &=\; \frac{p_n\, m_n(\gamma_k)}{p_n\, m_n(\gamma_k) \;+\; (1 - p_n)\, u_n(\gamma_k)}, \quad k = 1 \ldots K_\mathrm{B} \end{aligned} \tag{15}$$

For the M-step, we could maximize (14) over the entire range $\langle m(\gamma), u(\gamma) \rangle_{\gamma \in \Gamma}$, but so many parameters would overfit the data. So, we follow [10, 15] and assume that individual attribute matchings are conditionally independent given the "true matching" label M or U. For $\gamma \in \{0,1\}^d$ we get

$$\begin{aligned} m(\gamma) \;&=\; \textstyle\prod_{j=1}^d (m^j)^{\gamma^j} (1 - m^j)^{1-\gamma^j} \qquad & m^j \;&=\; \mathbf{P}[\gamma^j = 1 \,|\, M] \\ u(\gamma) \;&=\; \textstyle\prod_{j=1}^d (u^j)^{\gamma^j} (1 - u^j)^{1-\gamma^j} \qquad & u^j \;&=\; \mathbf{P}[\gamma^j = 1 \,|\, U] \end{aligned}$$

If a comparison vector $\gamma \in \{0, 1, *\}^d$ has missing values, it is treated as a set $I(\gamma)$ of possible complete vectors $\gamma' \in \{0,1\}^d$, as in (7), or equivalently as a predicate $P_\gamma(\gamma') \Leftrightarrow \gamma' \in I(\gamma)$. The probability of $P_\gamma(\gamma')$ to be satisfied given label M or U is

$$m(\gamma) = \prod_{j:\, \gamma^j \,\neq\, *} (m^j)^{\gamma^j} (1 - m^j)^{1-\gamma^j}, \;\; u(\gamma) = \prod_{j:\, \gamma^j \,\neq\, *} (u^j)^{\gamma^j} (1 - u^j)^{1-\gamma^j}.$$

With the above assumption, maximizing (14) computes the $n+1^{\text{st}}$ iteration parameters $\hat{p}$ and $\langle \hat{m}^j, \, \hat{u}^j \rangle_{j=1}^d$. The formulas for $\hat{p}$ and $\hat{m}^j$ are as follows:

$$\hat{p} \;=\; |S|^{-1} |Q|^{-1} \sum_{k=1 \ldots K_\mathrm{B}} \bar{z}_k, \quad \hat{m}^j \;=\; \sum_{\substack{k=1 \ldots K_\mathrm{B} \\ k:\, \gamma_k^j \,\neq\, *}} \bar{z}_k \, \gamma_k^j \bigg/ \sum_{\substack{k=1 \ldots K_\mathrm{B} \\ k:\, \gamma_k^j \,\neq\, *}} \bar{z}_k. \tag{16}$$

Since most tuple pairs in $S \times Q$ belong to U (are not "true matches"), the parameters $\langle u^j \rangle_{j=1}^d$ can be well approximated by ignoring the $\bar{z}_k$'s altogether (setting them all to 0) [15]:

$$u^j \;\approx\; \left| \left\{ k \;\middle|\; {\scriptstyle 1 \,\leqslant\, k \,\leqslant\, |S|\,|Q| \atop \gamma_k^j \,=\, 1} \right\} \right| \bigg/ \left| \left\{ k \;\middle|\; {\scriptstyle 1 \,\leqslant\, k \,\leqslant\, |S|\,|Q| \atop \gamma_k^j \,\neq\, *} \right\} \right| \tag{17}$$

We take advantage of this approximation, and use EM only to estimate p and $\langle m^j \rangle_{j=1}^d$. Once the EM iterations converge, we obtain all the parameters necessary to perform statistical tuple linkage between the tuples in S and in Q.

4.3 Proximity Measure

Return to the setup of Section 2 and consider a table S containing sensitive data and the query tables $Q_1, Q_2, \ldots, Q_n$ to be ranked by their proximity to S. The ranking is performed by optimally matching the tuples in each Q_i to the tuples in S and comparing the weights of these matches. According to Theorem 1, the fraction $m(\gamma)/u(\gamma)$ is the best measure to quantify whether or not a comparison vector γ indicates a true match. Let us make the following definition (inspired by [15]):

Definition 5. The *weight* of a tuple pair $\langle s, q \rangle$ from $S \times Q$, whose comparison vector is γ, is given by

$$w(s,q) \;=\; \log \frac{m(\gamma)}{u(\gamma)} \;=\; \sum_{j=1}^d \begin{cases} \log \frac{m^j}{u^j}, & \gamma^j = 1 \\ \log \frac{1-m^j}{1-u^j}, & \gamma^j = 0 \\ 0, & \gamma^j = * \end{cases}$$

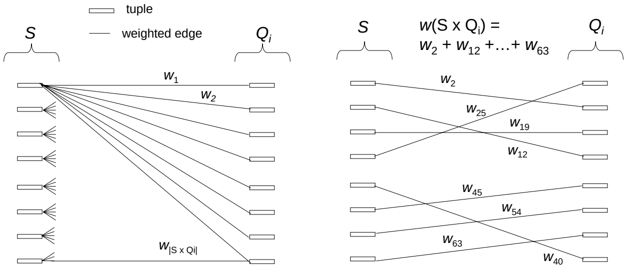
The *plus-weight* of $\langle s, q \rangle$ is 0 if this tuple pair is labeled with U by blocking, otherwise it is defined as

$$w^+(s,q) \;=\; \begin{cases} w(s,q), & w(s,q) \geqslant 0 \\ 0, & w(s,q) < 0 \end{cases} \tag{18}$$

We begin by computing the parameters $\hat{p}$ and $\langle \hat{m}^j, \, \hat{u}^j \rangle_{j=1}^d$ via the framework described in Section 4.2, where we set $Q = Q_1 \cup Q_2 \cup \ldots \cup Q_n$. We take this duplicate preserving union and run EM over Q to ensure that all parameters are the same for all Q_i's. Blocking assigns U-labels to all tuple pairs $\langle s, q \rangle$ that do not share at least one "discriminating" attribute value; see Section 7 for details.

Having estimated the m^j's and the u^j's, we use (18) to compute the plus-weights of all pairs in $S \times Q_i$ left unlabeled by blocking. All pairs labeled with U by blocking receive weight 0. Then for each Q_i we seek a maximum-weight matching that assigns each record in Q_i to one and only one record in S. The weight of a matching is defined as the sum of plus-weights of all matched pairs. Plus-weights are used so that negative weights never impact the matching process.

We compute the maximum-weight matching with the help of the Kuhn-Munkres algorithm for optimal matching over a bipartite graph, also known as the Hungarian algorithm [16, 22, 29]. The weight of the matching is the proximity measure between Q_i and S that we output, to be used in ranking queries and measuring disclosure.



(a) Assign weights to all edges among pairs of tuples in $S \times Q_i$.

(b) Find a 1-to-1 matching to maximize the sum of weights

Figure 1: Using statistical tuple linkage to measure proximity between tables S and Q_i.

Algorithm 2 : The statistical tuple linkage method (STL) of measuring proximity of tables $Q_1, \ldots, Q_n$ and S.

```
Require: $Q_1, \ldots, Q_n$ and $S$ are tables having the same schema $A_1 \times A_2 \times \ldots \times A_d$.
Ensure: The computation of the maximum matching weights $w(S,\, Q_i)$.
1: Set $Q = Q_1 \cup Q_2 \cup \ldots \cup Q_n$, locate missing values;
2: Perform blocking to label some tuple pairs $\langle s, q \rangle \in S \times Q$ as non-matching;
3: Compute comparison vectors $\gamma(s, q)$ for all tuple pairs in $S \times Q$, see Definition 3;
4: Initialize parameters (we set $\hat{p}$ to 0.05, $\hat{m}^1, \ldots, \hat{m}^d$ to 0.95) and compute $\hat{u}^1, \ldots, \hat{u}^d$ by (17);
5: Repeat E-step (15) and M-step (16) of EM algorithm over $\hat{p}, \hat{m}$ until expectations $\langle \bar{z}_k \rangle_{k=1}^{K_{\mathrm{B}}}$ stabilize, see (15);
6: For each $Q_i$, compute the plus-weights of all tuple pairs in $S \times Q_i$ using (18);
7: Find maximum 1-to-1 matchings using Kuhn-Munkres algorithm over complete bipartite graphs $S \times Q_i$;
8: Output the sum of plus-weights of matched tuple pairs.
```

Figures 1(a) and 1(b) graphically portray the application of the statistical tuple linkage method to the problem of query ranking. Fig. 1(a) shows computed weights for all edges in $S \times Q_i$, and Fig. 1(b) illustrates the result of using Kuhn-Munkres to maximize the sum of plus-weights assigned to edges while ensuring that each tuple in Q_i and S has at most one edge.

Algorithm 2 summarizes the computation steps involved in measuring proximity through statistical tuple linkage.

5. DERIVATION PROBABILITY GAIN

This method measures proximity between two tables Q and S based on the minimum-length (maximum-probability) derivation of S from Q. Intuitively, one can think of an archiver that tries to compress S given the tuples in Q. The compressed "file" includes both the new values in S recorded "as-is" and the link structure to copy the repeated values. The size of the archive, expressed through its probability, or more exactly the size difference made by the presence of Q, gives the proximity measure. We consider a specific compression procedure that uses the minimum spanning tree algorithm.

Definition 6. Given tables $Q = \langle q_1, q_2, \ldots, q_{|Q|} \rangle$ and $S = \langle s_1, s_2, \ldots, s_{|S|} \rangle$, a *derivation forest* from Q to S is a collection of disjoint rooted labeled trees $\{T_1, T_2 \ldots, T_k\}$ whose roots are in Q and non-root nodes are in S. The trees' bodies have to cover all tuples in S. A derivation forest defines for each $s_i \in S$ a single *parent* record $\pi(s_i) \in Q \cup S$.

Statement 1. *The number of possible derivation forests from Q to S equals* $|Q|\,(|S| + |Q|)^{|S|-1}$.

Proof. See [11] for a proof of a very similar result for trees; see also [27]. □

We consider a generative model for S given Q with two parameter groups, for each attribute $j = 1 \ldots d$:

- Matching probability $\mu^j \in [0, 1]$,
- Default distribution $p^j(v)$ over all $v \in A_j$.

In this model, we generate the tuples of S from the tuples of Q as follows:

1. Pick a derivation forest D uniformly at random. Forest D defines a parent $\pi(s_i)$ for each record $s_i \in S$. According to Statement 1, the probability of D is:
$$P[D] \;=\; \text{const} \;=\; \big(|Q|\,(|S| + |Q|)^{|S|-1}\big)^{-1}.$$
2. Generate the tuples of S in an order so that each s_i is always preceded by $\pi(s_i)$. To generate tuple $s_i = \langle s_i^1, s_i^2, \ldots, s_i^d \rangle$, for each $j = 1...d$ do: Toss a Bernoulli coin z_i^j with probability μ^j to fall 1 and $1 - \mu^j$ to fall 0. If $z_i^j = 1$, just copy the parent's j^{th} attribute value $\pi^j(s_i)$ into s_i^j; if $z_i^j = 0$, generate s_i^j independently according to the default distribution $p^j(s_i^j)$.

Denote by Z the outcomes of all Bernoulli coins z_i^j. The joint probability of everything being generated, both hidden variables $(D,\, Z)$ and observed tuples (S), given Q equals

$$\mathbf{P}\,[D, Z, S \mid Q] \;=\; \mathbf{P}\,[D] \cdot \prod_{i=1}^{|S|} \prod_{j=1}^{d} p^j(s_i^j)^{1-z_i^j} \;\times \\ \times\; (\mu^j)^{z_i^j}\,(1-\mu^j)^{1-z_i^j} \tag{19}$$

with the constraint that $s_i^j = \pi^j(s_i)$ wherever $z_i^j = 1$ (otherwise $\mathbf{P}\,[D, Z, S \,|\, Q] = 0$).

To measure proximity between tables Q and S, we use $\mathbf{P}\,[D, Z, S \,|\, Q]$ with hidden variables D and Z chosen to maximize this probability. This can be viewed as an instance of the minimum description length principle, where we choose best D and Z to describe S given Q. The "length" of description $\langle D, Z, S \rangle$ is computed as $-\log_2 \mathbf{P}\,[D, Z, S \,|\, Q]$.

Definition 7. Let us define the *weight* $w(s_i, t)$ of an edge between tuples $s_i \in S$ and $t \in Q \cup S$ to be:

$$w(s_i, t) \;:=\; \sum_{\substack{j\,=\,1\ldots d \\ s_i^j\,=\,t^j}} \max\left\{ -\log\left(\frac{1-\mu^j}{\mu^j}\, p^j(s_i^j)\right),\, 0 \right\}$$

Note the symmetricity: $w(s_i, t) = w(t, s_i)$; this is important for our weighted spanning tree representation. Note also that edges $\langle s_i, t \rangle$ whose matching attribute values $s_i^j = t^j$ have low probability to occur randomly are given more weight.

Statement 2. *Probability (19) reaches maximum when derivation forest D is chosen to maximize the sum*

$$w(D) \;:=\; \textstyle\sum_{i=1}^{|S|} w\big(s_i, \pi(s_i)\big). \tag{20}$$

PROOF. Formula (19) can be rewritten as follows:

$$\mathbf{P}\,[D, Z, S \,|\, Q] \;=\; \mathbf{P}\,[D]\,\frac{\prod_{i=1}^{|S|}\prod_{j=1}^{d} p^{\,j}(s_i^j)}{\prod_{i=1}^{|S|} W\big(z_i, s_i, \pi(s_i)\big)} \qquad (21)$$

$$\text{where}\;\; W\big(z_i, s_i, \pi(s_i)\big) \;=\; \prod_{j=1}^{d} \frac{p^{\,j}(s_i^j)^{z_i^j}}{(\mu^j)^{z_i^j}\,(1-\mu^j)^{1-z_i^j}}$$

Since $\mathbf{P}\,[D]$ = const, this term does not affect the value of (19). Once D is fixed, we can pick optimal $Z = Z^*(D)$ by independently minimizing each $W\big(z_i, s_i, \pi(s_i)\big)$, which becomes (recall that $s_i^j \neq \pi^j(s_i) \Rightarrow z_i^j = 0$):

$$W_{\text{opt}}\big(z_i^*, s_i, \pi(s_i)\big) \;=\; W'\big(s_i, \pi(s_i)\big)\cdot\prod_{j=1}^{d}\frac{1}{1-\mu^j}$$

$$\text{where}\;\; W'\big(s_i, \pi(s_i)\big) \;=\; \prod_{j:\, s_i^j = \pi^j(s_i)} \min\Big\{\tfrac{1-\mu^j}{\mu^j}\,p^{\,j}(s_i^j),\, 1\Big\}$$

By Definition 7, the weight $w(s_i, \pi(s_i))$ of an edge between tuples s_i and $\pi(s_i)$ is equal to the negative logarithm of $W'\big(s_i, \pi(s_i)\big)$. Therefore, we can rewrite (21) for optimal $Z = Z^*$ as below:

$$\log \mathbf{P}\,[D, Z^*\!, S \mid Q] \;=\; \log \mathbf{P}\,[D] + \sum_{i=1}^{|S|} w\big(s_i, \pi(s_i)\big)$$

$$+\; \sum_{i=1}^{|S|}\sum_{j=1}^{d} \log p^{\,j}(s_i^j) \;+\; |S|\sum_{j=1}^{d}\log\,(1-\mu^j). \qquad (22)$$

It is easy to see now that the optimal derivation forest D^* is such that the sum of edge weights $w\big(s_i, \pi(s_i)\big)$ over the trees in D^* is maximized. □

The search for the optimal maximum-weight D^* is easily converted into a minimum (or maximum) spanning tree problem. Given tables Q and S, let $G = (V, E)$ be an undirected graph with vertices $V = Q \cup S \cup \{\xi\}$ where ξ is a new special vertex, and with edges formed by all $(Q \cup S) \times S$ and $\{\xi\} \times Q$. Set edge weights according to Definition 7 for non-ξ edges, and set $w(\xi, q_i) = w_{\max}$ for all $q_i \in Q$ where $w_{\max}$ is chosen larger than any non-ξ weight.

The symmetricity of weight function $w(s_i, t)$ allows us to set one weight per edge, independently of its direction towards ξ.

STATEMENT 3. *There is a one-to-one correspondence between maximum spanning trees for G and optimal derivation forests from Q to S.*

PROOF. Given a forest D^*, a spanning tree is produced by adding vertex ξ and connecting all $q_i \in Q$ to ξ. Given a spanning tree T over G that includes all edges connecting ξ and Q, a derivation forest is formed by discarding ξ and its adjacent edges. This forest has exactly one Q-vertex per each tree:

- No Q-vertex would imply that some S-vertices are not connected to ξ in T;
- Two Q-vertices would create a cycle in T as they are connected through S and through ξ.

Any maximum spanning tree T over G includes all ξ-edges since these are the heaviest edges: a tree without edge (ξ, q_i) gains weight by adding (ξ, q_i) and discarding the lightest edge in the resulting cycle. If the derivation forest over $Q \cup S$ that corresponds to T is not optimal, the tree gains weight by replacing this forest with a heavier one; hence, a maximum spanning tree corresponds to an optimal derivation forest. Conversely, if the spanning tree that corresponds to forest D^* is not maximum-weight, the forest is not optimal because a heavier forest is given by any maximum spanning tree. □

COROLLARY 1. *Maximum probability $\mathbf{P}\,[D^*\!, Z^*\!, S \,|\, Q]$ can be computed by taking the weight $w(T)$ of a maximum spanning tree over graph G formed as above, subtracting the ξ-edge weights to get $w(D^*) = w(T) - |Q|\,w_{\max}$, and using formula (22):*

$$\log \mathbf{P}\,[D^*\!, Z^*\!, S \mid Q] \;=$$

$$=\; -\log|Q| \;-\; (|S|-1)\log\,(|S|+|Q|) \;+\; w(D^*) \;+$$

$$+\; \sum_{i=1}^{|S|}\sum_{j=1}^{d}\log p^{\,j}(s_i^j) \;+\; |S|\sum_{j=1}^{d}\log\,(1-\mu^j). \qquad (23)$$

PROOF. Follows from Statements 1, 2, and 3. □

We compute the proximity measure between Q and S by comparing $\mathbf{P}\,[D^*\!, Z^*\!, S \,|\, Q]$ to the maximum derivation probability of S without Q, written as $\mathbf{P}\,[D^{**}\!, Z^{**}\!, S\,]$. It is computed analogously to $\mathbf{P}\,[D^*\!, Z^*\!, S \,|\, Q]$ but with a "dummy" one-tuple Q, and represents the amount of information contained in S. The proximity between Q and S is defined as the log-probability gain for the optimal derivation of S caused by the presence of Q:

$$\text{prox}\,(Q,\, S) \;:=\; \log\,\frac{\mathbf{P}\,[D^*,\, Z^*,\, S \mid Q\,]}{\mathbf{P}\,[D^{**},\, Z^{**},\, S\,]} \qquad (24)$$

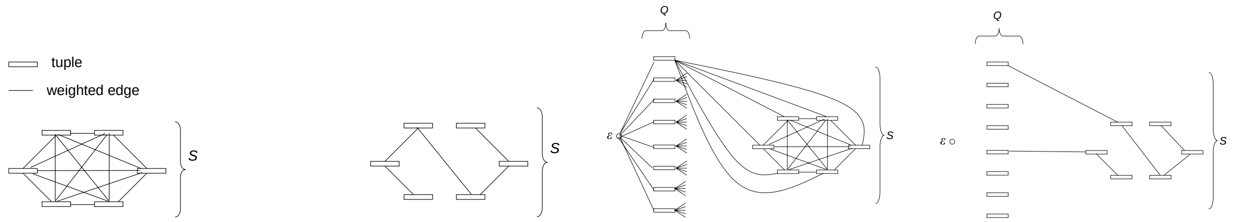
Algorithm 3 summarizes the computation steps for this method. In our experiments, we take $\forall\, j\colon \mu^j = 1/2$ and compute the default probabilities $p^{\,j}(v)$ of attribute values as frequency counts across all query tables.

Figures 2(a) through 2(d) graphically illustrate the DPG method. In Figure 2(a), weights are assigned to all edges among tuples of S, and in Figure 2(b), a maximum spanning tree is computed based upon these weights. Figure 2(c) adds the tuples of Q to the graph, computing and assigning weights betweens edges of $Q \times S$. In Figure 2(d), a new maximum spanning tree is computed now using edges inside S and in $Q \times (S \cup \{\xi\})$. The weights of the remaining edges are used to calculate the benefit of Q to S.

6. COMPARISON OF THE METHODS

Let us take a step back and look at the big picture: what are the similarities and differences between these three ranking methods? All three methods look for matching attributes between the tuples of sensitive table S and of each query table Q_i, yet each method uses different intuition and techniques, resulting in different behavior. We contrast some of the characteristics of our three methods in Table 4.

For Partial Tuple Matching (PTM) the most important ranking factor is the "document frequency" of partial tuples shared between S and Q_i: the number of other query tables that also contain these shared tuples. The two other methods compute their statistics over all tuples in the union $Q_1 \cup Q_2 \cup \ldots \cup Q_n$, which is vulnerable to the bias caused by repetitive data and by the variation in the query table size $|Q_i|$. On the other hand, document frequency may be



(a) Assign weights to all edges among tuples of S

(b) Compute MST

(c) Additionally assign weights to all edges in $Q \times (S \cup \{\xi\})$

(d) Compute MST

Figure 2: Illustration of the derivation probability gain method of measuring proximity of tables Q and S

```
Algorithm 3 : The derivation probability gain method of measuring proximity of tables $Q_1, \ldots, Q_n$ and $S$.
Require: $Q_1, \ldots, Q_n$ and $S$ are tables having the same
    schema $A_1 \times A_2 \times \ldots \times A_d$.
Ensure: The computation of prox$(Q_i,\, S)$
  Scan all query tables and estimate frequency $p^j(v) > 0$
    for $\forall\, v \in A_j$ and $\forall\, j = 1 \ldots d$;
  Use Definition 7 to compute weights $w(s,t)$ for $\forall\, s \in S$
    and $\forall\, t \in Q_i \cup S$;
  Form graph $G$ with vertices in $Q_i \cup S \cup \{\xi\}$ and edges
    from $\xi$ to $Q_i$ and from $S$ to $Q_i \cup S$. Assign a dominating
    weight $w_{\max}$ to $\xi$-edges;
  Run the Maximum Spanning Tree (MST) algorithm over
    $G$, let $w(T)$ be its weight;
  Set $w(D^*) = w(T) - |Q_i| w_{\max}$ and use (23) to compute
    $\log \mathbf{P}\,[D^*\!,\, Z^*\!,\, S \,|\, Q_i]$;
  Run MST algorithm over clique $S$ and use (23) to com-
    pute $\log \mathbf{P}\,[D^{**},\, Z^{**},\, S\,]$ taking $|Q| = 1$;
  Compute prox$(Q_i,\, S)$ using (24).
```

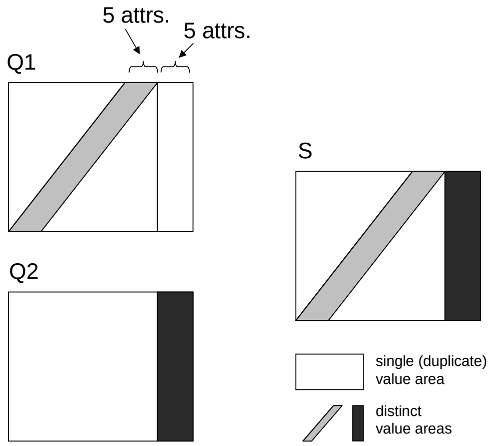


Figure 3: Impact of highly non-uniform attributes on ranking

a poor statistic if the number of queries is small. Thus, PTM ranking is combinatorial rather than statistical. The PTM method counts frequency of attribute combinations (partial tuples), while the other two methods account for each matching attribute individually in tuple comparisons.

The Statistical Tuple Linkage (STL) method stems from the assumption that the tuples in S and Q_i represent external entities, and works to identify same-entity tuples. Its probability parameters $\langle m^j, u^j \rangle_{j=1}^d$ treat equally all values of the same attribute and assume conditional attribute independence. If the values of a certain attribute have a strongly non-uniform distribution, some being rare and highly discriminative and others overly frequent, the algorithm will show suboptimal performance (see Example 2). Missing/default values receive special attention in STL since they differ significantly from other values, and blocking improves efficiency.

Example 2. In Figure 3, the white areas represent attributes all having the same value, say zero. The grey area represents attributes having unique values. Same-colored areas in Q_1, Q_2 match with S; the proportion of diagonal and vertical grey areas are equal. STL ranks Q_2 above Q_1 while PTM and DPG rank Q_1 and Q_2 equally. The difference for STL is due to the non-uniform distribution of values in "diagonal" attributes (some values are common and others unique).

The intuition behind Derivation Probability Gain (DPG) is that shared information between S and Q_i helps to compress S better in the presence of Q_i than alone. Because tuples in S can be "compressed" by deriving them from other S-tuples (even without Q_i), DPG may be better than the other two methods if S contains many duplicates or near-duplicates. However, DPG makes certain attribute independence assumptions and collects value statistics by counting tuples in query tables, which is prone to bias.

7. EXPERIMENTAL RESULTS

We implemented the three proposed methods as Java applications and performed experiments on a Windows XP Professional Version 2002 SP 2 workstation with 2.4GHz Intel Xeon dual processors, 2 GB of memory, and a 136 GB IBM ServeRAID SCSI disk drive.

We used the IPUMS data set [26]; the complete dataset consists of a single table with 30 attributes, and 2.8 million records with household census information. We used random samples from this dataset for our experiments below. For each attribute in the IPUMS dataset, missing values are represented by specific values. For example, a value of 99 for IPUMS attribute `statefip` represents an unknown

	PTM	**STL**	**DPG**
Intuition	We look for documents (the Q_is) that share with S terms (partial tuples) of low document frequency.	Each tuple represents an external entity; we look for Q_is that describe many entities in common with S.	We measure how much Q_i helps to "compress" the information in S, compared to "compressing" S alone.
Algorithmic techniques	Partial tuples (with wildcards), simple combinatorics	Mixture model, EM, optimal weighted bipartite matching	Derivation model, minimum spanning tree computation
Ranking measure	Document frequency histograms over select partial tuples shared by Q_i and S	Maximum weight of a 1-to-1 matching between tuples of Q_i and of S	Log-probability gain for the optimal derivation of S given Q_i vs. of S alone
Time/space optimization	Selecting only a few best partial tuples for each Q_i	Blocking: heuristic removal of some tuple non-matches	None at the moment
Sensitivity to attribute combinations	**High:** partial tuples can represent any possible attribute combination	**Limited:** attribute matches assumed independent given tuple match/non-match	**Limited:** given a derivation, each value contributes weight independently
Single-tuple sensitivity	**High:** a single rare tuple may decide the ranking	**Limited:** proximity measure is aggregated	**Limited:** proximity measure is aggregated
Robustness w.r.t. near-duplicates within one Q_i	**High:** only frequency across all Q_is affects ranking	**Limited:** they affect both EM and bipartite matching	**Moderate:** they affect value statistics, but not derivation
Robustness w.r.t. near-duplicates within S	**Limited:** each near-duplicate within S contributes independently	**Limited:** they affect both EM and bipartite matching	**High:** they add to derivation of S given Q_i as much as to derivation of S alone
Robustness w.r.t. highly non-uniform attributes	**High:** value-level partial tuples	**Limited:** attribute-level parameters (Example 2)	**High:** value-level parameters
Number of model parameters	N/A	**Small:** p and $\langle m^j, u^j \rangle_{j=1}^{d}$	**Moderate:** a probability for each attribute value
Penalty for $\lvert Q_i \rvert$ size	No penalty	No penalty	Yes, $\log\big(\lvert Q_i\rvert(\lvert S\rvert + \lvert Q_i\rvert)^{\lvert S\rvert - 1}\big)$ see Statement 1

Table 4: Comparison of the ranking methods

state of residence rather than a household's state of residence. For the STL method, missing attribute values are omitted from rank score calculations and from parameter estimation as described in Section 4.2. We used the following blocking strategy for the STL method. For a pair of tuples $\langle s, q\rangle \in S \times Q$ to be considered as a possible match, s and q must match on at least one of their discriminating attribute values. Otherwise, the pair is discarded or blocked. An attribute value v is considered discriminating depending upon the number of tuples in S and in Q with that attribute value; computed as the product $\rho(v)$ of the number of tuples in S having the value v in attribute A_j and the number of tuples in Q with the same value. If $\rho(v) < |Q|$, we consider v to be discriminating.

Ideally, we would like to rank queries higher if they have a greater chance of being a source of information contained in S. We formulate some desirable properties to compare our ranking methods in experiments:

1. Given a single query Q_1 whose tuples have been inserted into table S, and other queries $Q_2, \ldots, Q_n$ that have not contributed any tuples to S, no query $Q_2, \ldots, Q_n$ is ranked above Q_1.

2. Given queries Q_1, Q_2 whose tuples have been inserted into table S and other queries $Q_3, \ldots, Q_n$ that have not contributed any tuples to S, no query $Q_3, \ldots, Q_n$ is ranked above Q_1 or Q_2.

3. Given queries Q_1, Q_2 whose tuples have been inserted into table S, and the tuples inserted into S by Q_1 are a superset of those inserted by Q_2, Q_1 is ranked above Q_2.

4. Given queries Q_1, Q_2 having inserted the same subset of tuples into table S, and the number of tuples in Q_2 is larger than Q_1, Q_1 is ranked above Q_2.

5. Given that S may have been subsequently updated and thus some attribute values are retained while others are modified, the above properties hold.

Property 1 says that if S has been copied from a single query Q_1, then Q_1 should be ranked first. Properties 2 to 4 address the usage of multiple queries to populate S. Property 5 allows for the possibility that the data might have been updated over time and that tuples in Q_i and S now match only on some of their attribute values.

7.1 Match Set Size

We used queries $Q_0, \ldots, Q_5$, each with 1000 randomly selected tuples such that $|Q_i| = 1000, |Q_i \cap Q_j| = 0, i \neq j, |Q_0 \cap S| = 0, |Q_1 \cap S| = 200, |Q_2 \cap S| = 400, |Q_3 \cap S| = 600, |Q_4 \cap S| = 800, |Q_5 \cap S| = 1000, |S| = 3000$. For each $Q_i, Q_j, |Q_j \cap S| > |Q_i \cap S|, j > i$. Random selection was done by assigning each tuple a distinct random number $0, \ldots, n-1$, where n is the dataset size and selecting tuples on ranges of these numbers. This experiment is intended to give an indication of the goodness of each method with respect to Properties 1 to 3. All three methods exhibited similar goodness with respect to these properties since each Q_{i+1} ranked above Q_i.

7.2 Overlapping Matching Sets

In these experiments, $Q_i \subset Q_{i+1}, |Q_0| = 200, |Q_1| = 500, |Q_2| = 1000, |Q_3| = 2000, |Q_4| = 5000$. In a first experiment, the sensitive table S is identical to query Q_0 with 200 tuples. In a second experiment, the sensitive table S

	Elapse Time (Minutes)		
$\|S\|$	**PTM**	**STL**	**DPG**
$S \equiv Q_0$ (small)	5.7	0.73	5.95
$S \equiv Q_4$ (large)	171	16	116

Table 5: Impact of the size of S on the performance of each method; $Q_i \subset Q_{i+1}, |Q_0| = 200, |Q_1| = 500, |Q_2| = 1000, |Q_3| = 2000, |Q_4| = 5000$

is identical to query Q_4 with 5000 tuples. In both experiments, each larger query includes all tuples of the smaller sizes. These experiments are intended to give an indication of the goodness of each method with respect to Properties 1 through 4. In the first experiment, PTM and STL rank all queries equally since they have no penalty for query size. However, DPG has a penalty for query size and ranks Q_{i+1} below Q_i due to its greater size and extraneous tuples with respect to S. In the second experiment, all three methods have similar goodness as each Q_{i+1} ranked above Q_i.

7.3 Perturbation

This experiment was intended to give an indication of the goodness of each method with respect to Property 5. The perturbation reflects the fact that the tuples in S might, for example, have been updated after the time the data was acquired by the 3rd party to the time the data was recovered by the party claiming to be its rightful owner and source. In this experiment, $|Q_0| = 1000, |S| = 1000, |Q_0 \cap S| = 1000$ before tuples in S are perturbed, and $|Q_i| = 1000, |Q_i \cap S| = \emptyset, |Q_i \cap Q_j| = \emptyset, i \in 1, \ldots 5, i \neq j$. A percentage of values are perturbed in S (we perturbed 20%, 40%, 60%, 80% of values in S in separate experiments); perturbed values could appear in any attribute. All methods correctly ranked Q_0 above $Q_1, \ldots, Q_5$.

7.4 Performance

In Table 5, we show the elapse time in minutes that each method required to compute the results presented in Section 7.1. These results show the impact of the sensitive table size on the performance of each method. Table 5 contrasts a small size of S (S is $Q_0, |Q_0| = 200$) verses a large size (S is $Q_4, |Q_4| = 5000$). The results show that all methods are sensitive to both the size of S and Q, but that the STL method has overall the best performance. With the STL method, simple comparisons among attribute values in tuples of Q and S are used to generate the comparison verctor γ which is then used in the iterative step of the EM algorithm. The PTM method requires complex comparisons to determine if a tuple either matches or is partially matched by another tuple. Since the number of these comparisons is determined by $|S|$, the PTM method is significantly impacted by this cost when $|S|$ is large. We used indices to optimize these comparisons. However, these indices are in-memory Java objects that consume additional memory resources, thus also having an impact on performance. In comparison with the STL method, the DPG method computes comparisons among tuples in S in addition to comparisons between tuples of Q and S.

We note that the performance of the STL method can be further improved by increasing the level of blocking, as long as it does not significantly affect the accuracy of ranking. It may also be possible to apply similar types of optimizations to the DPG method to improve its performance.

8. CONCLUSION

In this paper, we have introduced and studied the problem of ranking a collection of queries $Q_1, \ldots, Q_n$ over a database D with respect to their proximity to a table S which is suspected to contain information misappropriated from the results of queries over D. We have proposed, developed and contrasted three conceptually different query ranking methods, and experimentally evaluated each method.

We conclude this paper with some remaining issues that need to be addressed in the future. First, our desirable properties used for empirical comparisons are defined informally and offer only an initial set of principles by which to evaluate ranking methods. A formal and complete set of properties would be needed to precisely and fully compare the quality of ranking methods and provide a basis for an in-depth empirical analysis of ranking methods. Second, a typical dataset consists of attributes that are categorical, continuous, and textual. The simple match/non-match dichotomy may be too simplistic for non-categorical values, which calls for an extension to similarity matching. Third, while each our ranking method has its own intuition and own advantages, they could do even better in combination, or borrow ideas from each other. Finally, our Java implementation of each method was useful for experimentation, but we hope that future work will improve performance.

Acknowledgements We wish to thank Tyrone Grandison and Christopher Johnson for their helpful comments.

9. REFERENCES

[1] R. Agrawal, R. Bayardo, C. Faloutsos, J. Kiernan, R. Rantzau, and R. Srikant. Auditing compliance using a hippocratic database. In *30th Int'l Conf. on Very Large Data Bases*, Toronto, Canada, August 2004.

[2] R. Agrawal, P. J. Haas, and J. Kiernan. Watermarking relational databases. *VLDB Journal*, 12(2):157–169, August 2003.

[3] Australian privacy act of 1998, 1998. http://www.privacy.gov.au/ACT/privacyact/.

[4] T. R. Belin and D. B. Rubin. A method for calibrating false match rates in record linkage. *Journal of the American Statistical Assocation*, 90(430):694–707, June 1995.

[5] Personal information protection and electronic documents act, second session, thirty-sixth parliament, 48-49 elizabeth ii, 1999-2000, statutes of canada, 2000.

[6] M. Cochinwala, S. Dalal, A. K. Elmagarmid, and V. S. Verykios. Record matching: Past, present and future. Technical Report CSD-TR #01-013, Department of Computer Sciences, Purdue University, July 2001.

[7] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

[8] European Union Directive on Data Protection, Official Journal of the European Communities, 1995.

[9] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64:1183–1210, December 1969.

[10] L. A. Goodman. Exploratory latent structure analysis using both identifiable and unidentifiable models. *Biometrika*, 61:215–231, 1974.

[11] R. L. Graham, M. Grötschel, and L. Lovász, editors. *Handbook of Combinatorics*, volume 2, chapter 21, page 1024. Elsevier Science B. V., 1995.

[12] L. Gu, R. Baxter, D. Vickers, and C. Rainsford. Record linkage: Current practice and future directions. Technical Report 03/83, CSIRO Mathematical and Information Sciences, GPO Box 664, Canberra 2601, Australia, April 2003.

[13] H. O. Hartley. Maximum likelihood estimation from incomplete data. *Biometrics*, 14:174–194, 1958.

[14] Health insurance portability and accountability act of 1996, united states public law 104-191, 1996. http://www.hhs.gov/ocr/hipaa.

[15] M. A. Jaro. Advances in record linkage methodology as applied to matching the 1985 census of Tampa, Florida. *Journal of the American Statistical Association*, 84:414–420, 1989.

[16] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quaterly*, 2:83–97, 1955.

[17] M. D. Larsen and D. B. Rubin. Iterative automated record linkage using mixture models. *Journal of the American Statistical Association*, 96:32–41, 2001.

[18] Y. Li, V. Swarup, and S. Jajodia. Fingerprinting relational databases: Schemes and specialties. *IEEE Trans. Dependable Sec. Computing (TDSC)*, 2(1):34–45, 2005.

[19] G. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. Wiley-Interscience, November 1996.

[20] G. McLachlan and D. Peel. *Finite Mixture Models*. Wiley-Interscience, October 2000.

[21] G. Miklau and D. Suciu. A formal analysis of information disclosure in data exchange. In *Proc. of the 2004 ACM SIGMOD Int'l Conf. on Management of Data*, Paris, France, June 2004.

[22] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society of Industrial and Applied Mathematics*, 5(1):32–38, March 1957.

[23] A. Nanda and D. K. Burleson. *Oracle Privacy Security Auditing*. Rampant, 2003.

[24] President's Information Technology Advisory Committee. Revolutionizing health care through information technology, June 2004.

[25] J. Rissanen. *Stochastic Complexity in Statistical Inquiry*. World Scientific Publ. Co., 1989.

[26] S. Ruggles, M. Sobek, T. Alexander, C. A. Fitch, R. Goeken, P. K. Hall, M. King, and C. Ronnander. Integrated public use microdata series: Version 3.0, 2004. Machine-readable database.

[27] V. N. Sachkov. *Combinatorial Methods in Discrete Mathematics*, chapter 4.2. Cambridge University Press, 1996. Result found in the 2-nd edition, in Russian, 2004.

[28] G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, New York, 1989.

[29] Wikipedia.org. Hungarian algorithm, March 2006.

[30] W. E. Winkler. Matching and record linkage. In B. G. Cox, editor, *Business Survey Methods*, pages 355–384. Wiley, 1995.