Managing Information Leakage

Steven Euijong Whang, Hector Garcia-Molina

Computer Science Department, Stanford University
353 Serra Mall, Stanford, CA 94305, USA
{swhang, hector}@cs.stanford.edu

Abstract. We explore the problem of managing information leakage by connecting two hitherto disconnected topics: entity resolution (ER) and data privacy (DP). As more of our sensitive data gets exposed to a variety of merchants, health care providers, employers, social sites and so on, there is a higher chance that an adversary can "connect the dots" and piece together our information, leading to even more loss of privacy. For instance, suppose that Alice has a social networking profile with her name and photo and a web homepage containing her name and address. An adversary Eve may be able to link the profile and homepage to connect the photo and address of Alice and thus glean more personal information. The better Eve is at linking the information, the more vulnerable is Alice's privacy. Thus in order to gain DP, one must try to prevent important bits of information being resolved by ER. In this paper, we formalize information leakage and list several challenges both in ER and DP. We also propose using disinformation as a tool for containing information leakage.

1 Introduction

In this paper we explore the connections between two hitherto disconnected topics: entity resolution and data privacy. In entity resolution (ER), one tries to identify data records that refer to the same real world entity. Matching records are often merged into "composite" records that reflect the aggregate information known about the entity. The goal of data privacy (DP) is to prevent disclosure to third parties of sensitive user (entity) data. For instance, sensitive data can be encrypted to make it hard for a third party to obtain, or the sensitive data can be "modified" (e.g., changing an age 24 to a range "between 20 and 30").

We will argue that in a sense ER and DP are opposites: the better one is at ER, them more one learns about real world entities, including their sensitive information. And to achieve DP, one must try to prevent the bits of information that have been published about an entity from being glued together by ER.

To illustrate, we present a simple motivating example. Consider an entity (person) Alice with the following information: her name is Alice, her address is 123 Main, her phone number is 555, her credit card number is 999, her social security number is 000. We represent Alice's information as the record: { ⟨N, Alice⟩, ⟨A, 123 Main⟩, ⟨P, 555⟩, ⟨C, 999⟩, ⟨S, 000⟩ }. Suppose now that Alice buys something on the Web and gives the vendor a subset of her information, say {⟨N, Alice⟩, ⟨A, 123 Main⟩, ⟨C, 999⟩}. By doing so, Alice has already partially compromised her privacy. We can quantify this "information leakage" in various ways: for instance we can say that the vendor has 3 out of 5 of Alice's attributes, hence the recall is $\frac{3}{5}$. We view leakage as a continuum, not as all-or-nothing. Low leakage (recall in our example metric) is desirable, since the vendor (or third party) know less about Alice, hence we try to *minimize* leakage. (Note we can actually weight attributes in our leakage computation by their sensitivity.)

Next, say Alice gets a job, so she must give her employer the following data: {⟨N, Alice⟩, ⟨A, 123 Main⟩, ⟨P, 555⟩, ⟨S, 000⟩}. In this case the leakage is $\frac{4}{5}$. This is where ER comes into play: If the employer and vendor somehow pool their data, they may be able to figure out that both records refer to the same entity. In general, in ER there are no unique identifiers: one must analyze the data and see if there is enough evidence. In our example, say the common name and address (and the lack of conflicting information) imply that the records match and are combined (and say the attributes are unioned). Then the third party has increased leakage (recall) to 1.

If Alice wants to prevent this increase in leakage, she may release *disinformation*, e.g., a new record that prevents the resolution that increased leakage. For example, say that Alice somehow gives the vendor the following additional record: {⟨N, Alice⟩, ⟨A, 123 Main⟩, ⟨P, 666⟩, ⟨C, 999⟩}. (Note the incorrect phone number.) Now the vendor resolves this third record with its first record, reaching the conclusion that Alice's phone number is 666. Now, when the vendor and employer pool their information, the different phone numbers lead them to believe that records correspond to different entities, so they are not merged and leakage does not increase. The incorrect phone number also decreases another metric we will consider, precision, since now not all of the third party's data is correct. Thus, leakage can decrease, not just by knowing less about Alice, but by mixing the correct data about Alice with incorrect data.

Before proceeding with the main body of our paper, we highlight the key features of our approach, which we believe make it well suited for studying ER and DP:

- Although not illustrated in our example, our model captures data confidences and multiple attribute values, both which arise naturally in ER. In particular, we believe less information has leaked if a third party is uncertain about Alice's attributes, as opposed to the case where the third party is certain.
- In most DP work, privacy is all-or-nothing, while as mentioned above, our leakage ranges between 0 (no information known by third party) to 1 (all information, and only correct information, is known). We believe that our continuous leakage model is more appropriate in our case: Alice *must* give out some sensitive data in order to buy products, get jobs, and so on. We cannot guarantee full privacy in this context; we can only quantify (and hopefully minimize) leakage. Furthermore, we are able to capture the notion that more leaked attributes is worse than fewer. For example, if a third party only knows our credit card number, that by itself is not a great loss. If the third party also learns our card expiration date, that is a more serious breach. If in addition they know our name and address, the information leakage is more serious.
- So far we have phrased leakage as a bad thing, something to be minimized. However, our model can also be used to study the mirror problem, where a good analyst is using ER to discover information about "bad guys". Here the goal is to maximize leakage, i.e., to discover how to perform ER so we can learn the most correct information about adversaries.

As we will show, our framework can help us answer fundamental questions on information leakage, for instance:

- Alice needs to give certain information to a store. She may want to know the impact of this release: Will the new information allow the store to "connect the dots" and piece together many previously released records? Or will the leakage increase be minimal?
- An analyst may want to understand what ER algorithms are best to increase information leakage.
- Alice may want to use disinformation to reduce the impact of previously leaked information. By adding some bogus information about herself, it becomes more costly for Eve to resolve Alice's correct information.

Table 1 summarizes the scenarios we have considered in our work. The first column shows whether or not the adversary (which we call Eve) uses confidences in the leakage model. The second column shows properties that are satisfied by the match and merge functions, which are the core functions used to compare and combine records, respectively (see Section 2.2). For each scenario (row) we have developed an algorithm that computes either query (defined in Section 2.2) or database (defined in Section 2.3) leakage given a reference record p for Alice, and a database R held by Eve. The third column shows the most efficient algorithm Eve can use for deriving the query leakage. The fourth column shows the most efficient algorithm Eve can use for deriving the database leakage. As one can see in the table, the more properties that hold, the more efficiently we can compute leakage.

The properties depend on the data semantics, and different applications (e.g., commercial products, publications, personal information) will have different properties. To show that the properties are achievable in practice, we have developed simple match and merge functions that satisfy some of the properties, as shown in Table 2.

In summary, our contributions on the convergence of ER and DP are two-fold:

- We propose a framework for measuring information leakage of the database for certain data and extend the framework to use uncertain data (Sections 2–3).

Table 1. Summary of Leakage Measurements

Confidence	Properties	Query	Database
No	(none)	Alg. 1	Alg. 4
No	Representativity	Alg. 2	Alg. 6
No	Neg. Representativity Representativity	Alg. 3	Alg. 8
Yes	(none)	Alg. 1	Alg. 4
Yes	Representativity Monotonicity Increasing	Alg. 2	Alg. 6
Yes	Neg. Representativity Representativity Monotonicity Increasing	Alg. 3	Alg. 8

Table 2. Summary of Match and Merge Functions

Names	Conf.	Properties	Section
M_c, μ_u	No	Neg. Representativity Representativity	2.2
M_e (or M_{em}), μ_u	No	Representativity	2.2
M_{cm}, μ_u	No	(none)	2.2
M_{et}, μ_{ui} (or μ_{ux})	Yes	Representativity Monotonicity Increasing	3.2

- We answer a variety of questions related to leakage and entity resolution. Among them, we study in detail the problem of releasing disinformation to minimize the database leakage (Sections 4–7).

2 Certain Data

We assume a database of records $R = \{r_1, r_2, \ldots, r_n\}$. The database could be a collection of social networking profiles, homepages, or even tweets. Or we can also think of R as a list of customer records of a company. Each record r is a set of attributes, and each attribute consists of a label and value. (In Section 3 we extend the model to values with confidences.) We do not assume a fixed schema because records can be from various data sources that use different attributes. As an example, the following record may represent Alice:

$$r = \{\langle \text{name}, \text{Alice}\rangle, \langle \text{age}, 20\rangle, \langle \text{age}, 30\rangle, \langle \text{zip}, 94305\rangle\}$$

Each attribute $a \in r$ is surrounded by angle brackets and consists of one label $a.lab$ and one value $a.val$. We also use the notation $r(l)$, which returns the set of all values of attributes having the label l (e.g., $r(\text{age}) = \{20, 30\}$). Notice that there are two ages for Alice. We consider $\langle$age, 20$\rangle$ and $\langle$age, 30$\rangle$ to be two separate pieces of information, even if they have the same label. Multiple label-value pairs with identical labels can occur when two records combine and the label-value pairs are simply collected (see Section 2.2 for details). In our example, Alice may have reported her age to be 20 in some case, but 30 in others. Although we cannot express the fact that Alice has only one age (either 20 or 30), the confidences we introduce in Section 3 can be used to indicate the likelihood of each value.

2.1 Record Leakage

We consider the scenario where Eve has one record r of Alice in her database R. (We consider the case where R contains multiple records in Section 2.2.) In this scenario, we only need to measure the information leaked by r in comparison to the "reference" record p that contains the complete information of Alice. We define $L_r(r,p)$ as the *record leakage* of r against p.

While leakage can be measured in a variety of ways, we believe that the well known concepts of precision and recall (and the corresponding F_1 metric) are very natural for this task. We first define the precision Pr of the record r against the reference p as $\frac{|r \cap p|}{|r|}$. Intuitively, Pr is the fraction of attributes in r that are also correct according to p. Suppose that $p = \{\langle \text{N, Alice}\rangle, \langle \text{A, 20}\rangle, \langle \text{P, 123}\rangle, \langle \text{Z, 94305}\rangle\}$ and $r = \{\langle \text{N, Alice}\rangle, \langle \text{A, 20}\rangle, \langle \text{P, 111}\rangle\}$. Then the precision of r against p is $\frac{2}{3} \approx 0.67$. We next define the recall Re of r against p as $\frac{|r \cap p|}{|p|}$. The recall reflects the fraction of attributes in p that are also found in r. In our example, the recall of r against p is $\frac{2}{4} = 0.5$. We can combine the precision and recall to produce a single metric called $F_1 = \frac{2 \times Pr \times Re}{Pr + Re}$. In our example, the F_1 value is $\frac{2 \times 0.67 \times 0.5}{0.67 + 0.5} \approx 0.57$.

There are many possible improvements for the proposed L_r, and we only list a few:

- We can give different weights to attributes based on the amount of information each attribute contains. For example, a record r that contains the address of Alice may leak more information than a record that only contains the zip code of Alice.
- We can discriminate labels that have multiple values when counting attributes. For instance, given a record $\{\langle \text{A, 10}\rangle, \langle \text{B, 20}\rangle, \langle \text{B, 30}\rangle\}$ where label B has two values, we may give more weight to the attribute $\langle \text{A, 10}\rangle$ than the attributes $\langle \text{B, 20}\rangle$ or $\langle \text{B, 30}\rangle$ when deriving the precision and recall.
- We can reflect the "effort" needed for the adversary to correctly resolve the records. For example, if Alice posts multiple profile pictures, Eve may have to ask someone who knows Alice to verify which photo is the right one. Alternatively, if Alice posts multiple phone numbers, then Eve must be able to figure out which phone number is correct by possibly calling the numbers one by one. The exact effort would thus depend on various factors including the attribute type and the background information Eve has on Alice.

For now, we believe that our simple definition of record leakage is a good starting point.

2.2 Query Leakage

We now consider the case where Eve has a database R containing multiple records. These records can represent information on different entities, and can be obtained directly from the entities, from public sources, or from other organizations Eve pools information with.

When Eve had a single record (previous section), we implicitly assumed that the one record was about Alice and we computed the resulting leakage based on what Eve knew about Alice. Now with multiple records, how does Eve know which records are "about Alice" and leak Alice's information? And what happens if multiple database records are about Alice?

To address these questions, we now define leakage, not as an absolute, but relative to a "query". For instance, Eve may pose the query "What do I know about $\{\langle \text{N, Alice}\rangle, \langle \text{A, 123Main}\rangle\}$". In this case, Eve is saying that the attributes "name: Alice" and "address: 123Main" identify an entity of interest to her, and would like to know what else is known about this entity. Note that this pair of attributes is not necessarily a unique key that identifies entity Alice; the two attributes are simply how Eve thinks of entity Alice. They may be insufficient to uniquely identify Alice, they may be more than is needed. Furthermore, there could be different attributes that also identify Alice.

Our next step is to compute leakage (relative to this query) by figuring out what Eve knows related to $\{\langle \text{N, Alice}\rangle, \langle \text{A, 123Main}\rangle\}$. But which database records are related to this query? And how are all the related records combined into what Eve knows about Alice?

To answer these questions, we introduce what we call the *match* and the *merge* functions in ER. A match function M compares two records r and s and returns true if r and s refer to the same real-world entity and false otherwise. A merge function μ takes two matching records r and s and combines them into a single record $\mu(r,s)$.

To illustrate how we use these functions to evaluate leakage, consider the database of Table 3, owned by Eve. Suppose that Eve identifies Alice by the query q=$\{\langle \text{N, Alice}\rangle, \langle \text{C, Google}\rangle\}$ (i.e., the Alice that works at Google). What else does Eve know about this Alice? We use a process called *dipping* to discover database records that match (defined by our function) the query record. That is, we first look for a database record that matches the query. When we find it, we merge the matching record with the query, using our merge function. In our example, say record r_2 matches q, so we obtain $r_q = \mu(q, r_2)$. Then we look for any database record that matches r_q, the expanded query, and merge it to our expanded record. For instance, say $M(r_q, r_1)$ evaluates to true, so we replace r_q by $\mu(r_q, r_1)$. We continue until no other database record matches. This process is called dipping because it is analogous to dipping say a pretzel (the query) into a vat of melted chocolate (the database). Each time we dip the pretzel, more and more chocolate may adhere to the pretzel, resulting in a delicious chocolate-covered pretzel (or an expanded query with all information related to Alice). Note that dipping is a type of entity resolution, where records in the database match against one record (the pretzel), as opposed to any record in the database.

Rec.	**Type**	**Attributes**
r_1	Social	⟨N, Alice⟩ , ⟨P, 123⟩ , ⟨B, Jan. 10⟩
r_2	Homepage 1	⟨N, Alice⟩ , ⟨C, Google⟩, ⟨A, 30⟩
r_3	Homepage 2	⟨N, Alice⟩, ⟨E, Stanford⟩ , ⟨A, 20⟩
r_4	Homepage 3	⟨N, Alice⟩, ⟨C, Boggle⟩, ⟨A, 50⟩

Table 3. Records of Alice on the Web

At the end of the dipping process, r_q represents what Eve knows about Alice (q), so we evaluate the leakage by comparing r_q to Alice's private information p, as before. Note that we will get different leakage for different queries. For instance, if Eve thinks of Alice as q={⟨N, Alice⟩}, more records will conglomerate in our example, which may lead to lower leakage (if r_3 and r_4 actually refer to different Alices) or higher leakage (if r_3 and r_4 are the same Alice as r_1 and r_2).

In the remainder of this section we define the dipping process more formally. We start by defining two properties that match and merge functions generally have (and that we assume for our work).

After defining leakage, in Section 2.2 we define additional optional properties for match and merge functions. If these properties hold (and they do not hold in all cases), dipping is substantially simplified. We also present simple match and merge functions that satisfy the optional properties, and in Section 2.2 we present efficient dipping algorithms that take advantage of the optional properties.

We assume two basic properties for M and μ – commutativity and associativity. Commutativity says that, if r matches s, then s matches r as well. In addition, the merged result of r and s should be identical regardless of the merge order. Associativity says that the merge order is irrelevant.

- Commutativity: $\forall r, s, M(r, s) =$ true if and only if $M(s, r) =$ true, and if $M(r, s) =$ true, $\mu(r, s) = \mu(s, r)$
- Associativity: $\forall r, s, t,\ \mu(r, \mu(s, t)) = \mu(\mu(r, s), t)$

We believe that most match and merge functions will naturally satisfy these properties. Even if they do not, they can easily be modified to satisfy the properties. To illustrate the second point, suppose that commutativity does not hold because $M(r, s)$ only compares r and s if r has an age smaller or equal to s and returns false otherwise. In that case, we can define the new match function $M'(r, s)$ to invoke $M(r, s)$ if r's age is smaller or equal to s's age and invoke $M(s, r)$ if s's age is smaller than r. In the case where the two properties are not satisfied, we only need to add a few more lines in our dipping algorithms for correctness (introduced later).

Before defining the dipping result of a set of records, we define the "answer sets" for Alice. Throughout the paper, we use the short-hand notation $\mu(S)$ for any associative merge function μ as the merged result of all records in the set of records S (if $S = \{r\}$, $\mu(S) = r$).

Definition 2.1 *Given a query q, a match function M, a merge function μ, and a set of record R, the collection of* answer sets *is $A = \{S_1, \ldots, S_m\}$ where each $S_i \in A$ is a set of records that satisfies the following conditions.*

- $q \in S_i$

- $S_i \subseteq R \cup \{q\}$
- *The records in* $S_i - \{q\}$ *can be reordered into a sequence* $[r_1, \ldots, r_m]$ *such that* $M(q, r_1) =$ *true,* $M(\mu(q, r_1), r_2)$ $=$ *true,* $\ldots$, $M(\mu(\{q, r_1, \ldots, r_{m-1}\}), r_m) =$ *true*

For example, suppose we have a database $R = \{r_1, r_2\}$ where r_1 and r_2 are clearly not the same person and have the names Alice and Alicia, respectively. However, say the query q matches either r_1 or r_2 because it contains both names Alice and Alicia and no other information. Then the answer set is $A = \{\{\}, \{q, r_1\}, \{q, r_2\}\}$. Notice that in this example the set $\{q, r_1, r_2\}$ is not in A because r_1, even after it merges with q, does not match r_2.

A dipping result of R is then the merged result of a "maximal" answer set from Definition 2.1 that has no other matching record in R.

Definition 2.2 *Given the collection of answer sets* A*, a match function* M*, and a merge function* μ*,* $r_q = \mu(S)$ *is a* dipping result *if* $S \in A$ *and* $\forall r \in R - S$*,* $M(r, \mu(S)) =$ *false.*

Continuing our example from above, the dipping result r_q can be either $\mu(r_1, q)$ or $\mu(r_2, q)$ because once q merges with r_1 (r_2), it cannot merge with r_2 (r_1). Notice that we exclude the case of merging multiple records with r_q at a time. For example, even if r_q matches with the merged record $\mu(r, s)$, but not with r or s individually, then we still cannot merge r_q with r and s.

We define the *query leakage* $L_q(p, q, M, \mu, R)$ of Alice as the maximum value of $L_r(p, r_q)$ for all possible dipping results r_q that can be produced using the match and merge functions M and μ on the database R. We first prove that, in general, deriving the query leakage is an NP-hard problem.

Proposition 2.3 *Without any properties, the problem of finding the query leakage* L_q *is NP-hard.*

Proof. We reduce the knapsack problem (which is known to be NP-complete) to the query leakage problem. One version of the knapsack problem can be stated as follows: Given a set of non-negative integers $\{x_1, \ldots, x_n\}$, does the sum of some non-empty subset equal a bag size of W? For any instance of the knapsack problem, we construct a corresponding set of records R for the dipping problem. For each integer x_i, we create a corresponding record $r_i = \{\langle A, x_i \rangle\}$. Say we denote the A value of the record r as $r.A$. We set the query q to any record, say, r_1. For the match function M, we define $M(r_i, r_j)$ to return true if $r_i.A < W \wedge r_j.A < W$. Otherwise, $M(r_i, r_j)$ returns false. For the merge function μ, we define $\mu(r_i, r_j)$ as $\{\langle A, r_i.A + r_j.A \rangle\}$. Also, we define the reference information (or simply reference) $p = \{\langle A, W \rangle\}$. We now show that query leakage $L_q(p, q, M, \mu, R)$ is nonnegative if and only if there is a non-empty subset of records $\{r_{i_1}, \ldots, r_{i_m}\}$ such that the sum of the corresponding integers $\Sigma_{i \in \{i_1, \ldots, i_m\}} x_i = W$ (i.e., there exists a non-empty subset of $\{x_1, \ldots, x_n\}$ whose sum is equal to W). Suppose that the query leakage is nonnegative. Then the dipping result $r_q = \{\langle A, W \rangle\}$, which means that there is a sequence of records whose corresponding integers add up to W. Suppose that there is a set of records whose corresponding set of integers add to W. Then we can merge the records in any order (since they all match with each other) until we obtain $r_q = \{\langle A, W \rangle\}$.

Algorithm 1 shows a brute-force algorithm that returns the dipping result r_q and query leakage L_q of a query q against a database R. For example, say that $R = \{r, s\}$, and only r matches q. For each permutation of R, we merge the records in the permutation with q sequentially until there is a record that does not match r'_q (Steps 5–13). In our example, suppose we are scanning the permutation $[r, s]$. Since r matches q, we merge the two records and, assuming that s does not match with $\mu(q, r)$, we have $r'_q = \mu(q, r)$. We then check if any remaining record in R matches with r'_q (Steps 9–12) in order to verify that r'_q satisfies Definition 2.1. If r'_q is indeed produced by an answer set, we then compare the record leakage of r'_q (in our example, we compute $L_r(p, \mu(q, r))$ with the current query leakage L_q and update r_q and L_q if necessary (Steps 14–16). After iterating through all the permutations of R, we return the dipping result r_q and corresponding query leakage L_q.

Algorithm 1 has a complexity of $O(|R| \times |R|!)$. In the following sections, we explore significantly more efficient ways to compute the query leakage given desirable properties for the match and merge functions. We give examples of match and merge functions satisfying the properties (summarized in Table 2).

Algorithm 1: Brute Force Query Leakage Algorithm

```
input : the reference $p$, a query $q$, a match function $M$, a merge function $\mu$, and a set of records $R$
output: the dipping result $r_q$ and query leakage $L_q(p, q, M, \mu, R)$
1  $L_q \leftarrow 0, r_q \leftarrow \{\}$;
2  foreach permutation $[r_{i_1}, \ldots, r_{i_{|R|}}]$ of the records in $R$ do
3      $r'_q \leftarrow q$;
4      $m \leftarrow$ false;
5      for $j = 1 \ldots |R|$ do
6          if $M(r'_q, r_{i_j})$ = true then
7              $r'_q \leftarrow \mu(r'_q, r_{i_j})$;
8          else
9              for $k = j \ldots |R|$ do
10                 if $M(r'_q, r_{i_k})$ then
11                     $m \leftarrow$ true;
12                     break;
13             break;
14         if $m$ = false $\wedge L_r(p, r'_q) > L_q$ then
15             $L_q \leftarrow L_r(p, r'_q)$;
16             $r_q \leftarrow r'_q$;
17 return $(r_q, L_q)$;
```

Properties We identify two desirable properties for M and μ: representativity and negative representativity. Representativity says that a merged record $\mu(r, s)$ "represents" r and s and matches with all records that match with either r or s. Intuitively, there is no "negative evidence" so merging r and s cannot create evidence that would prevent $\mu(r, s)$ from matching with any record that matches with r or s. It can be shown that representativity guarantees the uniqueness of a dipping result and is needed for efficient dipping. Negative representativity says that two records r and s that do not match will never match even if r or s merges with other records. That is, there is no "positive evidence" where r and s will turn out to be the same entity later on. The negative representativity property also enables efficient dipping. Note that the two properties above do not assume that r matches with s. We can show that none of the properties imply each other.

- Representativity: If $t = \mu(r, s)$, then for any u where $M(r, u)$ = true, we also have $M(t, u)$ = true
- Negative Representativity: If $t = \mu(r, s)$, then for any u where $M(r, u)$ = false, we also have $M(t, u)$ = false

Notice that none of the properties imply each other. For example, suppose that we have the database $\{r, s, t\}$. Suppose that $M(r, s)$ = true, $M(s, t)$ = false, and $M(\mu(r, s), t)$ = true, and that all the other pairs of records match with each other. The results satisfy representativity, but not negative representativity. Now suppose that $M(r, s)$ = true, $M(s, t)$ = true, and $M(\mu(r, s), t)$ = false, and that all the other pairs of records match with each other. While the results satisfy negative representativity, they violate representativity. We now illustrate sample match and merge functions that satisfy one, both, or none of the two properties.

We illustrate match and merge functions that satisfy commutativity, associativity, representativity, and negative representativity.

We define the match function M_c to use a single "key set" for comparing two records. A key set k is a minimal set of attribute labels $\{l_1, \ldots, l_m\}$ that are sufficient to determine if two records are the same using equality checks. All records are assumed to have values for the key-set attributes. The M_c function then matches r and s only if they have the exact same key-set attributes. For example, given the key set $k = \{A, B\}$, the record $r = \{\langle A, a\rangle, \langle B, b\rangle\}$ matches with $s = \{\langle A, a\rangle, \langle B, b\rangle, \langle C, c\rangle\}$, but not with $t = \{\langle A, a\rangle, \langle A, a'\rangle, \langle B, b\rangle\}$. As another example, the record $r = \{\langle A, a_1\rangle, \langle A, a_2\rangle, \langle B, b\rangle\}$ matches with $s =$

$\{\langle A, a_1\rangle, \langle A, a_2\rangle, \langle B, b\rangle\}$, but not with $t = \{\langle A, a_1\rangle, \langle B, b\rangle\}$. The character 'c' in M_c thus stands for the "complete comparison" of the key-set attributes between r and s.

The merge function μ_u unions the attributes of r and s (i.e., $\mu(r, s) = r \cup s$). For instance, if $r = \{\langle A, a\rangle, \langle B, b\rangle\}$ and $s = \{\langle A, a\rangle, \langle C, c\rangle\}$, then $\mu(r, s) = \{\langle A, a\rangle, \langle B, b\rangle, \langle C, c\rangle\}$. The character 'u' in μ_u stands for "union".

Proposition 2.4 *The match and merge functions M_c and μ_u satisfy commutativity, associativity, representativity, and negative representativity.*

Proof. The functions M_c and μ_u are commutative because if two records have the same key-set attributes, M_c returns true regardless of the ordering and μ_u simply unions the attributes. Next, μ_u is associative because of the union function. We now prove that M_c and μ_u are representative. Suppose that $t = \mu_u(r, s)$. Since we only merge matching records for a dipping solution, r and s have the same key-set attributes. Then for any u where $M_c(r, u)$ = true, we know that r and u have the same key-set attributes. Thus, $\mu_u(r, s)$ and u also have the same key-set attributes and match with each other. Finally, we prove that M_c and μ_u are negative representative. Suppose that $t = \mu_u(r, s)$. For any u where $M_c(r, u)$ = false, we know that r and u have different key-set attributes. Hence, $\mu_u(r, s)$ and u have different key-set attributes as well, so $M_c(t, u)$ = false.

We now illustrate match and merge functions that satisfy commutativity, associativity, and representativity, but not negative representativity. We define M_e to compare records based on one key set k (as in M_c; see Section 2.2), but now does an existential comparison of matching values for each key-set attribute. That is, r and s match when for each key-set label $l \in k$, there exists two attributes $a \in r$, $a' \in s$ where $a.lab = a'.lab$ and $a.val = a'.val$. For example, given the key set $k = \{A, B\}$, the record $r = \{\langle A, a\rangle, \langle B, b\rangle\}$ matches with $s = \{\langle A, a\rangle, \langle A, a'\rangle, \langle B, b\rangle\}$ because of the common attributes $\langle A, a\rangle$ and $\langle B, b\rangle$. (The two records do not match according to M_c.) As another example, the record $r = \{\langle A, a_1\rangle, \langle A, a_2\rangle, \langle B, b\rangle\}$ matches with $s = \{\langle A, a_1\rangle, \langle A, a_2\rangle, \langle B, b\rangle\}$ and also with $t = \{\langle A, a_1\rangle, \langle B, b\rangle\}$ (unlike M_c). The character 'e' in M_e stands for "existential" comparisons.

Proposition 2.5 *The match and merge functions M_e and μ_u satisfy commutativity, associativity, representativity, but not negative representativity.*

Proof. The functions M_e and μ_u satisfy commutativity because M_e and μ_u are not order sensitive in matching and merging records, respectively. Next, μ_u is associative because of the union operation. We prove that M_e and μ_u are representative. Suppose that $t = \mu_u(r, s)$. Then t contains all the attributes of r. For any u where $M_e(r, u)$ = true, we know that for each key-set label $l \in k$, there exists two attributes $a \in r$, $a' \in u$ where $a.lab = a'.lab$ and $a.val = a'.val$. Since t contains all attributes of r, it is also true that for each key-set label $l \in k$, there exists two attributes $a \in t$, $a' \in u$ where $a.lab = a'.lab$ and $a.val = a'.val$. Hence, $M_e(t, u)$ = true. Finally, we prove that M_e and μ_u do not satisfy negative representativity using a counter example. Suppose that $r = \{\langle A, a'\rangle\}$, $s = \{\langle A, a\rangle, \langle A, a'\rangle\}$, and $u = \{\langle A, a\rangle\}$. Using the key set $k = \{A\}$, we know that, while $M_e(r, u)$ = false, $M_e(\mu_u(r, s), u)$ = true because $\mu_u(r, s) = \{\langle A, a\rangle, \langle A, a'\rangle\}$.

We also define an extended version of M_e (called M_{em}) that uses multiple key sets instead of just one. That is, we now have a set of key sets $\mathcal{K} = \{k_1, \ldots, k_m\}$, and $M_{em}(r, s)$ is true if and only if there exists a key $k \in \mathcal{K}$ such that $M_e(r, s)$ is true. The character 'm' in M_{em} stands for "multiple" key sets. This time, we assume that each record has values for all the key sets.

Proposition 2.6 *The match and merge functions M_{em} and μ_u satisfy commutativity, associativity, representativity, but not negative representativity.*

Proof. The functions M_{em} and μ_u satisfy commutativity because M_{em} and μ_u are not order sensitive in matching and merging records, respectively. Next, μ_u is associative because of the union operation. We prove that M_{em} and μ_u are representative. Suppose that $t = \mu_u(r, s)$. Then t contains all the attributes of

r. For any u where $M_{em}(r,u)$ = true, we know that $M_e(r,u)$ = true for some key set $k \in \mathcal{K}$. Since t contains all the attributes of r, it is also true that $M_e(t,u)$ = true for k. Hence, $M_{em}(t,u)$ = true. To prove that M_{em} and μ_u do not satisfy negative representativity, we can use the same counter example in the proof of Proposition 2.5.

We now illustrate match and merge functions that satisfy commutativity, associativity, but neither negative representativity nor representativity. We extend M_c to support multiple key sets (called M_{cm}). We now use a set of key sets $\mathcal{K} = \{k_1, \ldots, k_m\}$, and $M_{cm}(r,s)$ is true if and only if there exists a key $k \in \mathcal{K}$ such that $M_c(r,s)$ is true.

Proposition 2.7 *The match and merge functions M_{cm} and μ_u satisfy commutativity, associativity, but neither negative representativity nor representativity.*

Proof. The functions M_{cm} and μ_u satisfy commutativity because M_{cm} and μ_u are not order sensitive in matching and merging records, respectively. Next, μ_u is associative because of the union operation. We prove that M_{cm} and μ_u do not satisfy negative representativity using a counter example. Suppose that $r = \{\langle B,b\rangle\}$, $s = \{\langle A,a\rangle, \langle B,b\rangle\}$, and $u = \{\langle A,a\rangle\}$. Using the key sets $k_1 = \{A\}$ and $k_2 = \{B\}$, we know that, while $M_{cm}(r,u)$ = false, $M_{cm}(\mu_u(r,s),u)$ = true because $\mu_u(r,s) = \{\langle A,a\rangle, \langle B,b\rangle\}$. Finally, we prove that M_{cm} and μ_u do not satisfy representativity using a counter example. Suppose that $r = \{\langle A,a\rangle, \langle B,b\rangle\}$, $s = \{\langle A,a'\rangle, \langle B,b\rangle\}$, and $u = \{\langle A,a\rangle\}$. Using the key sets $k_1 = \{A\}$ and $k_2 = \{B\}$, we know that, while $M_{cm}(r,u)$ = true, $M_{cm}(\mu_u(r,s),u)$ = false because $\mu_u(r,s) = \{\langle A,a\rangle, \langle A,a'\rangle, \langle B,b\rangle\}$.

We prove the uniqueness of the dipping result given the representativity property for the match and merge functions.

Proposition 2.8 *If M and μ satisfy representativity, the dipping result of Definition 2.2 is unique.*

Proof. Suppose there are two different sets S and S' that satisfy Definition 2.2. Without loss of generality, we assume that S contains at least one record that is not in S'. We prove that there exists a record $r \in R$ and $r \notin S'$, but $M(\mu(S'),r)$ = true, violating Definition 2.2 for S'. Let $r' \in S$ be the first record in the sequence $[r_1, \ldots, r_m]$ that is not in S' (r' always exists because of our assumption). We know that $M(\mu(\{q, r_1, \ldots, r_{i-1}\}), r_i)$ = true and that $\mu(\mu\{q, r_1, \ldots, r_{i-1}\}, \mu(S' - \{q, r_1, \ldots, r_{i-1}\})) = \mu(S')$. Thus by representativity $M(\mu(S'), r_i)$ = true. Hence, we have found a record $r_i \in R$ such that $r_i \notin S'$ and $M(\mu(S'), r_i)$ = true as we have claimed. Hence, $S = S'$, which means that a set S satisfying Definition 2.2 is unique.

Thus if M and μ satisfy representativity, then by Proposition 2.8 the query leakage $L_q(p, q, M, \mu, R) = L_r(p, r_q)$ for the unique dipping result r_q.

Dipping We now explore two efficient algorithms that dip a query q into the database R and collect Alice's information into r_q using a dipping algorithm D. The first algorithm assumes representativity while the second algorithm assumes both representativity and negative representativity. Since both algorithms assume representativity, the query leakage is then computed as $L_r(p, r_q)$ (where p is the reference) once r_q is computed.

Representativity Only We consider the case where M and μ satisfy representativity, but not negative representativity. As a result, merging records may trigger additional new matches. For instance, suppose that we use the database $R = \{r, s\}$ and a query q. Also, say we use a match function M and a merge function μ where $M(q,r)$ = true, $M(q,s)$ = false, and $M(\mu(q,r),s)$ = true. Clearly the negative representativity property is violated because $M(q,s)$ = false, but $M(\mu(q,r),s)$ = true. Algorithm 3 would compare q with the records r and s and return $r_q = \mu(q,r)$ as a result. Since $\mu(q,r)$ now matches with s, however, the correct answer should be $\mu(\{q,r,s\})$. Due to this additional "chaining" of matches, the record comparisons need to be repeated iteratively until there are no matches. We propose an iterative dipping algorithm that can use these match and merge functions to return a correct dipping result.

Algorithm 2 returns a dipping result using multiple iterations. For example, suppose that we have the set of records $R = \{r_1 = \{\langle\text{name, Alli}\rangle, \langle\text{email, alice@yahoo}\rangle, \langle\text{phone, 123}\rangle\}, r_2 = \{\langle\text{name, Alice}\rangle, \langle\text{phone, 123}\rangle\}\}$ and a query $q = \{\langle\text{name, Alice}\rangle, \langle\text{email, alice@yahoo}\rangle\}$. Also say that we use the match function M_e, the merge function μ_u, and the key sets $k_1 = \{\text{name}\}$ and $k_2 = \{\text{email, phone}\}$. Algorithm 2 starts reading the records in R and compares them with q (Step 5–10). Since r_1 does not match with q for any key set, we add r_1 in R' (Step 10). Next, r_2 matches with q according to k_1, so we merge r_2 with q into $\mu_u(r_2, q)$ (Step 7) and set m to true. Since m is true, we repeat the loop of Steps 2–13. We start with $R = \{r_1\}$ and $r_q = \{\mu_u(r_2, q)\}$. This time, $\mu_u(r_2, q)$ matches with r_1 according to k_2 because $\mu_u(r_2, q) = \{\langle\text{name, Alice}\rangle, \langle\text{email, alice@yahoo}\rangle, \langle\text{phone, 123}\rangle\}$ and thus has the same email and phone combination as r_1. We thus merge $\mu_u(r_2, q)$ with r_1 into $\mu_u(\{r_1, r_2, q\})$. Eventually, we terminate the loop and return $r_q = \mu_u(\{r_1, r_2, q\})$ and the query leakage $L_r(p, r_q)$.

Algorithm 2: Multi-Pass Dipping Algorithm

```
input  : a query $q$, a match function $M$, a merge function $\mu$, and a set of records $R$
output: the dipping result $r_q$ and its query leakage $L_q$
1  $r_q \leftarrow q$;
2  repeat
3      $m \leftarrow$ false;
4      $R' \leftarrow \{\}$;
5      for $r \in R$ do
6          if $M(r, r_q)$ = true then
7              $r_q \leftarrow \mu(r_q, r)$;
8              $m \leftarrow$ true;
9          else
10             $R' \leftarrow R' \cup \{r\}$;
11     if $m$ = false then
12         break;
13     $R \leftarrow R'$;
14 until;
15 return $(r_q, L_r(p, r_q))$ ;
```

Proposition 2.9 *Algorithm 2 returns a dipping result satisfying Definition 2.2.*

Proof. We first show that the records merged into r_q form a set S that satisfies Definition 2.1. First, $q \in S$ by Step 1. Second, $S \in R \cup \{q\}$ by Step 7. Third, by the behavior of Algorithm 2, we know that the sequence of records $[r_1, \ldots, r_m]$ that merge with q satisfies $M(q, r_1) = \text{true}$, $M(\mu(q, r_1), r_2) = \text{true}$, …, $M(\mu(\{q, r_1, \ldots, r_{m-1}\}), r_m) = \text{true}$. We next show that $\forall r \in R - S, M(r, r_q) = \text{false}$. If $M(r, r_q) = \text{true}$, then r would have merged with r_q at some point (Step 7) because Algorithm 2 simply loops until there are no matching records with r_q. Hence, S is in the answer set A, and Algorithm 2 returns a valid dipping result $r_q = \mu(S)$.

Proposition 2.10 *Algorithm 2 has a worst-case complexity of $O(|R|^2)$ and is optimal in the number of record comparisons in the worst case.*

Proof. We count the maximum number of record comparisons performed by Algorithm 2. Say that $n = |R|$. For the first loop iteration, n record comparisons are done. The loop only continues if there was at least one merge. In the worst case, there will be exactly one merge. Thus, in the next loop, $|R|$ decreases by 1, and we must do $n - 1$ comparisons. We continue the loop until all the records have merged with r_q. Thus, the total number of record comparisons is $n + (n-1) + \ldots + 1 = \frac{n \times (n+1)}{2}$ in the worst case.

We prove that no other algorithm that returns a dipping result r_q satisfying Definition 2.2 can perform better than Algorithm 2 in terms of record comparisons in the worst case. We construct an "adverserial" dataset consisting of n distinct records that behaves as follows: For each loop, we only find one match after comparing all records in R with r_q, thus effectively forcing the algorithm to perform $|R|$ comparisons. After r_q merges with one record, we compare r_q with all the records in R only to find a match at the very last comparison and so on. After $n-1$ merges, the algorithm was forced to perform at least $\frac{n\times(n+1)}{2}$ record comparisons.

One could improve the runtime of dipping given more details in the match and merge functions. For example, if we are using the match function M_{em}, we can use an index for reducing the number of record comparisons. Specifically, we can construct one index per key set $k \in \mathcal{K}$ and search the indices for records that match the query according to at least one key set.

Representativity and Negative Representativity We now show a dipping algorithm that scans R once and returns a dipping result satisfying Definition 2.2 given that the match and merge functions satisfy negative representativity. The match and merge functions must also satisfy representativity for the dipping result to be unique so that we can compute the correct query leakage L_q. Algorithm 3 merges all records that match with the query q. As an illustration, suppose that we use M_c and μ_u as the match and merge functions. Also, say we have the set of records $R = \{r_1 = \{\langle$name, Alli$\rangle$, $\langle$email, alice@yahoo$\rangle\}$, $r_2 = \{\langle$name, Alice$\rangle$, $\langle$phone, 123$\rangle\}\}$ and a query $q = \{\langle$name, Alice$\rangle$, $\langle$zip, 987$\rangle\}$. We set M_c to use the key set $k = \{$name$\}$. Algorithm 3 scans the records in R and merges those that match with q into r_q. In our example, only record r_2 matches with q, so $r_q = \mu(q, r_2) = \{\langle$name, Alice$\rangle$, $\langle$phone, 123$\rangle$, $\langle$zip, 987$\rangle\}$.

Algorithm 3: One-Pass Dipping Algorithm

```
  input  : a query $q$, a match function $M$, a merge function $\mu$, and a set of records $R$
  output: the dipping result $r_q$ and query leakage $L_q$
1 $r_q \leftarrow q$;
2 for $r \in R$ do
3     if $M(r, r_q)$ = true then
4         $r_q \leftarrow \mu(r_q, r)$;
5 return $(r_q, L_r(p, r_q))$;
```

Proposition 2.11 *If M and μ satisfy negative representativity, Algorithm 3 returns a dipping result satisfying Definition 2.2.*

Proof. We first show that the records merged into r_q form a set S satisfying Definition 2.1. First, $q \in S$ by Step 1. Second, $S \in R \cup \{q\}$ by Steps 2–4. Third, by the behavior of Algorithm 3, we know that the sequence of records $[r_1, \ldots, r_m]$ that merge with q satisfies $M(q, r_1)$ = true, $M(\mu(q, r_1), r_2)$ = true, ..., $M(\mu(\{q, r_1, \ldots, r_{m-1}\}), r_m)$ = true. We next show that $\forall r \in R - S, M(r, r_q)$ = false. At some point, r must have been compared with an intermediate version of r_q (Step 3) and failed to match. By negative representativity, the final r_q does not match with r as well. Hence, S is in the answer set A, and Algorithm 3 returns a valid dipping result $r_q = \mu(S)$.

The complexity of Algorithm 3 is $O(|R|)$ because it simply compares each record in R with q. The complexity of the algorithm shows that with more properties for the match and merge functions, it becomes easier for Eve to exhaustively figure out Alice's information. Again, one could enhance the dipping process given more details of the match and merge functions. For example, if we are using the match function M_c, we can improve the runtime of dipping by using an index that only points to records that have the same key-set values as q.

While Algorithm 3 returns a dipping result satisfying Definition 2.2 just by assuming negative representativity for M and μ, we still assume representativity for M and μ. Otherwise, computing $L_r(p, r_q)$ may not produce the correct query leakage of q against R.

2.3 Database Leakage

What happens if we do not know how Eve identifies Alice, i.e., if we do not have a specific query q? In such a case, we may assume that Eve can think of any one of the records in R as "Alice's record". Thus, for each R record we can compute a leakage number, and by taking the maximum value we can obtain a worst case leakage, representing what Eve can *potentially* know about Alice.

More formally, we define the database leakage $L_d(p, M, \mu, R)$ as $max_{q \in R}\ L_q(p, q, M, \mu, R - \{q\})$. That is, for each record $q \in R$, we compute the dipping of q on $R - \{q\}$ (i.e., the database without q) and choose the worst-case query leakage of Alice as the entire database leakage. We also define the corresponding dipping result with the maximum query leakage as the maximum dipping result.

No Properties Algorithm 4 shows a brute-force algorithm that uses Algorithm 1 to compute the database leakage L_d of R. For example, given $R = \{r, s, t\}$, we return the maximum value of $L_q(p, r, M, \mu, \{s, t\})$, $L_q(p, s, M, \mu, \{r, t\})$, and $L_q(p, t, M, \mu, \{r, s\})$ as the database leakage and the corresponding dipping result as the maximum dipping result. The complexity of Algorithm 4 is $O(|R|^2 \times |R|!)$ because Algorithm 1 already has the complexity of $O(|R| \times |R|!)$.

```
Algorithm 4: Brute Force Database Leakage Algorithm
  input  : the reference $p$, a match function $M$, a merge function $\mu$, and a set of records $R$
  output: the maximum dipping result $r_q$ and database leakage $L_d$
1 $L_d \leftarrow 0$;
2 for $r \in R$ do
3     $(r'_q, L_q) \leftarrow$ Algorithm 1$(p, r, M, \mu, R - \{r\})$;
4     if $L_q > L_d$ then
5         $L_d \leftarrow L_q$;
6         $r_q \leftarrow r'_q$;
7 return $(r_q, L_d)$;
```

Representativity Only We now assume that M and μ satisfy representativity, but not negative representativity. Again, we are assuming that there is no negative evidence where merging two records r and s cannot create evidence that would prevent $\mu(r, s)$ from matching with any record that matches with r or s. As a result, computing each query leakage of $q \in R$ becomes more efficient by using Algorithm 2, and thus the computation of the database leakage significantly improves as well compared to the brute-force approach.

Algorithm 5 is similar to Algorithm 4 except that it uses Algorithm 2 to compute the database leakage instead of Algorithm 1 assuming that representativity holds. Algorithm 5 computes the query leakage for each record $r \in R$ (Step 3) and returns the maximum value as the database leakage along with the corresponding dipping result as the maximum dipping result. The complexity of Algorithm 5 is $O(|R|^3)$ because Algorithm 2 already has a complexity of $O(|R|^2)$.

Enhancement using Clustering The computation time of the database leakage can be significantly improved by further exploiting the representativity property. We define the notion of a *valid* clustering of R using M and μ as any partition $C = \{c_1, \ldots, c_m\}$ where $\bigcup_{c \in C} c = R$ and $\forall c_i, c_j \in C$ where $c_i \neq c_j, M(\mu(c_i), \mu(c_j))$ = false. We can show that computing the database leakage for each cluster in C (which is much faster than

Algorithm 5: Database Leakage Algorithm assuming Representativity

```
input  : the reference $p$, a match function $M$, a merge function $\mu$, and a set of records $R$
output: the maximum dipping result $r_q$ and database leakage $L_d$
1 $L_d \leftarrow 0$;
2 for $r \in R$ do
3     $(r'_q, L_q) \leftarrow$ Algorithm 2$(p, r, M, \mu, R - \{r\})$;
4     if $L_q > L_d$ then
5         $L_d \leftarrow L_q$;
6         $r_q \leftarrow r'_q$;
7 return $(r_q, L_d)$;
```

computing the database leakage of the entire R), and then taking the maximum database leakage among all the results produces the same leakage value as the database leakage of R.

Given any algorithm that returns a valid clustering, Algorithm 6 improves Algorithm 5 by "localizing" the query leakage computation within the clusters. We first produce a valid clustering of R (see Algorithm 7 for details) at Step 2. For example, suppose that $R = \{r, s, t\}$ and that only the records r and s match with each other. Say that the clustering algorithm returns the clustering $C = \{\{r, s\}, \{t\}\}$. Next, Algorithm 6 computes the database leakage L_d for each cluster (Step 4). In our example, we compute the database leakage for the clusters $\{r, s\}$ and $\{t\}$. (The results are $L_r(p, \mu(r, s))$ and $L_r(p, t)$, respectively.) The final database leakage is then the maximum value of all the L_d's of the clusters. In our example, we return $max\{L_r(p, \mu(r, s)), L_r(p, t)\}$. The maximum dipping result r_q is $\mu(r, s)$ if $L_r(p, \mu(r, s)) \geq L_r(p, t)$ and t otherwise.

Algorithm 6: Improved Database Leakage Algorithm assuming Representativity

```
input  : the reference $p$, a match function $M$, a merge function $\mu$, and a set of records $R$
output: the maximum dipping result $r_q$ and database leakage $L_d$
1 $L_d \leftarrow 0$;
2 $\{c_1, \ldots c_m\} \leftarrow$ ClusteringAlgorithm$(M, \mu, R)$;
3 for $c \in \{c_1, \ldots, c_m\}$ do
4     $(r'_q, L'_d) \leftarrow$ Algorithm 5$(p, M, \mu, c)$;
5     if $L'_d > L_d$ then
6         $L_d \leftarrow L'_d$;
7         $r_q \leftarrow r'_q$;
8 return $(r_q, L_d)$;
```

We show that Algorithm 6 returns the correct database leakage given that M and μ satisfy representativity and the clustering algorithm returns a valid clustering.

Proposition 2.12 *Suppose that M and μ satisfy representativity. Then given a clustering algorithm that returns a valid clustering, Algorithm 6 returns the database leakage L_d of the database R.*

Proof. The only difference between Algorithm 6 and Algorithm 5 is that the dipping of each record $r \in R$ is done on the cluster $c \in C$ (where $r \in c$) instead of the entire database R. Hence, we only need to prove that the dipping result r_q of r on c where $c \in C$ and $r \in c$ is always the same as the dipping result r'_q of r on R. (Consequently, the query leakage of r would be the same for both cases.) Suppose that r_q and r'_q are not the same. Then by the representativity property and Proposition 2.8, the set of records S and S' used to produce r_q and r'_q (i.e., $\mu(S) = r_q$ and $\mu(S') = r'_q$) are not the same. Suppose that $S - S' \neq \{\}$. Then given the sequence of records $[r, r_1, \ldots, r_{|S|}]$ used to produce r_q, let r_i be the first record not in S'. Then we know that $M(\mu(\{r, r_1, \ldots, r_{i-1}\}), r_i) =$ true and that $\{r, r_1, \ldots, r_{i-1}\} \subseteq S'$. Hence by representativity, $M(S', r_i) =$

true, which contradicts the assumption that r_i does not match S'. Now suppose that $S' - S \neq \{\}$. Then given the sequence of records $[r, r_1, \ldots, r_{|S'|}]$ used to produce r'_q, let r_i be the first record not in S. If $r_i \in c$, then we can use an argument similar to the one above to prove that $M(S, r_i) =$ true. If $r_i \notin c$, then r has merged with a record outside c. However, $M(\mu(\{r, r_1, \ldots, r_{i-1}\}), r_i) =$ true and $\{r, r_1, \ldots, r_{i-1}\} \subseteq c$, so $M(\mu(c), r_i) =$ true by representativity. Suppose the cluster $c' \in C$ contains r_i. Then again by representativity, $M(\mu(c), \mu(c'))$ = true, violating the assumption that the clustering $\{c_1, \ldots, c_k\}$ of R is valid. Hence we conclude that $S = S'$, and thus the dipping results r_q and r'_q are the same. Consequently, the query leakage of r is the same for Algorithms 5 and 6 and so is the final database leakage.

Assuming that each cluster has an average size of Z, the complexity of Algorithm 6 is $O(C(|R|) + \frac{|R|}{Z} \times Z^3) = O(C(|R|) + |R| \times Z^2)$ where $C(|R|)$ indicates the clustering algorithm complexity. (We later show that Algorithm 7 below has a complexity of $O(|R|^2)$) Although this complexity does not improve the complexity of Algorithm 5 (i.e., when $Z = |R|$, then there is no gain in complexity), the cluster sizes are very small related to $|R|$ in practice, and the computation of the database leakage can significantly improve.

We now elaborate on a clustering algorithm that can be used in Algorithm 6. Given M, μ, and R, Algorithm 7 produces a valid clustering result. For example, suppose that $R = \{r, s, t\}$, and the only matches are $M(s, t) =$ true and $M(r, \mu(s, t)) =$ true. We first convert R to $C = \{\{r\}, \{s\}, \{t\}\}$ (Step 1), and read in the cluster $\{r\}$ as c (Step 4). Since $\mu(\{r\}) = r$ does not match with $\mu(\{s\}) = s$ or $\mu(\{t\}) = t$, we move $\{r\}$ to C' (Step 12). Next, we read $\{s\}$ as c. Although $\mu(\{s\})$ does not match with $\mu(\{r\})$, it does match with $\mu(\{t\})$, and we merge $\{s\}$ and $\{t\}$ into $\{s, t\}$ and add it to C (Steps 14–15). We now set c as $\{r, s\}$. Since $\mu(\{r, s\})$ matches with $\{t\}$, we merge $\{r, s\}$ and $\{t\}$ into $\{r, s, t\}$ and add it to C. After a few more steps, we return $C' = \{\{r, s, t\}\}$.

Algorithm 7: Clustering Algorithm assuming Representativity

```
   input : a match function $M$, a merge function $\mu$, and a set of records $R$
   output: a set of clusters $C = \{c_1, \ldots, c_m\}$
 1 $C \leftarrow \{\{r\} | r \in R\}$;
 2 $C' \leftarrow \{\}$;
 3 while $C \neq \{\}$ do
 4     $c \leftarrow$ a cluster from $C$;
 5     $C \leftarrow C - \{c\}$;
 6     $s \leftarrow$ null;
 7     for $c' \in C'$ do
 8         if $M(\mu(c), \mu(c'))$ = true then
 9             $s \leftarrow c'$;
10             break;
11     if $s$ = null then
12         $C' \leftarrow C' \cup \{c\}$;
13     else
14         $C' \leftarrow C' - \{s\}$;
15         $C \leftarrow C \cup \{c \cup s\}$;
16 return $C'$;
```

We prove two interesting properties of Algorithm 7. First, the result C of Algorithm 7 is the maximum valid clustering of R. That is, no valid clustering has a larger number of clusters than C. In fact, we can also prove that the maximum valid clustering is unique. Second, the complexity of Algorithm 7 is $O(|R|^2)$, and the number of record comparisons is optimal in the worst case.

Proposition 2.13 *Algorithm 7 returns the maximum valid clustering of R.*

Proof. Say that Algorithm 7 returns the clustering result C_1 and there exists a larger clustering result C_2 (i.e., $|C_1| < |C_2|$). Suppose that we keep track of the set $C \cup C'$ for each record merge in Algorithm 7. Initially, $C \cup C' = \{\{r\}|r \in R\}$. After the last merge, $C \cup C' = C_1$. We now start merging records from the initial state until we merge the clusters c and s (Step 15) where $c \subseteq c'$ and $s \notin c'$ for some cluster $c' \in C_2$. Such an instance is guaranteed to exist because $|C_1| < |C_2|$. Say that $s \subseteq c''$ where $c'' \in C_2$. We thus know that $M(c, s) =$ true and $c \subseteq c'$. By representativity, $M(c', s) =$ true. Since $s \in c''$, we also know that $M(c', c'') =$ true, which contradicts the assumption that C_2 is a valid clustering. Hence, there can be no clustering result that contains more clusters than C_1.

Proposition 2.14 *Algorithm 7 has a complexity of $O(|R|^2)$ and is optimal in the number of record comparisons in the worst case.*

Proof. We first count the maximal number of record comparisons by Algorithm 7. For each cluster c removed by C, at most $|C'|$ comparisons are conducted. For each merge, we perform an additional $|C'|-1$ comparisons. Hence, in the worst case, no records match until C contains one record and C' contains $|R|-1$ records (we perform $\frac{|R|\times|R|-1}{2}$ comparisons). If a merge occurs at the very last comparison, we can perform $|R|-2$ additional comparisons. If the next merge occurs at the very last comparison, we can perform an additional $|R|-3$ comparisons and so on. Hence, the maximum number of comparisons is $\frac{|R|\times(|R|-1)}{2} + (|R|-2) + (|R|-3) + \ldots + 1 = (|R|+1)^2$.

We now prove that any exhaustive algorithm will do the same number of comparisons in the worst case. We construct "adversarial" match and merge functions that behave as follows. The match function returns true only after it was asked to compare all pairwise combinations of the records in R (this process requires $\frac{|R|\times(|R|-1)}{2}$ comparisons). For the two matching records, the merge function creates a new record that forces the match function to wait for the record to be compared against all the remaining records (thus performing $|R|-2$ additional comparisons) before declaring a match. The merge function will again create a new record that forces the match function to perform $|R|-3$ more comparisons and so on. Hence, the total number of comparisons is $\frac{|R|\times(|R|-1)}{2} + (|R|-2) + (|R|-3) + \ldots + 1 = (|R|+1)^2$ as well.

Representativity and Negative Representativity We now consider the case where M and μ satisfy both representativity and negative representativity. In addition to assuming no negative evidence when merging two records as in Section 2.3, we also assume no positive evidence where two records r and s that do not match will later on turn out to be the same entity. Algorithm 8 computes the database leakage by clustering the matching records (Steps 2–10) and computing the database leakage for each cluster (Steps 12–15). For example, suppose we have the database $R = \{r, s, t\}$ and that only r and s match. We iterate the records in R and compare them to the records in S. Since there are no records in S yet, we insert r into S (Step 10). We then read s and compare it with the records in S. Since r and s match, we merge s with r into $\mu(r, s)$ (Steps 6–7). Next, we start comparing t with the records in S. Since t does not match with $\mu(r, s)$, we add t as a record in S. We now start measuring the record leakage of each record in S. In our example, we compute $L_r(p, \mu(r, s))$ and $L_r(p, t)$ and return the maximum value as the database leakage. The maximum dipping result r_q is $\mu(r, s)$ if $L_r(p, \mu(r, s)) \geq L_r(p, t)$ and t otherwise.

We prove that Algorithm 8 correctly returns the database leakage of R given that M and μ satisfy representativity and negative representativity.

Proposition 2.15 *If M and μ satisfy representativity and negative representativity, Algorithm 8 returns the database leakage of R.*

Proof. We show that after Steps 2–10, each record $s \in S$ is the unique dipping result of any record $r \in R$ that was merged to create s. Suppose that s was a result of merging the sequence of records $[r_1, \ldots, r_m]$, and r is some record r_i within the sequence. Say we construct the new sequence $[r_i, r_{j_1}, \ldots r_{j_{m-1}}]$ where r_i is the first record and the other records are listed in any order. All the records $r_{j_1}, \ldots, r_{j_{m-1}}$ match with r_i by negative representativity (otherwise, we would never have produced s in the first place). Also, any record r_{j_k} $(k = 1 \ldots m-1)$ matches with $\mu(\{r_i, r_{j_1}, \ldots, r_{j_{k-1}}\})$ by representativity. Hence, the set $\{r_{j_1}, \ldots r_{j_{m-1}}\}$

Algorithm 8: Database Leakage Algorithm assuming Representativity and Negative Representativity

```
input  : the reference $p$, a match function $M$, a merge function $\mu$, and a set of records $R$
output: the maximum dipping result $r_q$ and database leakage $L_d$
1  $S \leftarrow \{\}$;
2  for $r \in R$ do
3      match $\leftarrow$ false;
4      for $r' \in S$ do
5          if $M(r, r')$ = true then
6              $S \leftarrow S - \{r'\}$;
7              $S \leftarrow S \cup \{\mu(r, r')\}$;
8              match $\leftarrow$ true;
9      if match = false then
10         $S \leftarrow S \cup \{r\}$;
11 $L_d \leftarrow 0$;
12 for $r' \in S$ do
13     if $L_d < L_r(p, r')$ then
14         $L_d \leftarrow L_r(p, r')$;
15         $r_q \leftarrow r'$;
16 return $(r_q, L_d)$;
```

is an answer set for r_i satisfying Definition 2.1. We now show that s does not match with any other record t in $R - \{r_1, \ldots, r_m\}$. If s matches with t, then t matches with every record in $\{r_1, \ldots, r_m\}$ by negative representativity. Consequently, t would have merged into s in Steps 5–8 at some point by representativity. Hence, s is a (unique) dipping result satisfying Definition 2.2. The record leakage $L_r(p, s)$ is the query leakage of r, and the maximum value of all record leakages is the database leakage of R.

The complexity of Algorithm 8 is $O(|R|^2)$ because of the double for-loop in Steps 2–10. However, given more details of the match and merge functions, we can improve the complexity of Algorithm 8. For instance, if we use M_c (see Section 2.2) as the match function, then we can replace the inner for loop with a hash table lookup for the key-set value of r against all the key-set values of the records in S. As a result, the double-loop in Steps 2–10 can reduce to a single scan of records, and the entire algorithm complexity reduces to $O(|R|)$.

2.4 Scaling the Leakage Computation

We cover techniques that can further scale the measurement of the database leakage. The first technique is to run the database leakage algorithms on a smaller candidate set of records that are likely to be related to Alice. The second technique is to produce a bound of the database leakage using faster algorithms.

Using Candidate Sets Until now, we have proposed algorithms that propose the exact database leakage of R. As a result, if the match and merge functions do not satisfy certain properties, then measuring the leakage can be an expensive process. However, one may be satisfied with an approximate result of the database leakage with more efficient computation of leakage. Running the database leakage algorithms on a candidate set of records in R instead of the entire database can significantly improve the runtime with the cost of returning approximate results. One way is to first collect all the records that are at least distantly related to Alice using an efficient method. The exhaustive database leakage algorithm can then run on the candidate set, resulting in much faster computation of the database leakage than computing leakage on the entire database. For example, one could run the database leakage algorithm on a search engine result of Alice instead of on the entire Web. The downside of using a candidate set is that one may miss matching records of Alice outside the candidate set.

Computing Bounds of the Leakage We propose an efficient method for computing a bound for L_d when a general match function M and the union merge function μ_u (see Section 2.2) are used by Eve. In general, since neither representativity or negative representativity hold, we cannot use Algorithms 6 or 8 to compute the database leakage. However, we can efficiently provide (hopefully tight) upper and lower bounds of the database leakage.

We attempt to compute bounds for the database leakage when the record linkage function L_r is the F_1 metric. (Note that the approximation techniques depend on the definition of L_r.) We first show the following properties of upper and lower bounds for the precision and recall for any dipping result.

Proposition 2.16 *Given a reference p, the database R, and any dipping result r_q, the following relations hold:*

- $min_{r\in R}\frac{Pr(r)}{|R|} \leq Pr(r_q) \leq min\{1, \Sigma_{r\in R}Pr(r)\}$
- $max_{r\in R}Re(r) \leq Re(r_q)$

Proof. Suppose that r_q is the result of merging the records in the set S (i.e., $\mu(S) = r_q$).

We prove that $min_{r\in R}\frac{Pr(r)}{|R|} \leq Pr(r_q)$. It is sufficient to show that $min_{r\in S}\frac{Pr(r)}{|S|} \leq Pr(r_q)$ because $min_{r\in R}\frac{Pr(r)}{|R|} \leq min_{r\in S}\frac{Pr(r)}{|S|}$. Let $S = \{r_1, \ldots, r_{|S|}\}$ and for each $r_i \in S$, X_i is the number of "positive" attributes (which are in both r and p) and Y_i is the number of "negative" attributes (which are in r, but not in p). In the worst case for the precision of $\mu(S)$, the numerator does not increase from $max_{r_i\in S}|X_i|$, and the sets of attributes $X_1 \cup Y_1, \ldots, X_{|S|} \cup Y_{|S|}$ are all distinct. Hence $Pr(r_q) \geq \frac{max_{r_i\in S}|X_i|}{\Sigma_{i=1\ldots|S|}(|X_i|+|Y_i|)}$. Also, $\frac{max_{r_i\in S}|X_i|}{\Sigma_{i=1\ldots|S|}(|X_i|+|Y_i|)} \geq \frac{max_{r_i\in S}|X_i|}{|S|\times max_{r_i\in S}(|X_i|+|Y_i|)} \geq min_{r_i\in S}\frac{|X_i|}{|S|\times(|X_i|+|Y_i|)} = min_{r\in S}\frac{Pr(r)}{|S|}$.

Next, we prove that $Pr(r_q) \leq min\{1, \Sigma_{r\in R}Pr(r)\}$. We first show that $Pr(r_q) \leq \Sigma_{r\in S}Pr(r)$. Again, let $S = \{r_1, \ldots, r_{|S|}\}$ and for each $r_i \in S$, X_i is the number of "positive" attributes (which are in both r and p) and Y_i is the number of "negative" attributes (which are in r, but not in p). In the best case for the precision of $\mu(S)$, the denominator does not increase from $max_{r_i\in S}(|X_i| + |Y_i|)$, and the sets of positive attributes $X_1, \ldots, X_{|S|}$ are all distinct. Hence, $Pr(r_q) \leq \frac{\Sigma_{i=1\ldots|S|}|X_i|}{max_{r_i\in S}(|X_i|+|Y_i|)} \leq \Sigma_{i=1\ldots|S|}\frac{|X_i|}{|X_i|+|Y_i|} = \Sigma_{r\in S}Pr(r)$. It is straightforward to show that $\Sigma_{r\in S}Pr(r) \leq \Sigma_{r\in R}Pr(r)$. Since the precision does not exceed 1, we can say that $Pr(r_q) \leq min\{1, \Sigma_{r\in R}Pr(r)\}$.

The second property can be easily shown by observing that $Re(\mu(S)) = \frac{|(\bigcup_{r\in S} r)\cap p|}{|p|}$ is larger or equal to $Re(r') = \frac{|r'\cap p|}{|p|}$ for any $r' \in S$.

We can thus compute an upper bound of precision, a lower bound of precision, and a lower bound of recall for any dipping result by simply scanning all the records in R and computing the values $min_{r\in R}\frac{Pr(r)}{|R|}$, $min\{1, \Sigma_{r\in R}Pr(r)\}$, and $max_{r\in S}Re(r)$, respectively. A potential problem of estimating the precision is that the bound may not be tight, especially when R is large. A remedy is to compute the estimates on a small candidate set R' (where $|R'| \ll |R|$) assuming that the candidate set contains all the records that have a positive precision.

For example, suppose we have the database $R = \{r = \{\langle A, a'\rangle, \langle B, b\rangle, \langle C, c\rangle\}, s = \{\langle A, a\rangle, \langle B, b'\rangle, \langle C, c\rangle\}, t = \{\langle A, a\rangle, \langle B, b\rangle, \langle C, c'\rangle\}\}$ and the reference $p = \{\langle A, a\rangle, \langle B, b\rangle, \langle C, c\rangle\}$. Suppose the match function M only matches records that have exactly one identical attribute. Hence, the three records in R match with each other, but no merged record matches with the remaining record in R. The database leakage is the maximum value of the query leakage of r, s, and t. A dipping result of r is $r_q = \mu(r, s) = \{\langle A, a\rangle, \langle B, b\rangle, \langle B, b'\rangle, \langle C, c\rangle, \langle C, c'\rangle\}$, so the precision and recall of r_q are $\frac{3}{5}$ and 1, respectively. The query leakage is then $\frac{2\times\frac{3}{5}\times 1}{\frac{3}{5}+1} = \frac{3}{4}$. We can show that the query leakages of s or t also have the same value. Thus, the database leakage is $\frac{3}{4}$. We now compute the lower bound of the precision as $min_{r\in R}\frac{Pr(r)}{|R|} = \frac{2}{3\times 3} = \frac{2}{9}$. The upper bound of the precision is $min\{1, \Sigma_{r \in R} Pr(r)\} = min\{1, \frac{2}{3} \times 3\} = 1$. Finally, the lower bound of the recall is $max_{r \in S} Re(r) = \frac{2}{3}$.

So far we have computed upper and lower bounds of precision and a lower bound of recall. We now compute the upper bound of recall by modifying M to satisfy representativity (combined with μ_u). Suppose that $B(r)$ returns the records in R that were merged to produce r. We then extend M to M_r where $M_r(r, s)$ $= M(r, s) \vee \bigvee_{r' \in B(r), s' \in B(s)} M(r', s')$. That is, in addition to the original match function, we also check if any of the base records of r and s match. If so, we consider r and s to match even if $M(r, s)$ does not return true. We can easily see that M_r now satisfies representativity. Also, the recall of the dipping result r_q produced by M_r and μ_u is higher than r_q produced by M and μ because M_r matches any pair of records that match according to M. We modify Algorithm 6 to return the database recall instead of the database leakage by modifying Step 4 of Algorithm 5 (which is called by Algorithm 6) to "$L_q \leftarrow Re(p, r_q)$;". The final L_d value then gives the maximum recall among all the dipping results of records in R. The recall is an upper bound for the recall value of any dipping result.

Continuing our example above, we now modify M to match any merged record with the remaining record in R. We then run the modified Algorithm 6, which produces $Re(p, \mu(\{r, s, t\})) = 1$ as the upper bound of recall because the dipping result of any record $r' \in R$ is always $\mu(\{r, s, t\})$.

Combining the lower bounds of the precision and recall into a record leakage L_r^{min} and the upper bounds to the record leakage L_r^{max}, we now have a bound $[L_r^{min}, L_r^{max}]$ for the database leakage L_d. In our example, we get the range $[\frac{2 \times \frac{2}{9} \times \frac{2}{3}}{\frac{2}{9} + \frac{2}{3}}, \frac{2 \times 1 \times 1}{1+1}] = [\frac{1}{3}, 1]$ while the actual database leakage is $\frac{3}{4}$.

3 Uncertain Data

As we argued in the introduction, data confidence plays an important role in leakage. For instance, Eve "knows more about Alice" if she is absolutely sure Alice is 50 years old (correct value), as opposed to thinking she might be 50 years old with say 30% confidence, or thinking Alice is either 30 or 50 years old. To capture this intuition, we extend our model to include uncertain data values. Note that there are many ways to model data uncertainty, and our goal here is not to use the most sophisticated model possible. Rather, our goal is to pick a simple uncertainty model that is sufficient for us to study the interaction between uncertainty and information leakage.

Thus, in our extended model, each record r in R consists of a set of attributes, and each attribute contains a label, a value, and a confidence (from 0 to 1) that captures the uncertainty of the attribute (from Eve's point of view). Any attribute that does not exist in r is assumed to have a confidence of 0. As an example, the following record may represent Alice:

$$r = \{\langle \text{name}, \text{Alice}, 1\rangle, \langle \text{age}, 20, 0.5\rangle, \langle \text{age}, 30, 0.4\rangle, \langle \text{zip}, 94305, 0.3\rangle\}$$

That is, Eve is certain about Alice's name and age, but is only 50% confident about Alice being 30 years old, 40% confident in Alice being 30 years old, and 30% confident about Alice's zip code 94305. For each attribute $a \in r$, we can access a's label $a.lab$, a single value $a.val$, and confidence $a.cnf$. We also use the notation $r(l, v)$ to access the confidence of an attribute that has the label l and value v. (If no such attribute exists, then the confidence 0 is returned.) In Figure 3, the outdated record r_3 of Alice can be viewed to have a lower confidence than the up-to-date record r_2. We assume that attributes in the reference p always have a confidence of 1. We require that no two attributes in the same record can have the same label and value pair.

The confidences within the same record are independent with each other and reflect "alternate worlds" on Eve's belief of the correct information of Alice. For example, if we have $r = \{\langle$name, Alice, 1$\rangle$, $\langle$age, 20, 0.5$\rangle$, $\langle$phone, 123, 0.5$\rangle\}$, then there are four possible alternate worlds of r with equal probability: $\{\langle$name, Alice$\rangle\}$, $\{\langle$name, Alice$\rangle$, $\langle$age, 20$\rangle\}$, $\{\langle$name, Alice$\rangle$, $\langle$phone, 123$\rangle\}$, and $\{\langle$name, Alice$\rangle$, $\langle$age, 20$\rangle$, $\langle$phone, 123$\rangle\}$

Note that we are using a single number (in the range of 0 to 1) to represent the confidence of an attribute. More complex confidence models are possible. For example, a confidence could include lineage information explaining how the confidence was derived. Or we may have two separate sets of confidences where one set reflects Alice's view on the records while the other set reflects Eve's view. However, we believe that single numbers are a reasonable way to represent confidences in the dipping process.

3.1 Record Leakage

We now extend our record leakage metric in Section 2.1 to use confidences. We first define the notation $IN(r, s)$ for two records r and s as $\{a \in r \mid \exists a' \in s \text{ s.t. } a.lab = a'.lab \wedge a.val = a'.val\}$. That is, $IN(r, s)$ denotes the attributes of r whose label-value pairs exist in s.

The precision Pr or a record r against the reference p is defined as $\frac{\Sigma_{t \in IN(r,p)} t.cnf}{\Sigma_{t \in r} t.cnf}$. Compared to the previous definition in Section 2.1, we now sum the confidences of the attributes in r whose label-value pairs are also in p and divide by the total confidence of r. For example, if $r = \{\langle$name, Alice, 1$\rangle$, $\langle$age, 20, 0.5$\rangle$, $\langle$age, 30, 0.4$\rangle$, $\langle$phone, 111, 0.3$\rangle\}$ and $p = \{\langle$name, Alice, 1$\rangle$, $\langle$age, 20, 1$\rangle$, $\langle$phone, 123, 1$\rangle$, $\langle$zip, 94305, 1$\rangle\}$, then $Pr(r, p) = \frac{1+0.5}{1+0.5+0.4+0.3} \approx 0.68$. Notice that Pr penalizes higher confidence on incorrect attribute values. For instance, if the confidence of r's phone (which has an incorrect value 111) was 1, then the precision of r would decrease to $\frac{1+0.5}{1+0.5+0.4+1} \approx 0.52$.

The recall Re of r against p is defined as $\frac{\Sigma_{t \in (r \cap p)} t.cnf}{\Sigma_{t \in p} t.cnf}$. This time, we divide the sum of confidences of the attributes in r whose label-value pairs are also in p by the total confidence of p. Continuing our example above, we have $Re(r, p) = \frac{1+0.5}{1+1+1+1} \approx 0.38$. Compared to Pr, Re is more sensitive to low confidences of correct attributes in r. For instance, if the confidence of r's age being 20 was 0.1, then the recall of r would decrease to $\frac{1+0.1}{1+1+1+1} \approx 0.275$.

Finally, we define the record leakage L_r as the F_1 metric $\frac{2 \times Pr \times Re}{Pr + Re}$. In our example, $L_r(r, p) = F_1(r, p) = \frac{2 \times 0.68 \times 0.38}{0.68 + 0.38} \approx 0.49$. In general, L_r can be defined as any metric.

3.2 Matching and Merging with Confidences

In this section, we discuss how the matching and merging of records can be done using confidences. For instance, even if the two records r= $\{\langle$name, Alice, 0.1$\rangle$, $\langle$age, 20, 0.2$\rangle\}$ and s= $\{\langle$name, Alice, 0.2$\rangle$, $\langle$phone, 123, 0.3$\rangle\}$ have the same name Alice, the confidences (0.1 and 0.2) may be too small for r and s to represent the same real-world entity.

There are various ways to accommodate confidence in the match function ranging from simply ignoring confidences to using confidences when comparing each pair of attributes. In our model, we extend any match function M that only operates on attributes without confidences to have an additional "trigger condition" T that takes two records r and s and returns true if the confidences in the records are high enough to allow a match between records and false otherwise. Hence, extending M with T is equivalent to using the match function $M \wedge T$. For example, say we have two records $r = \{\langle A, a, 0.9\rangle\}$ and $s = \{\langle A, a, 0.1\rangle\}$. The match function M may say that r and s match because they have the same value a for the attribute A. However, the trigger condition T may say that s's confidence 0.1 is too low for the two records to match. Hence, $M \wedge T$ = true $\wedge$ false = false, and the two records do not match according to the extended match function.

We define the notion of domination in confidence for two records.

Definition 3.1 *A record s* dominates *another record r in confidence (denoted as $r \leq_C s$) if $\forall a \in r, \exists a' \in s$ where $a.lab = a'.lab$, $a.val = a'.val$, and $a.cnf \leq a'.cnf$.*

For example, if $r = \{\langle A, a, 0.2\rangle\}$ and $s = \{\langle A, a, 0.2\rangle, \langle B, b, 0.1\rangle\}$, then $r \leq_C s$, but $s \not\leq_C r$. Again, notice that we are implicitly assuming that any attribute that does not exist in a record has a confidence of 0.

We now present a desirable property for a match function extended with a trigger condition called monotonicity, which states that higher confidences only makes it more likely for two records to match.

Definition 3.2 *A match function M extended with a trigger condition is* monotonic *in confidence if for any records r, s, r', s' such that $M(r, s)$ = true, $r \leq_C r'$, and $s \leq_C s'$, then $M(r', s')$ = true.*

As an illustration, we extend the match function M_e (see Section 2.2) with a trigger condition T_t into a monotonic match function M_{et}. The trigger condition T_t checks if all the key-set attributes of r and s exceed a given threshold t. That is $T_t = \bigwedge_{l \in k}(\bigwedge_{v \in r(l)} r(l,v) \geq t \wedge \bigwedge_{v \in s(l)} s(l,v) \geq t)$ given the key set k. For example, suppose that $r = \{\langle \text{A, a, 0.1} \rangle\}$, $s = \{\langle \text{A, a, 0.2} \rangle\}$, and we use the match function M_e (see Section 2.2) with the key set $\{A\}$. However, if $t = 0.15$, then r and s do not match because r does not have a sufficiently high confidence for its key-set attribute $\langle \text{A, a, 0.1} \rangle$ (i.e., $r(A,a) \leq t$). The trigger condition T_t is monotonic because if $T_t(r,s) = \text{true}$, $r \leq_C r'$, and $s \leq_C s'$, then $T_t(r', s') = \text{true}$ because the confidences of the key-set attributes of r' and s' (being larger or equal to those of r and s) are surely larger or equal to the threshold t.

The merging of records must also deal with confidences. We extend any merge function μ with a confidence function f that computes confidences for the attributes in a merged record. That is, given two records r and s that merge, $f(r,s,l,v)$ returns the confidence for the attribute $a \in \mu(r,s)$ where $a.lab = l$ and $a.val = v$. Hence, the extended merge function μ' now works in two phases.

In phase 1, we remove the confidences in r and s and merge the records using μ. Formally, we compute $\mu(r', s')$ where $r' = \{\langle a.lab, a.val \rangle | a \in r\}$ and $s' = \{\langle a.lab, a.val \rangle | a \in s\}$. For example, suppose that we merge two records $r = \{\langle A, a, 1\rangle, \langle B, b, 0.2\rangle\}$ and $s = \{\langle A, a', 1\rangle, \langle B, b, 0.3\rangle\}$ using an extension of μ_u (see Section 2.2). We first compute $r' = \{\langle A, a\rangle, \langle B, b\rangle\}$ and $s' = \{\langle A, a'\rangle, \langle B, b\rangle\}$ and then produce $\mu_u(r', s') = \{\langle A, a\rangle, \langle A, a'\rangle, \langle B, b\rangle\}$.

In phase 2, we iterate through each attribute $a \in \mu(r', s')$ and add the confidence $a.cnf = f(r, s, a.lab, a.val)$ to a. In our example, suppose that we use the confidence function $f(r,s,l,v) = max\{r(l,v), s(l,v)\}$. Then $\mu_u(r', s') = \{\langle A, a, max\{1, 0\}\rangle, \langle A, a', max\{0, 1\}\rangle, \langle B, b, max\{0.2, 0.3\}\rangle\} = \{\langle A, a, 1\rangle, \langle A, a', 1\rangle, \langle B, b, 0.3\rangle\}$.

We define the increasing property for a merge function extended with a confidence function, which states that confidences are always increasing as records merge.

Definition 3.3 *A merge function μ using a confidence function f is* increasing *if for any records r and s, $\forall a \in r \cup s$, for any $a' \in \mu(r,s)$ such that $a.lab = a'.lab$ and $a.val = a'.val$, $a.cnf \leq a'.cnf$.*

As an illustration, we extend the merge function μ_u (see Section 2.2) with a confidence function $f(r,s,l,v) = r(l,v) + s(l,v) - r(l,v) \times s(l,v)$ into an increasing merge function μ_{ui} (i.e., f assumes that the attributes from r and s are independent). For example, suppose that we merge the records $r = \{\langle A, a, 1\rangle, \langle B, b, 0.2\rangle\}$ and $s = \{\langle A, a', 1\rangle, \langle B, b, 0.3\rangle\}$ using μ_{ui}. In phase 1, we compute $\mu_u(\{\langle A, a\rangle, \langle B, b\rangle\}, \{\langle A, a'\rangle, \langle B, b\rangle\}) = \{\langle A, a\rangle, \langle A, a'\rangle, \langle B, b\rangle\}$. Next, in phase 2, we add the confidences as follows: $\mu_{ui} = \{\langle A, a, 1 + 0 - 1 \times 0\rangle, \langle A, a', 0 + 1 - 0 \times 1\rangle, \langle B, b, 0.2 + 0.3 - 0.2 \times 0.3\rangle\} = \{\langle A, a, 1\rangle, \langle A, a', 1\rangle, \langle B, b, 0.44\rangle\}$. The extended merge function μ_{ui} is increasing because the equation $c_1 + c_2 - c_1 \times c_2$ is larger or equal to c_1 or c_2. As another example for an increasing merge function, we define μ_{ux} that is identical to μ_{ui}, but uses the confidence function $f(r,s,l,v) = max\{r(l,v), s(l,v)\}$ (we illustrated μ_{ux} above). By definition, the confidences are always increasing as records merge.

We assume that the trigger condition T and confidence function f still preserve the commutativity and associativity properties of M and μ. In order to satisfy the properties, $T(r,s)$ should be the same as $T(s,r)$ for any records r and s. Also, the confidences in the merged record should not depend on the order of merging.

We now show that the monotonicity and increasing properties preserve representativity of unextended match and merge functions.

Proposition 3.4 *If M and μ are representative without confidences, and their extensions M' and μ' are monotonic and increasing, then M' and μ' are also representative.*

Proof. Suppose that $t = \mu'(r,s)$. Also say there is a record u such that $M'(r,u) = \text{true}$. Since μ' is increasing, we know that $r \leq_C u$. By monotonicity of M and representativity of M' and μ', we know that $M'(t,u) = M(t,u) \wedge T(t,u) = \text{true}$, satisfying the representativity condition for M' and μ'.

We also show that the monotonicity and increasing properties preserve the negative representativity property of unextended match and merge functions.

Proposition 3.5 *If M and μ are negative representative without confidences, and their extensions M' and μ' are monotonic and increasing, then M' and μ' are also negative representative.*

Proof. If M and μ satisfy negative representativity, then any extended match and merge functions M' and μ' also satisfy negative representativity because M guarantees that any two records r and s that do not match will never match even if r or s merge with other records.

We now show that M_{et} and μ_{ui} (or μ_{ux}) satisfy commutativity, associativity, and representativity, but not negative representativity.

Proposition 3.6 *The match and merge functions M_{et} and μ_{ui} (or μ_{ux}) satisfy commutativity, associativity, representativity, but not negative representativity.*

Proof. The match and merge functions M_{et} and μ_{ui} satisfy commutativity and associativity because comparing the key-set attribute confidence with a threshold t does not depend on the order of records compared, and the confidence function f used in μ_{ui} does not depend on the order of records merged. (We can also show that M_{em} and μ_{ux} satisfy the two properties.) It directly follows from Propositions 2.5 and 3.4 that M_{et} and μ_{ui} (or μ_{ux}) satisfy representativity. Finally, it follows from Propositions 2.5 and 3.5 that M_{et} and μ_{ui} (or μ_{ux}) do not satisfy negative representativity.

If the conditions of Proposition 3.4 are satisfied, the dipping result that satisfies Definition 2.2 is unique by Proposition 2.8. However, if the extended match and merge functions are not monotonic and increasing, a dipping result satisfying Definition 2.2 may not be unique. For example, suppose we use the match function M_{et} with the key set $\{A\}$ and the threshold $t = 0.5$. However, say we extend the merge function μ_u into μ'_u using the confidence function $f(r, s, l, v) = r(l, v) + s(l, v) - r(l, v) \times s(l, v) \times \Pi_{l \in k}(\Pi_{v \in r(l)} r(l, v) \times \Pi_{v \in s(l)} s(l, v))$. That is, in addition to using the equation $c_1 + c_2 - c_1 \times c_2$ for combining two confidences c_1 and c_2, we also multiply each confidence in the merged record by the product of the key-set attribute confidences of r and s. Now say that we are given the database $R = \{r_1 = \{\langle A, a, 0.5\rangle, \langle B, b_1, 1\rangle\}, r_2 = \{\langle A, a, 0.5\rangle, \langle B, b_2, 1\rangle\}\}$, and the query $q = \{\langle A, a, 0.5\rangle\}$. If q is compared with r_1 first, then the merged result $\mu'_u(q, r_1) = \{\langle A, a, 0.5 \times 0.5 \times (0.5 + 0.5 - 0.5 \times 0.5)\rangle, \langle B, b_1, 0.5 \times 0.5 \times (1 + 0 - 1 \times 0)\rangle\} = \{\langle A, a, 0.1875\rangle, \langle B, b_1, 0.25\rangle\}$. Notice that $\mu'_u(q, r_1)$ no longer matches with r_2 due to the low confidence $0.1875 < T = 0.5$. Hence, the final result $r_q = \mu'_u(q, r_1)$. However, if q is compared with r_2 first, we will get the different result $r_q = \mu'_u(q, r_2)$.

3.3 Query and Database Leakage Algorithms

We can reuse Algorithms 2 and 3 to compute the dipping result r_q and query leakage L_q of q. If M and μ only satisfy representativity, Algorithm 2 returns a valid and unique dipping result for any extensions of M and μ. If the unextended match and merge functions M and μ satisfy negative representativity (see Section 2.2), then by Proposition 3.5 any extended match and merge functions M' and μ' also satisfy negative representativity. Hence, Algorithm 3 returns a valid and unique dipping result for any extensions of M and μ using a trigger condition and confidence function.

Furthermore, the database leakage algorithms can also be reused. If the extended M and μ do not satisfy any property, then Algorithm 4 is the default (but inefficient) algorithm that returns the maximum dipping result and database leakage of q. If the representativity property is satisfied, then we can use Algorithm 6 to compute the maximum dipping result and database leakage. If both representativity and negative representativity are satisfied, we use Algorithm 8.

4 Releasing Disinformation

Our framework can be used to answer a variety of questions as we illustrate from this section on. As we use our framework, it is important to keep in mind "who knows what". In particular, if Alice is studying leakage of her information, she needs to make assumptions as to what her adversary Eve knows (database

R) and how she operates (the match and merge functions, and dipping algorithm Eve uses). These types of assumptions are common in privacy work, where one must guess the sophistication and compute power of an adversary. On the other hand, if Eve is studying leakage she will not have Alice's reference information p. However, she may use a "training data set" for known individuals in order to tune her dipping algorithms, or say estimate how much she really knows about Alice.

In this section, we propose disinformation as a key technique for managing information leakage. Given previously released information R, a match function M, and a merge function μ, Alice may want to release either a single record or multiple records that can decrease the query or database leakage. We call records that are used to decrease the database leakage *disinformation* records. Of course, Alice can reduce the query or database leakage by releasing arbitrarily large disinformation. However, disinformation itself has a cost. For instance, adding a new social network profile would require the cost for registering information. As another example, longer records could require more cost and effort to construct. For now, we assume that the cost to create and add r is the number of attributes in r. We use $C(r)$ to denote the entire cost of creating r.

We define the problem of minimizing the database leakage using one or more disinformation records. Given a set of disinformation records S and a maximum budget of C_{max}, the optimal disinformation problem can be stated as the minimization function presented below:

$$\begin{array}{l} \text{minimize} \quad L_d(p, M, \mu, R \cup S) \\ \text{subject to } \Sigma_{r \in S} C(r) \leq C_{max} \end{array}$$

The problem of minimizing the query leakage can also be stated by replacing L_d by L_q in the above formula. The set of records S that minimizes the database leakage within our cost budget C_{max} is called "optimal" disinformation. The problem of releasing a single record (i.e., $|S| = 1$) is a special case and is covered in Section 4.1 while the general problem of releasing multiple records is covered in Section 4.2.

Alternatively, one might want to maximize the database leakage with the released information. This dual problem arises when Alice might want to expose her correct information as much as possible (e.g., when Alice is looking for a job, she might want her correct profile to be visible). We believe our disinformation techniques can also be used for this leakage maximization problem.

4.1 Single Record Release

We study the problem of releasing a single record for disinformation. The disinformation record r_d can reduce the database leakage in two ways. First, r_d can perform *self disinformation* by acting as disinformation itself and add its irrelevant information to the dipping result r_q with the maximum query leakage. For example, given the database $R = \{r, s, t\}$ and a reference p, suppose the database leakage is $L_r(p, \mu(r, s))$. Then r_d can be created to match with $\mu(r, s)$ and thus decrease the database leakage to $L_r(p, \mu(\{r, s, r_d\}))$. Second, r_d can perform *linkage disinformation* by reducing the database leakage by linking irrelevant records in R to r_q. For example, say that t contains totally irrelevant information to the target. If r_d can be made to match with both $\mu(r, s)$ and t, then the database leakage could decrease to $L_r(p, \mu(\{r, s, t, r_d\}))$ because of r_d. Of course, r_d can also use both self and linkage disinformation.

When creating a record, we use a user-defined function called *Create*(S, L) that creates a new minimal record that has a size less or equal to L and is guaranteed to match all the records in the set S. If there is no record r such that $|r| \leq L$ and all records in S match with r, the *Create* function returns the empty record $\{\}$. A reasonable assumption is that the size of the record produced by *Create* is proportional to $|S|$ when $L > |S|$. We also assume a function called *Add*(r) that appends a new attribute to r. The new attribute should be "incorrect but believable" (i.e., bogus) information. We assume that if two records r and s match, they will still match even if *Add* appends bogus attributes to either r or s. (Notice that this property is similar to the representativity property for match and merge functions.) The *Create* function is assumed to have a time complexity of $O(|S|)$ while the *Add* function $O(|r|)$.

As an example, suppose that we use the match function M_c and the key set k. If $|k| \leq L$, *Create*(r, L) returns a record s that has the exact same key-set values as r. Otherwise, *Create* returns $\{\}$. The same construction works for M_e. As another example, suppose we use the match function M_{em}. Then we first choose the key set k_{min} with the minimum size among all the key sets in $\mathcal{K}$. Then if $|k_{min}| \leq L$, $Create(r, L)$ returns a record s that has the exact same key-set values of k_{min} as r. Otherwise, $Create$ returns $\{\}$.

There are many details we can consider when adding bogus attributes to r_d using the function *Add*. We list a few examples below.

- Natural Attribute: a bogus attribute should be natural enough for the adversary to think it real. For instance, an attribute ⟨XXX, YYY, 1⟩ does not seem real while ⟨name, Alicia, 1⟩ may be more convincing.
- Relevant Attribute: a new record should be relevant to its format in the database. For instance, if we want to post a new record in the form of a homepage, then the contents of the attributes should be relevant to homepages.
- Diverse Attribute: the new information we add for Alice must sufficiently differ from Alice's original information in order to hide sensitive data. For instance, if we want to hide Alice's exact address, we may want to add a new address that is not too close to Alice's real address.
- Sophisticated Cost Function: the cost function for creating a record should depend on many factors of the attributes including their contents, ordering, size, and difficulty to look realistic to Eve.

In the following sections, we show various algorithms that return a single record for disinformation. As more properties are satisfied by the match and merge functions, the more efficient the disinformation algorithms become.

No Properties We first assume that the match and merge functions do not satisfy any properties. A brute-force algorithm for the single record disinformation problem is shown in Algorithm 9. Among all records of size C_{max}, we choose the one that gives the minimum database leakage. Obviously, the brute-force algorithm is very inefficient and is only stated to give a baseline implementation.

Algorithm 9: Brute Force Single-Record Disinformation Algorithm

```
input : the budget $C_{max}$, the reference $p$, a match function $M$, a merge function $\mu$, and a set of records $R$
output: a disinformation record $r_d$
1 $L_d \leftarrow 0$;
2 $r_d \leftarrow \{\}$;
3 for every possible record $r$ constructed with the functions Create and Add where $|r| \leq C_{max}$ do
4     $(r_q, L'_d) \leftarrow$ Algorithm 4$(p, M, \mu, R \cup \{r\})$;
5     if $L'_d < L_d$ then
6         $L_d \leftarrow L'_d$;
7         $r_d \leftarrow r$;
8 return $r_d$;
```

One can prove that the single-record disinformation problem is NP-hard. In fact, even minimizing the query leakage turns out to be still NP-hard according to the following Proposition 4.1.

Proposition 4.1 *If M and μ do not satisfy any properties, the problem of minimizing the query leakage L_q with a single disinformation record r_d is NP-hard.*

Proof. We reduce the set-cover decision problem (which is known to be NP-complete) to the single-record disinformation problem for the query leakage where M and μ do not satisfy any properties. Given a universe U, a family F of subsets of U, and the maximum number of sets k, the set-cover decision problem returns Yes if there exists k or fewer subsets in F that form U when unioned and No otherwise. We can construct a corresponding instance for the dipping problem. For each set $s_i \in F$, we create a record $r_i = \{\langle A, i\rangle, \langle B, s_i\rangle\}$

where A and B are new attribute labels while i and s_i are their attribute values. We set the query $q = \{\langle C, c\rangle\}$ where C is a new attribute label, and c is a distinguished value different from all others.

The match function M is defined as follows: two records r and s match if either one of the following two conditions hold.

1. Both r and s have the same attribute $\langle C, c\rangle$, and none of the records have a (non-empty) B value.
2. Both r and s have the same attribute $\langle A, i\rangle$ (for some i), and at least one record contains the attribute $\langle C, c\rangle$.

Before defining the merge function, we define the new notation $BSet(r)$, which refers to the B value of r. As we shall see in our definition of μ, we guarantee that any record except for q contains exactly one B value, so there is no ambiguity when using the B value of a non-reference record. Using this notation, the merge function μ takes the records r and s and returns $\{\langle A, i\rangle | i \in r(A) \cup s(A)\} \cup \{\langle B, BSet(r) \cup BSet(s)\rangle\}$ $\cup$ $\{\langle C, v\rangle | v \in r(C) \cup s(C)\}$. For creating the disinformation record, we set C_{max} as $k + 1$. We define the reference p as $\{\langle B, s\rangle | s \in (2^U - \{U\})\}$ so that the information leakage is minimized only if the dipping result r_q has U as its B value.

We show that M and μ do not satisfy representativity using a counter example. Say that $r = \{\langle C, c\rangle\}$, $s = \{\langle B, s_1\rangle, \langle C, c\rangle\}$, and $t = \{\langle C, c\rangle\}$. Then although $M(r, t)$ = true (i.e., r and s share $\langle C, c\rangle$ and both records do not have a B attribute), $M(\mu(r, s), t)$ = false (because $\mu(r, s)$ now has a B attribute). We also show that M and μ do not satisfy negative representativity either using a counter example. This time, say r $= \{\langle A, 1\rangle\}$, $s = \{\langle A, 2\rangle, \langle C, c\rangle\}$, and $t = \{\langle A, 2\rangle\}$. Then although $M(r, t)$ = false (no A attributes are shared and no records contain $\langle C, c\rangle$), $M(\mu(r, s), t)$ = true (because $\mu(r, s)$ and t share $\langle A, 2\rangle$ and $\mu(r, s)$ contains $\langle C, c\rangle$).

The only way the dipping result r_q can have a B value of U is when r_d matches with q and also with r_i's whose B values form a set cover of U. (Notice that q does not match with any of the r_i records because r_i's do not contain $\langle C, c\rangle$ and q does not contain A attributes.) For r_d to match with q, r_d needs to contain the attribute $\langle C, c\rangle$ because q does not have an A attribute. Since r_d now contains $\langle C, c\rangle$, it can add (k or fewer) $\langle A, i\rangle$ attributes to match with r_i records. Notice that r_d cannot add a B attribute because then it can never match with q according to the first condition of the match function. Once we obtain a solution r_d and if $\bigcup_{r \in \{r \in R | r(A) \subseteq r_d(A)\}} BSet(r) = U$, we know that the dipping result r_q is $\{\langle C, c\rangle, \langle B, U\rangle, \ldots\}$ and thus the query leakage $L_q = 0$. Consequently, the set $\{BSet(r) | r \in R \wedge r(A) \subseteq r_d(A)\}$ forms a set cover of U with k or fewer sets. If $\bigcup_{r \in \{r \in R | r(A) \subseteq r_d(A)\}} BSet(r) \neq U$, we can conclude that there is no set cover that uses k or fewer sets.

Instead of computing the optimal disinformation record, we can use a greedy approximate algorithm to compute a reasonable solution. Heuristic Algorithm 10 does not assume any properties and returns a disinformation single record that uses both self and linkage disinformation. The idea of Algorithm 10 is to decide whether to use both linkage and self disinformation or self disinformation only. Suppose we have the database $R = \{r, s, t\}$ and that only r and s match with each other. Step 1 produces the maximum dipping result of R. That is, we find the dipping result of $r' \in R$ where $L_q(p, r', M, \mu, R)$ is the largest. Assuming that the leakage $L_r(p, \mu(r, s))$ is higher than $L_r(p, t)$, r_q in Step 1 returns $\mu(r, s)$. We then create a disinformation record r_1 that matches $\mu(r, s)$ (Step 2) and add as many OK possible bogus attributes (Steps 3–4). The smallest database leakage using self disinformation is L_{self} (Step 5). Next, we identify the dipping result r_{min} that would minimize the query leakage of r_q when combined (i.e., the query leakage of $\mu(r_q, r_{min})$ is minimal). We can derive r_{min} by evaluating $L_q(p, \mu(r_q, r'_q), M, \mu, R)$ for every dipping result r'_q that can be generated with a record in R, and then setting r_{min} to the r'_q with the smallest $L_q(p, \mu(r_q, r'_q), M, \mu, R)$ value. We create a disinformation record r_2 that matches with both r_q and r_{min} (Step 7). Again, we fill r_2 with as many OK possible bogus attributes (Steps 8-9) and derive the resulting database leakage L_{link} using r_2 (Step 10). Finally, if L_{self} is smaller than L_{link}, self disinformation without linkage disinformation is considered as the better disinformation strategy, and we return r_1 as the disinformation record (Step 12). Otherwise, we return r_2 (Step 14).

Obviously, Algorithm 10 does not return an optimal single-record disinformation. For example, linking more than two records (instead of performing just one linkage) may further reduce the database leakage.

Algorithm 10: Heuristic Single-Record Disinformation Algorithm without assuming Properties

```
    input  : the budget $C_{max}$, the reference $p$, a match function $M$, a merge function $\mu$, and a set of records $R$
    output: a disinformation record $r_d$
 1  $(r_q, L_d) \leftarrow$ Algorithm 4$(p, M, \mu, R)$;
 2  $r_1 \leftarrow Create(\{r_q\}, C_{max})$;
 3  while $r_1 \neq \{\} \wedge |r_1| < C_{max}$ do
 4      $Add(r_1)$;
 5  $(r_0, L_{self}) \leftarrow$ Algorithm 4$(p, M, \mu, R \cup \{r_1\})$;
 6  $r_{min} \leftarrow$ dipping result $r'_q$ for any record in $R$ where $L_q(p, \mu(r_q, r'_q), M, \mu, R)$ is minimal;
 7  $r_2 \leftarrow Create(\{r_q, r_{min}\}, C_{max})$;
 8  while $r_2 \neq \{\} \wedge |r_2| < C_{max}$ do
 9      $Add(r_2)$;
10  $(r_0, L_{link}) \leftarrow$ Algorithm 4$(p, M, \mu, R \cup \{r_2\})$;
11  if $L_{self} < L_{link}$ then
12      return $r_1$;
13  else
14      return $r_2$;
```

However, if most optimal disinformation results involve at most one linkage, then Algorithm 10 can return a reasonable result much faster than the brute-force algorithm.

Representativity Only For the case where M and μ satisfy representativity only, the single-record disinformation problem is still NP-hard. The complexity of the problem comes from the fact that the disinformation record r_d can link any records in R, and finding the best combination for minimizing the database leakage is not trivial.

Proposition 4.2 *If M and μ satisfy representativity, but not negative representativity, the problem of minimizing the database leakage L_d with a single disinformation record r_d is NP-hard.*

Proof. We reduce the set-cover decision problem (which is known to be NP-complete) to the single-record disinformation problem for the database leakage where M and μ satisfy representativity, but not negative representativity. Given a universe U, a family F of subsets of U, and the maximum number of sets k, the set-cover decision problem returns Yes if there exists k or fewer subsets in F that form U when unioned and No otherwise. We can construct a corresponding instance for the dipping problem. For each set $s_i \in F$, we create a record $r_i = \{\langle A, i\rangle, \langle B, s_i\rangle\}$ where A and B are new attribute labels while i and s_i are their attribute values. We set the query $q = \{\langle C, c\rangle\}$ where C is a new attribute label, and c is a distinguished value different from all others.

The match function M is defined as follows: two records r and s match if they have any common A attribute. That is, $M(r, s) = (r(A) \cap s(A) \neq \emptyset)$ (recall that the $r(A)$ notation returns the set of A values of r). Similar to the proof of Proposition 4.1, we define the notation $BSet(r)$, which refers to the B value of r. Notice that none of the r_i records match with each other. The merge function μ then takes the records r and s and returns $\{\langle A, i\rangle | i \in r(A) \cup s(A)\} \cup \{\langle B, BSet(r) \cup BSet(s)\rangle\}$. Notice that M and μ satisfy representativity, but not negative representativity. To see that representativity holds, suppose that $t = \mu(r, s)$ for any two records r and s, and that there exists some record u where $M(r, s) =$ true. By the definition of M, we know that r and u have at least one common A attribute. Also, since $\mu(r, s)$ simply unions the A attributes, t and u share at least one A attribute as well (i.e., $M(t, u) =$ true). However, M and μ do not satisfy negative representativity. For example, if $r = \{\langle A, 1\rangle\}$, $s = \{\langle A, 2\rangle\}$, and $t = \{\langle A, 2\rangle\}$, then although $M(r, t) =$ false, $M(\mu(r, s), t) =$ true.

For creating the disinformation record, we set C_{max} as $k+1$. We define the reference p as $\{\langle B, s\rangle | s \in (2^U - \{U\})\}$ so that the information leakage is minimized only if the dipping result r_q has U as its B value.

Once we obtain a solution r_d and if $\bigcup_{r \in \{r \in R | r(A) \subseteq r_d(A)\}} BSet(r) = U$, we know that the dipping result r_q is $\{\langle C, c\rangle, \langle B, U\rangle, \ldots\}$ and thus the query leakage $L_q = 0$. Consequently, the set $\{BSet(r) | r \in R \wedge r(A) \subseteq r_d(A)\}$ forms a set cover of U with k or fewer sets. If $\bigcup_{r \in \{r \in R | r(A) \subseteq r_d(A)\}} BSet(r) \neq U$, we can conclude that there is no set cover that uses k or fewer sets.

Using representativity, we can implement Algorithm 10 more efficiently. First, we can now use Algorithm 6 instead of Algorithm 4 for computing the maximum dipping result r_q in Step 1. Second, L_{self} in Step 5 can now be computed using Algorithm 6 as well. Third, Step 6 can be computed by iterating through each record r in R and running Algorithm 2 on $\mu(r_q, r'_q)$, and then taking the dipping result with the minimum query leakage. Fourth, Step 10 can also be computed using Algorithm 6. Finally, Steps 3–4 and 8–9 run in $O(C_{max})$ time. Since Algorithm 6 runs in $O(|R|^3)$ time and Algorithm 2 in $O(|R|^2)$ time, the total complexity of Algorithm 10 is $O(|R|^3 + C_{max})$.

Representativity and Negative Representativity We assume that M and μ satisfy both representativity and negative representativity. Algorithm 11 exploits the fact that r_d cannot connect any records r and s where $M(r, s) =$ false by negative representativity. Hence, we create r_d that matches with the dipping result r_q with the maximum query leakage. (By negative representativity, r_d cannot be made to match with any other record in R that does not form r_q.) For example, suppose that we have the database $R = \{r, s, t\}$ and that only r and s match. In Proposition 2.15 we showed that after Steps 2–8 of Algorithm 8, each record $s' \in S$ (S is a variable inside Algorithm 8) is the unique dipping result of any record $r \in R$ that was merged to create s'. Hence after Step 8 of Algorithm 8, we have $S = \{\mu(r, s), t\}$. Assuming that $L_r(p, \mu(r, s)) > L_r(p, t)$, r_q becomes $\mu(r, s)$ (Step 1 of Algorithm 11). We then create a record r_d that matches with r_q (Step 2 of Algorithm 11). If the *Create* function successfully generates r_d within the budget C_{max} (i.e., $r_d \neq \{\}$ and $|r_d| \leq C_{max}$), we add as many OK possible bogus attributes (Steps 3–4 of Algorithm 11). Finally, we return r_d.

Algorithm 11: Optimal Single-Record Disinformation Algorithm assuming Representativity and Negative Representativity

```
    input  : the budget $C_{max}$, the reference $p$, a match function $M$, a merge function $\mu$, and a set of records
             $R$
    output: a disinformation record $r_d$
  1 $(r_q, L_d) \leftarrow$ Algorithm 8$(p, M, \mu, R)$;
  2 $r_d \leftarrow Create(\{r_q\}, C_{max})$;
  3 while $|r_d| < C_{max}$ do
  4     $Add(r_d)$;
  5 return $r_d$;
```

Proposition 4.3 *If M and μ satisfy representativity and negative representativity, Algorithm 11 returns an optimal disinformation record.*

Proof. We first show that any disinformation record r_d produced cannot match with two different records r_i and r_j in S, which is produced in Steps 2–8 of Algorithm 8 during Step 1 of Algorithm 11. Suppose that r_d does match with both r_i and r_j. Say we use the notation $r.L$ to indicate the set of records in R merged to create r. Then r_d matches with all the records in $r_i.L$ as well as with the records in $r_j.L$ by negative representativity. However, for any record $r_1 \in r_i.L$, we know $M(r_1, r_j) =$ false by negative representativity (otherwise, $M(\mu(\mu(r_i.L - \{r_1\}), r_1), r_j) = M(r_i, r_j) =$ true, which is a contradiction). We thus know that $M(r_1, \mu(r_j, r_d))$ = false (by negative representativity) and consequently $M(r_i, \mu(r_j, r_d))$ = false (again by negative representativity). However, since $M(r_i, r_d)$ = true, $M(r_i, \mu(r_j, r_d))$ = true by representativity, contradicting our previous statement. Thus r_d can only match with exactly one record in S. Therefore, linkage disinformation is not feasible. So the best solution is to create an r_d that matches to r_q, the largest dipping result, and add bogus attributes to that result.

In Step 2, Algorithm 11 creates r_d to match with the dipping result r_q used for computing the database leakage. Since we assume that the record produced by $Create(S, L)$ is minimal if $|S| = 1$, we know that $r_d = Create(\{r_q\}, L)$ is always the smallest record that matches with r_q. Hence, we can now minimize the database leakage by adding as many OK possible bogus attributes to r_d.

Notice that while Algorithm 11 returns a disinformation record that is optimal according to the minimization problem defined in the beginning of Section 4, the length of the disinformation record may not be the shortest possible length. That is, we only focus on minimizing the database leakage, but do not try to minimize the disinformation record length as long as it is within C_{max}. One could define a more sophisticated optimization problem where we obtain the minimum database leakage using the shortest disinformation record possible. In this case, bogus attributes should not be appended to r_q once the database leakage no longer decreases.

The complexity of Algorithm 11 is $O(|R|^2 + C_{max})$ because the time to run Algorithm 8 takes $O(|R|^2)$ time and Steps 3–4 take $O(C_{max})$ time to run.

4.2 Multiple Record Release

We now consider the problem of adding multiple records for disinformation. Compared to adding a single disinformation record, adding multiple records adds a new dimension of complexity where we must now consider how the multiple records "influence" each other. For example, one must be careful in adding bogus attributes to one disinformation record when adding them to another disinformation record could decrease the database leakage by a larger amount. As another example, if we can perform linkage disinformation, the records that one disinformation record links may affect which records another disinformation should link in order to minimize the database leakage.

Just like in Section 4.1, we use the $Create(S, L)$ function for creating new records that have a length within L and the $Add(r)$ function that appends new attributes to r. We now show algorithms that return a set of records (instead of a single record) for disinformation. Again, as more properties are satisfied by the match and merge functions, the more efficient the disinformation algorithms become.

No Properties The brute-force algorithm for the multiple-record disinformation problem (which assumes no properties to be satisfied by M and μ) is a slight variant of Algorithm 9 where we replace one disinformation record r with a set of records S. In addition, the total cost of the records in S (i.e., $\Sigma_{r \in S}|r|$) must not exceed C_{max}.

We propose a greedy heuristic algorithm for multiple-record disinformation. Heuristic Algorithm 12 calls Algorithm 10 multiple times by assigning a portion of the entire budget C_{max}. For example, say we have the database $R = \{r, s, t\}$ and that only r and s match. Suppose $C_{max} = 2 \times C_{min}$ and $c = C_{min}$ where C_{min} is some constant cost that is always enough for $Create$ to return a non-empty record. When running Algorithm 10, say we generate the disinformation record u. Then S becomes $\{u\}$ (Step 5). Running Algorithm 10 again on $R \cup \{u\}$, say we now generate the record v. After updating S to $\{u, v\}$, C_{max} becomes 0, and we return S as the final multiple-record disinformation result.

A key part of Algorithm 12 is determining the maximum budget c for a single-record disinformation. If c is too small, then Algorithm 10 either cannot create a new disinformation record or will not be able to sufficiently lower the database leakage. If c is too large, then too much budget may be used to create a single disinformation record that lowers the leakage of one dipping result when it may be better to create several disinformation records that "simultaneously lower" the leakage values of multiple dipping results. Since Algorithm 10 creates records that match with at most two records (see Step 7 of Algorithm 10), a

Algorithm 12: Heuristic Multiple-Record Disinformation Algorithm

```
input : the budget $C_{max}$, the reference $p$, a match function $M$, a merge function $\mu$, and a set of records $R$
output: a set $S$ of disinformation records
1 $c \leftarrow$ Maximum budget for single record disinformation;
2 $S \leftarrow \{\}$;
3 while $C_{max} \geq c$ do
4     $r \leftarrow$ Algorithm 10$(c, p, M, \mu, R \cup S)$;
5     $S \leftarrow S \cup \{r\}$;
6     $C_{max} \leftarrow C_{max} - c$;
7 return $S$;
```

possible value of c is the larger of the following two values: the minimum length such that $Create(\{r, s\}, c)$ does not return $\{\}$ given two arbitrary records r and s, and the minimum length such that $Create(\{r\}, c)$ does not return $\{\}$ for any record r plus the cost to append one bogus attribute using the *Add* function. For example, if we use M_e for the match function and the key set $k = \{A, B\}$, then the minimum length of a record to connect two arbitrary records is 4. On the other hand, the minimum length of a record to match one arbitrary record is 2, and the cost of adding one bogus attribute is 1. Hence in this example, we set $c = max\{4, 2+1\} = 4$. An alternate strategy is to simply try out several values for c and choose the set S of disinformation records that minimizes the database leakage. For example, we may run Algorithm 12 multiple times where c is set to the values $\frac{C_{max}}{2}$, $\frac{C_{max}}{3}$, ..., $\frac{C_{max}}{100}$.

Representativity only Given that M and μ satisfy representativity, we can enhance the runtime of Algorithm 12 by enhancing the runtime of Algorithm 10. Techniques similar to the ones mentioned in Section 4.1 can be used.

Representativity and Negative Representativity Given that M and μ satisfy both representativity and negative representativity, we can compute the multiple-record disinformation more efficiently. Just like in Section 4.1, we exploit the fact that a disinformation record r_d cannot connect any records r and s where $M(r, s)$ = false by negative representativity. Before introducing the optimal disinformation algorithm, we first define one preliminary function.

Algorithm 13 returns the minimal set of disinformation records needed to lower the database leakage to a specified threshold T or less. If the option *op* = '$\leq$', we only require the database leakage to be less or equal to T. If the option *op* = '$<$', we require the database leakage to be strictly less than T. Suppose that *op* = '$\leq$'. Then for each distinct dipping result r_q, we first check if the query leakage $L_r(p, r_q)$ exceeds T. If so, we create a new record r_d and append OK bogus attributes until $L_r(p, \mu(r_q, r_d)) \leq T$ (Steps 5–7). If *op* = '$<$', the only difference is that we add bogus attributes to each disinformation record r_d until $L_r(p, \mu(r_q, r_d)) < T$. After creating a disinformation record (using the same method) for each dipping result whose query leakage exceeds T (Step 8), we return the set of disinformation records created (Step 9). For example, suppose that we have the database $R = \{r, s\}$ where r and s do not match and *op* = '$\leq$'. If both $L_r(p, r) > T$ and $L_r(p, s) > T$, we create two new records r_1 and r_2 that match r and s, respectively, and add bogus attributes to r_1 and r_2 until both $L_r(p, \mu(r, r_1)) \leq T$ and $L_r(p, \mu(s, r_2)) \leq T$. The result of Algorithm 13$(p, M, \mu, R, T, \text{'}\leq\text{'})$ is then $\{r_1, r_2\}$.

The worst-case complexity of Algorithm 13 is $O(|R| \times (|R| + L_d(p, M, \mu, R)))$ because Step 2 takes $O(|R|^2)$ time and Steps 3–8 take $O(|R| \times L_d(p, M, \mu, R))$ time in the worst case when $T = 0$.

Algorithm 14 returns an optimal multiple-record disinformation for R. The algorithm exploits the fact that there is no linkage disinformation (by the representativity and negative representativity properties) and creates at most one disinformation record per dipping result. The key is to find the lowest database leakage T we can obtain while limiting the total disinformation cost to be within C_{max}. Algorithm 14 uses a binary

Algorithm 13: Disinformation Algorithm assuming Representativity and Negative Representativity

```
input : the reference $p$, a match function $M$, a merge function $\mu$, a set of records $R$, a threshold $T$, and
        an operator $op$
output: a set of disinformation records $S$
1  $S \leftarrow \{\}, D \leftarrow \{\}$;
2  Steps 2–10 of Algorithm 8;
3  for $r \in S$ do
4      if $L_r(p, r)$ $op$ $T$ then
5          $r_d \leftarrow Create(\{r\}, \infty)$;
6          while $L_r(p, r \cup r_d)$ $op$ $T$ do
7              $Add(r_d)$;
8      $D \leftarrow D \cup \{r_d\}$;
9  return $D$;
```

search to find T (Steps 1–13) and then creates the disinformation records that lower the database leakage to T or less (Step 14). We note that that Algorithm 14 does not consider the cost of creating "intermediate" disinformation records that are used to find T as being part of the entire budget C_{max}. That is, the costs of the *Create* and *Add* functions that are called by Algorithm 13 within Steps 3, 4, 5, and 8 of Algorithm 14 are considered free, and only Step 14 that produces the final disinformation records uses up the budget. While one may argue that the overhead of creating intermediate records should also be part of the budget, we are simplifying the problem by assuming intermediate records are much cheaper to create and can immediately be discarded once they are used. We omit the details on how Algorithm 13 creates intermediate records with low cost.

Algorithm 14: Optimal Multiple-Record Disinformation Algorithm assuming Representativity and Negative Representativity

```
input : the budget $C_{max}$, the reference $p$, a match function $M$, a merge function $\mu$, and a set of records
        $R$
output: an optimal set of disinformation records $S$
1  $L \leftarrow 0, H \leftarrow$ Algorithm 8$(p, M, \mu, R)$;
2  repeat
3      $S_L \leftarrow$Algorithm 13$(p, M, \mu, R, L, `\leq')$;
4      $S_H \leftarrow$Algorithm 13$(p, M, \mu, R, H, `\leq')$;
5      $S'_H \leftarrow$Algorithm 13$(p, M, \mu, R, L_d(p, M, \mu, R \cup S_H), `<')$;
6      if $\Sigma_{r \in S_L}|r| = \Sigma_{r \in S'_H}|r|$ then
7          break loop
8      $S_M \leftarrow$Algorithm 13$(p, M, \mu, R, \frac{L+H}{2}, `\leq')$;
9      if $\Sigma_{r \in S_M}|r| \leq C_{max}$ then
10         $H \leftarrow \frac{L+H}{2}$;
11     else
12         $L \leftarrow \frac{L+H}{2}$;
13 until;
14 return Algorithm 13$(p, M, \mu, R, H, `\leq')$;
```

As a running example, suppose that we have the database $R = \{r = \{\langle A, a_1\rangle, \langle B, b_1\rangle, \langle B, b_2\rangle\}, s = \{\langle A, a_2\rangle, \langle B, b_3\rangle\}\}$. Say that the reference p contains all the existing B attributes (i.e., $p = \{\langle B, b_1\rangle, \langle B, b_2\rangle,$

$\langle B, b_3\rangle, \ldots\}$). Also suppose we use the match and merge functions M_c and μ_u and the key set $k = \{A\}$. For simplicity, say we define the record leakage function $L_r(p, r)$ as $(|p \cap r| - |r - p|)$. (We call this new measure L_{diff}; exploring various other record leakage functions is an interesting future work.) That is, we simply count the attributes in r that are already in p minus the number of attributes only in r. Finally, we set $C_{max} = 3$.

Initially, the dipping results of $R \cup S$ are r and s, and they have the query leakages 2 and 1, respectively. For instance, the record leakage of r is 2 because $L_r(p, r) = 2 - 0 = 2$. The initial database leakage L_d is thus $max\{2, 1\} = 2$. Hence, L is initially set to 0 while H is set to the database leakage 2 (Step 1).

We now start a binary search for the threshold T that can be used to minimize the database leakage given the budget C_{max}. We first compute S_L, S_H, and S'_H (Steps 3–5). First S_L is the minimal disinformation needed to reduce the database leakage to 0. The invocation of Algorithm 13$(p, M, \mu, R, 0, `\leq')$ returns the disinformation records that lower the queries leakages of r and s to 0. For r, we can create the disinformation record $r_1 = \{\langle A, a_1\rangle, \langle C, c_1\rangle, \langle C, c_2\rangle\}$ that reduces the query leakage to $L_r(p, \mu(r, r_1)) = 2 - 2 = 0$. (Notice that all C attributes are bogus.) For s, we can create the disinformation record $r_2 = \{\langle A, a_2\rangle, \langle C, c_3\rangle\}$ that reduces the query leakage to $L_r(p, \mu(s, r_2)) = 1 - 1 = 0$. Thus $S_L = \{r_1, r_2\}$. Next, we compute $S_H =$ Algorithm 13$(p, M, \mu, R, 2, `\leq')$. Since the database leakage does not have to decrease, $S_H = \{\}$. We now compute $S'_H =$ Algorithm 13$(p, M, \mu, R, L_d(p, M, \mu, R \cup \{\}), `<')$ =Algorithm 13$(p, M, \mu, R, 2, `<')$. Since the threshold T is 2 and we use the '$<$' option, we must reduce the query leakage of r by adding the disinformation record $r_3 = \{\langle A, a_1\rangle, \langle C, c_4\rangle\}$ for the database leakage to be strictly less than 2. Hence, $S'_H = \{r_3\}$. Intuitively, S'_H is the smallest disinformation that reduces the database leakage of $R \cup S_H$.

By comparing S_L and S'_H, we can now tell if there exists a disinformation that has a cost larger than that of S_L, but smaller than S_H. If so, we can continue to "narrow" our search for the right threshold T. Otherwise, we know there is no disinformation that lowers the database leakage to be less than H while using a total cost within C_{max}. In our example, since $\Sigma_{r \in S_L}|r| = 5$ while $\Sigma_{r \in S'_H}|r| = 2$, we continue to Step 8 (i.e., there is still more searching to do). In Step 8, we compute the disinformation that lowers the database leakage to be less or equal to $\frac{L+H}{2}$. If the total cost of the disinformation is less than C_{max}, we can safely reduce H to $\frac{L+H}{2}$. Otherwise, we can instead increase L to $\frac{L+H}{2}$. In our example, since $\frac{L+H}{2} = 1$, we now need to obtain the disinformation that reduces the query leakages of r and s to 1 or less. Since only r has a query leakage that exceeds 1, we construct $r_4 = \{\langle A, a_1\rangle, \langle C, c_5\rangle\}$ (which matches r) and set $S_M = \{r_4\}$. In Steps 9–10, we set H to 1 because the disinformation cost is smaller than C_{max}, i.e., $\Sigma_{r \in S_M}|r| = 2 \leq C_{max} = 3$.

During the second iteration of the while loop (Steps 2–13) where $L = 0$ and $H = 1$, we obtain $S_L = \{r_5, r_6\}$ and $S_H = \{r_7\}$ where we create the disinformation records $r_5 = \{\langle A, a_1\rangle, \langle C, c_6\rangle, \langle C, c_7\rangle\}$, $r_6 = \{\langle A, a_2\rangle, \langle C, c_8\rangle\}$, and $r_7 = \{\langle A, a_1\rangle, \langle C, c_9\rangle\}$. As a result, $S'_H =$ Algorithm 13$(p, M, \mu, R, L_d(p, M, \mu, R \cup \{r_7\}), `<')$ =Algorithm 13$(p, M, \mu, R, 1, `<') = \{r_8, r_9\}$ where we again create the disinformation records $r_8 = \{\langle A, a_1\rangle, \langle C, c_{10}\rangle, \langle C, c_{11}\rangle\}$ and $r_9 = \{\langle A, a_2\rangle, \langle C, c_{12}\rangle\}$. This time, $\Sigma_{r \in S_L}|r| = \Sigma_{r \in S'_H}|r| = 5$, which means that there does not exist any disinformation that lowers the database leakage to some value between L and H. Hence, we now exit the while loop with $H = 1$.

After the while loop, in Step 14 we create the disinformation record $r_{10} = \{\langle A, a_1\rangle, \langle C, c_{13}\rangle\}$ that reduces the database leakage to $H = 1$. We then return $S = \{r_{10}\}$ as the final set of disinformation records.

Recall that in the previous example we used a modified record leakage function called L_{diff} to simplify the illustration. We now define a general property for record leakage measures called strict monotonicity (not to be confused with the monotonicity for a match function with confidences in Definition 3.2).

Definition 4.4 *The record leakage function L_r is* strictly monotonic *in leakage if $L_r(p, r)$ is strictly decreasing as we append more bogus attributes to r.*

The F_1 measure satisfies strict monotonicity because adding bogus attributes for F_1 results in a decrease in precision and thus a decrease in F_1 as well. The L_{diff} measure also satisfies strict monotonicity because adding bogus attributes increases the attributes that are in r, but not in the reference p, lowering the L_{diff} result.

We now prove the correctness of Algorithm 14 given any strictly monotonic record leakage function.

Proposition 4.5 *Given that the record leakage function L_r is strictly monotonic, Algorithm 14 returns an optimal disinformation of records.*

Proof. Given a total budget of C_{max}, say the minimum obtainable database leakage is L'_d. For each dipping result r_q of R that has a query leakage exceeding L'_d, the only way to reduce the query leakage L_q to be less or equal to L'_d is to create a new record r that matches r_q and add enough bogus attributes (by representativity and negative representativity). Since L_r is strictly monotonic, there exists a minimum number n_r of bogus attributes to append to r that drops the query leakage to L'_d or less. Algorithm 14 does a binary search to find a range of database leakage values $[L, H]$ such that the disinformation cost for reducing the database leakage to L is larger or equal to C_{max} while that of H is smaller or equal to C_{max}. The exception for L is when C_{max} is more than enough for reducing the database leakage to 0. In this case, we can safely reduce C_{max} to the exact cost needed to reduce the database leakage to 0. Computing S'_H in Step 5 guarantees that we terminate the while loop in Steps 2–13. Intuitively, we are trying to find an "in the middle" disinformation S'_H that has a cost smaller than that of S_L and larger than that of S_H. If there is no such S'_H, then we can safely conclude that S_L and S_H will never change even if we further narrow the range $[L, H]$. Without Step 5, the algorithm may run indefinitely where L and H approach each other forever (while S_L and S_H do not change anyway). Hence, the resulting threshold H is one that can be used to minimize the database leakage as much as possible using resources within the budget C_{max}. Hence, the database leakage of $R \cup S$ where S = Algorithm 13$(p, M, \mu, R, H, `\leq')$ is L'_d.

The efficiency of Algorithm 14 depends on how fast $[L, H]$ converges to the range where there is no disinformation (i.e., S'_H) that has a leakage larger than L and smaller than H. If records in general have only a few attributes, then the convergence would be fast because there are not many possible disinformation results with different leakages. On the other hand, the more attributes records have, the more variety of possible leakage values, which can make the binary search in Steps 2–13 more time consuming. Since we always halve the range, the number of while-loop iterations is roughly proportional to $log(L_d(p, M, \mu, R))$. Since the complexity of Algorithm 13 is $O(|R| \times (|R| + L_d(p, M, \mu, R)))$, and the rest of the steps in the while loop take constant time to run, the total complexity of Algorithm 14 is $O(|R| \times (|R| + L_d(p, M, \mu, R)) \times log(L_d(p, M, \mu, R)))$.

5 Releasing Critical Information

Suppose that Alice tracks R, the information she has given out in the past. She now wants to release a new record r (e.g., her credit card information) which may fall in the hands of the adversary who uses the dipping algorithm D. Alice can compute the direct leakage involved in releasing the record r, i.e., $L_r(p, r)$. However, we may want to capture the information leaked by r only instead of computing the entire leakage of R. We thus define the incremental leakage of r as follows.

$$I(p, M, \mu, R, S) = L_d(p, M, \mu, R \cup S) - L_d(p, M, \mu, R)$$

Notice that, since we do not have a specific query q for Alice, we use the database leakages for computing the incremental leakage. Since r may make it possible for Eve to piece together big chunks of information about Alice, the incremental leakage may be large, even if r contains relatively little data.

To illustrate incremental leakage, say that Alice wants to purchase a cellphone app from one of two stores S_1 and S_2, and is wondering which purchase will lead to a more significant loss of privacy. Both stores require Alice to submit her name, credit card number, and phone number for the app. However, due to Alice's previous purchases, each store has different information about Alice. In particular:

- Alice's reference information is $p = \{\langle \text{N}, n_1, 1\rangle, \langle \text{C}, c_1, 1\rangle, \langle \text{C}, c_2, 1\rangle, \langle \text{P}, p_1, 1\rangle, \langle \text{A}, a_1, 1\rangle\}$ where N stands for name, C for credit card number, P for phone, and A for address.
- Store S_1 has one previous record $R_1 = \{r = \{\langle \text{N}, n_1, 1\rangle, \langle \text{C}, c_1, 1\rangle, \langle \text{A}, a_1, 1\rangle\}\}$. That is, Alice bought an item using her credit card number and shipping address. (We omit the item information in any record for brevity.)

- Store S_2 has two previous records $R_2 = \{s = \{\langle \text{N}, n_1, 1\rangle, \langle \text{C}, c_1, 1\rangle, \langle \text{P}, p_1, 1\rangle\}, t = \{\langle \text{N}, n_1, 1\rangle, \langle \text{C}, c_2, 1\rangle, \langle \text{A}, a_1, 1\rangle\}\}$. Here, Alice has bought items using different credit cards. The item of s could be a ringtone that required a phone number for purchasing, but not a shipping address.
- Both S_1 and S_2 require the information $u = \{\langle \text{N}, n_1, 1\rangle, \langle \text{C}, c_2, 1\rangle, \langle \text{P}, p_1, 1\rangle\}$ for the cellphone app purchase. Since Alice is purchasing an app, again no shipping address is required.

To compute leakages, say Alice is only concerned with the previously released information, so she assumes that the database at store S_1 only contains record r, while the database at store S_2 only contains s and t. (The stores are not colluding in this example.) Alice also assumes that two records match if their names and credit card numbers are the same or their names and phone numbers are the same, and that merging records simply performs a union of attributes.

Under these assumptions, before Alice's app purchase, the database leakage for both stores is $\frac{3}{4}$. For the first store, R_1 only contains one record r, so the database leakage is $L_r(p, r) = \frac{2\times1\times3/5}{1+3/5} = \frac{3}{4}$. For the second store, R_2 contains two records s and t, so we need to take the maximum of the query leakages of s and t. Since s and t do not match with each other (i.e., they do not have the same name and credit card or name and phone combination), the dipping result of s is s while the dipping result of t is t. Hence, the database leakage is $max\{L_r(p, s), L_r(p, t)\} = max\{\frac{3}{4}, \frac{3}{4}\} = \frac{3}{4}$.

If Alice buys her app from S_1, then its database will contain two records, r and u. In this case, the database leakage at S_1 is still $\frac{3}{4}$ because r and u do not match and thus have the same query leakage $\frac{3}{4}$ (the maximum query leakage is thus $\frac{3}{4}$). On the other hand, if Alice buys from S_2, the database at S_2 will contain s, t, and u. Since u matches with both s and t, the dipping result of u is $\mu(\{s, t, u\})$, which is identical to p. Hence, the database leakage becomes 1.

To compare Alice's two choices, it is useful to think of the incremental leakage, in this example, the change in leakage due to the app purchase. In our example, the incremental leakage at S_1 is $\frac{3}{4} - \frac{3}{4} = 0$ while the incremental leakage at S_2 is $1 - \frac{3}{4} = \frac{1}{4}$. Thus, in this case Alice should buy her app from S_1 because it preserves more of her privacy.

6 Enhancing a Composite Record

From the adversary's point of view, there may also be interesting "optimization" questions to ask. Since Eve does not know Alice's full record p, the questions cannot be phrased in terms of p. Consider a composite record r_c that Eve has inferred from a set of facts in a set R. For whatever reason, Eve is very interested in r_c, but unfortunately there is some uncertainty in the attributes in r_c. We define r_p to be the same as r_c except that all confidences in r_p are set to 1. $L_r(r_p, r_c)$ is a measure of how certain r_c is: the closer $L_r(r_p, r_c)$ is to 1, the more certain Eve is of the information in r_c.

To improve $L_r(r_c, p)$ (i.e., make it closer to 1), Eve can try to increase her confidence in the attributes in R. For any given attribute $a = \langle l, v, c\rangle$ in some $r_i \in R$, Eve can improve the confidence of a by doing more research, bribing someone, issuing a subpoena, etc. The increase in the confidence of a will clearly have a cost associated with it. There are again many ways to model the cost, but for simplicity let us assume that the cost in changing the confidence from its current value of c to 1 is $C(a) = 1 - c$.

Now the question is, what is the most cost effective way to increase Eve's confidence in r_c. If Eve only wants to verify one attribute, then we want the one $a \in r_i$ that maximizes

$$\frac{L_r(r_p, r'_c) - L_r(r_p, r_c)}{C(a)}$$

where r'_c is the composite record Eve can infer when the confidence in a is increased to 1.

For example, suppose that we have the database $R = \{r_1 = \{\langle \text{N, Alice}, 1\rangle, \langle \text{A}, 20, 1\rangle\}, r_2 = \{\langle \text{N, Alice}, 0.9\rangle, \langle \text{P}, 123, 0.8\rangle, \langle \text{C}, 987, 1\rangle\}\}$ where N stands for name, A stands for age, P stands for phone, and C stands for credit card number. Suppose that Alice's dipping result is $r_c = \mu_{ux}(r_1, r_2) = \{\langle \text{N, Alice}, 1\rangle, \langle \text{A}, 20, 1\rangle, \langle \text{P}, 123, 0.8\rangle, \langle \text{C}, 987, 1\rangle\}$. (Recall that μ_{ux}, defined in Section 3.2, takes the maximum confidence value when merging two attributes with the same label and value pair.) Then $r_p = \{\langle \text{N, Alice}, 1\rangle, \langle \text{A}, 20,$

1⟩, ⟨P, 123, 1⟩, ⟨C, 987, 1⟩}. If we enhance the name in r_2 to have a confidence of 1, then r'_c = {⟨N, Alice, 1⟩, ⟨A, 20, 1⟩, ⟨P, 123, 0.8⟩, ⟨C, 987, 1⟩} (which is identical to r_c), and $C(\langle \text{N}, \text{Alice}, 0.9\rangle) = 1 - 0.9 = 0.1$. Then $\frac{L_r(r_p, r'_c) - L_r(r_p, r_c)}{C(\langle \text{N}, \text{Alice}, 0.9\rangle)} = \frac{\frac{2\times1\times0.95}{1+0.95} - \frac{2\times1\times0.95}{1+0.95}}{0.1} = 0$. On the other hand, if we enhance the phone number of r_2, then r'_c = {⟨N, Alice, 1⟩, ⟨A, 20, 1⟩, ⟨P, 123, 1⟩, ⟨C, 987, 1⟩} with the cost $C(\langle \text{P}, 123, 0.8\rangle) = 1 - 0.8 = 0.2$. Then $\frac{L_r(r_p, r'_c) - L_r(r_p, r_c)}{C(\langle \text{P}, 123, 0.8\rangle)} = \frac{\frac{2\times1\times1}{1+1} - \frac{2\times1\times0.95}{1+0.95}}{0.2} \approx 0.13$. Hence, it is better to enhance the phone number of r_2 instead of its name.

A variant of this problem is to find out the most cost effective way to disprove r_c. For instance, say that if we show that Joe does not really use the pseudonym "Scarface," then our record r_c would break apart completely. Then it would be important to investigate if the record that links the two names together is really valid. Thus, we may want to find the record in R whose removal would cause the greatest decrease in $L_r(r_p, r_c)$, relative to the cost of checking if the record is valid.

7 Checking a Hypothesis

Eve may be interested in a "hypothesis" record r that *cannot* be currently inferred from a set of records R. Instead of enhancing existing records, Eve now needs to add a new record (or multiple records) to R in order to substantiate the hypothesis. The question is which one (or more) records are the best ones to add, again given that there is a cost associated with adding new information. Once Eve identifies this most cost-effective record, she can go out and investigate it. If the record happens to be true, Eve can add it to R and then have solid evidence for the hypothesis r.

8 Related Work

Many works have proposed privacy schemes for data publishing in the context of linkage attacks. The k-anonymity [6] model guarantees that linkage attacks on certain attributes cannot succeed. Subsequent works such as the l-diversity model [3] have identified and fixed new weaknesses of the k-anonymity scheme. Rastogi et al. [4] shows the tradeoff between privacy and utility in data publishing in the context of maintaining the accuracy of counting queries. In contrast, we assume that the data is already published and that we want to manage and minimize the leakage of sensitive information.

A closely related work is the P4P framework [1], which seeks to contain illegitimate use of personal information that has already been released to an external (possibly adversarial) entity. For different types of information, general-purpose mechanisms are proposed to retain control of the data. More recently, a startup called ReputationDefender [5] has started using disinformation techniques for managing the reputation of individuals focusing on improving search engine results (e.g., adding to the web positive or neutral information about its customers by either creating new web pages or by multiplying links to existing ones). The focus of ReputationDefender is to make one's correct information clearly visible. Hence, the disinformation is being used to maximize the database leakage. TrackMeNot [7] is a browser extension that helps protect web searchers from surveillance and data-profiling by search engines using noise and obfuscation. In comparison, our work complements the above works by formalizing information leakage and proposing disinformation techniques against data dipping algorithms.

Data dipping has also been studied under the name of query-time entity resolution [2] where a query is efficiently resolved against a database. In comparison, our focus is on computing the database leakage based on data dipping results as well as proposing efficient disinformation algorithms as well.

9 Conclusion and Future Work

We have propose a framework for managing information leakage and various algorithms that compute the query and database leakages. The more properties are satisfied by the match and merge functions, the more efficient the algorithms for computing leakage. We have posed a number of challenges both in ER and

DP. Among them, we have studied in detail the problem of using disinformation as a tool for containing information leakage. We believe our techniques are preliminary steps to the final goal of truly managing public data, and that many interesting problems remain to be solved.

There are many challenges we would like to address in future work including the following:

- A comparison of our privacy guarantees to those in the privacy literature.
- How to create bogus information that is not too different from the original data (otherwise the adversary will not believe in it), but not too similar either (the similar data may actually confirm certain information to be true).
- How to manage information leakage in the presence of data updates and deletes (in addition to inserts).
- Defining utility, i.e., how useful the data is to the user in the presence of disinformation.
- How to use our techniques for, say, millions of users where the adversary may now have the incentive to produce counter attacks.
- Evaluating the negative effects of disinformation where an honest person may get incorrect information of Alice.
- Solving the dual problem of disinformation where we would like to emphasize certain data (e.g., Alice may want people to know for sure that she is moving).

10 Acknowledgements

We thank Narayanan Shivakumar and Aleksandra Korolova for helpful discussions.

References

1. G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, N. Mishra, R. Motwani, U. Srivastava, D. Thomas, J. Widom, and Y. X. 0002. Vision paper: Enabling privacy for the paranoids. In *VLDB*, pages 708–719, 2004.
2. I. Bhattacharya and L. Getoor. Query-time entity resolution. *J. Artif. Intell. Res. (JAIR)*, 30:621–657, 2007.
3. A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *ICDE*, page 24, 2006.
4. V. Rastogi, S. Hong, and D. Suciu. The boundary between privacy and utility in data publishing. In *VLDB*, pages 531–542, 2007.
5. ReputationDefender. http://www.reputationdefender.com.
6. L. Sweeney. k-anonymity: A model for protecting privacy. *IJUFKS*, 10(5):557–570, 2002.
7. TrackMeNot. http://cs.nyu.edu/trackmenot.