Disinformation Techniques for Entity Resolution

ABSTRACT
We study the problem of disinformation. We assume that an “agent” has some sensitive information that the “adversary” is trying to obtain. For example, a camera company (the agent) may secretly be developing its new camera model, and a user (the adversary) may want to know in advance the detailed specs of the model. The agent’s goal is to disseminate false information to “dilute” what is known by the adversary. We model the adversary as an Entity Resolution (ER) process that pieces together available information. We formalize the problem of finding the disinformation with the highest benefit given a limited budget for creating the disinformation and propose efficient algorithms for solving the problem. We then evaluate our disinformation planning algorithms on real and synthetic data and compare the robustness of existing ER algorithms. In general, our disinformation techniques can be used as a framework for testing ER robustness.

1. INTRODUCTION
Disinformation is a well-known strategy used to perturb known information by adding false information. A classic example is the Normandy landing during World War II, where the Allied forces used disinformation to make the Germans believe an attack was imminent on Pas de Calais instead of Normandy. As a result, the German forces were concentrated in Pas de Calais, which made the Normandy landing one of the turning points in the war. To present a more modern example, consider the way life insurers are predicting the life spans of their customers by piecing together health-related personal information on the Web [12]. Here the customers cannot prevent the sensitive information from leaking, as they need to give it out piecemeal to purchase items, interact with their friends, get jobs, etc. However, disinformation could be used to protect sensitive information by preventing the ER algorithm used by the insurance companies from identifying with certainty the customer’s say habits or genes.

We adapt disinformation to an information management setting where parts of an agent’s critical information has leaked to the public. For example, suppose a camera manufacturer called Cakon is working on its latest camera product called the C300X. Although the new model is supposed to be secret, some details on the specs of the C300X may have been leaked to the public as rumors by early testers, insiders, and even competitors. Such leaks may be damaging for Cakon because customers that know about the C300X may delay their purchase of old camera models until the new camera is manufactured, potentially lowering the profits of Cakon. Or a competitor camera company may realize Cakon’s strategy and develop a new camera with better specs.

It is usually very hard to “delete” public information. For example, once the information of the C300X is leaked on rumor sites, Cakon may not be able to ask the person who wrote the rumor to delete her remark. Even if the rumor was deleted, several copies of the information may remain in backup servers or other web sites.

An alternative strategy is to use disinformation techniques and add even more information to what the public (adversary) knows. Specifically, the agent generates “bogus” records such that the adversary will have more difficulty resolving the records correctly. As a result, we can effectively “dilute” the existing information.

We assume that the adversary performs an Entity Resolution (ER) operation, which is the process of identifying and merging records judged to represent the same real-world entity. In our example, the adversary can then piece together various rumors about the C300X to get a more complete picture of the model specifications, including its release date.

To illustrate how disinformation can be used, suppose there are five records \(r, s, t, u\), and \(v\) that represent camera rumors as shown in Table 1. Suppose that the five records refer to four different camera models where the ER algorithm correctly clusters the records by producing the ER result \(E_1 = \{\{r\}, \{s\}, \{t\}, \{u, v\}\}\). Here, we use curly brackets to cluster the records that refer to the same entity. The ER result says that the adversary considers \(u\) and \(v\) to refer to the same camera model while considering the remaining three records to be different models.

Now say that the agent’s sensitive information is \(\{r\}\), which is the target cluster that refers to the new camera model C300X (which in reality will have 30M pixels and sell for 8K dollars). The agent can reduce what is known about the C300X by generating a record that would make the adversary confuse the clusters \(\{r\}\) and \(\{s\}\). Generating the disinformation record would involve creating a model number that is similar to both C300X and C300 and then taking some realistic number of pixels between 20M and 30M and

<table>
<thead>
<tr>
<th>Record</th>
<th>Model</th>
<th>Pixels (M)</th>
<th>Price (K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(r)</td>
<td>&quot;C300X&quot;</td>
<td>30</td>
<td>8</td>
</tr>
<tr>
<td>(s)</td>
<td>&quot;C300&quot;</td>
<td>20</td>
<td>7</td>
</tr>
<tr>
<td>(t)</td>
<td>&quot;C200&quot;</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>(u)</td>
<td>&quot;C100&quot;</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>(v)</td>
<td>&quot;C100&quot;</td>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1: Camera Rumor Records

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...$15.00.
a price between 7K and 8K dollars. Suppose that the agent has generated the disinformation record \(d_2 = \{\langle \text{Model}, \langle \text{C300X} \rangle, \{\text{Pixels}, 20\}, \langle \text{Price}, 7\rangle \}\) as a result, \(d_2\) may match with \(r\) because they have the same model name. The ER algorithm can conceptually merge the two records into \(r + d_2 = \{\langle \text{Model}, \langle \text{C300X} \rangle, \{\text{Pixels}, 20\}, \langle \text{Price}, 7\rangle \} \) where the ‘+’ operation denotes the merged result of two records. Now \(r + d_2\) and \(s\) are now similar and might match with each other because they have the same number of pixels and price. As a result, \(r + d_2\) and \(s\) may merge to produce the new ER result \(E_2 = \{\langle r, s, d_1 \rangle, \{t\}, \{u, v\}\}\). While \(r\) and \(s\) were considered different entities in \(E_1\), they are now considered the same entity in \(E_2\) with the disinformation record \(d_1\).

As a result of the disinformation, the adversary is confused about the upcoming C300X: Will it have 20M or 30M pixels? Will it sell for 7K or 8K dollars? With all the uncertainty, the adversary may be even less confident that the C300X is a real upcoming product. (In Section 5 we define a concrete metric for confusion.)

We could further reduce the correctness of the target cluster by merging it with even more clusters. For example, if we also add the disinformation record \(d_2 = \{\langle \text{Model}, \langle \text{C3000} \rangle, \{\text{Pixels}, 10\}, \langle \text{Price}, 5\rangle \}\) to Table 1, then the ER result may now consider the C300 and C200 to be the same model as well, leading to an ER result of \(E_3 = \{\langle r, s, d_1, d_2, t \rangle, \{u, v\}\}\) where \(r, s, \) and \(t\) have all merged together. While \(E_3\) dilutes C300X more than \(E_2\), there is also a price to pay for generating more disinformation records. Hence, creating disinformation now becomes an optimization problem where we would like to maximize the “confusion” around an entity as much as possible using a total cost for creating the disinformation records within a fixed budget.

A practical application of our disinformation techniques is evaluating the “robustness” of ER algorithms against disinformation. Note that in our example above, whether \(r\) and \(s\) and \(d\) merge is very dependent on the ER algorithm in use. Some ER algorithms may be more susceptible to merging unrelated records while others may be more robust to a disinformation attack. The robustness may also depend on the comparison threshold used for matching records. For example, an ER algorithm with a more relaxed comparison threshold is more likely to mistakenly merge records compared to the same ER algorithm with a stricter comparison threshold. Our disinformation techniques can be used to evaluate how sensitive each ER algorithm and threshold combination is to various data changes.

In summary, the main contributions of this paper are:

- We formalize the notion of disinformation in an ER setting. We also define a disinformation optimization problem that maximizes confusion for a given disinformation budget. We analyze the hardness of this problem and propose a desirable ER property that makes disinformation more effective (Section 2).
- We propose efficient exact and approximate algorithms for a restricted version of the disinformation problem. We then propose heuristics for the general disinformation problem (Section 3).
- We propose techniques for generating the values of disinformation records based on the values of existing records (Section 4).
- We experiment on synthetic and real data to demonstrate the effectiveness of disinformation and compare the robustness of existing ER algorithms (Section 5).

2. FRAMEWORK

In this section, we formalize ER and the disinformation problem. We first define our ER model. We then introduce a pairwise approach for evaluating the cost of merging clusters. Next, we define the benefit obtained by merging clusters and develop a global strategy for generating disinformation. Finally, we introduce two ER algorithms in the literature and identify a desirable property of ER algorithms that makes our pairwise approach effective.

2.1 ER Model

We assume a database of records \(R = \{r_1, r_2, \ldots, r_n\}\). The database could be a collection of rumors, homepages, tuples, or even tweets. Each record \(r\) is a set of attributes, and each attribute can be thought of as a label and value pair, although this view is not essential for our work. 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 refer to the camera model C300X:

\[r = \{\langle \text{N}, \langle \text{C300X} \rangle, \{\text{X}, 30\}, \{\text{P}, 8\}\}\}\]

Each attribute \(a \in r\) is surrounded by angle brackets and consists of one label \(a.lab\) and one value \(a.val\).

An ER algorithm \(E\) takes as input a set of records and groups together records that represent the same real world entity. We represent the output of the ER process as a partition of the input. Given an input set of records \(R = \{r, s, t, u, v\}\), an output can be \(\{\langle r, s, t \rangle, \{u, v\}\}\), where the inner curly brackets indicate the records that refer to the same entity. In this example, two real world entities were identified, with \(\langle r, s, t \rangle\) representing the first, and \(\{u, v\}\) representing the second. Intuitively, the more accurately \(E\) clusters the records, the more useful information the adversary obtains.

We assume the ER algorithm used by the adversary is known to the agent, but cannot be modified. This assumption is common where one must guess the sophistication and compute power of an adversary. For instance, Cakon may know all the possible websites containing rumors of Cakon products and may have an idea on how an adversary might piece together the rumors from the websites. (In Section 5.1.3 we discuss what happens when the agent does not know the adversary’s ER algorithm.) In addition, we assume the agent cannot delete or modify records in the database \(R\).

2.2 Pairwise Approach for Merging Clusters

To generate disinformation, we first take a pairwise approach of analyzing the costs of merging clusters. We assume that inducing the merge of two clusters \(c_i\) and \(c_j\) has a well-defined cost function \(D(c_i, c_j)\) that is non-negative and commutative. That is, \(\forall i, j, D(c_i, c_j) \geq 0\) and \(D(c_i, c_j) = D(c_j, c_i)\). The cost function measures the agent’s effort to generate disinformation records that would merge \(c_i\) and \(c_j\).

In addition, we assume a function \(PLAN(c_i, c_j)\) that specifies the steps for actually generating the disinformation records in order to merge \(c_i\) and \(c_j\). The definitions of the two functions depend on the ER algorithm and the records as we illustrate below.

The cost function \(D\) can reflect the amount of disinformation that needs to be generated. For example, suppose that all the records are in a Euclidean space, and the ER algorithm always clusters records that are within a Euclidean distance of 1. If there are two singleton clusters \(c_1\) and \(c_2\) that have a Euclidean distance of 10, then we need to generate at least 9 records between \(c_1\) and \(c_2\) that have a distance of 1 between each other and with \(c_1\) and \(c_2\). If the cost of creating the records is estimated as the number of disinformation records that need to be created, then \(D(c_1, c_2) = 9\) and \(PLAN(c_1, c_2)\) provides the steps for actually creating the 9 disinformation records.

A more sophisticated cost function may also reflect the effort needed to create the specific disinformation values. For example,
creating a new public camera record would require some person to post a rumor about the camera on a public website or blog, and the effort may vary depending on the contents of the rumor. A rumor about a camera with unrealistically-high specs may actually damage the reputation of Cakon and can be viewed as a costly disinformation value to create. The result of PLAN(c_i, c_j) may now include instructions for logging into the website and posting the rumor.

In the case where there is no way to induce the merge of c_i and c_j (e.g., due to the ER algorithm or restrictions in the records that can be generated), then the distance D(c_i, c_j) is given as ∞. In general, the more different c_i and c_j are, the costlier it is to generate the disinformation needed to merge the clusters.

For the optimization problem we define next, we assume that the merging costs for different pairs of clusters are independent of each other. That is, the value of D(c_i, c_j) is fixed and not affected by whether other clusters were merged. For example, if D(c_1, c_2) = 5 and D(c_1, c_3) = 4, then the cost of merging c_1 and c_2 is always 5 regardless of whether we will merge c_1 and c_3. In reality, however, the costs may not be independent because the merging of multiple pairs of clusters may be affected by the same disinformation.

For instance, if the disinformation record d is in the middle of the clusters c_1 and c_2 as well as the clusters c_3 and c_4, then by adding d, we may end up merging c_1 and c_2 as well as c_3 and c_4 at the same time. Hence, once c_1 and c_2 are merged, the merging cost of c_3 and c_4 reduces to 0. Note that when we evaluate our disinformation strategies in Section 5, we do not enforce this assumption for the actual ER algorithms. In Section 5.2, we show that even without the independence assumption, our techniques work well in practice.

### 2.3 Disinformation Problem

We consider the problem of maximizing the “confusion” of one entity e, which we call the target entity. Given an ER algorithm E and a database R, we call the cluster c_0 ∈ E(Δ) that represents the information of e the target cluster. Intuitively, by merging other clusters in E(Δ) to the target cluster, we can dilute the information in the target cluster and thus increase the confusion. In our motivating example, the camera company Cakon was increasing the confusion on the target entity C300X by merging the C300 cluster {s} to the target cluster {r}. If there are multiple clusters that represent e, we choose the cluster that “best” represents e and set it as the target cluster c_0. For example, we could define the best cluster as the one containing the largest number of records that refer to e.

The records in Δ do not necessarily have to refer to real entities. For example, one can create records of a fake entity in order to merge them with the target cluster. While computing the cost of creating the fake entities is out of the scope of this paper, our disinformation techniques can be applied to maximize confusion once all the fake entities are created.

The confusion of an entity is an application-specific measure. For example, we can define the confusion of e as the number of incorrect attributes of e minus the number of correct attributes of e where we count duplicate attributes. The amount of confusion we gain whenever we merge a cluster c_i ∈ E(Δ) with the target cluster c_0 can be captured as the benefit of c_i, which is computed as N(c_i) using a benefit function N. In our example above, we can define the benefit of c_i to be the number of incorrect attributes in c_i about e. Suppose that e can be represented as the record r = \{(Model, “C300X”), (Pixels, 20)\}. Then a cluster e containing the records s = \{(Model, “C300X”), (Pixels, 20)\} and t = \{(Model, “C200”), (Pixels, 20)\} has one correct attribute (i.e., (Model, “C300X”)) and three incorrect attributes (i.e., one (Model, “C200”) and two (Pixels, 20)’s). As a result, the benefit of e is 3. As a default, we always define the benefit N(c_0) to be 0 because c_0 does not need to merge with itself. In Section 5.1 we define a concrete metric for confusion.

Given our knowledge on the ER algorithm E, database R, cost function D, and the benefit function N, we now define an optimization problem of producing the best set of pairwise cluster merges that can maximize the total benefit while using a total cost for merging clusters within a fixed budget. We first draw an undirected cost graph G among the clusters in E(Δ) where each edge between the clusters c_i and c_j (denoted as c_i → c_j) has a weight of D(c_i, c_j).

For example, suppose that E(Δ) = \{⟨t⟩, {s}, {t}, {{u, v}}\} and the target cluster c_0 = {r}. Also suppose that D({r}, {s}) = 1, D({s}, {{u, v}}) = 2, and the rest of the edges have the weight 4. In this example, we also assume that the benefits for all clusters have the value 10, except for c_0, which has a benefit of 0. The resulting cost graph G is shown in Figure 1 (ignore the double lines for now).

We view any subtree J in G that has the target cluster c_0 as its root a disinformation plan of the entity e that specifies which pairs of clusters should be merged together through disinformation. Just like in G, we denote the set of vertices in J as J.V and the set of edges in J as J.E. The cost of merging the clusters connected by J is then \(\sum_{(c_i, c_j) \in J.E} D(c_i, c_j)\). Continuing our example above, suppose the subtree J of G connects the three clusters \{r\}, \{s\}, and \{t\} with the two edges \{r\}→\{s\} and \{s\}→\{t\}. Here, the plan is to add disinformation records between \{r\} and \{s\}, and between \{s\} and \{t\} to merge the three clusters. As a result, the total merging cost of J is 1 + 2 = 3, and the total benefit obtained is 0 + 10 + 10 = 20.

Figure 1 depicts the plan J by drawing double lines for the edges in J. If the subtree J instead contained the edges \{r\}→\{s\} and \{r\}→\{t\}, then the total merging cost would be 1 + 4 = 5 (but the benefit would be the same, i.e., 20).

Given a budget B that limits the total cost of generating disinformation, we define the optimal disinformation plan of e as follows.

**Definition 2.1.** Given a cost graph G, a target cluster c_0, a cost function D, a benefit function N, and a cost budget B, the optimal disinformation plan J is the subtree of G that contains c_0 and has the maximum total benefit \(\sum_{c_i \in J.V} N(c_i)\) subject to \(\sum_{(c_i, c_j) \in J.E} D(c_i, c_j) \leq B\).

Using the cost graph in Figure 1, suppose that c_0 = {r} and the cost budget B = 3. As a result, the subtree J with the largest benefit connects the clusters \{r\}, \{s\}, and \{t\} with the edges \{r\}→\{s\} and \{s\}→\{t\} and has a total benefit of 0 + 10 + 10 = 20 and a total merging cost of D({r}, {s}) + D({s}, {t}) = 1 + 2 = 3 ≤ B. Merging \{u, v\} to c_0 will require a total merging cost of 4, which exceeds B.

A disinformation plan provides a guideline for creating disinformation. Since we assume that all the merging costs are independent of each other, a disinformation plan satisfying Definition 2.1
does not necessarily lead to an optimal disinformation in the case where the costs are not independent. However, the independence assumption allows us to efficiently find out which clusters should be merged in order to increase the confusion significantly. In Section 5 we will study the effectiveness of disinformation plans based on the independence assumption in scenarios where the merging costs are not independent.

In general, the total benefit of the merged clusters may not be expressible as a linear sum of fixed benefits and may depend on the specific combination of clusters. The new problem can be stated by replacing the sum \( \sum_{c_i \in C} N(c_i) \) in Definition 2.1 with some general function \( F(J,V) \) that reflects the total benefit. While this generalization may capture more notions of confusion, there is less opportunity for an efficient computation of the optimal plan.

We now show that the disinformation problem is NP-hard in the strong sense [4], which means that the problem remains NP-hard even when all of its numerical parameters are bounded by a polynomial in the length of the input. In addition, it is proven that a problem that is NP-hard in the strong sense has no fully polynomial-time approximation scheme unless \( P = NP \). The proofs for the complexity of the disinformation problem can be found in our technical report [17].

**Proposition 2.2.** Finding the optimal disinformation plan (Definition 2.1) is NP-hard in the strong sense.

Given that the disinformation problem is NP-hard in the strong sense, we now consider a more restricted version of the disinformation problem where we only consider disinformation plans that have heights of at most \( h \). Here, we define the height of a tree as the length of the longest path from the root to the deepest node in the tree. For example, if a tree has a root node and two child nodes, the height is 1. We can prove that even if \( h = 2 \), the disinformation problem is still NP-hard in the strong sense. However, if \( h = 1 \) then the problem becomes polynomial-time solvable.

**Proposition 2.3.** Finding the optimal disinformation plan (Definition 2.1) with \( h = 1 \) is NP-hard in the weak sense.

The disinformation problem with \( h = 1 \) is interesting because it captures the natural strategy of comparing the target entity \( e \) with one other entity at a time, making it a practical approach for disinformation. In Section 3, we show there are an exact pseudo-polynomial algorithm and an approximate polynomial-time algorithm for the \( h = 1 \) problem. In Section 5, we show that disinformation plans with \( h = 1 \) perform just as good as general disinformation plans in terms of maximizing the confusion of \( e \) while taking much less time to generate.

### 2.4 ER Algorithms

We illustrate two ER algorithms in the literature. In Section 5, we use these algorithms for evaluating our disinformation techniques.

The Single-link Hierarchical Clustering algorithm [6, 8] (HC) merges the closest pair of clusters (i.e., the two clusters that have the smallest distance) into a single cluster until the smallest distance among all pairs of clusters exceeds a certain threshold \( T \). When measuring the distance between two clusters, the algorithm takes the smallest possible distance between records within the two clusters. Suppose we have the input set of records \( R = \{r_1, r_2, r_3\} \) where \( T = 2 \), and the distances between records are set as \( d(r_1, r_2) = 2 \), \( d(r_2, r_3) = 4 \), and \( d(r_1, r_3) = 5 \). The HC algorithm creates a partition of singletons \( \{r_1\}, \{r_2\}, \{r_3\} \) and first merges \( \{r_1\} \) and \( \{r_2\} \), which contain the closest records that have a distance smaller or equal to \( T \), into \( \{r_1, r_2\} \). The cluster distance between \( \{r_1, r_2\} \) and \( \{r_3\} \) is the minimum of \( d(r_1, r_3) \) and \( d(r_2, r_3) \), which is 4. Since the distance exceeds \( T \), \( \{r_1, r_2\} \) and \( \{r_3\} \) do not merge, and the final ER result is \( \{\{r_1, r_2\}, \{r_3\}\} \).

The Sorted Neighborhood (SN) algorithm [5] initially sorts the records in \( R \) using a certain key assuming that closer records in the sorted list are more likely to match. For example, suppose that we have the input set of records \( R = \{r_1, r_2, r_3\} \) and sort the records by their names (which are not visible in this example) in alphabetical order to obtain the list \( \{r_1, r_2, r_3\} \). The SN algorithm then slides a fixed-sized window and compares all the pairs of clusters that are inside the same window at any point. If the window size is 2 in our example, then we compare \( r_1 \) with \( r_2 \) and then \( r_2 \) with \( r_3 \), but not \( r_1 \) with \( r_3 \) because they are never in the same window. We thus produce pairs of records that match with each other. We can repeat this process using different keys (e.g., we could also sort the person records by their address values). After collecting all the pairs of records that match, we perform a transitive closure on all the matching pairs of records. For example, if \( r_1 \) matches with \( r_2 \) and \( r_2 \) matches with \( r_3 \), then we merge \( r_1, r_2, r_3 \) together into the ER result \( \{\{r_1, r_2, r_3\}\} \).

### 2.5 Monotonicity

We now define a property of an ER algorithm that makes our disinformation techniques more effective.

**Definition 2.4.** An ER algorithm \( E \) is monotonic if for any database \( R \) and disinformation record \( d \), \( \forall c_i \in E(R), \exists c_j \in E(R) \cup \{d\} \) where \( c_i \subseteq c_j \).

For example, say that the ER algorithm \( E \) is monotonic and \( E(R) = \{\{r, s\}, \{t\}\} \). Then if we add a disinformation record \( d \) and compute \( E(R \cup \{d\}) \), the records \( r \) and \( s \) can never split. Thus a possible ER result would be \( \{\{r, s, d\}, \{t\}\} \), but not \( \{\{r, d\}, \{s\}, \{t\}\} \).

The monotonicity property is helpful in the agent’s point of view because we do not have to worry about the ER algorithm splitting any clusters when we are trying to merge two clusters. As a result, the analysis of the cost graph is accurate, and the agent can better predict how the ER result would change if we add disinformation records according to the optimal disinformation plan.

We now show which ER algorithms in Section 2.4 satisfy the monotonicity property. The proof can be found in our technical report [17].

**Proposition 2.5.** The HC algorithm is monotonic, but the SN algorithm is not monotonic.

Notice that if the window size is at least \( |R| \) (i.e., the total number of records), then the SN algorithm does satisfy monotonicity because the algorithm reduces to a pairwise comparison of all records followed by a transitive closure.

The monotonicity property is desirable because the disinformation we generate (see Section 4 for details) is more likely to merge clusters according to the disinformation plan without any clusters splitting in the process.

### 3. PLANNING ALGORITHMS

We start by proposing an algorithm that returns an optimal disinformation plan (Definition 2.1) where \( h = 1 \). Restricting \( h \) to 1 gives us the insight for solving the general problem later on.
We propose a pseudo-polynomial algorithm that uses dynamic programming and runs in \(O((G.V) \times B)\) time, which is polynomial to the numerical value of the budget, but still exponential to the length of the binary representation of \(B\). We assume that \(B\) is an integer and that all the edges in the cost graph \(G\) have integer values. Next, we propose a 2-approximate greedy algorithm that runs in \(O((G.V) \times \log((G.V)))\) time. Finally, we propose two heuristics for the general disinformation problem based on the first two algorithms for the restricted problem.

### 3.1 Exact Algorithm for 1-Level Plans

The exact algorithm for 1-level plans uses dynamic programming to solve the disinformation problem where \(h = 1\). This algorithm is similar to a dynamic programming technique used to solve the 0–1 Knapsack problem and is described in detail in our technical report [17].

Given the cost graph \(G\), the root node \(c_0 \in G.V\), the cost function \(D\), the benefit function \(N\), and the budget \(B\), we first assign sequential ids starting from 1 to the vertices other than \(c_0\) in \(G\). Each subproblem in \((t, i)\) \(t \in \{0, \ldots, |G.V| - 1\}\) \(i \in \{0, 1\}\) is defined as solving the disinformation problem for a subgraph of \(G\) that contains all the vertices up to the id \(i\) along with the edges among those vertices while using the cost budget \(t\). We use a 2-dimensional array \(m\) where \(m[i, t]\) contains the maximum benefit for each subproblem \((i, t)\). In addition, we store the clusters in the optimal disinformation plan for each subproblem in the array \(s\). After running the algorithm, the optimal disinformation plan \(J\) has a total benefit of \(m[|G.V| - 1, B]\).

To illustrate the exact algorithm, suppose we have a cost graph \(G\) that contains the vertices \(c_0, c_1, c_2\) and edges with the weights \(D(c_0, c_1) = 1\), \(D(c_0, c_2) = 2\), and \(D(c_1, c_2) = \infty\). Also, suppose that \(N(c_1) = N(c_2) = 1\), and the cost budget \(B = 1\). We first initialize the arrays \(m\) and \(s\) to 0 (and \(c_0\)), respectively, if either the first or second index is 0. We then solve the subproblem \((1, 1)\). Since the cost \(D(c_0, c_1) = 1\) is within the current budget \(t = 1\), we check if adding \(c_1\) to the plan is beneficial by comparing \(m[0, t - D(c_0, c_1)] + N(c_1) = m[0, 0] + N(c_1)\) (i.e., the benefit for adding \(c_1\) into the plan) with \(m[0, t]\) (i.e., the benefit not for adding \(c_1\) to the plan). Since \(m[0, t] = D(c_0, c_1) + N(c_1) = 1 > m[0, 0] = 0\), we decide to add \(c_1\) to the plan and set \(m[1, 1] = 1\) and \(s[1, 1] = \{c_0, c_1\}\). We then solve the subproblem \((2, 1)\). Since the cost of adding \(c_2\) to the plan is \(D(c_0, c_2) = 2\), which is already larger than the threshold \(t = 1\), we do not add \(c_2\) to the plan and set \(m[2, 1] = m[1, 1] = 1\) and \(s[2, 1] = \{c_0, c_1\}\). Finally, we construct the optimal disinformation plan \(J\) by setting \(J.V = \{2[1] = \{c_0, c_1\}\) and \(J.E = \{c_0, c_1\}\). The optimal benefit is \(m[2, 1] = 1\).

The proof of the correctness and completeness of the exact algorithm can be found in our technical report [17].

**Proposition 3.1.** The exact algorithm generates the optimal disinformation plan with \(h = 1\).

**Proposition 3.2.** The time complexity of the exact algorithm is \(O(|G.V| \times B)\), and the space complexity is \(O(|G.V|^2 \times T)\).

### 3.2 Approximate Algorithm for 1-Level Plans

We now propose a 2-approximate greedy algorithm that runs in polynomial time. The algorithm is similar to a 2-approximation algorithm that solves the 0–1 Knapsack problem. We first add \(c_0\) to the disinformation plan. Then we select the clusters where \(D(c_0, c_i) \leq B\) and sort them by the benefit-per-cost ratio \(\frac{N(c_i)}{D(c_0, c_i)}\) in decreasing order into the list \([c_1, \ldots, c_n]\). We then iterate through the sorted list of clusters and add each cluster to the disinformation plan \(J\) until the current total cost exceeds \(B\). Suppose that we have added the sequence of clusters \([c_1', \ldots, c_n']\) where \(k \leq n\). If \(k = n\) or \(\sum_{i=1}^{n} N(c_i) > N(c_{k+1})\), we return the disinformation plan \(J\) where \(J.V = \{c_0, c_1', \ldots, c_n'\}\) and \(J.E = \{c_0-c_1'\} = \{1, \ldots, k\}\). Otherwise, we return the plan \(J\) where \(J.V = \{c_0, c_k+1\}\) and \(J.E = \{c_0-c_k+1\}\).

For example, suppose we have a cost graph \(G\) that contains the vertices \(c_0, c_1, c_2, c_3\) and edges that have the weights \(D(c_0, c_1) = 1\), \(D(c_0, c_2) = 2\), \(D(c_0, c_3) = 3\), and \(D(c_1, c_4) = 6\). (The other weights are not needed to solve the problem.) Also say that the benefits are \(N(c_1) = 3\), \(N(c_2) = 6\), \(N(c_3) = 6\), and \(N(c_4) = 12\), and the budget \(B = 5\). We first sort the clusters other than \(c_0\) that have a cost \(D(c_0, c_i) \leq B\) by their \(\frac{N(c_i)}{D(c_0, c_i)}\) values in decreasing order. The benefit-per-cost ratios of \(c_1, c_2, c_3, c_4\) are 3, 2, 2, 2, respectively. Since \(c_4\) cannot be in the list because \(D(c_0, c_4) = 6 > B = 5\), we obtain the sorted list \([c_1, c_2, c_3]\). Then we scan the sorted list and add each cluster to the disinformation plan \(J\) until the total cost exceeds \(B\). Since \(N(c_1) + N(c_2) + N(c_3) = 9 > N(c_4) = 6\), we have \(J.V = \{c_0, c_1, c_2\}\) and \(J.E = \{c_0, c_1, c_0-c_2\}\) with a total benefit of 3 + 6 = 9. The optimal solution turns out to be \(J.V = \{c_0, c_1, c_3\}\) and \(J.E = \{c_0-c_2, c_0-c_3\}\) with a benefit of 6 + 6 = 12, demonstrating that the greedy algorithm is not an optimal algorithm.

The proof of the correctness and complexity of the approximate algorithm can be found in our technical report [17].

**Proposition 3.3.** The greedy algorithm generates a 2 approximate optimal disinformation plan with \(h = 1\).

**Proposition 3.4.** The time complexity of the greedy algorithm is \(O(|G.V|^2 \times \log(|G.V|))\), and the space complexity is \(O(|G.V|)\).

### 3.3 Heuristics for General Plans

Since the general disinformation problem (Definition 2.1) is NP-hard in the strong sense, there is no exact pseudo-polynomial algorithm or approximate polynomial algorithm for the problem. Instead, we propose two heuristics that extend the algorithms in Sections 3.1 and 3.2 to produce disinformation plans with no restriction in the length. The full description of the algorithms can be found in our technical report [17].

The first heuristic (called \(EG\)) repeatedly calls the exact algorithm in Section 3.1 for constructing each level of the disinformation plan. As a result, the \(EG\) algorithm always returns a disinformation plan that is at least as good as the best 1-level plan.

To illustrate \(EG\), suppose we are using the cost graph \(G\) in Figure 1 and set \(B = 3\). We initialize the disinformation plan \(J\) by setting \(J.V = \{0\}\) and \(J.E = \{\}\). We then run the exact algorithm for 1-level plans to derive the best 1-level plan. In our example, the result is \(J\) where \(J.V = \{0, 0\}\) and \(J.E = \{0\}\). Then we merge the clusters \([0]\) and \([0]\) within \(G\) and update the edges accordingly. As a result, the new cost graph contains the edges \(\{(t \rightarrow [0, u, v], (r, s) \rightarrow [t, r, s] \rightarrow [t, r, s] \rightarrow [t, u, v]\}\), and the new weights are set as follows: \(D([r, s], [t]) = \min\{D([r], [t]), D([s], [t])\} = 2\) and \(D([r], [u, v]) = \min\{D([r], [u, v]), D([s], [u, v])\} = 4\). We now run the 1-level algorithm again with a remaining budget of 2. As a result, we merge \([t] \rightarrow [r, s]\) and update the cost graph to be \(G.V = \{r, s, t, [u, v]\}\) and \(G.E = \{r, s, t \rightarrow [u, v]\}\). At this point, \([u, v]\) cannot merge with \([r, s, t]\) because there is no budget left. Hence, the final plan is \(J\) where \(J.V = \{r, s, t, [u, v]\}\) and \(J.E = \{r \rightarrow [s], [s] \rightarrow [t]\}\).

The proof for the time and space complexities of \(EG\) can be found in our technical report [17].

**Proposition 3.5.** The time complexity of the \(EG\) algorithm is \(O(|G.V|^2 \times B + |G.V|^3)\), and the space complexity is \(O(|G.V|^2 \times B)\).
Our second heuristic (called AG) extends the greedy algorithm in Section 3.2. Again, we first sort the clusters other than c0 that have a cost \( D(c_0, c_i) \leq B \) by their \( \frac{N(c_i)}{D(c_0, c_i)} \) values in decreasing order. The algorithm then only merges the cluster with the highest benefit-per-cost ratio to the closest cluster in the current plan and updates the edges and the budget just like in the EG algorithm. We repeat the process of sorting the remaining clusters and merging the best one with the closest cluster in the plan until no cluster can be be merged without costing more than the budget.

To illustrate AG, suppose we again use the cost graph \( G \) in Figure 1 and set \( B = 3 \). We first sort the clusters by their benefit per cost ratio. As a result, we get the sorted list \( \{s, t\} \) where \( \{u, v\} \) is not in the sorted list because its cost 4 already exceeds the budget \( B \). We then merge \( \{s\} \) with \( \{r\} \) and create the edges \( \{r, s\} \)–\( \{t, u\} \) and \( \{r, s\} \)–\( \{u, v\} \) with the weights \( D(r, s), \{t, u\} = \min \{2, 4\} = 2 \) and \( D(r, s), \{u, v\} = \min \{4, 4\} = 4 \), respectively. We then re-sort the remaining clusters according to their benefit per cost ratio. This time, we get the sorted list \( \{\{\}\} \) where \( \{u, v\} \) again has a cost of 4, which exceeds the current budget \( B \). We then merge \( \{t\} \) with \( \{r, s\} \) and create the edges \( \{r, s\} \)–\( \{t, u\} \) with the weight \( D(r, s), \{t, u\} = 4 \). Since no cluster can now merge with \( \{r, s\} \), we terminate and return the plan \( J \) where \( J.V = \{(r, \{s\}, \{t\}\)} and \( J.E = \{(r)\} \).

The proof for the time and space complexities of AG can be found in our technical report [17].

**Proposition 3.6.** The time complexity of the AG algorithm is \( O(|G.V|^2 \times \log(|G.V|)) \), and the space complexity is \( O(|G.V|) \).

### 4. Creating New Records

In this section, we discuss how to create new records for disinformation based on existing records. We assume that the records are in a Euclidean space. (In our technical report [17], we also propose various strategies for generating disinformation when the records are not in a Euclidean space.) Suppose the agent is inducing a merge between two clusters \( c_i \) and \( c_j \) in a Euclidean space by creating disinformation records in between. One method is to find the centroids \( r_1 \) and \( r_2 \) of \( c_i \) and \( c_j \), respectively, by averaging the values of the records for each cluster, and then creating new records on the straight line connecting \( r_1 \) and \( r_2 \). For example, if there are two clusters \( c_1 \) : \{\{(X, 20), (Y, 7)\}\} and \( c_2 \) : \{\{(X, 30), (Y, 8)\}, \{(X, 50), (Y, 8)\}\}, then the agent first generates the centroids \( r_1 \) : \{\{(X, 20), (Y, 7)\}\} and \( r_2 \) : \{\{(X, 40), (Y, 8)\}\}. If the agent wants to generate a point exactly in the middle of \( r_1 \) and \( r_2 \) according to the Euclidean space, she can create the record \( t \) : \{\{(X, 30), (Y, 7.5)\}\} by averaging the values for each attribute. If generating one disinformation record is not enough, the agent can further generate disinformation records that are between \( r_1 \) and \( t \) and between \( r_2 \) and \( t \). In our example, the agent can create the disinformation records \( \{(X, 25), (Y, 7.25)\}\) and \( \{(X, 35), (Y, 7.75)\}\). Hence, the agent can easily create disinformation records based on existing values in a Euclidean space.

### 5. Experiments

We evaluate the disinformation planning algorithms in Section 3 on synthetic data (Section 5.1) and then on real data (Section 5.2). We compare the robustness of the ER algorithms defined in Section 2.4. Our algorithms were implemented in Java, and our experiments were run in memory on a 2.4GHz Intel(R) Core 2 processor with 4 GB of RAM.

**Confusion Metric.** We define a confusion metric \( C \) for a cluster \( c \). In the motivating example of Section 1, the adversary was confused into whether the records in the cluster \( c = \{r, s\} \) (excluding the disinformation record) represented the same camera model C300X. However, the correct information of the target entity \( c = C300X \) was \( \{r\} \). We first define the precision \( Pr \) of \( c \) as the fraction of non-disinformation records in \( c \) that refer to \( c \). In our example, \( Pr = \frac{1}{3} \). We also define the recall \( Re \) of \( c \) as the fraction of non-disinformation records that refer to \( c \) that are also found in \( c \). Since there is only one record that refers to the C300X, \( Re = \frac{1}{3} \).

The \( F_1 \) score [8] represents the overall accuracy of the information in \( c \) and is defined as \( \frac{2 \times Pr \times Re}{Pr + Re} \). In our example, the \( F_1 \) score of \( c \) is \( \frac{2 \times \frac{1}{3} \times \frac{1}{3}}{\frac{1}{3} + \frac{1}{3}} = \frac{2}{3} \). Finally, we define the confusion \( C \) of \( c \) as \( 1 - F_1 \) where we capture the notion that the lower the accuracy, the higher the confusion the adversary has on \( c \). In our example, the confusion of \( c \) is \( C(c) = 1 - \frac{2}{3} = \frac{1}{3} \).

**Benefit and Cost Functions.** We define the benefit function \( N \) to return the size \( |c| \) of each cluster \( c \). If we use the plan \( J \) for generating disinformation, we obtain a total benefit of \( \sum_{c \in J.V} |c| \) and a confusion of \( C(c_0 \cup \sum_{c \in J.V} c) \). While maximizing \( \sum_{c \in J.V} |c| \) does not necessarily maximize \( C(c_0 \cup \sum_{c \in J.V} c) \), we will see that we can still obtain a high confusion in practice. In the special case where the recall \( Re \) of the target cluster \( c_0 \) is 1, we can show that maximizing \( C(c_0 \cup \sum_{c \in J.V} c) \) is in fact equivalent to maximizing \( \sum_{c \in J.V} |c| \).

We define the cost function \( D \) to return the number of disinformation records that need to be created when merging two clusters. For example, if we need to generate two disinformation records \( d_1 \) and \( d_2 \) to merge the clusters \( \{r\} \) and \( \{s\} \), then \( D(\{r\}, \{s\}) = 2 \). Notice that the budget \( B \) thus specifies the maximum number of disinformation records that can be generated.

### Disinformation Plan Algorithms

In our experiments, we use the four disinformation planning algorithms defined in Section 3, which are summarized in Table 2.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E2 )</td>
<td>Exact algorithm for 1-level plans</td>
</tr>
<tr>
<td>( EG )</td>
<td>Heuristic extending ( E2 ) for general plans</td>
</tr>
<tr>
<td>( AG )</td>
<td>Greedy algorithm for 1-level plans</td>
</tr>
<tr>
<td>( AG )</td>
<td>Heuristic extending ( AG ) for general plans</td>
</tr>
</tbody>
</table>

### 5.1 Synthetic Data Experiments

We evaluate our disinformation techniques using synthetic data. The main advantage of synthetic data is that they are much easier to generate for different scenarios and provide more insights into the operation of our planning algorithms. In general, there are two types of attributes in records: attributes used for matching records and ones that contain additional properties. For our synthetic data, we only create attributes needed for record matching and do not model the additional properties. We consider a scenario where records in a non-Euclidean space are converted to records in a Euclidean space using a mapping function \( M \). As a result, all the converted records contain real numbers in their attributes. We then run ER and generate the disinformation records in the Euclidean space. The disinformation records could then be converted back into the non-Euclidean space using an inverse mapping function \( M^{-1} \) (See our technical report [17] for more details on generating \( M \) and \( M^{-1} \) functions.) We do not actually use the functions \( M \) and \( M^{-1} \), but directly generate the mapped synthetic records in the Euclidean space.

Table 3 shows the parameters used for generating the synthetic
Table 3: Parameters for generating synthetic data

<table>
<thead>
<tr>
<th>Par.</th>
<th>Description</th>
<th>Val.</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>Number of entities</td>
<td>100</td>
</tr>
<tr>
<td>u</td>
<td>Avg. number of duplicate records per entity</td>
<td>10</td>
</tr>
<tr>
<td>f</td>
<td>Zipfian exponent number of # duplicates</td>
<td>1.0</td>
</tr>
<tr>
<td>d</td>
<td>Number of attributes (dimensions) per record</td>
<td>2</td>
</tr>
<tr>
<td>i</td>
<td>Minimum value difference between entities</td>
<td>50</td>
</tr>
<tr>
<td>v</td>
<td>Maximum deviation of value per entity</td>
<td>50</td>
</tr>
<tr>
<td>g</td>
<td>Zipfian exponent number of deviation</td>
<td>1.0</td>
</tr>
<tr>
<td>t</td>
<td>Record comparison threshold</td>
<td>50</td>
</tr>
</tbody>
</table>

Table 3: Parameters for generating synthetic data

database $R$ and the default values for the parameters. There are $s$ entities in the dataset that are distributed on a $d$-dimensional Euclidean space. For each dimension, we randomly assign the values in the list $[0, i, 2 \times i, \ldots, (s - 1) \times i]$ to the $s$ entities. As a result, any two entities have a distance of at least $i$ from each other for any dimension. For each entity $e$, the data set contains an average of $u$ records that represent that entity, where the number of duplicates form a Zipfian distribution with an exponent of $f$. Each record $r$ generated for the entity $e$ contains $d$ attributes. For each attribute $a$, $r$ contains a value selected from a Zipfian distribution with an exponent of $g$ within the range of $[x - v, x + v]$ where $x$ is the $a$ value of $e$ and $v$ is the maximum deviation of a duplicate record's value from its entity value.

**ER Algorithms.** We experiment on the $HC$ and $SN$ algorithms defined in Section 2.4. The $HC$ algorithm repeatedly merges the closest clusters together until the closest-cluster distance exceeds the comparison threshold $t = 50$. The default value for $t$ was set to be equal to the minimum value difference parameter $i$. When comparing two records $r$ and $s$ across clusters, the match function computes the Euclidean distance between the two records and checks if the distance is within $t$. For example, if $r = \{ (v_1, 1), (v_2, 1) \}$ and $s = \{ (v_1, 2), (v_2, 3) \}$, then the Euclidean distance is $\sqrt{(2 - 1)^2 + (3 - 1)^2} = \sqrt{5}$, which is smaller than $t = 50$. The $SN$ algorithm sorts the records according to their first dimension value (of course, there are other ways to sort the records) and then uses a sliding window of size $W$ to compare the records using a Boolean match function that returns true if the Euclidean distance of two records is within $t$ and false otherwise. The smaller the window size $W$, the fewer records are compared. However, a window size that is too small will prevent $SN$ from properly resolving the records. In our experiments, we set a window size of $W = 20$ so that $SN$ was efficient and yet had nearly identical ER results as the $HC$ algorithm when resolving $R$.

**Target Entity and Cluster.** Before generating the disinformation, we choose one entity as the target entity $e$ and then choose the cluster in the agent's ER result that "best" represents $e$ as the target cluster $c_0$. There may be several clusters that contain records of $e$ when the $ER$ algorithm does not properly cluster the records that refer to $e$. In this case, we set the cluster with the lowest confusion as the target cluster. For example, suppose that the agent's ER result of $R$ is $\{ \{ r_1, r_2, s_1 \}, \{ r_3, s_2 \} \}$ where each record $r_1$ refers to the entity $e_1$ and each record $s_i$ refers to the entity $e_2$. If $e_1$ is our target entity, the confusion values of the two clusters are $1 - \frac{1}{\frac{2}{3} + \frac{2}{3} - \frac{1}{3}} = \frac{1}{5}$ and $1 - \frac{1}{\frac{2}{3} + \frac{2}{3} - \frac{1}{3}} = \frac{1}{5}$, respectively. Since $\frac{1}{5} < \frac{5}{2}$, we set $\{ r_1, r_2, s_1 \}$ as the target cluster $c_0$.

As a default, we choose the entity with the largest number of duplicates to be the target entity $e$. According to our data generation method, there is only one entity that has the most duplicates because of the Zipfian distribution of the number of duplicates per entity. Notice that we are using a worst-case scenario where the many duplicates of $e$ makes it difficult to dilute $e$’s information by merging clusters.

**Disinformation Generation.** When creating disinformation records to merge the two clusters $c_1$ and $c_2$, we first measure the distance between the centroids of the clusters. We then create disinformation records along the straight line in the Euclidean space connecting the two centroids with an interval of at most $t$ so that any two consecutive records along the line are guaranteed to match with each other. For example, if $c_1 = \{ r : \{ (v, 1) \}, s : \{ (v, 3) \} \}$ and $c_2 = \{ t : \{ (v, 7) \} \}$, then the centroid of $c_1$ is $\{ (v, 2) \}$, and the centroid of $c_2$ is $\{ (v, 7) \}$. If the distance threshold $t = 2$, the merging cost is $\left\lceil \frac{2}{3} - 1 \right\rceil = 2$, and we can create the two disinformation records $\{ \{ v, 4 \} \}$ and $\{ \{ v, 6 \} \}$.

We evaluate the four disinformation algorithms in Table 2 on the synthetic data using the $HC$ and $SN$ algorithms. Although not presented here due to space restrictions, we also show in our technical report [17] how the disinformation algorithms perform with restrictions on creating values, on higher-dimensional data, and on larger data.

**5.1.1 ER Algorithm Robustness**

We compare the robustness of the $HC$ and $SN$ algorithms against the $E2$ planning algorithm. (Using any other planning algorithm produces similar results.) We vary the budget $B$ from 100 to 400 records and see the increase in confusion as we generate more disinformation records. Since we choose the target entity as the one with the largest number of duplicates, it takes many disinformation records to significantly increase the confusion. For target entities with fewer duplicates, the increase of confusion is much more rapid (see Section 5.1.5).

Figure 2 shows that the overall confusion results for the $SN$ algorithm are lower than those of the $HC$ algorithm. Initially, the ER results without the disinformation were nearly the same where the $SN$ algorithm produced 105 clusters with the largest cluster of size 195 while the $HC$ algorithm produced 104 clusters with the largest cluster of size 196. However, as we add disinformation records, the $SN$ algorithm shows a much slower increase in confusion, demonstrating that it is more robust to disinformation than the $HC$ algorithm. The main reason is that $HC$ satisfies monotonicity, so clusters are guaranteed to merge by adding disinformation whereas the $SN$ algorithm may not properly merge the same clusters despite the disinformation.

Figure 3 compares the four planning algorithms using the $SN$ algorithm. We can observe in the figure that the $EG$, $E2$, and $A2$ algorithms have similar confusion results. Interestingly, the $AG$ algorithm performs consistently worse than the other three algorithms when the budget exceeds 100. The reason is that the $AG$ algorithm was generating disinformation plans with large heights (e.g., the optimal plan when $B = 200$ had a height of 8), but the $SN$ algorithm was not able to merge all the clusters connected by the plan due to the limited sliding window size. For example, even if two clusters $c_1$ and $c_2$ were connected with a straight line of disinformation records, the records of some other cluster $c_3$ were preventing some of the records connecting $c_1$ and $c_2$ from being compared within the same sliding window.

Another observation is that the confusion plots do not necessarily increase as the budget increases. For example, the confusion of the $A2$ algorithm decreases when $B$ increases from 50 to 100. Again, the reason is that the disinformation was not merging clusters as planned due to the random intervention of other existing clusters.
The frequency of failing to merge clusters strongly depends on the data and sliding window size. That is, if we were to use a database other than $R$, then the $AG$ algorithm could have different confusion results compared to the one shown in the graph. A smaller window size will make the clusters unlikely to merge, which results in lower confusion (see our technical report [17]). We conclude that the 1-level planning algorithms can actually perform better than the general planning algorithms when using the $SN$ algorithm.

For the $HC$ algorithm, all four planning algorithms have plots that resemble the $HC$ plot in Figure 2 (see our technical report [17] for details). The results suggest that the 1-level plan algorithms have confusion performances comparable to the general plan algorithms using the $HC$ algorithm.

5.1.2 Entity Distance Impact

We investigate how the distances among entities influence the confusion results. Figure 4 shows how the accuracies of the $HC$ and $SN$ algorithms change depending on the minimum value difference $i$ between entities using a budget of $B = 200$ records. The closer the entities are with each other (i.e., as $i$ decreases), the more likely the ER algorithm will mistakenly merge different clusters, which leads to a higher confusion. The $HC$ algorithm plot clearly shows this trend. The only exception is when $i$ decreases from 10 to 0. The confusion happens to slightly decrease because some of the records that were newly merged with the target cluster were actually correct records that referred to $e$. The $SN$ algorithm plot becomes increasingly unpredictable as $i$ decreases. The reason is that when merging two clusters with disinformation, there is a higher chance for other clusters to randomly interfere with the disinformation.

5.1.3 Universality of Disinformation

In practice, the agent may not be able to tell which ER algorithm the adversary will use on her database. Hence, it is important for our disinformation techniques to be universal in a sense that the disinformation records generated from the agent’s ER algorithm should increase the confusion of the target entity even if the adversary uses any other ER algorithm. We claim that, as long as the ER algorithm used for generating the disinformation “correctly” clusters the records in the database, the optimal disinformation generated by using the agent’s ER algorithm are indeed applicable when the adversary uses a different ER algorithm.

Figure 5 shows the results of using the disinformation generated when the agent assumes the $SN$ ($HC$) algorithm while the adversary actually uses the $HC$ ($SN$) algorithm. We observe that there is almost no change in the confusion results compared to when the agent and adversary use the same ER algorithms. The reason is that the $HC$ and $SN$ algorithms identified nearly the same entities when resolving $R$, so the disinformation records that was generated were nearly the same as well.

In a worst-case scenario, the ER algorithms of the agent and adversary may produce very different ER results for $R$, leading to different disinformation results as well. However, the different ER results means that one (or both) of the ER algorithms must have incorrectly resolved $R$. Suppose that the agent’s ER algorithm clustered the records correctly and the disinformation was generated using that ER algorithm. Then although the disinformation may not significantly increase the confusion of the the adversary’s (incorrect) ER algorithm, the adversary’s ER algorithm produced a high confusion in the first place, so it is natural that the disinformation cannot further increase the confusion. Thus, as long as we generate the disinformation records from correct ER results, the records can be universally applied to any other correct ER algorithm.

5.1.4 Partial Knowledge

Until now, we have assumed the agent to have a complete knowledge of the database $R$. In reality, the agent may not know all the information in the public. For example, Cakon may not know every single rumor of its new camera model on the Web. Hence, we investigate how the agent only having a partial knowledge of the database influences the confusion results. We first compute the ER result of the $HC$ algorithm on $R$ and select the target cluster. We then take a random sample of the clusters in the ER result (without the target cluster) and add them to the partial information. We then generate the disinformation records based on the target cluster and partial information.

Table 4 shows the decrease in confusion (%) relative to the confusion using the full information without sampling. We vary the sampling rate from 20 to 80%. As the sampling rate goes up, the closer the confusion values are to those of the full information. For example, if we use a budget of 200 records, a sampling rate of 20% decreases the full-information confusion by 13.4% while a sampling rate of 80% only decreases the confusion by 3.4%. Nevertheless, we conclude that the disinformation generated from partial information is still effective.
5.1.5 Target Entities with Fewer Duplicates

In this section, we consider target entities that have fewer duplicates and observe how their confusion values increase against disinformation. The fewer the duplicates, the more rapidly the confusion increases as a result of merging clusters. For example, suppose that Cakon has made an official announcement of a new camera model. With a lot of press coverage (i.e., there are many duplicate records about the model), it is hard to confuse the adversary of this information even with many false rumors. However, if Cakon has not made any announcements, and there are only speculations about the new model (i.e., there are few duplicate records), then it is much easier to confuse the adversary by adding just a few false rumors.

Figure 6 shows the confusion results when we use the entities with the $k$-th most duplicates as the target entities where $k$ varied from 1 to 50. (Recall there is a total of $s = 100$ entities.) For each entity, we measure its confusion against disinformation generated by the $E2$ algorithm using a budget $B$ of at most 10. The other parameters in Table 3 were set to their default values. As a result, the entities with fewer duplicates tend to have a more rapid increase in confusion against the same budget. For example, we only need to generate 3 disinformation records to increase the confusion of the entity with the 50-th largest number of duplicates to 0.53. Our results show that it is easier to confuse the adversary on entities with fewer duplicates.

### 5.2 Real Data Experiments

We now evaluate our disinformation techniques on real data to see how disinformation works in two domains where records are not necessarily in a Euclidean space. Suppose that a celebrity wants to hold an event in a secret location without letting the public know. She might want to confuse the adversary by creating false information about locations. Using this scenario, we experimented on a hotel database where the hotel records simulate possible locations for the secret event. The hotel data was provided by Yahoo! Travel where tens of thousands of records arrive from different travel sources (e.g., Orbitz.com), and must be resolved before they are shown to the users. Each hotel record contains a name, street address, city, state, zip code, and latitude/longitude coordinates. We experimented on a random subset of 5,000 hotel records located in the United States.

Resolving hotel records involves the comparison of multiple non-Euclidean values. In particular, the match function $B_H$ compares two hotel records and first checks if two records differ by their state+city combinations, zip codes, or latitude+longitude combinations. If there is no difference, then $B_H$ computes the string similarities between the names and street addresses of the two records using the Jaro distance [15], which ranges from 0 to 1 and is higher for closer strings. If the two hotel names have a Jaro distance of at least 0.7 while the two street addresses have a distance of 0.95, $B_H$ returns true. Or if the two records have the exact same phone numbers, $B_H$ also returns true. Otherwise, $B_H$ returns false. We use the $HC$ algorithm for resolving records while using a Boolean match function as the distance function. That is, two records have a distance of 0 if they match according to the match function and 1 otherwise. We set the comparison threshold to be 0.5.

Creating the disinformation involves generating non-Euclidean values as well. When inducing a merge between two clusters $c_i$ and $c_j$ with disinformation, we first choose the records $r \in c_i$ and $s \in c_j$ that require the fewest disinformation records to merge $r$ and $s$ together. Then we create a series of disinformation records between $r$ and $s$ where the names and street addresses first resemble $r$ and then gradually resemble $s$. For the attributes other than the name and street address, we simply add the union of the values to all the disinformation records. More details on generating the disinformation records can be found in our technical report [17].

We set the target entity $e$ for the hotel data to be the one with the largest number of duplicates, which is a worst-case scenario where increasing the confusion of $e$ is difficult. The maximum number of duplicates per entity turns out to be 3 because the hotel data was collected from only a few data sources that did not contain duplicates within themselves.

In Figure 7, we evaluate the four disinformation planning algorithms on the hotel records. Since the target cluster only had a size of 3, the confusion of the target entity was sensitive to even a few records merging with the target cluster. For example, using 10 disinformation records, the confusion of the target entity increased to 0.4. The four planning algorithms produce identical confusion results as the budget increases because the cluster sizes were very uniform, so there was little incentive to use multi-level plans so that “far away” clusters would merge with the target cluster. The results show that, even if we generate disinformation on a partial order space, we were still able to significantly increase the confusion for an adversary ER algorithm that uses the Jaro distance for comparing the names and street addresses of hotels.

We also experimented on a comparison shopping database that simulates our Cakon scenario in Section 1 where there are various rumors of items on the Web, and the agent wants to hide the information about a specific product by introducing disinformation. The shopping data was provided by Yahoo! Shopping where millions of records arrive on a regular basis from different online stores and must be resolved before they are used to answer customer queries. Each record contains attributes including the title, price, and cat-

### Table 4: Decrease in confusion (%) with sampling

<table>
<thead>
<tr>
<th>Sampling Rate</th>
<th>50</th>
<th>100</th>
<th>150</th>
<th>200</th>
<th>250</th>
<th>300</th>
<th>350</th>
<th>400</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>0</td>
<td>0.5</td>
<td>1.1</td>
<td>1.2</td>
<td>1.5</td>
<td>1.8</td>
<td>2.2</td>
<td>2.6</td>
</tr>
<tr>
<td>40</td>
<td>0</td>
<td>0.5</td>
<td>0.9</td>
<td>1.1</td>
<td>1.4</td>
<td>1.6</td>
<td>1.9</td>
<td>2.2</td>
</tr>
<tr>
<td>60</td>
<td>0</td>
<td>0.3</td>
<td>0.5</td>
<td>0.7</td>
<td>0.9</td>
<td>1.1</td>
<td>1.3</td>
<td>1.5</td>
</tr>
<tr>
<td>80</td>
<td>0</td>
<td>0.3</td>
<td>0.5</td>
<td>0.7</td>
<td>0.9</td>
<td>1.1</td>
<td>1.3</td>
<td>1.5</td>
</tr>
</tbody>
</table>
category of an item. We experimented on a random subset of 5,000 shopping records that had the string “iPod” in their titles. The full details of the experiments can be found in our technical report [17].

The titles of the shopping records are on average shorter than the names and street addresses of the hotel data, so it was more difficult to create disinformation records that guaranteed the merge of two clusters. In fact, there were cases where two disinformation records could not merge even if they had titles that differed by only one character edit because the titles still did not have a Jaro distance that exceeded the comparison threshold. As a result, the disinformation records occasionally failed to merge clusters, which is illustrated by the decrease of confusion in Figure 8 when the budget increases from 10 to 11. However, the confusion of the target entity eventually increases to high values for larger budgets as shown in the figure. All the four planning algorithms show near-identical performances.

In conclusion, we have shown that our disinformation techniques are effective for two real-world applications where the comparisons of records is sophisticated and involves multiple types of non-Euclidean data. In addition, the 1-level plans perform just as well as the general plans regardless of the application.

6. RELATED WORK

Entity Resolution has been studied under various names including record linkage, merge/purge, deduplication, reference reconciliation, object identification, and others (see [15, 3] for recent surveys). Most work focuses on improving the ER quality or scalability. In contrast, our approach is to dilute the information of ER results by adding disinformation records. Our techniques can be useful when sensitive information has leaked to the public and cannot be deleted.

The problem of managing sensitive information in the public has been addressed in several works. The P4P framework [1] seeks to contain illegitimate use of personal information that has already been released to an adversary. For different types of information, general-purpose mechanisms are proposed to retain control of the data. Measures based on ER [13, 14] have been proposed to quantify the amount of sensitive information that has been released to the public. Reference [7] defines the leakage of information in a general data mining context and provides detection and prevention techniques for leakage. In comparison, our work models the adversary as an ER operator and maximizes the confusion of the target entity.

A recent line of work uses disinformation for managing sensitive information in the public. Reference [9] uses disinformation while distributing data to detect if any information has leaked and to tell who was the culprit. Reputation.com [10] uses disinformation techniques for managing the reputation of individuals on the Web. For instance, Reputation.com suppresses negative information of individuals in search engine results by creating new web pages or by multiplying links to existing ones. TrackMeNot [11] 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 uses disinformation against an ER algorithm to increase the confusion of the target entity.

Clustering techniques that are robust against noise have been studied extensively in the past [2, 16]. Most of these works propose clustering algorithms that find the right clusters in the presence of unnecessary noise. In contrast, we take an opposite approach where our goal is to intentionally confuse the ER algorithm for the target entity as much as possible. The disinformation records we generate can thus be viewed as an extreme case of noise where the ER algorithm is forced to produce incorrect results.

7. CONCLUSION

Disinformation is an effective strategy for an agent to prevent an adversary from piecing together sensitive information in the public. In addition, disinformation can be used to evaluate the robustness of ER algorithms. We have formalized the disinformation problem by modeling the adversary as an Entity Resolution process and proposed efficient algorithms for generating disinformation that induces the target cluster to merge with other clusters. Our experiments on synthetic data show that the optimal disinformation can significantly increase the confusion of the target entity, especially if the ER algorithm satisfies monotonicity. We have shown that the optimal disinformation generated from correct ER results can be applied when the adversary uses a different (but correct) ER algorithm. Our disinformation techniques perform reasonably well even with partial information on the ER results. Also, our techniques are more effective when there are fewer duplicates of the target entity. Finally, we have demonstrated with real data that our disinformation techniques are effective when the records are not in a Euclidean space and the match functions are complex.

8. REFERENCES

[17] XXX. Details omitted due to double-blind reviewing.