Disinformation Techniques for Entity Resolution

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 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 synthetic and real 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.

Disinformation can be viewed as a tool for safeguarding sensitive information that for one or another reason may have "leaked". That is, an adversary may obtain some sensitive information, e.g., preparations for the real Normandy invasion. The disinformation (Calais invasion) causes the adversary either to disregard the true information, or at least to have less confidence in it. Thus, in a way the sensitive information is protected.

Disinformation and this type of information protection are closely related to entity resolution. An *entity resolution* (ER) algorithm takes as input a set of records, and identifies those that refer to the same real world entity. Records that refer to the same entity can be combined to form a more complete picture of the entity. In the invasion example, resolution was done by humans, combining evidence records (e.g., intercepted messages, shipping activity, etc.) to form the adversary's best guess for the invasion. In the absence of disinformation, the better the adversary is at ER, the better he will be at "connecting the dots" and learning our sensitive information. Disinformation is a direct attack on the ER process, confusing its results, and protecting information.

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 [13]. 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. As another example, people search engines like iSearch.com [1] resolve hundreds of millions of records crawled from the Web to create one profile per person. Again, it is very hard to remove what is already on the Web, but it is possible to confuse the ER algorithm. For instance, by creating a fake Web page with person X's name and person Y's address and phone number, we may cause the ER algorithm to combine person X and Y into a single output profile, thus muddling what is published.

We are definitely *not* advocating that everyone should generate disinformation (and as we will discuss, there are also drawbacks to generating disinformation). However, we do believe that anyone using ER needs

Record	Model	Pixels (M)	Price (K)
r	"C300X"	30	8
s	"C300"	20	7
t	"C200"	10	5
u	"C100"	10	1
v	"C100"	10	1

Table 1. Camera Rumor Records

to understand disinformation attacks. That is, if we now take the role of the "adversary", and we have a choice of ER algorithms, we should prefer the one that is more robust against disinformation, i.e., the one that is less easily fooled by disinformation. In this paper we will present a model for disinformation attacks, study the implications for ER, and will illustrate robustness comparisons among ER algorithms.

1.1 Motivating Example

To further motivate our work, let us consider yet another example, in simplified fashion. Say that a camera manufacturer called Cakon is working on its latest camera called the C300X. Cakon needs to keep its new camera secret, for once customers know with certainty the details, they are much less likely to buy the older models (and the new model will not be available for purchase for months). Furthermore, Cakon does not want its competitors to know exactly what they are releasing. However, information about the camera will leak to the public by early testers, component suppliers, or insiders eager to brag. Indeed, there are multiple rumor Web sites that specialize in gathering leaks about upcoming cameras, manually doing ER, and predicting the new models. (Do a Web search for "rumors" plus your favorite camera brand.) As in our previous example, Cakon is unable to eliminate leaked information about the C300X, but it may be able to generate disinformation, e.g., by sending out prototypes with different specifications to testers, or by asking a printer to design a pamphlet with incorrect details.

To be more concrete, say a rumors site (the adversary in this case) has collected five rumor records r, s, t, u, and v as shown in Table 1. (Yes, rumor sites do their resolution manually, probably without even using a database, but please bear with us as we illustrate and formalize the process.) Each record provides the anticipated camera model, number of megapixels (MP), and price.

Say the ER algorithm identifies records that refer to the same real world entity (camera) by computing the similarity in model names, MP counts, and prices. In this case, say all cameras look dissimilar except for u and v. Thus, the ER result is $E_1 = \{\{r\}, \{s\}, \{t\}, \{u, v\}\}$. Here, we use curly brackets to cluster the records that refer to the same entity.

We call the camera manufacturer the *agent*, and say its sensitive information is record $\{r\}$. That is, the new camera model is C300X, with 30M pixels and will 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 a price between 7K and 8K dollars. Suppose that the agent has generated the disinformation record d_1: $\{\langle$Model, "C300X"$\rangle$, $\langle$Pixels, 20$\rangle$, $\langle$Price, 7$\rangle\}$. As a result, when the adversary runs ER, d_1 may match with r because they have the same model name. The ER algorithm can conceptually merge the two records into $r + d_1$: $\{\langle$Model, "C300X"$\rangle$, $\langle$Pixels, 20$\rangle$, $\langle$Pixels, 30$\rangle$, $\langle$Price, 7$\rangle$, $\langle$Price, 8$\rangle\}$ where the '+' operation denotes the merged result of two records. Now $r + d_1$ and s are now similar and might match with each other because they have the same number of pixels and price. As a result, $r + d_1$ and s may merge to produce the new ER result $E_2 = \{\{r, s, d_1\}, \{t\}, \{u, v\}\}$. While r and s were considered different entities in E_1, they are now considered the same entity in E_2 due to 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 5.1 we define a concrete metric for confusion.)

The agent can further increase confusion by causing the target cluster to merge with even more clusters. For example, if the agent generates disinformation record $d_2 = \{\langle$Model, "C300"$\rangle$, $\langle$Pixels, 10$\rangle$, $\langle$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 $E_3 = \{\{r, s, d_1, d_2, t\}, \{u, v\}\}$ where r, s, and t have all merged together.

Our example has illustrated the effects of disinformation, but disinformation also has a cost for the agent. The disinformation must be "believable" so as discussed earlier it may require creating Web sites or prototype cameras. Thus, one of the problems we will address is how to maximize the "confusion" around an entity as much as possible using a total cost for creating the disinformation records within a fixed budget.

As mentioned earlier, one of the main uses of our model and work is as a framework for evaluating the robustness of ER algorithms. Note that in our example, 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. We will use our disinformation techniques to evaluate how sensitive each ER algorithm and threshold combination is to various attacks.

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 several 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:

$$r = \{\langle \text{N}, \text{"C300X"}\rangle, \langle \text{X}, 30\rangle, \langle \text{P}, 8\rangle\}$$

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 $\{\{r, s, t\}, \{u, v\}\}$, where the inner curly brackets indicate the records that refer to the same entity. In this example, two real world entities were identified, with $\{r, s, t\}$

representing the first, and $\{u, v\}$ representing the second. Intuitively, the better 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 positive and commutative. 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 $c_0 \in E(R)$ that represents the information of e the *target cluster*. Intuitively, by merging other clusters in $E(R)$ 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 that contains the largest number of records that refer to e.

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 $c_i \in E(R)$ 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 $r = \{\langle$Model, "C300X"$\rangle$, $\langle$Pixels, 30$\rangle\}$. Then a cluster c containing the records $s = \{\langle$Model, "C300"$\rangle$, $\langle$Pixels, 20$\rangle\}$ and $t = \{\langle$Model, "C200"$\rangle$, $\langle$Pixels, 20$\rangle\}$ has one correct attribute (i.e., $\langle$Model, "C300X"$\rangle$) and three incorrect attributes (i.e., one $\langle$Model, "C200"$\rangle$ and two $\langle$Pixels, 20$\rangle$'s). As a result, the benefit of c 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 $E(R)$ where each edge between the clusters c_i and c_j (denoted as c_i–c_j) has a weight of $D(c_i, c_j)$. We denote the set of vertices in G as $G.V$ and the set of edges as $G.E$. For example, suppose that $E(R) = \{\{r\}, \{s\}, \{t\}, \{u, v\}\}$ and the target cluster $c_0 = \{r\}$. Also suppose that $D(\{r\}, \{s\}) = 1$, $D(\{s\}, \{t\}) = 2$, 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).

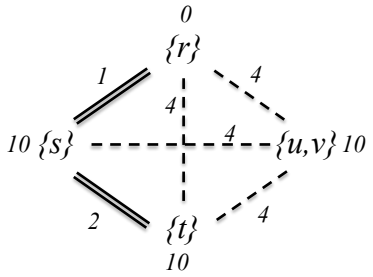

Fig. 1. Cost Graph

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 $D(\{r\}, \{s\}) + D(\{s\}, \{t\}) = 1 + 2 = 3 \leq B$. 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 $\sum_{c_i \in J.V} N(c_i)$ 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 [5], 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.

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

Proof. The decision version of the disinformation problem is to determine whether there exists a disinformation plan J that has a total benefit larger or equal to a given budget B while satisfying the constraints in Definition 2.1. We show that the decision version of the disinformation problem is NP-complete in the strong sense, which means that the problem remains NP-complete even when all of its numerical parameters are bounded by a polynomial in the length of the input. First, the decision version of the disinformation problem is in NP because we can verify any solution by comparing the total benefits and weights with the budgets in linear time. Next, we reduce the decision version of the Set Cover problem, which is known to be NP-complete in the strong sense, to the decision version of the disinformation problem. The Set Cover problem has a finite set U, a family C of subsets of U, and $k \in \mathbb{N}$. The question is whether there is a cover D of U in C with at most k sets. Let $U = \{a_1, \ldots, a_n\}$ and $C = \{s_1, \ldots, s_m\}$. We can construct an instance of the decision version of the disinformation problem as follows. We first create a vertex v_0 for c_0 with benefit 0. We then create m vertices $x_1, \ldots, x_m$ with benefit 0, each of which has an edge from v_0 with weight 1. Finally, we create n vertices $y_1, \ldots, y_n$ with benefit 1. Also, each vertex y_i has an edge from each vertex x_j with a weight of $k+1$ if $a_i \in s_j$. As a result, there is a cover of U in C with at most k sets if and only if the graph G has a subtree J that contains v_0 such that the total benefit of J is at least n and the total weight of J is at most $k + n \times (k + 1)$. Hence, the decision version of the disinformation problem is NP-complete in the strong sense, which makes the optimization version in Definition 2.1 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 $h = 1$ where all the clusters other than c_0 can only be directly connected to c_0 in a disinformation plan, the disinformation problem is NP-hard in the weak sense [5] where there exists a pseudo-polynomial algorithm that returns the optimal disinformation plan.

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

Proof. We first prove that the disinformation problem with $h = 1$ is NP-hard by showing a reduction from the 0–1 Knapsack optimization problem, which is known to be NP-hard in the weak sense. Suppose there are n items $i_1, \ldots, i_n$ with the weights $w_1, \ldots, w_n$ and values $v_1, \ldots, v_n$. Given a capacity C, the 0–1 Knapsack problem tries to find the subset of items that have a total weight within C, but with a maximal total value. We can reduce this problem to a disinformation problem by first creating a cluster c_0 with a benefit of $N(c_0) = 0$. We then create for each item i_j a cluster c_j where $D(c_j, c_0) = w_j$ and $N(c_j) = v_j$. For any subtree J in G, the merge cost is $\sum_{c_i \in J.V \setminus \{c_0\}} w_i$ while the total benefit is $\sum_{c_i \in J.V \setminus \{c_0\}} v_i$. Hence, if we find the optimal disinformation S, the optimal solution for the Knapsack problem is $\{i_j | c_j \in J.V \setminus \{c_0\}\}$. Next, we prove that the disinformation problem with $h = 1$ is NP-hard in the weak sense by reducing it to the 0–1 Knapsack problem. For any instance of the restricted disinformation problem with $h = 1$, we can create an instance of the 0–1 Knapsack problem by creating for each cluster $c_j \neq c_0$ an item i_j that has the value $v_j = N(c_j)$ and weight $w_j = D(c_j, c_0)$. Since the 0–1 Knapsack problem is NP-hard in the weak sense, the disinformation problem with $h = 1$ is NP-hard in the weak sense as well.

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 [7, 9] (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 [6] first 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 on the sorted list of records 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.

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

Proof. We first prove that the HC algorithm is monotonic. According to the algorithm, two records r and s merge with each other if there exists a sequence of records $[r_1(= r), \ldots, r_n(= s)]$ where for each pair (r_i, r_{i+1}) in the path, $D(r_i, r_{i+1}) \leq T$. Thus, even if we add a new record d to R, the records r and s that used to merge will have the same sequence of records that can merge them together. We now prove that the SN algorithm is not monotonic by providing a counter example. Suppose the database is $R = \{r, s\}$ where r and s match with each other. If the window size W is 2 and the sorted list of records is $[r, s]$, the SN algorithm will compare r and s in the same window and cluster them together. Say that we add a disinformation record d that does not match with either r or s, but has a key value between those of r and s. As a result, the SN algorithm will not be able to compare r and s within the same window and thus will not cluster them together.

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

The monotonocity 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 B, 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

Algorithm 1 is an exact algorithm that uses dynamic programming to solve the disinformation problem where $h = 1$. This algorithm is similar to a dynamic algorithm used to solve the 0–1 Knapsack problem. 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 $\{(i, t) |\ i = 0, \ldots, |G.V| - 1$ and $t = 0, \ldots, B\}$ 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

Algorithm 1: Exact algorithm for 1-level plans

input : the cost graph G, the root node c_0, the cost function D, the benefit function N, the cost budget B, the array of the optimal benefits m, the array of the optimal disinformation plans s
output: the optimal disinformation plan J

1 **for** $t = 0, \ldots, B$ **do**
2 $m[0, t] = 0$;
3 $s[0, t] = \{c_0\}$;
4 **for** $i = 0, \ldots, |G.V| - 1$ **do**
5 $m[i, 0] = 0$;
6 $s[i, 0] = \{c_0\}$;
7 **for** $i = 1, \ldots, |G.V| - 1$ **do**
8 **for** $t = 1, \ldots, B$ **do**
9 **if** $D(c_0, c_i) \leq t \wedge m[i-1, t - D(c_0, c_i)] + N(c_i) > m[i-1, t]$ **then**
10 $m[i, t] = m[i-1, t - D(c_0, c_i)] + N(c_i)$;
11 $s[i, t] = s[i-1, t - D(c_0, c_i)] \cup \{c_i\}$;
12 **else**
13 $m[i, t] = m[i-1, t]$;
14 $s[i, t] = s[i-1, t]$;
15 $J.V \leftarrow s[|G.V| - 1, B]$;
16 $J.E \leftarrow \{c_0\text{–}c_i | c_i \in s[|G.V| - 1, B] \setminus \{c_0\}\}$;
17 **return** J;

(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 Algorithm 1, 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 $B = 1$. In Steps 1–6, we initialize the arrays m and s when either the first or second index is 0. We then solve the remaining subproblems in Steps 7–14. In Steps 7–8, we set $i = 1$ and $t = 1$. In Step 9, $D(c_0, c_1) = 1 \leq t = 1$, so we check the second condition. Since $m[i-1, t - D(c_0, c_i)] + N(c_1) = m[0, 0] + 1 = 1 > m[i-1, t] = m[0, 1] = 0$, we set $m[1, 1] = 1$ and $s[1, 1] = \{c_0, c_1\}$. We then continue to Steps 7–8 and set $i = 2$ and $t = 1$. In Step 9, $D(c_0, c_2) = 2 > t = 1$, so we set $m[2, 1] = m[1, 1] = 1$ and $s[2, 1] = \{c_0, c_1\}$. Finally, we continue to Steps 15–17 and construct the optimal disinformation plan J by setting $J.V = s[2, 1] = \{c_0, c_1\}$ and $J.E = \{c_0\text{–}c_1\}$ and then return J. The optimal benefit is $m[2, 1] = 1$.

Proposition 3.1 *Algorithm 1 generates the optimal disinformation plan with $h = 1$.*

Proof. We prove by induction. In the base case, if either i or t is 0, then the only possible disinformation plan is $\{c_0\}$. Now suppose neither i nor t are 0. We need to decide whether to add c_i to the disinformation plan or not. As long as $D(c_0, c_i) > t$, we can compute the optimal benefit including c_i by adding the optimal benefit of the subproblem $(i, t - D(c_0, c_i))$ and $N(c_i)$. The only other option is to compute the optimal benefit of the subproblem $(i - 1, t)$. The larger benefit is clearly the optimal benefit for the subproblem (i, t). Hence, we can derive the optimal benefit and disinformation plan for all subproblems.

Proposition 3.2 *The time complexity of Algorithm 1 is $O(|G.V| \times B)$, and the space complexity is $O(|G.V|^2 \times T)$.*

Proof. The time complexity of Algorithm 1 is $O(|G.V| \times B)$ because the double loop in Steps 7–14 iterates $(|G.V| - 1) \times B$ times. The space complexity of Algorithm 1 is $O(|G.V|^2 \times T)$ because we need to store the optimal disinformation plan for each entry in s. Although not shown in the algorithm, the space complexity can be reduced to $O(|G.V| \times B)$ if we only store the incremental changes in the disinformation plan for each subproblem.

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. We then 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 $[c'_1, \ldots, c'_k]$ where $k \leq n$. If $k = n$ or $\sum_{i=1,\ldots,k} N(c_i) > N(c_{k+1})$, we return the disinformation plan J where $J.V = \{c_0, c'_1 \ldots, c'_k\}$ and $J.E = \{c_0\text{–}c'_i | i = 1, \ldots, k\}$. Otherwise, we return the plan J where $J.V = \{c_0, c'_{k+1}\}$ and $J.E = \{c_0\text{–}c'_{k+1}\}$.

For example, suppose we have a cost graph G that contains the vertices c_0, c_1, c_2, and c_3 with edges that have the weights $D(c_0, c_1) = 1$, $D(c_0, c_2) = 2$, $D(c_0, c_3) = 3$, and $D(c_0, c_4) = 6$. (The other weights are not needed to solve the problem.) Also say that the benefits are $N(c_1) = 3$, $N(c_2) = 6$, $N(c_3) = 6$, $N(c_4) = 12$, 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, 3, 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]. We then scan the sorted list and add each cluster to the disinformation plan J until the total cost exceeds B. Since $N(c_1) + N(c_2) = 9 > N(c_3) = 6$, we have $J.V = \{c_0, c_1, c_2\}$ and $J.E = \{c_0\text{–}c_1, c_0\text{–}c_2\}$ with a total benefit of 3 + 6 = 9. The optimal solution turns out to be $J.V = \{c_0, c_2, c_3\}$ and $J.E = \{c_0\text{–}c_2, c_0\text{–}c_3\}$ with a benefit of 6 + 6 = 12, demonstrating that the greedy algorithm is not an optimal algorithm.

Proposition 3.3 *The greedy algorithm generates a 2-approximate optimal disinformation plan with $h = 1$.*

Proof. Let $d = \sum_{j=1\ldots k} D(c_0, c_j)$ and $b = \sum_{j=1\ldots k} N(c_j)$. Since $d + D(c_0, c_{k+1}) > B$ and all the clusters are sorted by their $\frac{N(c_i)}{D(c_0,c_i)}$ ratios, we conclude that the optimal benefit OPT $< b + N(c_{k+1})$. The greedy algorithm returns a total benefit of $max\{b, N(c_{k+1})\}$ when $k < n$ (if $k = n$, all the clusters are included in the plan, so the greedy algorithm is optimal). Hence, OPT $< b + N(c_{k+1}) \leq 2 \times max\{b, N(c_{k+1})\}$, which makes the solution of the greedy algorithm 2-approximate.

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

Proof. Since the greedy algorithm first needs to sort the clusters in $G.V$, it takes $O(|G.V|^2 \times log(|G.V|))$ time to run. The space complexity is $O(|G.V|)$ since we only need to store the sorted information of clusters.

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 heights.

Algorithm 2 (called *EG*) is a heuristic that repeatedly runs Algorithm 1 for constructing each level of the disinformation plan. As a result, Algorithm 2 always returns a disinformation plan that is at least as good as the 1-level plan generated by Algorithm 1. To illustrate Algorithm 2, suppose we are using the cost graph G in Figure 1 and set $B = 3$. In Steps 1–2, we initialize the disinformation plan J by setting $J.V = \{\{r\}\}$ and $J.E = \{\}$. In Steps 3–5, we set $c = \{r\}$, $G' = G$, and $B' = 3$. In Step 7, we run Algorithm 1 to derive the best 1-level plan for G' and B'. In our example, the result is J' where $J'.V = \{\{r\}, \{s\}\}$ and $J'.E = \{\{r\}\text{–}\{s\}\}$. Since $|J'.V| = 2 > 1$ in Step 8, we continue to Steps 10–20 where we merge the clusters $\{r\}$ and $\{s\}$ within G' and update the edges accordingly. In Step 10, we update the budget B' to $3 - 1 = 2$. In Steps 11–12, we update J by setting $J.E$ to $\{\{r\}\text{–}\{s\}\}$ and $J.V$ to $\{\{r\}, \{s\}\}$. In Step 13, we updated c to the newly merged cluster $\{r, s\}$. In Steps 14–15, we update $G'.V$ to $\{\{r, s\}, \{t\}, \{u, v\}\}$. In Step 16, we remove the unnecessary edges in G' by setting $G'.E$ to $\{\{t\}\text{–}\{u, v\}\}$. In Steps 17–20, we add the new edges

connecting c to the vertices in G' and update the cost function D accordingly. As a result, $G'.E$ becomes $\{\{t\}$–$\{u,v\}, \{r,s\}$–$\{t\}, \{r,s\}$–$\{u,v\}\}$ and the following weights are updated: $D(\{r,s\}, \{t\}) = D(\{t\}, \{r,s\}) = min\{D(\{r\}, \{t\}), D(\{s\}, \{t\})\} = 2$ and $D(\{r,s\}, \{u,v\}) = D(\{u,v\}, \{r,s\}) = min\{D(\{r\}, \{u,v\}), D(\{s\}, \{u,v\})\} = 4$. We now repeat the loop in Steps 6–20 with the updated B', G', and c values. This time, we merge $\{t\}$ to $\{r,s\}$ and get the results $T' = 0$, $G'.V = \{\{r,s,t\}, \{u,v\}\}$, $G'.E = \{\{r,s,t\}$–$\{u,v\}\}$, and $c = \{r,s,t\}$. In the next loop, $\{u,v\}$ cannot merge with $\{r,s,t\}$ because $B' = 0$, so we break the while loop at Step 9 and return J in Step 21 where $J.V = \{\{r\}, \{s\}, \{t\}\}$ and $J.E = \{\{r\}$–$\{s\}, \{s\}$–$\{t\}\}$.

Algorithm 2: Heuristic using Algorithm 1 for general plans

input : the budget B, the cost graph G, the root node c_0, the cost function D, and the benefit function N
output: a disinformation plan J

1 $J.V \leftarrow \{c_0\}$;
2 $J.E \leftarrow \{\}$;
3 $c \leftarrow c_0$;
4 $G' \leftarrow G$;
5 $B' \leftarrow B$;
6 **while** $B' > 0$ **do**
7 $J' \leftarrow$ Alg. 1$(B', G', D, N, c, [], [])$;
8 **if** $|J'.V| = 1$ **then**
9 **break**;
10 $B' \leftarrow B' - \sum_{(c_i,c_j)\in J'.E} D(c_i, c_j)$;
11 $J.E \leftarrow J.E \cup \{c_i$–$c_j | c_i \in J'.V \setminus \{c\}$ and $c_j = argmin_{d\in J.V} D(d, c_i)\}$;
12 $J.V \leftarrow J.V \cup (J'.V \setminus \{c\})$;
13 $c \leftarrow \bigcup_{d\in J.V} d$;
14 $G'.V \leftarrow G'.V \setminus J'.V$;
15 $G'.V \leftarrow G'.V \cup \{c\}$;
16 $G'.E \leftarrow G'.E \setminus \{(c_i, c_j) | (c_i, c_j) \in G'.E \wedge (c_i \in J'.V \vee c_j \in J'.V\})$;
17 **for** $d \in G'.V \setminus \{c\}$ **do**
18 $G'.E \leftarrow G'.E \cup \{c$–$d\}$;
19 $D(d, c) \leftarrow min_{e\in J'.V} D(d, e)$;
20 $D(c, d) \leftarrow D(d, c)$;
21 **return** J;

Proposition 3.5 *The time complexity of Algorithm 2 is $O(|G.V|^2 \times B + |G.V|^3)$, and the space complexity is $O(|G.V|^2 \times B)$.*

Proof. Algorithm 2 has a time complexity of $O(|G.V|^2 \times log(|G.V|))$ because each time a cluster with the highest benefit per cost ratio is merged to c_0, we re-sort the list of remaining clusters. The space complexity is $O(|G.V|)$ because we only need to store a sorted list of clusters in G.

Our second heuristic (called *AG*) extends the greedy algorithm in Section 3.2. Again, 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 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 also update the disinformation plan J by setting $J.E = J.E \cup \{c_i$–$c_j | c_j = argmin_{c\in J.V} D(c, c_i)\}$ and then $J.V = J.V \cup \{c_i\}$. (The ordering of the updates is important.) 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 $\{r,s\}$–$\{t\}$ and $\{r,s\}$–$\{u,v\}$ with the weights $D(\{r,s\},\{t\}) = D(\{t\},\{r,s\}) = min\{2,4\} = 2$ and $D(\{r,s\},\{u,v\}) = D(\{u,v\},\{r,s\}) = min\{4,4\} = 4$, respectively. We then re-sort the remaining clusters according to their benefit per cost ratio. This time, we get the sorted list $[\{t\}]$ where $\{u,v\}$ again has a cost of 4, which exceeds the current budget 2. We then merge $\{t\}$ with $\{r,s\}$ and create the edge $\{r,s,t\}$–$\{u,v\}$ with the weight $D(\{r,s,t\},\{u,v\}) = D(\{u,v\},\{r,s,t\}) = 4$. Since no cluster can now merge with $\{r,s,t\}$, we terminate and return the plan J where $J.V = \{\{r\}, \{s\}, \{t\}\}$ and $J.E = \{\{r\}$–$\{s\}, \{s\}$–$\{t\}\}$.

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

Proof. The *AG* algorithm has a time complexity of $O(|G.V|^2 \times log(|G.V|))$ because each time a cluster with the highest benefit per cost ratio is merged to c_0, we re-sort the list of remaining clusters. The space complexity is $O(|G.V|)$ because we only need to store a sorted list of clusters in G.

4 Creating New Records

In this section, we discuss how to create new records for disinformation based on existing records. At first, we assume that the records are in a Euclidean space. We then propose various strategies for generating disinformation when the records are not in a Euclidean space.

4.1 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_i and r_j 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_i and r_j. 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.

4.2 Non-Euclidean Space

Suppose that the clusters to merge – c_i and c_j – are in a non-Euclidean space, but the agent can still compute the pairwise distances between records. (An ER algorithm typically computes distances or similarities between records.) Again, the idea is to find the records r_i and r_j that represent c_i and c_j, respectively, and then create disinformation records between r_i and r_j. Since the two clusters are not in a Euclidean space, however, the agent can no longer compute the centroids of the clusters. An alternative way to choose r_i and r_j is to identify the clustroids of c_i and c_j based on the pairwise distances between records. Intuitively, a clustroid of a cluster c is the record $r \in c$ that is "closest" to the other points in c. For example, r may have the smallest average distance to the other points in c. Thus if the cluster is c: {{⟨X, "abc"⟩}, {⟨X, "bcd"⟩}, {⟨X, "cde"⟩}}, and the distance between two records is computed by taking the edit distance of their string values, the clustroid is {⟨X, "bcd"⟩}. The agent can also choose the records $r_i \in c_i$ and $r_j \in c_j$ that are closest to (or furthest from) each other.

The agent can create disinformation records between r_i and r_j by mapping them into more restricted spaces (e.g., a Euclidean space). For each attribute, say that the agent wants to find the middle point v_m of two values v_1 and v_2 such that the distances $d(v_1, v_m)$ and $d(v_2, v_m)$ are approximately the same in the non-Euclidean space. We assume there is a mapping function M for each attribute of a record that converts the

values v_1 and v_2 into the values u_1 and u_2 in the restricted space. Once the agent operates in the restricted space, she can more easily find the mid-point u_m where the distance $d'(u_1, u_m)$ is the same as $d'(u_2, u_m)$. The agent can then use an inverse function M^{-1} to map u_m back into a non-Euclidean value, which becomes the desired midpoint v_m.

Suppose the agent is using a Euclidean space as the restricted space, and there are two records containing strings: r: {⟨X, "C300X"⟩} and s: {⟨X, "C300"⟩}. The agent can convert each string into a real number that reflects the string's rank in the alphabetically sorted list of all possible strings. Here, we assume that strings are matched based on their closeness in alphabetical order and not by other measures such as edit distance. Once the agent computes the average value of M("C300X") and M("C300") and converts that value back into a string using M^{-1}, she obtains the string "C300LMM...", so the new record is {⟨X, "C300LMM..."⟩}.

The agent can also create disinformation records in a non-Euclidean space using a total order of values in a dictionary as the restricted space. This mapping is useful if the agent wants to make sure the values in a disinformation record are realistic. In our example above, the agent created the camera model "C300LMM...". However, if the adversary is knowledgeable in camera products, she could easily distinguish the disinformation records from the real ones. The solution is to only use values in a dictionary that contains a sequence of all the possible realistic values in the non-Euclidean space. For instance, if the dictionary is all the valid camera models in alphabetical order, then the model that is closest to "C300LMM..." could be "C300S", so the agent can create the disinformation record {⟨X, "C300S"⟩}.

If the values of existing records cannot be mapped to a total ordering of values in a dictionary, the agent can use a partial ordering of values that form a lattice instead. For example, say that the agent has two records containing sets: r: {⟨X, {1, 2}⟩} and s: {⟨X, {2, 3}⟩}. Also say the agent uses the Jaccard similarity when measuring the distances between sets. As a result, the similarity between r and s is $\frac{1}{3}$. Since sets of elements form a lattice, the agent can assign the common ancestor of the two sets (i.e., the union of the two sets {1, 2, 3}) as the value of the new record. The resulting disinformation record {⟨X, {1, 2, 3}⟩} has a distance of $\frac{2}{3}$ from each record.

Finally, if the agent cannot create new values at all, she can still create new records by only using the values in existing records. For example, if r = {⟨X, "C300"⟩, ⟨Y, 20⟩} and s = {⟨X, "C300X"⟩, ⟨Y, 30⟩}, then she can create the disinformation record {⟨X, "C300"⟩, ⟨Y, 30⟩}.

In general, the more restrictive the mapping space is, the costlier it is to create disinformation records.

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.

5.1 Synthetic Data Experiments

We first 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 disinformation 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 (see Section 4.2). 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 the inverse mapping function M^{-1}. We do not actually use the functions M and M^{-1}, but directly generate the mapped synthetic records in the Euclidean space.

Table 2 shows the parameters used for generating the synthetic 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 $[x, x+v]$ where x is the a value of e and v is the maximum deviation of a duplicate record's value from its entity value.

Par.	**Description**	**Val.**
Data Generation		
s	Number of entities	100
u	Avg. number of duplicate records per entity	10
f	Zipfian exponent number of # duplicates	1.0
d	Number of attributes (dimensions) per record	2
i	Minimum value difference between entities	50
v	Maximum deviation of value per entity	50
g	Zipfian exponent number of deviation	1.0
Comparison Rule		
t	Record comparison threshold	50

Table 2. Parameters for generating synthetic data

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 $r = \{\langle v_1, 1\rangle, \langle v_2, 1\rangle\}$ and $s = \{\langle v_1, 2\rangle, \langle v_2, 3\rangle\}$, 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.

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 e = C300X was $\{r\}$. We first define the precision Pr of c as the fraction of non-disinformation records in c that refer to e. In our example, $Pr = \frac{1}{2}$. We also define the recall Re of c as the fraction of non-disinformation records that refer to e that are also found in c. Since there is only one record that refers to the C300X, $Re = \frac{1}{1}$. The F_1 score [9] 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 $\frac{2 \times \frac{1}{2} \times \frac{1}{1}}{\frac{1}{2} + \frac{1}{1}} = \frac{2}{3}$. 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 $C(c) = 1 - \frac{2}{3} = \frac{1}{3}$.

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_i 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 $1 - \frac{2\times2/3\times2/3}{2/3+2/3} = \frac{1}{3}$ and $1 - \frac{2\times1/2\times1/3}{1/2+1/3} = \frac{3}{5}$, respectively. Since $\frac{1}{3} < \frac{3}{5}$, 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.

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 $C(c_0 \cup \bigcup_{c\in J.V} c)$. While maximizing $\sum_{c\in J.V} |c|$ does not necessarily maximize $C(c_0 \cup \bigcup_{c\in J.V} c)$, 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 $C(c_0 \cup \bigcup_{c\in J.V} c)$ 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 Generation When creating disinformation records to merge the two clusters c_i and c_j, 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 $c_1 = \{r : \{\langle v, 1\rangle\}, s : \{\langle v, 3\rangle\}\}$ and if $c_2 = \{t : \{\langle v, 7\rangle\}\}$, then the centroid of c_1 is $\{\langle v, 2\rangle\}$, and the centroid of c_2 is $\{\langle v, 7\rangle\}$. If the distance threshold $t = 2$, the merging cost is $\lceil \frac{7-2}{2} \rceil - 1 = 2$, and we can create the two disinformation records $\{\langle v, 4\rangle\}$ and $\{\langle v, 6\rangle\}$. In our experiments, we use the four disinformation planning algorithms defined in Section 3, which are summarized in Table 3.

Algorithm	Description
$E2$	Exact algorithm for 1-level plans
EG	Heuristic extending $E2$ for general plans
$A2$	Greedy algorithm for 1-level plans
AG	Heuristic extending $A2$ for general plans

Table 3. Disinformation Plan Algorithms

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 5.1.7).

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.

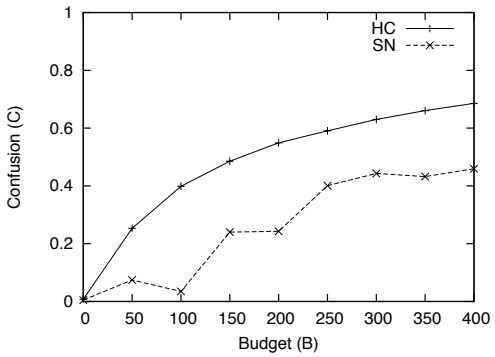

Fig. 2. Robustness of the HC and SN algorithms

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.

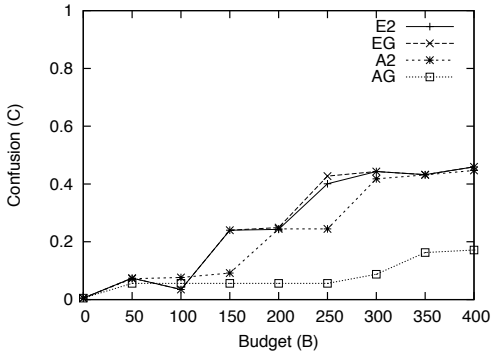

Fig. 3. Comparison of disinformation algorithms

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. We conclude that the 1-level planning algorithms can actually perform better than the general planning algorithms when using the SN algorithm.

To see the relation between the sliding window size of the SN algorithm and the increase of confusion, in Figure 4 we vary the window size from 10 to 50 and observe the confusion of the SN algorithm against the $E2$ disinformation planning algorithm using a budget of $B = 200$ records. As the window size increases, the confusion increases because the disinformation records have a higher chance of merging the clusters they connect.

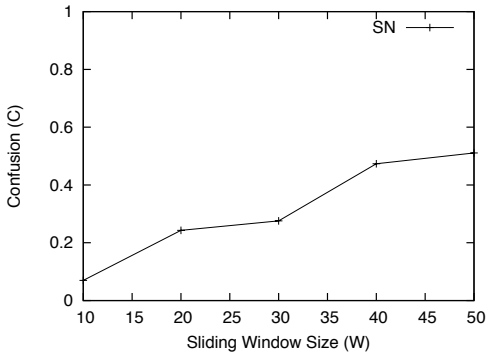

Fig. 4. Window size impact on confusion for SN

We now test the robustness of the HC algorithm against the four disinformation planning algorithms. We vary the budget B from 100–400 records and see the increase in confusion as we generate more disinformation records. Figure 5 shows that all four planning algorithms have smoothly increasing plots where the confusion initially increases rapidly and then slows down as merging large clusters to the target cluster becomes more difficult. The EG algorithm performs slightly better than the $E2$ algorithm while the $A2$ algorithm performs slightly worse than the $E2$ algorithm. The AG algorithm consistently outperforms the three other algorithms by up to a confusion difference of 0.11. Nonetheless, the results suggest that the 1-level plan algorithms have confusion performances comparable to the general plan algorithms using the HC algorithm.

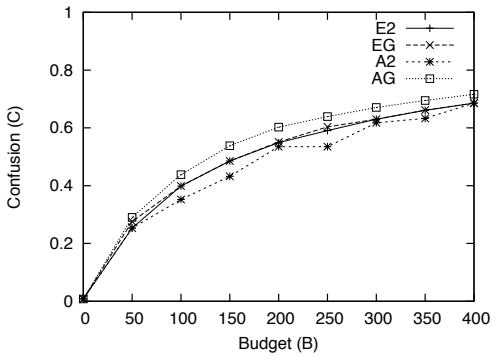

Fig. 5. Robustness of the HC algorithm

5.1.2 Entity Distance Impact We investigate how the distances among entities influence the confusion results. Figure 6 shows how the accuracies of the SN and HC 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.

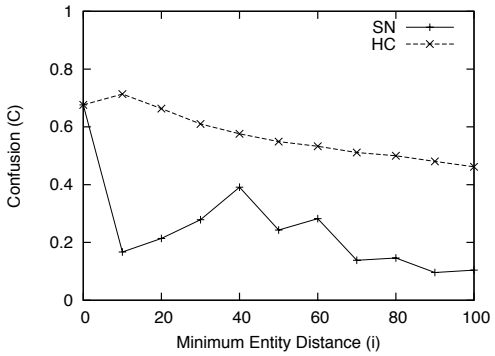

Fig. 6. Entity distance impact on confusion

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 7 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 SN and HC algorithms identified nearly the same entities when resolving R, so the disinformation 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 correctly running 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

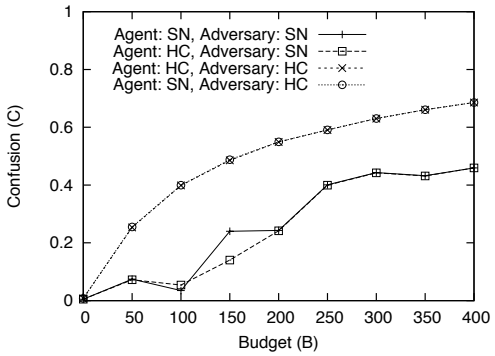

Fig. 7. Universal disinformation

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.

Table 4. Decrease in confusion (%) with sampling

Sampling Rate	Budget							
	50	100	150	200	250	300	350	400
20	15.6	14.1	12.1	13.4	9.6	9.5	10.6	8.2
40	7	10.3	11.8	10.5	9.8	10.2	9.2	8.3
60	3.5	5.4	5	7.2	6.9	6.6	7	7.1
80	0	3.9	3.5	3.4	2.8	2.2	2.6	3

5.1.5 Restricted Values We explore how restrictions in creating values can influence the confusion results. When creating disinformation records, one may need to create values that are realistic to the adversary. For example, when creating fake camera models, the agent may only be able to use a valid model name. In our experiments, we simulate this restriction by only allowing an attribute value of a disinformation record to be multiples of an integer, which we call the granularity. If the granularity is 2, then the disinformation record $\{\langle v_1, 0\rangle, \langle v_2, 2\rangle\}$ is valid, but $\{\langle v_1, 0\rangle, \langle v_2, 3\rangle\}$ is not valid.

When generating disinformation records that connect two centroids r and s, we start from one of the centroids (say r) and create the disinformation record d that satisfies the granularity constraint and has the shortest Euclidean distance to s while being within a Euclidean distance of the comparison threshold t from r. We then repeat the same search starting from d and continue creating disinformation records until the current disinformation record is within a Euclidean distance of t from s. For example, suppose that $r = \{\langle v_1, 0\rangle, \langle v_2, 0\rangle\}$ and $s = \{\langle v_1, 5\rangle, \langle v_2, 3\rangle\}$. Suppose that we only allow the attribute values of disinformation records to be multiples of 2. If the comparison threshold $t = 2\sqrt{2}$ and we start from r, then we create the disinformation record $d_1 = \{\langle v_1, 2\rangle, \langle v_2, 2\rangle\}$, which is the closest record to s that has a distance within $2\sqrt{2}$ from r. We then create the next disinformation record $d_2 = \{\langle v_1, 4\rangle, \langle v_2, 2\rangle\}$. Alternatively, we could have created the record $\{\langle v_1, 4\rangle, \langle v_2, 4\rangle\}$. Since d_2 is within a distance of $\sqrt{2}$ from s, we return the two records

$\{d_1, d_2\}$ as the final disinformation. Table 5 shows the difference in confusion as we increase the granularity from 20 to 40 when we use the HC algorithm. Any granularity larger than the comparison threshold $t = 50$ is not feasible because there is no way to create two disinformation records that match with each other. As the granularity increases, the confusion values decrease as well for the same budget because it takes more disinformation records to merge clusters. When $B = 100$, setting the granularity to 20 temporarily results in a higher confusion than the result without restrictions because there were relatively more disinformation records that were being generated, and these records were merging more clusters to the target cluster beyond what was planned. We conclude that the disinformation generated with value restrictions is still effective unless the restrictions make it absolutely infeasible to create disinformation.

Table 5. Decrease in confusion (%) with restrictions

Granularity	Budget							
	50	100	150	200	250	300	350	400
20	13.1	-2.3	3.8	1.6	3.5	2.3	2.7	2.3
40	26.7	15.8	9.9	8.7	6.8	5.8	5.3	5.2

5.1.6 Number of Dimensions In practice, records may contain more than two attributes and thus be in high-dimensional spaces. To see how our disinformation planning algorithms work for high-dimensional data, we increase the number of dimensions d of the synthetic data. The other parameters in Table 2 were set to their default values. We also increased the comparison threshold t in proportion to $\sqrt{d}$. For example, if t was the threshold used in a 2-dimension space, we use the threshold $\frac{\sqrt{3}}{\sqrt{2}}t$ in a 3-dimensional space.

Figure 8 shows the confusion results when we use the HC algorithm and increase the number of dimensions from 1 to 8. As a result, the confusion values decrease as d increases, but converge for large dimensions. The reason that the confusion values decrease is because the average distance between entities increases slightly faster than the comparison threshold t. As a result, fewer clusters are merged together, so the confusion decreases. For example, suppose that we create three entities in an n-dimensional space. If $n = 1$, then according to our construction method, the three entities have the values 0, i, and $2i$. The average entity distance is always $\frac{4i}{3}$. Now if we create three entities on a 2-dimensional space, we can show that the average entity distance for any assignment of entities is at least $\sqrt{2}\frac{4i}{3}$. For example, one possible assignment of the values could be (0, $2i$), (i, 0), and ($2i$, i). In this case, the average entity distance is $\frac{(2\sqrt{5}+\sqrt{2})i}{3}$, which is larger than $\sqrt{2}\frac{4i}{3}$. Since t only increases by $\sqrt{2}$, the entities are now relatively further away from each other.

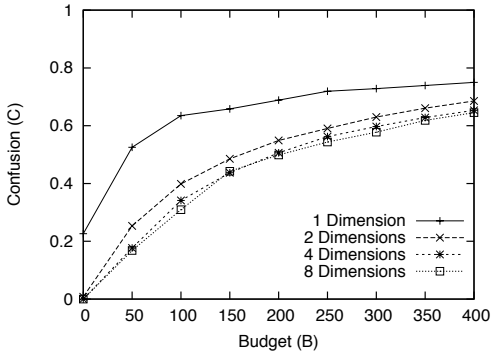

Fig. 8. Number of dimensions impact on confusion

5.1.7 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 9 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 2 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.

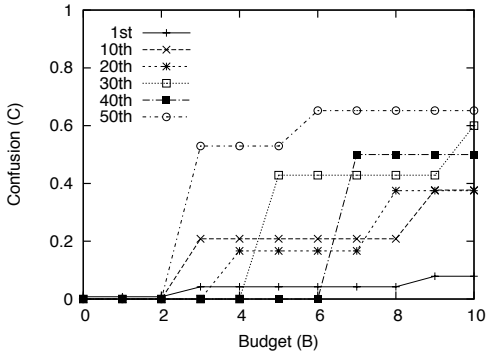

Fig. 9. Entities with fewer duplicates

5.1.8 Scalability We now test the scalability of our techniques on large numbers of entities. Figure 10 shows the runtime increase of the planning algorithms as the number of entities increases from 1,000 to 16,000. The target entity and other parameters in Table 2 were set to their default values. We used the SN algorithm with a window size of 20 as the ER algorithm and set the budget B to 200. As a result, the EG and AG algorithms are 2 to 6x slower than the $E2$ and $A2$ algorithms because of the time to generate the general plans. All the runtimes increase in a quadratic fashion against the number of entities. We conclude that the 1-level plans are practical because they are just as effective as the general plans and can also be generated more efficiently.

5.2 Real Data Experiments

We evaluate our disinformation techniques on real data to see how disinformation works in two domains where records are not necessarily in a Euclidean space.

5.2.1 Hotel data results 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

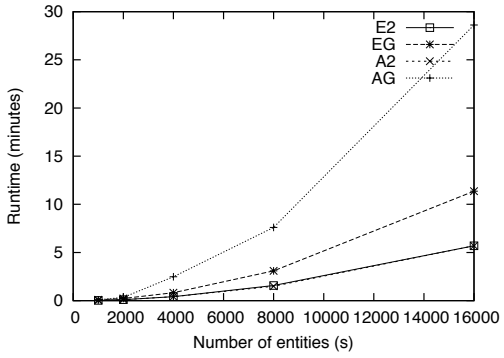

Fig. 10. Scalability of disinformation generation

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 now involves the generation of 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. We then 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.

When creating the names and street addresses for the disinformation records between r and s, it is hard to directly generate the strings using the Jaro distance on the non-Euclidean space. Instead, we use a mapping function M (see Section 4.2) that converts the records into a partial order space where strings are related to each other based on character inserts, deletes, and updates. During the conversion, M does not actually change the strings themselves. In addition, we implicitly use an identity function as the inverted mapping function M^{-1}.

To generate the names and street addresses, we first compute the Levenshtein distance l_n between the names of r and s, which is the minimum number of character inserts, deletes, and updates to convert r's name to s's name. In addition, we set the maximum possible number of edits d_n between consecutive names. We then set d's name as the string that has an edit distance of $min\{l_n, d_n\}$ from r's name and $l_n - min\{l_n, d_n\}$ from s's name. Similarly, we create d's street address using the Levenshtein distance l_a between the street addresses of r and s and the maximum possible number of edits d_a between consecutive street addresses. For example, suppose that $r = \{\langle$name, "Hyatt"$\rangle$, $\langle$street address, "111 Main"$\rangle$, $\langle$phone, 123$\rangle\}$ and $s = \{\langle$name, "Hilton"$\rangle$, $\langle$street address, "222 Main"$\rangle$, $\langle$phone, 456$\rangle\}$. The Levenshtein distances for the names and street addresses are $l_n = 4$ and $l_a = 2$, respectively. If we set $d_n = 2$ and $d_a = 1$, we can create the disinformation record $d = \{\{\langle$name, "Hyaton"$\rangle$, $\langle$street address, "122 Main"$\rangle$, $\langle$phone, 123$\rangle$, $\langle$phone, 456$\rangle\}$. In general, we

need to generate $max\{\lceil \frac{l_n}{d_n} \rceil - 1, \lceil \frac{l_a}{d_a} \rceil - 1, 0\}$ disinformation records. In our experiments, we set $d_n = 37$ and $d_a = 1$.

We set the target entity e_h 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_h 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 11, 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.

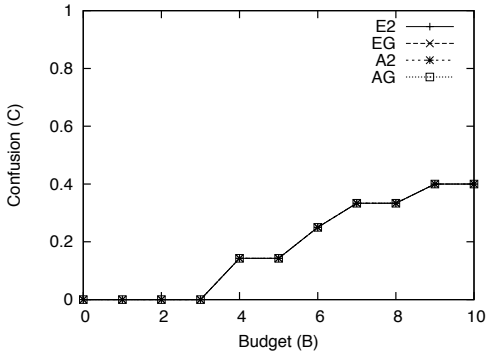

Fig. 11. Hotel data confusion results

5.2.2 Shopping data results 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 category of an item. We experimented on a random subset of 5,000 shopping records that had the string "iPod" in their titles.

The match function B_S compares two shopping records and checks if the two records have similar titles, prices, and categories. When comparing the titles, B_S checks if the Jaro distance is at least 0.85. When comparing the prices, B_S checks if the smaller price is within 33% of the larger price. For the categories, B_S performs an equality check. If one of the three values are not similar enough, then B_S returns false. Again, we use the HC algorithm for resolving records while using a Boolean match function as the distance function.

When creating the first disinformation record d between two shopping records r and s, we first make sure r's price $r.p$ is higher than or equal to s's price $s.p$. If not, we swap r and s. We then create a new title as we do for hotel names based on the Levenshtein distance l_t between the titles of r and s and the maximum possible number of edits d_t between consecutive titles. We also create a new price that is $X\%$ smaller than $r.p$ unless $s.p$ is already within $X\%$ of $r.p$. Finally, we union the categories of r and s and add them to d. For example, suppose that $r = \{\langle$title, "iPod Pink"$\rangle$, $\langle$price, 100$\rangle$, $\langle$category, electronics$\rangle\}$ and

$s = \{\langle$title, "iPod Purple"$\rangle$, $\langle$price, 20$\rangle$, $\langle$category, mp3$\rangle\}$. If $d_t = 2$ and $X = 50$, then we can create the disinformation records $d_1 = \{\langle$title, "iPod Pinkle"$\rangle$, $\langle$price, 50$\rangle$, $\langle$category, electronics$\rangle$, $\langle$category, mp3$\rangle\}$ and $d_2 = \{\langle$title, "iPod Pirple"$\rangle$, $\langle$price, 25$\rangle$, $\langle$category, electronics$\rangle$, $\langle$category, mp3$\rangle\}$. In general, we need to generate $max\{\lceil \frac{l_t}{d_t} \rceil - 1, \lceil log_X \frac{r.p}{s.p} \rceil - 1, 0\}$ disinformation records. For our experiments, we set $d_t = 1$ and $X = 50$.

For the shopping data, we unfortunately did not have a gold standard, so we could not select the target entity based on its number of true duplicates. Instead, we simply assumed that the initial ER result of the shopping data (without any disinformation) was the gold standard. That is, each cluster in the ER result is assumed to contain the exact set of records that refer to a certain entity. We set the target entity e_s for the shopping data to be the one represented by the 10^{th} largest cluster in the ER result, which had a size of 20.

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 12 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.

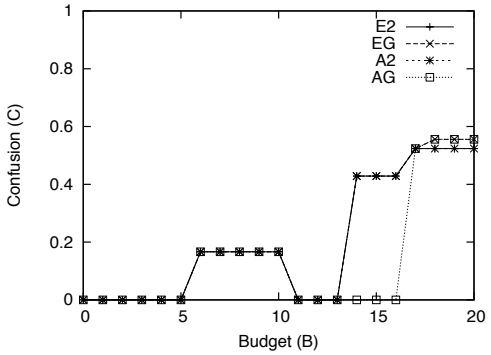

Fig. 12. Shopping data confusion results

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 [4, 15] for recent surveys). Most works focus 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 [2] 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 [14] have been proposed to quantify the amount of sensitive information that has been released to the public. Reference [8] 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 works uses disinformation for managing sensitive information in the public. Reference [10] uses disinformation while distributing data to detect if any information has leaked and to tell who was the culprit. A startup called Reputation.com [11] 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 [12] 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 [3, 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. We have compared the scalability of our disinformation planning algorithms and have shown that 1-level plans can be generated quickly while having confusion results comparable to those of general plans. Finally, we have demonstrated with real data that our disinformation techniques are effective even when the records are not in a Euclidean space and the match functions are complex.

References

1. iSearch. http://isearch.com.
2. G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, N. Mishra, R. Motwani, U. Srivastava, D. Thomas, J. Widom, and Y. Xu. Vision paper: Enabling privacy for the paranoids. In *VLDB*, pages 708–719, 2004.
3. M. Berthold and D. J. Hand, editors. *Intelligent Data Analysis: An Introduction.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1999.
4. A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.
5. M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1990.
6. M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *Proc. of ACM SIGMOD*, pages 127–138, 1995.
7. A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, 1999.
8. S. Kaufman, S. Rosset, and C. Perlich. Leakage in data mining: formulation, detection, and avoidance. In *KDD*, pages 556–563, 2011.
9. C. D. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval.* Cambridge University Press, New York, NY, USA, 2008.

10. P. Papadimitriou and H. Garcia-Molina. Data leakage detection. *IEEE Trans. Knowl. Data Eng.*, 23(1):51–63, 2011.
11. Reputation.com. http://www.reputation.com.
12. TrackMeNot. http://cs.nyu.edu/trackmenot.
13. Wall Street Journal. Insurers test data profiles to identify risky clients, 2011.
14. S. E. Whang and H. Garcia-Molina. Managing information leakage. In *CIDR*, pages 79–84, 2011.
15. W. Winkler. Overview of record linkage and current research directions. Technical report, Statistical Research Division, U.S. Bureau of the Census, Washington, DC, 2006.
16. R. Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.