A Privacy Preserving Efficient Protocol for Semantic Similarity Join Using Long String Attributes

Bilal Hawashin, Farshad Fotouhi
Dept. of Computer Science
Wayne State University
Detroit, MI 48202, USA
{hawashin, fotouhi}@wayne.edu

Traian Marius Truta
Dept. of Computer Science
Northern Kentucky University
Highland Heights, KY 41099, USA
trutat1@nku.edu

ABSTRACT

During the similarity join process, one or more sources may not allow sharing the whole data with other sources. In this case, privacy preserved similarity join is required. We showed in our previous work [4] that using long attributes, such as paper abstracts, movie summaries, product descriptions, and user feedbacks, could improve the similarity join accuracy under supervised learning. However, the existing secure protocols for similarity join methods can not be used to join tables using these long attributes. Moreover, the majority of the existing privacy-preserving protocols did not consider the semantic similarities during the similarity join process. In this paper, we introduce a secure efficient protocol to semantically join tables when the join attributes are long attributes. Furthermore, instead of using machine learning methods, which are not always applicable, we use similarity thresholds to decide matched pairs. Results show that our protocol can efficiently join tables using the long attributes by considering the semantic relationships among the long string values. Therefore, it improves the overall secure similarity join performance.

Categories and Subject Descriptors

H.2.4 [Database Management]: Systems - *Relational Databases*.
K.4.1 [Computers and Society]: Public Policy Issues – *Privacy*.

General Terms

Algorithms, Security, Performance, Experimentation.

Keywords

Similarity Join, Privacy Preserving Protocol, Diffusion Maps, Latent Semantic Analysis, Locality Preserving Projection, Cosine Similarity.

1. INTRODUCTION

Similarity join consists of grouping pairs of records whose similarity is greater than a threshold, *T*. In some cases, one or more sources may refuse partially or totally to share its whole data with other sources during the similarity join process, and only a few researchers have concentrated on performing similarity join under privacy constraints.

Examples of such works includes [5], which introduced a protocol to perform similarity join using phonetic encodings, [3], which proposed a privacy preserving record matching protocol on both data and schema levels, [7], which concentrated on the e-health applications and its intrinsic private nature, and [17], which used a Term Frequency – Inverse Document Frequency (TF.IDF) based comparison function and a secure blocking schema. Other methods concentrated on using encryption to preserve privacy such as [1][2].

To our knowledge, the existing protocols were proposed to perform similarity join under privacy constraints when the join attribute is a short attribute. The term *short attribute* refers to any attribute of *short string* data type, whereas the term *short string* refers to the data type representing any string value of limited length, such as person name, paper title, and address. On the other hand, the term *long attribute* refers to any attribute of *long string* data type, whereas the term *long string* refers to the data type representing any string value with unlimited length, such as paper abstract, movie summary, and user comment.

Obviously, long string values contain much more information than short string values, and we proved that using long string values can improve the similarity join semantic accuracy under supervised learning [4]. Adding to that, most databases include attributes of long string values. However, the previously stated protocols use measurements that are not suitable for such long values. For example, [1] [2] [5] [17] used methods that do not consider the semantic similarities among the string values. While [3] introduced the use of embedded vectors for mapping, their embedding method was applicable to short string attributes. Moreover, the previous work concentrated on using machine learning methods, and such methods are not always applicable. Here, we use similarity thresholds to decide matched records, which are much simpler and of comparable efficiency if used carefully. Finally, the previous methods concentrated on the syntax representations of the string values without considering the underlying semantics. It is worthwhile to find an efficient semantic protocol for joining long

string values under privacy constraints when similarity thresholds are used.
As part of our solution, we compare diffusion maps [9], latent semantic indexing [10], and locality preserving projection [12]. These methods have strong theoretical foundations and have proved their superiority in many applications. Therefore, we compare their performance as candidate semantic similarity join methods for joining long attributes using similarity thresholds. It should be noted that the existing protocols are not included in this comparison because of their high running time cost and low accuracy when applied to long string values. In order to evaluate the performance of our suggested methods, we use two datasets, ***Amazon Products dataset*** [14] and ***IMDB Movies dataset*** [13]. We use various similarity threshold values to define the matched records and evaluate the protocol.

The contributions of this work are as follows:

- Proposing an efficient secure protocol to perform similarity join when the join attribute is a long attribute under privacy constraints, which can improve the secure similarity join accuracy.
- Finding an existing method that can be used efficiently for joining values of long attributes under privacy constraints when similarity thresholds are used.
- Considering the semantic similarities among the string values during the secure similarity join process.
- Our protocol can assist the existing protocols, which are used mainly with short attributes, to improve the overall secure similarity join performance.

The rest of this paper is organized as follows. Section 2 describes the candidate semantic methods to be compared with respect to joining long string values when similarity thresholds are used. Section 3 describes our secure protocol for semantic similarity join. Section 4 represents the experimental part where we compare the previous candidate semantic methods and study the performance of our protocol upon using the best performing methods from the previous comparison. Section 5 is the conclusion

2. SEMANTIC METHODS FOR JOINING LONG STRING VALUES UNDER SIMILARITY THRESHOLDS

In the following subsections, we describe the candidate semantic methods for joining long string values when similarity thresholds are used.

2.1 Diffusion Maps

Diffusion maps is a dimensionality reduction method proposed by Lafon [9]. Initially, a weighted graph is constructed whose nodes are labeled with long string values and whose edge labels correspond to the similarity between the corresponding node values. A similarity function called the kernel function, *W,* is used for this purpose. The first-order neighborhood structure of the graph is constructed using a Markov matrix *P*. In order to find similarities among non-adjacent nodes, forward running in time of a random walk is used. A Markov chain is computed for this purpose by raising the Markov matrix *P* to various integer powers. For instance, according to P_t, the t^{th} power of *P*, the similarity between two long string values *x* and *y* represents the probability of a random walk from *x* to *y* in *t* time steps. Finally, Single Value Decomposition (SVD) dimensionality reduction function is used to find the eigenvectors and the corresponding eigenvalues of $P_{t,t\geq 1.}$ The approximate pairwise long string value similarities are computed using the significant eigenvectors only. The similarity between any two long string values using such a method is called *diffusion maps similarity*. The mathematical details of diffusion maps are given below.

Consider a dataset *V* of *N* long string values, represented as vectors. Let *x*, *y* be any two vectors in $V_{,}$ $1 \leq x, y \leq N$. A weighted matrix W_σ (*x,y*) can be constructed as

$$W_\sigma(x,y) = \exp(-\frac{D\cos(x,y)}{\sigma}), \tag{1}$$

where σ specifies the size of the neighborhoods that defines the local data geometry. As suggested in [11],

$$Dcos(x,y) = 1 - \frac{x.y}{\|x\|.\|y\|}. \tag{2}$$

We can create a new kernel as follows:

$$W_\sigma^\alpha(x,y) = \frac{W_\sigma(x,y)}{q_\sigma^\alpha(x) q_\sigma^\alpha(y)}, \tag{3}$$

Where α deals with the influence of the density in the infinitesimal transitions of the diffusion, and

$$q_\sigma(x) = \sum_{y \in V} W_\sigma(x,y). \tag{4}$$

Suppose $d_\sigma(x) = \sum_{y \in V} W_\sigma^\alpha(x,y)$, (5)

we can normalize the previous kernel to get an anisotropic transition kernel *p*(*x,y*), as follows:

$$\underset{\sigma}{p}(x,y) = \frac{W_\sigma^\alpha(x,y)}{d_\sigma(x)}. \tag{6}$$

$\underset{\sigma}{p}$ *(x,y)* can be considered a transitional kernel of a Markov chain on *V*. The diffusion distance D_t between *x* and *y* at time *t* of the random walk is

$$D_t^2(x,y) = \sum_{z \in V} \frac{(p_t(x,z) - p_t(y,z))^2}{\phi_0(z)}, \tag{7}$$

where ϕ_0 is the stationary distribution of the Markov chain.

After using the SVD operation, the Markov chain eigenvalues and eigenvectors can be obtained. Therefore, the diffusion distance D_t can be written as:

$$D_t^2(x,y) = \sum_{j\geq 1}^{2t} \lambda_j (\varphi_j(x) - \varphi_j(y))^2 . \tag{8}$$

We can reduce the number of dimensions by finding the summation up to a specific number of dimensions z. Thus, the mapping would be:

$$\omega : x \rightarrow (\lambda_1 \varphi_1(x), \lambda_2 \varphi_2(x), ..., \lambda_z \varphi_z(x)) . \tag{9}$$

We set the values of σ and α to 10 and 1 respectively for our experiments, as used in [4].

2.2 Latent Semantic Indexing (LSI)

LSI [10] uses the Singular Value Decomposition operation to decompose the term by long string value matrix M, which contains terms (words) as rows and long string values as columns, into three matrices: T, a term by dimension matrix, S, a singular value matrix, and D, a long string value by dimension matrix. The original matrix can be obtained through matrix multiplication of TSD^T. In order to reduce the dimensionality, the three matrices are truncated to z user selected reduced dimensions. Dimensionality reduction reduces noise and reveals the latent semantics in the dataset. When the components are truncated to z dimensions, a reduced representation matrix, M_z is formed as

$$M_z = T_z S_z D_z^T . \tag{10}$$

Refer to [10] for a detailed explanation of this method.

2.3 Locality Preserving Projection

Locality preserving projection [12] is described briefly as follows. Given a set of long string values represented in the vector space $x_1, x_2, x_3, \ldots, x_n$ in R_m, where m represents the terms. This method finds a transformation matrix A that maps these long values into $y_1, y_2, y_3, \ldots, y_n$ in a new reduced space R_l, $l<m$, such that $y_i = A^T x_i$. This method is particularly applicable when $x_1, x_2, x_3, \ldots, x_n \in O$, where O is a nonlinear manifold embedded in R_m. Refer to [12] for a detailed explanation of this method.

3. SECURE PROTOCOL FOR SEMANTIC SIMILARITY JOIN USING LONG ATTRIBUTES

In this section, our proposed protocol is described. As stated before, this protocol is efficient in joining tables using their long string attributes. Up to our knowledge, no protocols were proposed to be used with such long attributes, and as proved in [4], using such attributes provides a better semantic join accuracy than using short attributes.

In the algorithm, we have two parties A and B, each of which has a relation, R_a and R_b respectively. First, the two parties share the similarity threshold value T that will be used later to decide similar pairs. Next, each party generates the term by long string value matrix from its long attribute, such that each row represents a term (word) and each column represents a long string value. The result is M_a and M_b for A and B respectively. For example, if A contains 1000 paper abstract values in its Paper Abstract attribute, each row in M_a represents a term, and each column represents an abstract. Later, the TF.IDF weighting is applied to both matrices. TF.IDF weighting is commonly used in information retrieval. TF.IDF weighting of a term w in a long string value x is given as follows:

$$\text{TF.IDF}(w,x)=\log(tf_{w,x}+1).\log(idf_w), \tag{11}$$

where $tf_{w,x}$ is the frequency of the term w in the long string value x, and idf_w is $\frac{N}{n_w}$, where N is the number of long string values in the relation, and n_w is the number of long string values in the relation that contains the term w.

Upon applying TF.IDF, both $WeightedM_a$ and $WeightedM_b$ are generated. Every row in this matrix represents a term, every column represents a long string value, and every entry represents the weight of the term in that long string value.

In the next step, both parties share the MeanTF.IDF threshold value [16] to be used and apply the MeanTF.IDF unsupervised feature selection method to both $WeightedM_a$ and $WeightedM_b$. This method assigns a numerical value for each term in both $WeightedM_a$ and $WeightedM_b$. The value of a term w is calculated as follows:

$$\text{Val}(w) = \frac{\sum_{x=1}^{N} TF.IDF(w,x)}{N} , \tag{12}$$

where TF.IDF(w,x) is the TF.IDF weight of the term w in the long string value x, and N represents the number of long string values in the relation. The value of each term represents its importance. The terms with the highest values are the most important terms. It should be noted that terms and features are used alternatively in this context and have the same meaning.

The features from A with the highest values are concatenated with randomly generated features by A and are sent to a third party, C. B does the same. Later, C finds the intersection and returns those shared features, SF, that exist in both parties. Both parties remove their randomly generated features from SF and generate new matrices, SF_a and SF_b, where each row represents an important term from SF, each column represents a long string value, and each entry represents the TF.IDF weighting. Later, every party adds random records to its corresponding matrix to hide its original data. It should be noted that in this step, every record, including the randomly generated ones, is assigned a random index number. The generated matrices, *Rand_Weighted_a* and *Rand_Weighted_b* are sorted according to their index number to guarantee that the randomly generated records are randomly distributed in both matrices. Next, both matrices are sent to C. C performs the semantic operation on both matrices to produce *Red_Rand_Weighted_a* and *Red_Rand_Weighted_b*.These matrices have the concept terms as rows and the long string values as columns. In the experiments section of the paper, we will compare different candidate semantic methods when various thresholds are used, and the best method will be used here. Also, we will study the effect of adding random records on the semantic operation performance in the experimental part. The protocol continues by finding the cosine similarities for all the pairs (x,y), $x \in$ *Red_Rand_Weighted_a* and $y \in$ *Red_Rand_Weighted_b*, and if the cosine similarity is greater than a threshold *T*, the pair of the two vectors is considered a match and inserted into *Matched*. *Matched* is returned to both *A* and *B* to delete the pairs that include randomly generated records. Finally, both parties can share their Matched list after deleting the random records.

```
Algorithm1: Secure Protocol for Semantic Similarity Join
            using Long Attributes
Input:      Two parties A and B, each has a long attribute X.
Output:     Set of matched records sent to both A and B.
Algorithm:
(1) Both A and B share the similarity threshold T to decide
    matched pairs.

(2) A and B generate their term by long string value matrices
    $M_a$ and $M_b$ from $R_a.X$ and $R_b.X$.

(3) TF.IDF weighting is calculated from $M_a$ and $M_b$ to
    generate $WeightedM_a$ and $WeightedM_b$.

(4) Both A and B share the MeanTF.IDF threshold value to
    perform MeanTF.IDF unsupervised feature selection.

(5) Both A and B return their selected features along with
    some randomly generated features to a third party C.

(6) C finds the shared features in both parties, SF, and returns
    them to both A and B.

(7) A and B generate reduced weighted matrices $SF_a$ and $SF_b$
    from $WeightedM_a$ and $WeightedM_b$ using SF after
    removing the randomly generated features.

(8) A generates random records, each of which has SF entries
    and add them randomly to $SF_a$. B does similarly.

(9) Every origional and random record in both $SF_a$ and $SF_b$ is
    assigned a random index number, and both parties keep
    track of the index numbers that belong to the randomly
    generated records.

(10) Both $SF_a$ and $SF_b$ are sorted according to the index
     number to generate Rand_Weighted_a and
     Rand_Weighted_b, which are sent later to C.

(11) C performs the semantic operation to generate
     Red_Rand_Weighted_a and Red_Rand_Weighted_b.

(12) C finds the pairwise cosine similarities among the
     generated two matrices.

(13) If the cosine similarity for a pair is greater than the
     predefined threshold T, this pair is inserted into
     Matched.

(14) C returns Matched to both A and B.

(15) Both A and B delete from Matched the randomly
     generated records.

(16) A and B share their Matched lists.
```

One issue with the algorithm is having a randomly generated feature in the returned *SF*. This could occur when the two parties generate randomly the same feature or when one party generates a random feature that matches an important feature in the other party. In order to calculate the probability of such scenarios, we assume that the randomly generated strings have length up to *k* characters. For a specific length *s*, the number of generated strings is 26^s for English alphabet. Therefore, the probability of generating a string that matches with an existing feature is

$$P = \frac{1}{\sum_{j=1}^{k}(26^j)}, \quad (13)$$

and the probability of generating the same random feature by both parties is P^2.

For example, if we generate lengths up to 5 characters, the probability of the first scenario will be around 10^{-19} and the probability of the second one is 10^{-38}, which are very unlikely. Furthermore, these cases will not affect the running of the algorithm, but will make SF_a and SF_b different in the number of rows (features). However, adding a few features to one matrix will not affect significantly the results because we use the main eigenvectors and eigenvalues in the semantic methods.

4. EXPERIMENTS

In order to evaluate the previous methods on long string values, two datasets were used, Amazon Products Dataset and the IMDB Movies Dataset. Table 1 below describes the use of these datasets in the experimental part.

The following is a brief description of each dataset.

4.1 Amazon Products

We collected 700 records from Amazon website via http://amazon.com. In this work, we are interested in the product descriptions, which provide detailed information about the products. The product descriptions were divided into categories, such as computers, perfumes, cars, and so on. All product descriptions that belong to the same category are considered similar. The total number of categories in the collected dataset is 13 categories. The categories of the collected descriptions were of various complexities, and the number of records in all categories is approximately equal.

4.2 Internet Movies Database (IMDB)

We collected 1000 records from the IMDB Movies database, which is available online via http://imdb.com. Typically, every movie has multiple summaries, written by various users. All summaries that belong to the same movie are considered of the same category. The total number of categories in the collected dataset is 10 categories. As in the previous dataset, the categories of the collected movies were of various complexities, and the number of records in all categories is approximately equal.

For our experiments, we used an Intel® Xeon® server with 3.16GHz CPU and 2GB RAM, with Microsoft Windows Server 2003 Operating System. Also, we used Microsoft Visual Studio 6.0 to read the datasets, Matlab 2008a for the implementations of the candidate semantic methods. For diffusion maps, we used Lafon implementation [9]. Regarding LSI, we used the Matlab svds( ) operation, and for locality preserving projection, we used implementation provided in [15].

Table 1. Datasets Description

Dataset	Used Number of Records	Number of Categories
Amazon Products	700	13
IMDB (Movies)	1000	10

In order to evaluate the performance, we used F_1 measurement, *preprocessing time*, *operation time*, and *matching time*. They are defined as follows:

- F_1 rating is the harmonic mean of the method *recall* and the *precision*. It is given as

$$F_1 = \frac{2*R*P}{R+P}, \qquad (14)$$

where R represents the recall, which is the ratio of the relevant data among the retrieved data, and P represents the precision, which is the ratio of the accurate data among the retrieved data. Their formulas are given as follow:

$$R = \frac{TP}{TP+FP}, \text{ if } TP+FN > 0,$$
otherwise undefined. (15)

$$P = \frac{TP}{TP+FN}, \text{ if } TP+FN > 0,$$
otherwise undefined. (16)

In order to find these measurements, a two-by-two contingency table is used for each category. Table 2 below represents this contingency table.

- Preprocessing time is the time needed to read the dataset and generate matrices that could be used later as an input to the semantic operation.
- Operation time is the time needed to apply the semantic method.
- Matching time is the time required by the third party, *C*, to find the cosine similarity among the records provided by both *A* and *B* in the reduced space and compare the similarities with the predefined similarity threshold.

Table 2. The Contingency Table to Describe the Components of the Recall and Precision

Actual Class	Predicted Class		
		Class = Yes	*Class = No*
	Class = Yes	TP	FN
	Class = No	FP	TN

In phase one, we want to find the best semantic candidate method to be used with long string values when similarity thresholds are used. We compared diffusion maps, latent semantic indexing, and locality preserving projection. As every method is a dimensionality reduction method, we used the optimal number of dimensions for each method that maximizes the F_1 measurement. Figure 1 shows an example of selecting the optimal number of dimensions for diffusion maps experimentally. In that Figure, we found the F_1 measurement for various numbers of dimensions ranging from 5 to 25. We used a fixed similarity threshold value in this case. Obviously, the maximum F_1 measurement was obtained using ten dimensions. The optimal number of dimensions for the remaining methods was calculated similarly. The best number of dimensions for LSI was eight, while it was five for LPP. Therefore, using this optimal number of dimensions for each method will result in the best performance for that method. Figure 2 depicts the comparison of the three methods using various similarity thresholds on IMDB dataset. According to the Figure, both LSI and Diffusion Maps worked efficiently, with advantage given to LSI. The maximum F_1 measurement for LSI was 0.81, with threshold 0.7, while the maximum F_1 measurement for Diffusion Maps was 0.71, with threshold 0.5. Locality Preserving Projection showed the worse performance due to its linear nature.

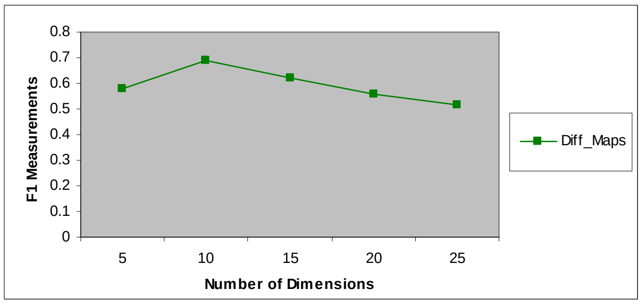

Figure 1. Finding best number of dimensions for diffusion maps experimentally. IMDB dataset was used. The best number of dimensions was 10 dimensions, which resulted in the highest F_1 Measurement.

For Amazon Products dataset, Figure 3 displays the results. Clearly, diffusion maps and LSI outperformed LPP. The performance of LSI dropped significantly in this dataset in comparison with diffusion maps. We concluded from phase one that both diffusion maps and LSI showed efficient performance in joining long string values with advantage given to diffusion maps due to its stable performance.

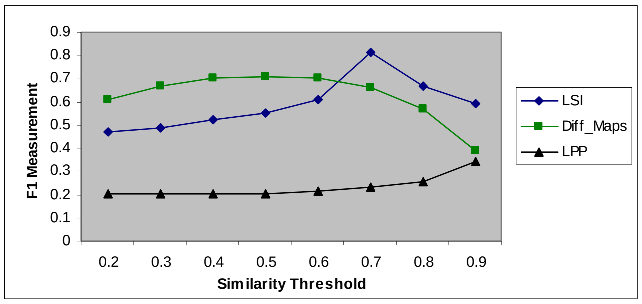

Figure 2. Comparing LSI, diffusion maps, and locality preserving projection to find the best semantic method for long attributes. IMDB dataset was used. Both LSI and diffusion maps showed the best performance.

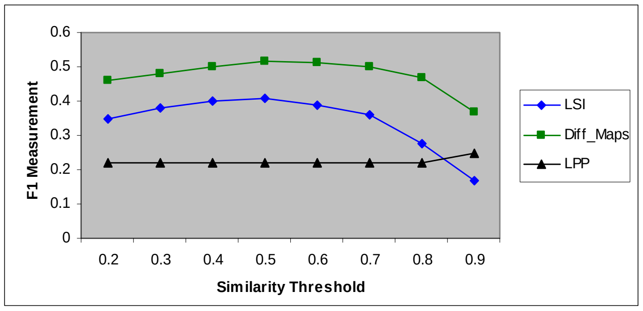

Figure 3. Comparing LSI, diffusion maps, and locality preserving projection to find the best semantic method for long attributes. Amazon Products dataset was used. Diffusion maps showed the best performance.

In phase two of the experimental part, we used diffusion maps and LSI, as they showed the best performance in phase one. We used them separately with our protocol and studied the protocol performance. We used both datasets in this phase. The evaluation measurements used here are preprocessing time, operation time, and matching time. It should be noted that the F_1 measurement for both methods was studied in phase one, where both methods showed efficient performance with advantage given to diffusion maps.

Regarding the preprocessing time, it took 12 seconds to read 1000 records from IMDB, while it took one second to find TF.IDF weighting, and 0.5 second to apply MeanTF.IDF. Time to find shared features by A and B was negligible (approximately zero). For Amazon Products dataset, similar trends were found.

For operation time, Figure 4 represents the results for LSI and diffusion maps with various dimensions in both IMDB and Amazon Products datasets. Obviously, the time needed to perform LSI is less than that in Diffusion Maps. The difference increases with the increase in the number of dimensions. For Amazon Products dataset, similar trends were found.

It is worthwhile to mention that this operation is done once only, and therefore, does not highly affect the protocol performance. Also, it is not necessary to have large number of dimensions for diffusion maps to get the optimal performance. The optimal number of dimensions for diffusion maps in IMDB dataset was ten, while it was five for Amazon Products dataset.

Regarding the matching time, and due to the small number of dimensions used to represent each long string value, this time was negligible, even with the Cartesian product comparison of 5000 records. For Amazon Products dataset, similar trends were found.

Moreover, we studied the effect of adding random records, as stated in steps 8-10 in the algorithm, on the performance of the semantic operation, which is done in step 11. We added various portions of random records that are dataset size dependant, and we found their effect on both the F_1 measurement and the number of suggested matches. Regarding F_1 measurement, the performance increased slightly when small portion of the random records were added, then it started to decrease. This is due to the mechanism of the semantic operation itself. In diffusion maps, the important eigenvalues and eigenvectors are extracted from the dataset. The more random records are inserted, the more their effect on the real eigenvectors and eigenvalues. At some point, the algorithm will extract eigenvector(s) and eigenvalue(s) that represent the random records, which will decrease the accuracy significantly. Figure 5 depicts this step. What was not expected is the slight enhancement in the accuracy upon adding small portion of random records. Theoretically, adding random records will increase the number of features in the kernel matrix, which can make the categories more separable and increase the classification accuracy providing that the eigenvectors are not affected by the added noise. Regarding the number of suggested matches, trivially, increasing the number of records by adding random records will increase the number of candidate pairs, which in turn will increase the suggested matches. Adding random records will increase the number of candidate pairs to be compared, which will increase the number of suggested matches. Adding more random records will consume more time and place more overhead. Figure 6 illustrates this step. Overall, we conclude that adding random records which compose 10% of the whole data size will hide the real data, without much effect on both the semantic operation accuracy and running overhead.

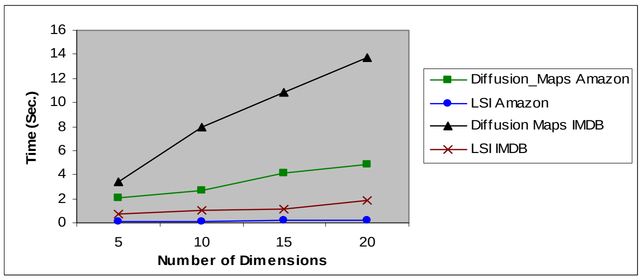

Figure 4. Operation Time for Diffusion Maps and LSI with various number of dimensions. IMDB and Amazon Products datasets were used. Operation time for LSI was less than that in diffusion maps.

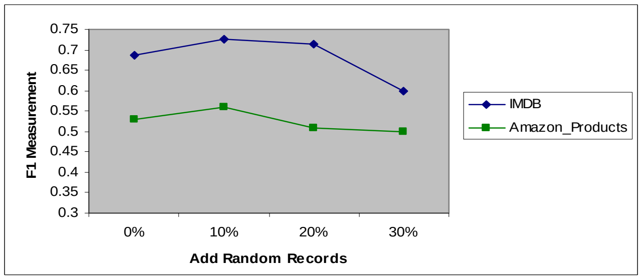

Figure 5. The effect of adding random records on the F_1 measurment upon using diffusion maps. F_1 measurement decreased rapily when the inserted random records size exceedes 10% of the dataset size.

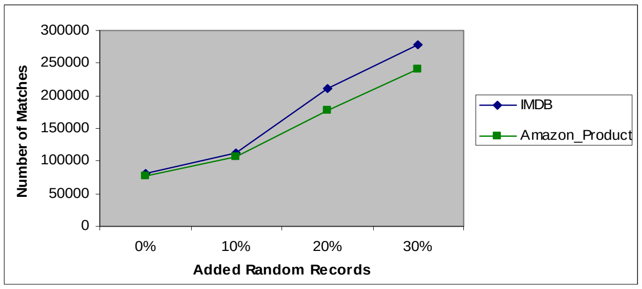

Figure 6. The effect of adding random records on the number of suggested matches upon using diffusion maps. Adding more reocrds introduced more overhead by increasing the number of suggested matched records.

5. CONCLUSION

In this work, we proposed an efficient secure similarity join protocol for long string join attributes. We showed that diffusion maps method provides the best performance, when compared with other semantic similarity methods for long strings. Both mapping into the diffusion map space and adding small portion of randomly generated records can hide the original data without affecting accuracy.

In the future work we intend to compare diffusion maps with other semantic similarity methods such as Laplacian Eigenmap(LapEig) and Local Tangent Space Alignment(LTSP) and study their effect on our protocol. We also aim to study the performance of the protocol on huge databases.

6. REFERENCES

[1] Agrawal, R., Evfimievski, A., and Srikant, R. 2003. Information Sharing Across Private Databases. *In Proceedings of SIGMOD* (San Diego, California, USA, June 09 – 12, 2003), 86-97.

[2] Freedman, M.J., Nissim, K., and Pinkas, B. 2004. Efficient Private Matching and Set Intersection. *In Proceedings of EUROCRYPT* (Interlaken, Switzerland, May 02 – 05, 2004), 1-19.

[3] Scannapieco, M., Figotin, I., Bertino, E., and Elmagarmid, A. K. 2007. Privacy Preserving Schema and Data Matching. *In Proceedings of SIGMOD* (Beijing, China, June 12 – 14, 2007), 653-664.

[4] Hawashin, B., Fotouhi, F., and Grosky, W. 2010. Diffusion Maps: A Superior Semantic Method to Improve Similarity Join Performance. *In Proceedings of ICDM Workshops* (Sydney, Australia, December 15-17 , 2010), 9-16.

[5] Karakasidis, A. and Verykios, V.S. 2009. Privacy Preserving Record Linkage Using Phonetic Codes. *In Proceedings of BCI* (Thessaloniki, Greece, September 17 – 19, 2009), 101-106.

[6] Hjaltason, G.R. and Sarnet, H. 2003. Properties of Embedding Methods for Similarity Searching in Metric Spaces. *IEEE TPAMI* 25, 5(May 2003), 530-549.

[7] Churces, T. and Christen, P. 2004. Some Methods for Blindfolded Record Linkage. *BMC Medical Informatics and Decision Making*, 4,9 (June 2004), 1-17.

[8] Elmagarmid, A., Panagiotis, G., and Verykios, S. 2007. Duplicate Record Detection: A Survey. *IEEE TKDE,* 19,1 (January 2007), 1-16.

[9] Coifman, R.R. and Lafon, S. 2006. Diffusion Maps. *Applied and Computational Harmonic Analysis* 12,1 (July 2006), 5-30.

[10] Deerwester, S., Dumais, S., Furnas, G., Landauer, T., and Harshman, R. 1990. Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*, 41, 6 (September 1990), 391-407.

[11] Ataa-Allah, F., Grosky, W. I., and Aboutajdine, D.2008. Document Clustering Based on Diffusion Maps and a Comparison of the k-Means Performances in Various Spaces. *In Proceedings of IEEE Symposium on Computers and Communications (ISCC)*(Marrakech, Morocco, July 6-9, 2008), 579-584.

[12] He, X. and Niyogi, P. 2003. Locality Preserving Projections. *Advances in Neural Information Processing Systems*(2003).

[13] IMDB Website: http://imdb.com.

[14] Amazon Website: http://amazon.com.

[15] http://www.zjucadcg.cn/dengcai/Data/data.html

[16] Tang, B., Shepherd, M., Milios, E., and Heywood, M. 2005. Comparing and Combing Dimension Reduction Techniques For Efficient Test Clustering. *In Proceedings of SIAM Workshops* (Newport Beach, CA, April 21-23, 2005).

[17] Al-Lawati, A., Lee, D., McDaniel, P. 2005. Blocking-aware Private Record Linkage. *In Proceedings of ACM SIGMOD workshops*(Baltimore, MD, USA, June 17, 2005), 59-69.