# Similarity Joins of Text with Incomplete Information Formats

Shaoxu Song and Lei Chen

Department of Computer Science
Hong Kong University of Science and Technology
{sshaoxu,leichen}@cs.ust.hk

**Abstract.** Similarity join over text is important in text retrieval and query. Due to the incomplete formats of information representation, such as abbreviation and short word, similarity joins should address an asymmetric feature that these incomplete formats may contain only partial information of their original representation. Current approaches, including *cosine similarity with q-grams*, can hardly deal with the asymmetric feature of similarity between words and their incomplete formats. In order to find this type of incomplete format information with asymmetric features, we develop a new similarity join algorithm, namely **IJoin**. A novel matching scheme is proposed to identify the overlap between two entities with incomplete formats. Other than *q-grams*, we reconnect the sequence of words in a string to reserve the abbreviated information. Based on the asymmetric features of similar entities with incomplete formats, we adopt a new similarity function. Furthermore, an efficient algorithm is implemented by using the join operation in SQL, which reduces pairs of tuples in similarity comparison. The experimental evaluation demonstrates the effectiveness and the efficiency of our approach.

## 1 Introduction

Similarity Join is an important operation in data cleaning and data integration [4]. It has been studied by various aspects and referred by a variety of names, including record linkage [7], entity identification [9] and approximate join [5]. The key issue is to identify whether two entities (e.g., relational tuples) are approximately the same[7]. Owning to the poor data quality with various errors caused by human factors and technique problems(e.g., database system problems), it is difficult to identify the same entities exactly by traditional join operation in SQL. For example, "International" and "Intenational" with spelling mistakes do not match exactly. By using *edit distance* [10], we can deal with the spelling mistake. Furthermore, due to various kinds of formats in representing information, it becomes even harder to detect approximate entities, such as different orders ("Shaoxu song" with given name first and "'SONG, Shaoxu" with surname first). *Cosine similarity with q-grams* [6] is used in dealing with block orders, which is also effective in spelling mistakes. In addition to effectiveness, efficiency is another key issue. A recent work [2] applied a join operation in

© Springer-Verlag Berlin Heidelberg 2007

SQL to identify entities with overlaps first, which reduced the times of comparisons between unrelated entities greatly. They proposed an efficient similarity join operators that can be used in many similarity functions.

However, the current existing approaches, including cosine similarity with *q-grams*, can hardly deal with the similarity between words and their incomplete formats, such as abbreviation, short word and incomplete information. For instance, there are various representing formats in bibliography references, including abbreviation ("VLDB" for "Very Large Databases"), short word ("Conf." for "Conference"), incomplete information ("In VLDB" for "In Proceedings of VLDB"). Those incomplete formats have a common asymmetric feature that incomplete formats contain only partial information of their original representation, which make it difficult to find and quantify the similarity between words and their incomplete formats. Current *q-grams* approach cannot identify the abbreviation information, while *cosine similarity* is not effective in dealing with such asymmetric features. There is a solution by Rohit [1] which tries to handle the abbreviation case. However, it needs several attributes with hierarchies, such as, County, State, City and Street. So it is NOT a common solution without humans domain knowledge.

In this paper, we propose a novel text similarity join approach, **IJoin**, to address both the effectiveness and efficiency issues of identifying the similarity between text entities with incomplete formats in similarity joins. In text matching, we connect first letter of each word to reserve potential abbreviation information and enhance the importance of the first few grams to find high similarity between words and their short formats. We also consider the asymmetric features of similarity between entities with incomplete formats in similarity function. Our contributions in this paper are summarized as follows:

(1) We propose a novel matching scheme to identify overlaps between text entities and their incomplete formats;
(2) We design a similarity function which can calculate the similarity of entities with asymmetric features when incomplete formats exist;
(3) We present an efficient implementation of our similarity joins algorithm which uses the join operation in SQL.

The rest of the paper is organized as follows. Section 2 summarizes text matching schemes, and introduces our IJoin matching approach. Section 3 presents our IJoin similarity function. In Section 4, we illustrate the basic and extended implementations of our IJoin approach. Section 5 reports the experimental results on the effectiveness and scalability of IJoin. Finally, we conclude in Section 6.

## 2 Distance Based Matching

In this section, we first summarize text matching schemes in similarity joins. Then, we illustrate our IJoin matching scheme, considering asymmetric features of incomplete information formats.

In similarity joins, we consider two relations (e.g. $R$ and $S$) with common attributes (e.g. $R.A$ and $S.A$). The main issue is to find tuples in $R$ and $S$ with exactly same or approximate values of the common attribute $A$. If the similarity between two tuples $r$ and $s$ from $R$ and $S$ respectively, satisfies the user specified similarity threshold $\eta$, then these two tuples will be the join result. Different from *edit distance* [10] based measure, we first map text strings in tuples to a set of elements as entity features. Based on the overlap of the sets, we calculate the similarity value by certain similarity functions (discussed in Section 3).

## 2.1 Existing Matching Approaches

One common text matching approach used in text retrieval, is to map a string to a set of word tokens [3]. For example, the string "Computer Science Department" can be mapped to a set of words, {'Computer', 'Science', 'Department'}. The word token based matching scheme can identify the similarity of same entities with different representing orders, where each word is treated as a block. A block move in the string affects the mapping set slightly. The strings "Computer Science Department" and "Department of Computer Science" have high overlap in their mapping sets. However, the word based approach is not effective in dealing with spelling errors. A spelling error, such as "Conputer", may affect the similarity of strings significantly.

Another widely used matching method is *q-grams*, which cuts a string into several substrings of length $q$. For the same example, "Computer Science" can be mapped to {'Com', 'omp', 'mpu', 'put', 'ute', 'ter', 'er ', 'r S', ' Sc', 'Sci', 'cie', 'ien', 'enc', 'nce'}. The *q-grams* method is more robust under spelling mistakes and keeps high similarity with different representing orders [6]. The spelling mistake of "Conputer" only affect three grams {'Con', 'onp', 'npu'}, which take up small parts of the whole string.

Different tokens in the mapping sets may have different ability of discrimination. For example, the token 'ing', which appears frequently in words, may have lower significance in discriminating different strings. Tokens are always associated with weights to represent their importance in a string, where text retrieval techniques are commonly used, like Inverse Document Frequency (IDF).

The approaches mentioned above can process entities without many incomplete information formats, but cannot identify the similarity between words and their abbreviation formats. For example, there is even no overlap at all between the q-grams of "Computer Science Department" and its abbreviation "CSD".

## 2.2 Matching of Incomplete Formats

We consider two kinds of incomplete information formats, the abbreviation and short words, in our IJoin matching. For the abbreviation, we generate several new elements by connecting the first letters of each word to reserve potential abbreviation information. We also enhance the importance of first few letters in each word by using a decay factor to enlarge the similarity between words and their short formats. The matching steps are described as follows:

***Step 1.*** In order to be robust under different representing orders, we first cut a string into word tokens. Other than *q-grams*, we do not record any information of word order, such as the connecting token 'r S' between two words in the *3-grams* sets of "Computer Science".

***Step 2.*** We cut each word into *q-grams* to in order to deal with spelling errors. When we search a word in the dictionary, we can find the word with high probability by looking up the first few letters of it. Motivated by this, we associate the first few letters with higher weight than the other ones, by setting a decay factor $\gamma (0 < \gamma \leq 1)$. The weight of *k-th* gram $g_k$ in a word is:

$$w(g_k) = w(g_{k-1}) \cdot \gamma \tag{1}$$

All grams are ordered by the sequence of letters in the word. Note that short word is always few several letters in its original word (e.g. "Conf." for "Conference"). The decay factor can increase the weight of overlap between the short word and the original one, since we associate higher weight to first few letters (e.g. $w('Con') > w('onf')$ in the word "Conference").

***Step 3.*** We reconnect the first letters in a string in order to identify the abbreviation of the string. Then, the string of first letters is mapped into *q-grams*. Each gram in this step has equal importance to represent abbreviation, so we do not take decay factor here.

Table 1 shows an example of matching scheme in our IJoin approach. The length of abbreviation word is probably short, so the decay factor does not affect the weight of abbreviation word significantly.

**Table 1.** Matching scheme in IJoin

| *String 1, String 2* | | {Computer | Science | Department} | , | {CSD} |
|---|---|---|---|---|---|---|
| *Q-grams of first letters* | | {CSD, | | | | |
| *Q-grams of each word* | $g_1$ | Com , | Sci , | Dep , | | {CSD} |
| | $g_2$ | omp , | cie , | ept , | | |
| | ... | ..., | ..., | ..., | | |
| | $g_k$ | ter , | nce , | ent } | | |

## 3 Similarity Function

In this section, we illustrate our similarity function in IJoin. After mapping strings to sets with associated weight of elements, tuples (or entities) can be represented by *vector-space model* [11]. Each tuple $t$ is represented by a vector of weights of $p$ grams (or tokens):

$$t_i = (g_{i1}, \ldots, g_{ip}) \tag{2}$$

where $t_i$ is the vector of tuple $i$ and $g_{ik}$ is the weight of gram $k$ in tuple $i$.

### 3.1 Cosine Similarity

The similarity between tuples can be quantified by the correlation, $\phi_{ij}$, which is so-called the *Cosine Measure*:

$$\phi(t_i, t_j) = \frac{|t_i \bigcap t_j|}{|t_i \bigcup t_j|} = \frac{\sum_{k=1}^{p} g_{ik} g_{jk}}{\sqrt{\sum_{k=1}^{p} g_{ik}^2 \sum_{k=1}^{p} g_{jk}^2}} \tag{3}$$

where $\phi(t_i, t_j)$ is the cosine similarity value between tuples $t_i$ and $t_j$. Clearly, a measure based on cosine similarity can be used when all terms are measured on the same scale. However, cosine similarity is not so effective in dealing with asymmetric features of similarity between words and their incomplete formats. Abbreviation and short word can hardly have high similarity value with their original representing formats.

### 3.2 IJoin Similarity

Asymmetric features exist in the similarity measure between words and their incomplete formats. Considering the asymmetric features that incomplete formats only take up slight parts of the original information, we calculate the similarity in IJoin as follows:

$$\varphi(t_i, t_j) = \frac{|t_i \bigcap t_j|}{\min(|t_i|, |t_j|)} = \frac{\sum_{k=1}^{p} g_{ik} g_{jk}}{\min(\sum_{k=1}^{p} g_{ik}^2, \sum_{k=1}^{p} g_{jk}^2)} \tag{4}$$

Other than computing the total weight of two tuples, we use only the smaller one of them. Incomplete formats, such as abbreviation and short word, can keep high similarity with their original formats.

Let us see the previous example in Table 1, where the overlap of two strings' 3-grams is {'CSD'}. We assume that the weight of each gram to be 1. According to the formula (3), the cosine similarity between these two strings is $\phi(string1, string2) = 1/\sqrt{20} = 0.223$. In fact, the cosine similarity value equals to 0 (no overlap), if the IJoin matching is not adopted. For our IJoin similarity function, the similarity value is $\varphi(string1, string2) = 1$.

## 4 Algorithm Implementation

In this section, we first introduce a basic implementation of our IJoin approach. Then, we discuss a more efficient way to process similarity joins with incomplete information.

### 4.1 Basic IJoin Implementation

Given two relations $R$ and $S$, the similarity join operation returns all pairs of tuples $r$ and $s$ from $R$ and $S$ respectively, which satisfy the similarity threshold $\eta$ (e.g. $\phi(r, s) \geq \eta$). During the preprocessing, we apply the decay factor $\gamma$ to each gram of words. *Q-grams* of the words' first letter sequence are added to reserve the potential information of abbreviation.

Finally, we compare each pair of tuples from two relations based on the similarity function in Formula (4). The number of candidates of pairs to be compared is large, especially when the data scale increases. An index of tuples may reduce the accessing time, however, the number of comparisons cannot be decreased. We will discuss a solution to reduce the number of candidates of comparing pairs in the next section.

## 4.2 Extended IJoin Implementation

In order to reduce the candidates of comparisons, we need to filter those pairs of tuples with no relevancy. A recent study [2] develops a method, namely SSJoin, by exploiting some new attributes for each tuple, which can be used to decide whether two tuples are relevant. The relation $R(A)$ is extended to $R(A, B)$, where $B$ is one of $A$'s *q-grams*. For example, the string "Computer Science Department" can be extended to 20 tuples as follows:

**Table 2.** Tuples in SSJoin

| R.A String | R.B 3-grams |
|---|---|
| Computer Science Department | Com |
| Computer Science Department | omp |
| Computer Science Department | mpu |
| ... | ... |
| Computer Science Department | men |
| Computer Science Department | ent |

Then, a join operation in SQL (exactly matching) is processed on attribute $R.B$, which finds pairs of tuples with common grams. Those tuples without common grams are totally irrelevant and filtered out. Although the rest pairs of tuples are relevant with common tuples, the number of candidate tuples is still large. The authors use a prefix-filter to reduce duplicate pairs.

In order to improve the efficiency, we do not consider all *q-grams* of $A$ in $B$ like the basic SSJoin. As mentioned before, we can detect a word probably by the first few letters of it. Therefore, in order to improve the efficiency, we consider the first *q-grams* of each word only in the first SQL join operation. This predigest operation will not miss the relevancy between words and their short word formats (e.g. "Conference" and "Conf." have the common gram "Con"). Furthermore, we also add *q-grams* of $A$'s first letters into $B$, in order to reserve the potential relevancy between strings and their abbreviation. Table 3 shows an example of tuples in our Extended IJoin approach.

After mapping $R(A)$ and $S(A)$ to $R(A, B)$ and $S(A, B)$ respectively, we operate a SQL join on the attribute $B$. The result is a group of tuples $(R.A, S.A)$

**Table 3.** Tuples in IJoin

| R.A String | R.B 3-grams |
|---|---|
| Computer Science Department | Com |
| Computer Science Department | Sci |
| Computer Science Department | Dep |
| Computer Science Department | CSD |

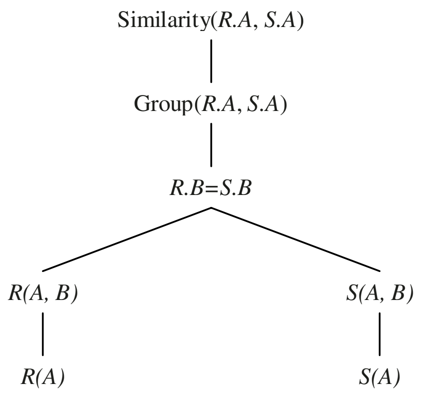

**Fig. 1.** Extended IJoins steps

with common grams. Each pair$(r, s)$ in the result can get its similarity value $\varphi(r, s)$ by using the similarity function (4). If the similarity value satisfies the user specified threshold $\eta$, $(r, s)$ will be the final join result. Fig 1 shows the process of similarity joins by Extended IJoin.

## 5 Experimental Evaluation

In this section, we illustrate the results of our experiments which evaluate the effectiveness and efficiency of our IJoin approach. The experiments were performed on a PC, with 2.0GHz CPU and 2GB memory. All programs were implemented in C# and SQL Server.

### 5.1 Data Sets

We use the real bibliography records from the DBLP web site [1]. The data set consists of all records from 4 conferences, including "VLDB", "ICDE", "ICLP" and "FSTTCS". For each bibliography record, we generated several kinds of citation formats (as shown in Table 4), which are frequently used in bibliography references. We divide all citation records (about 10,000) into two parts, and the

[1] http://www.informatik.uni-trier.de/~ley/db/

**Table 4.** Example of different citation formats

| ID | Citations |
|---|---|
| 1 | In Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005, 145-156 |
| 2 | In Proc. of the 31st Int. Conf. on Very Large Data Bases, Trondheim, Norway, Aug 30 - Sep 2, 2005, 145-156 |
| 3 | In the International Conference on Very Large Data Bases, 2005, 145-156 |
| 4 | In the Int. Conf. on Very Large Data Bases, 2005, 145-156 |
| 5 | In the VLDB, 2005, 145-156 |

similarity joins are performed to find all pairs of citations which represent the same bibliography records.

### 5.2 Evaluation Criteria

We use the F-Measures with Precision and Recall [8] to evaluate the effectiveness of join operations. Let $S_a$ be actual pairs of citations which represent the same bibliography records, and $S_f$ be pairs found by join operations with high similarity.

$$Recall(S_a, S_f) = \frac{|S_a \bigcap S_f|}{S_a} \tag{5}$$

$$Precision(S_a, S_f) = \frac{|S_a \bigcap S_f|}{S_f} \tag{6}$$

$$F(S_a, S_f) = \frac{2 * Recall(S_a, S_f) * Precision(S_a, S_f)}{Recall(S_a, S_f) + Precision(S_a, S_f)} \tag{7}$$

### 5.3 Effectiveness

In the first experiment, we evaluate the effectiveness by comparing the accuracy of cosine similarity with q-grams, basic IJoin and Extended IJion. We divide 2,000 records of citations into two groups, and process them by different similarity joins approaches. Fig. 2 shows the precision, recall and F-Measure under different specified thresholds of minimum similarity.

Fig. 2 (a) shows that cosine similarity with *q-grams* has a low accuracy in both precision and recall, which means that this approach can hardly find similar entities with incomplete information formats and the obtained results contain many errors. In Fig. 2 (b)(c), we can find that our IJoin approach achieves higher precision and recall. When the minimum similarity equals 0.925, it obtains the best balance between precision and recall. We did not apply the decay factor $\gamma$ to IJoin in this experiment (e.g. $\gamma = 1.0$), which will be evaluated later. And in Fig. 2 (d), we compare F-Measure among these three approaches which

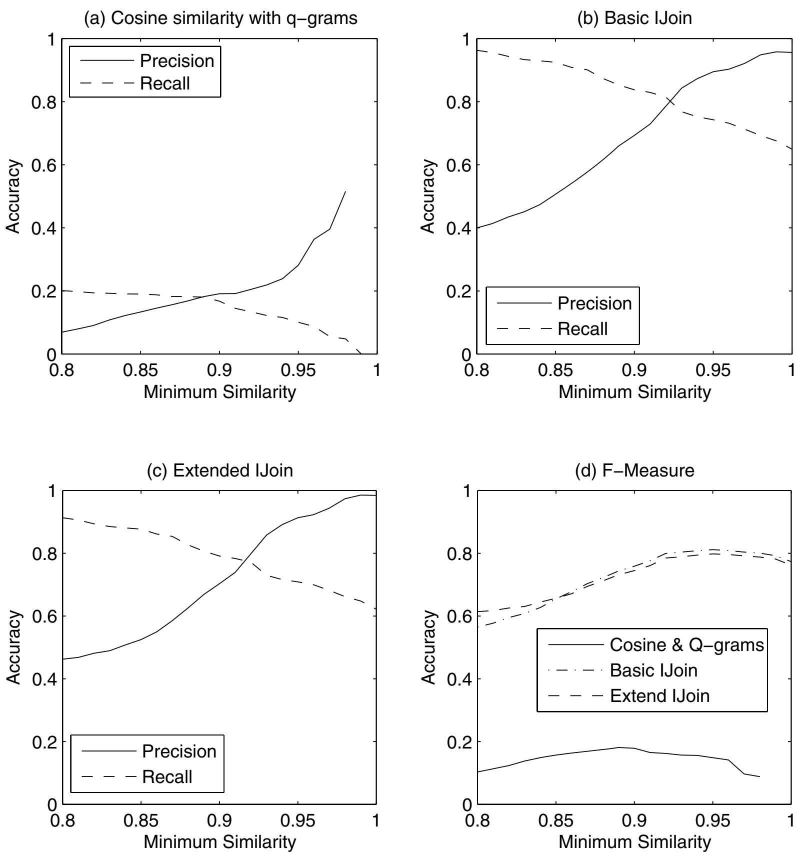

**Fig. 2.** Accuracy of different approaches

demonstrates the superiority of IJoin in effectiveness. From the figure, we also indicate that Extended IJoin achieves as high accuracy as the basic one, which denotes that the filter operation in Extended IJoin does not influence the effectiveness of IJoin too much. We will further discuss it in the next section.

Then, we evaluate the Extended IJoin with different decay factors to validate that the first several letters have greater importance in a word and can help to improve the accuracy of similarity joins. The experiment is also performed in 2,000 records of citations. Fig. 3 shows the results. With the decrease of decay factor $\gamma$, the first few letters (grams) get higher significance in the word which enhances the similarity value between words and their short word formats. As shown in figures, by enhancing the importance of first letters, the accuracy improves when incomplete information formats exist. Note that the decay factor decreases the whole value of all elements in the entity. It is the reason why the best balance point between precision and recall decreases together with the decay factor.

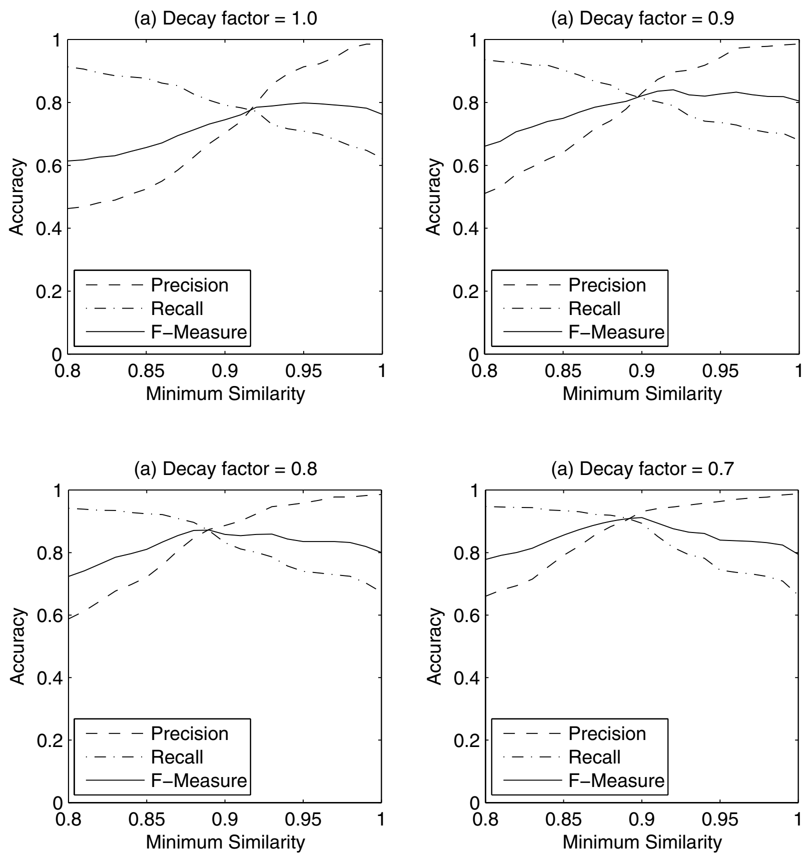

**Fig. 3.** Decay factor in Extended IJoin

## 5.4 Scalability

In this experiment, we evaluate the scalability of three approaches. Different number of citation records are performed under the best minimum similarity threshold of each approach, e.g. cosine similarity with q-grams (short as cosine & q-grams) achieves the best accuracy at $\eta = 0.89$, basic IJoin at $\eta = 0.925$ and Extended IJoin at $\eta = 0.915$. In order to be comparable with the cosine & q-grams approach, no decay factor is adopted in both IJoin (e.g. $\gamma = 1.0$). Fig. 4 shows the results with different data scales. Fig. 4 (a) illustrates the efficiency of the Extended IJoin approach. Its running time remains low even though the number of citations is multiplied. The time performance of cosine & q-grams and basic IJoin are quite similar and increase exponentially. We also show the accuracy with different data scales in Fig. 4 (b). IJoin approach achieves almost constant accuracy under the same similarity threshold in different numbers of citations. The results confirm the scalability of our Extended IJoin approach in both effectiveness and efficiency.

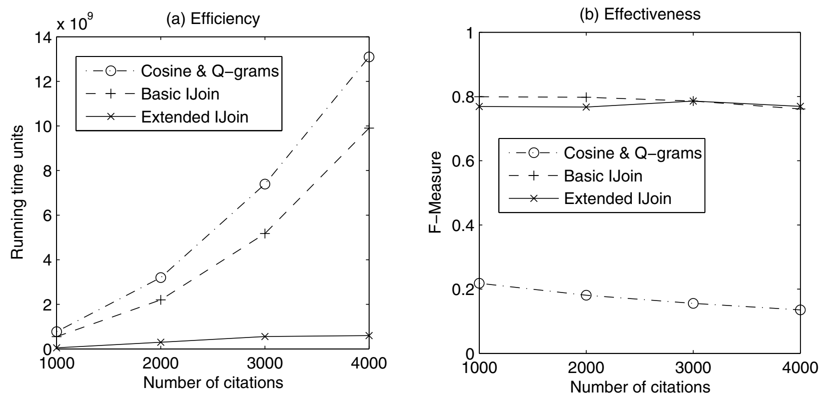

**Fig. 4.** Different data scales of citations

**Table 5.** Basic and Extended IJoin

| | Pairs of Comparison | Time Units | Precision | Recall |
|---|---|---|---|---|
| Basic IJoin | 3,998,000 | 9,901,249 | 0.714 | 0.814 |
| Extended IJoin | 102,080 | 596,875 | 0.748 | 0.792 |

Finally, we compare the basic and Extended IJoin in our experiment. Table 5 shows the results in 4,000 records of citations. The filter operation in Extended IJoin reduces the number pairs of comparison greatly by finding out pairs with common elements. The total number of comparisons in basic IJoin is about 40 times greater than that of Extended IJoin. As shown in “Time Units” column, the time performance improved greatly in Extended IJoin. For the effectiveness, although some pairs that are actually similar may be filtered out, the number of such false negatives is not so large and affects the result slightly as shown in the table column “Recall”. Only about 2.2% of similar pairs are missed by Extended IJoin in this experiment. It is interesting that Extended IJoin even achieves higher accuracy than the basic one. This is because most of irrelevant pairs are filtered out and the remaining pairs are probably similar.

## 6 Conclusions

In this paper, we proposed a novel approach, IJoin. to handle similarity joins of text with incomplete formats, such as abbreviation and short words, are considered in our text matching scheme and similarity function. We connect the first letter of each word to reserve potential abbreviation information and enhance the importance of the first few grams to find high similarity between words and their short formats. The similarity function in IJoin is based on the asymmetric features of similarity between entities with incomplete formats. We also illustrated an efficient implement of our approach (Extended IJoin). Our experiments showed the advantage of our approach in efficiency and effectiveness when dealing with text entities with incomplete information formats.

## References

1. R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *VLDB*, pages 586–597, 2002.
2. S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *ICDE*, page 5, 2006.
3. W. W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *SIGMOD Conference*, pages 201–212, 1998.
4. H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C.-A. Saita. Declarative data cleaning: Language, model, and algorithms. In *VLDB*, pages 371–380, 2001.
5. L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, pages 491–500, 2001.
6. L. Gravano, P. G. Ipeirotis, N. Koudas, and D. Srivastava. Text joins in an rdbms for web data integration. In *WWW*, pages 90–101, 2003.
7. N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: similarity measures and algorithms. In *SIGMOD Conference*, pages 802–803, 2006.
8. B. Larsen and C. Aone. Fast and effective text mining using linear-time document clustering. In *KDD*, pages 16–22, 1999.
9. E.-P. Lim, J. Srivastava, S. Prabhakar, and J. Richardson. Entity identification in database integration. In *ICDE*, pages 294–301, 1993.
10. G. Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001.
11. G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, Inc., New York, NY, USA, 1986.