Fast Duplicate Document Detection using Multi-level Prefix-filter

Kenji Tateishi and **Dai Kusui**
NEC Corporation
Takayama, Ikoma, Nara, 630-0101, Japan
{k-tateishi@bq, kusui@ct}.jp.nec.com

Abstract

Duplicate document detection is the problem of finding all document-pairs rapidly whose similarities are equal to or greater than a given threshold. There is a method proposed recently called prefix-filter that finds document-pairs whose similarities never reach the threshold based on the number of uncommon terms (words/characters) in a document-pair and removes them before similarity calculation. However, prefix-filter cannot decrease the number of similarity calculations sufficiently because it leaves many document-pairs whose similarities are less than the threshold. In this paper, we propose multi-level prefix-filter, which reduces the number of similarity calculations more efficiently and maintains the advantage of prefix-filter (no detection loss, no extra parameter) by applying multiple different prefix-filters.

1 Introduction

Duplicate Document Detection (DDD) is the problem of finding all document-pairs rapidly whose similarities are equal to or greater than a given threshold. DDD is often used for data cleaning of customer databases, trend analysis of failure case databases in contact centers, and can be applied for spam filtering by detecting duplicate blog documents. After receiving target documents and the similarity threshold (ST), the Duplicate Document Detection System (DDDS) shows users all document pairs whose similarities are equal or greater than ST, or document groups these document pairs unify. In the case of data cleaning, DDDS additionally requires users to confirm whether each document pair result is truly duplicated.

The naive implementation of DDD requires similarity calculations of all document pairs, but it demands huge time according to the number of target documents. The current techniques apply the two-stage approach: (i) Reduce document pairs using shallow filtering methods, and then (ii) calculate similarities between the remaining document pairs. Among them, prefix-filter(Sarawagi and Kirpal, 2004)(Chaudhuri et al., 2006)(Bayardo et al., 2007) is a filtering method that finds document-pairs whose similarities never reach the threshold based on the number of uncommon terms (words/characters) in a document-pair, and that removes them before similarity calculation.

For example, suppose that a document pair is composed of 10 terms, and 80% similarity means 8 terms are in common in the document pair. In this case, if the similarity of a document pair is equal to or greater than 80% and 3 terms are selected from one document, the other document must contain at least one of the 3 terms. Therefore, prefix-filter can remove document pairs where one document does not contain any of the 3 terms selected from the other. It can be implemented rapidly by index files. Prefix-filter has two advantages compared with other filtering methods: (i) All document pairs equal to or greater than the similarity threshold (ST) are obtained without any detection loss, and (ii) no extra parameter for filtering is required other than ST.

The problem with prefix-filter is that it cannot reduce similarity calculations sufficiently because it leaves many document-pairs whose similarities are less than ST. Document-pairs that prefix-filter can remove depend on terms selected from each document (in the above example, which 3 terms are selected). At worst, document pairs where only one term is in common might remain. The processing time of DDD can be approximated by the product of the number of similarity calculations and the pro-

cessing time of each similarity calculation. In order to identify the same document pairs correctly, a deep similarity function considering synonyms and variants is essential. Therefore, the number of similarity calculations should decrease as mush as possible.

In this paper, we propose multi-level prefix-filter, which reduces the number of similarity calculations more efficiently and maintains the advantages of prefix-filter (no detection loss, no extra parameter) by applying multiple different prefix-filters. Each prefix-filter chooses terms from each document based on a different priority decision criterion, and removes different document-pairs. It finally calculates the similarities of the document-pairs left by all of the prefix-filters. We conducted an experiment with a customer database composed of address and company name fields, and used edit-similarity for the similarity calculation. The result showed that multi-level prefix-filter could reduce the number of similarity calculations to 1/4 compared with the current prefix-filter.

2 Prefix-filter

Prefix-filter finds document-pairs whose similarities never reach the similarity threshold (ST) based on the number of uncommon terms in a document-pair, and that removes them before the similarity calculation. A DDDS with prefix-filter processes the following four steps. [1]

Step 1: Define x: the minimum proportion of common terms in a document pair whose similarity is equal to or greater than $ST (0 \leq ST \leq 1)$.

Step 2: Decide priorities of all terms on target documents.

Step 3: Select terms from each document according to the priorities in Step 2 until the proportion of selected terms exceeds $1 - x$.

Step 4: Remove document pairs that share no terms selected in Step 3, and calculate the similarities of the remaining document pairs.

Let us illustrate how prefix-filter works briefly. For example, a user inputs 6 documents as in Fig.1 and sets the similarity threshold at $ST = 0.6$ and chooses edit-similarity as the similarity function. Note that edit-similarity between document d1 and document d2, denoted as $edit\_sim(d1, d2)$, is defined as follows.

$$edit\_sim(d1, d2) = 1 - \frac{edit\_distance(d1, d2)}{max(|d1|, |d2|)}$$

Here, $|d1|$ and $|d2|$ denotes the length of $d1$ and $d2$ respectively, and $edit\_distance(d1, d2)$ represents the minimum number of edit operations (insertion, deletion, and substitution) that convert $d1$ to $d2$. For example, $edit\_distance(d1, d5)$ in Fig.1 is 4: delete E, H, and I, and insert M. Then, $max(|d1|, |d5|)$ is 9, derived from $|d1| = 9$ and $|d5| = 7$. Therefore, $edit\_sim(d1, d5) = 1 - (4/9) = 0.45$.

In the first step, when the similarity function is edit-similarity, the minimum proportion of common terms (characters) in a document pair whose similarity is equal or greater than $ST = 0.6$ is $x = 0.6$. This means the similarity of a document pair in which the proportion of common terms is less than 0.6 never reaches 0.6. x can be derived from the similarity function (see Appendix A).

In step 2, DDDS decides the priorities of all terms on target documents. Fig. 1 (a) gives all terms contained in the 6 documents priorities from the lowest document frequency (if the same frequency, alphabetical order). Regardless of the priority decision criteria, the similarities of document pairs removed are always less than ST, but document pairs removed differ. Empirically, it is known that giving high priority from the term of the lowest frequency is effective because the lower the frequency of a term, the lower the probability of a document pair containing that term(Chaudhuri et al., 2006).

In step 3, DDDS chooses terms from each document according to the priority decision criterion of step 2 in Fig.1 (a) until the proportion of selected terms exceeds $1 - x = 0.4$. For example, the proportion is over 0.4 when DDDS selects 4 terms from d1, composed of 9 terms. DDDS selects 4 terms according to (a): $\{A, B, C, I\}$. Fig.1 (b) shows selected terms using boldface and background color.

Finally, DDDS removes document pairs that share no terms selected in step 3, and calculates similarities of the remaining document pairs. The similarities of document pairs with no common terms never

[1] Here, we show the simplest prefix-filter of (Chaudhuri et al., 2006)

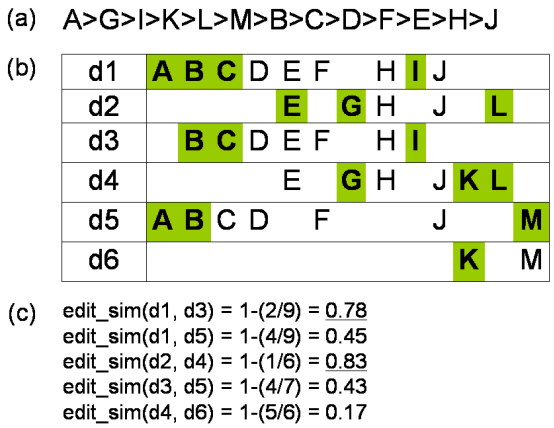

Figure 1: Overview of prefix-filter.

reach 0.6 because the proportion of common terms is less than 0.6. Prefix-filter can be implemented easily using an index file, storing the relation of each selected term and the list of document IDs including the term. As a result, document d1 targets d3 and d5 on similarity calculation. Finally, the number of similarity calculations can be reduced by 5 times while naive solution requires (6*5)/2=15 times.

3 Multi-level prefix-filter

The problem with prefix-filter is that it cannot reduce similarity calculations sufficiently because it leaves many document-pairs whose similarities are less than ST. Document-pairs that prefix-filter can remove depend on terms selected from each document. At worst, document pairs where only one term is in common might remain. In the case of selecting terms according to priority decision criterion (a) in Fig.1, for example, a document pair {d4,d6} on (b) remains although only K is in common. In order to identify the same document pairs correctly, a deep similarity function such as edit-similarity is essential. Therefore, the number of similarity calculations should be decreased as much as possible.

We propose multi-level prefix-filter, which reduces the number of similarity calculations more efficiently by applying multiple different prefix-filters. Each prefix-filter chooses terms from each document based on different priority decision criteria, and removes different document-pairs. It finally calculates the similarities of document-pairs left by all of the prefix-filters. That is why multi-level prefix-filter can reduce the number of document pairs more comprehensively than the current prefix-filter (without any detection loss). Fig.2 illustrates an example of multi-level prefix-filter, applying prefix-filter twice. After DDDS changes priority decision criterion between the first and second prefix-filter, terms selected from each document vary. As a result, document pairs filtered by each prefix-filter change as well. The product of document pairs each prefix-filter leaves leads to the reduction of similarity calculations by 3 times.

Let us explain two kinds of priority decision criteria of terms in the following sections.

3.1 Priority decision using $Score(n, w)$

We define Score(n,w), the score of a term w on n-th prefix-filter, as follows, and give a higher priority to a smaller value of Score(n,w).

$$Score(n,w) = \begin{cases} df(w) & n = 1 \\ 0.1 * df(w) + \sum_{i=1}^{n-1} sdf(i,w) & n \geq 2 \end{cases}$$

where $df(w)$ is the document frequency of w over the target documents, and $sdf(i, w)$ denotes the number of documents in which w was selected on i-th prefix-filter. The basic concept is to give a higher priority to a term of smaller frequency. As mentioned before, this is effective because the lower the frequency of a term, the lower the probability of a document pair containing that term. On the other hand, it is expected that a multi-level prefix-filter becomes more effective if each prefix-filter can filter different document pairs. Therefore, after the second prefix-filter ($n \geq 2$), we give a higher priority to a term whose frequency is small (first term) and which was not selected by previous prefix-filters (second term).

Fig.3 illustrates the process of multi-level prefix-filter based on this creterion. This multi-level prefix-filter can be implimented using two kinds of index files (W_INDEX, D_INDEX) rapidly. If PC with multiple processers, it is easy to parallelize filtering process.

3.2 Priority decision using $Score(d, n, w)$

We define $Score(d, n, w)$, the score of a term w contained in document d on n-th prefix-filter, as fol-

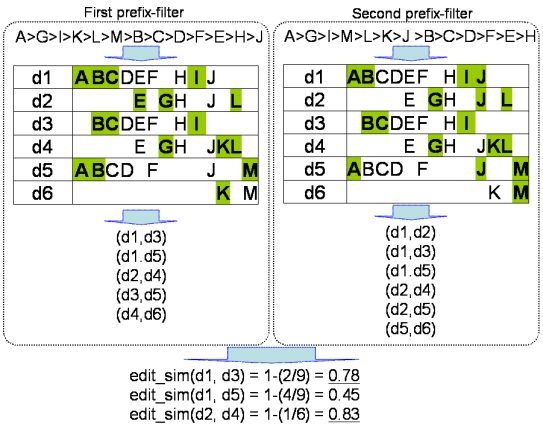

Figure 2: Overview of multi-level prefix-filter.

lows, and give a higher priority to a smaller value of $Score(d, n, w)$.

$$Score(d, n, w) = \begin{cases} df(w) & n = 1 \\ |DS^d_{n-1} \cap DSS_w| & n \geq 2 \end{cases}$$

where DS^d_{n-1} is target documents of similarity calculation of d left after the $n-1$-th prefix-filter, and DSS_w is documents containing a term w. The basic concept is to give a higher priority to a term that can filter many document pairs. It decides the priorities of terms on n-th prefix-filter after waiting for the result of $n-1$-th prefix-filter.

4 Experiments

4.1 Experimental method

We compared multi-level prefix-filter with the current prefix-filter in order to clarify how much the proposed method could reduce the number of similarity calculations. We used a customer database in Japanese, composed of 200,000 records, and had been used for data cleaning. Each record has two fields, company name and address, averaging 11 terms and 18 terms, respectively. We selected edit-similarity as the similarity function, and set 80% as ST. The database contains 86031 (43%) duplicated documents (records) in the company name, and 123068 (60%) in the address field when we assumed document pairs whose similarity was equal to or greater than 80%. A DDDS with multi-level prefix-filter ran on an NEC Express 5800 with Windows 2000, 2.6GHz Pentium Xeon and 3.4 GByte of memory.

N: Number of applying prefix-filter, D: Target documents, ST: Similarity Threshold
Index creation process:
1. for(w∈D)
2. Score(1,w) = df(w)
3. end for
4. for(i=1; i≦N; i++)
5. for(j=0; j≦|D|; j++)
6. W= terms chosen from w∈d_i of the smallest Score(i,w) until the proportion of selected terms exceeds 1-x.
7. for(w∈W)
8. push(D_INDEX(i,d_j), w)
9. push(W_INDEX(i,w), d_j)
10. end for
11. end for
12. for(w∈D)
13. Score(i+1,w)= 0.1 * df(w) + $\sum_{1 \leq k \leq i}$ sdf(k,w)
14. end for
15. end for

Matching process:
16. for(i=0; i≦|D|;i++)
Filtering process:
17. DS = D
18. for(j=1; j≦N; j++) {
19. for(w∈W_INDEX(j,d_j))
20. DSS_w ={d_k | d_k ∈ W_INDEX(j,w), k > i}
21. DS_j = DSS_w ∪ DS_j
22. end for
23. DS = DS ∩ DS_j
24. end for
Similarity calculation process:
25. for(ds∈DS)
26. push(RESULT, {d,ds}) if (sim(d, ds)≧ST)
27. end for
28. end for

Figure 3: Multi-level prefix-filter with Score(n,w).

4.2 Experimental result

Fig.4 (a) shows the comparison between multi-level prefix-filter using $Score(d, n, w)$ and $Score(n, w)$ under the condition that the number of prefix-filters is one or two. The company name field was used for target documents. Although multi-level prefix-filter using Score(n,w) succeeded in the reduction of processing time, Score(d,n,w) failed because of too many score calculations. Therefore, we used $Score(n, w)$ in the following experiments.

Fig.4 (b) shows the number of similarity calculations when the number of applied prefix-filters varies. In this figure, $n = 1$ means the current prefix-filter. The number of similarity calculations decreased most sharply in the case of applying prefix-filters twice on both the company name and address fields, and converged in 10 times. Multi-level prefix-filter reduced the number of similarity calculations by 10 times, about to 1/4 (77% reduction) in the company name field, and about to 1/3 (69% reduction) in the address field.

Fig.4 (c) shows total processing time when the

number of applied prefix-filters varies. It represents the sum of index creation/filtering time and similarity calculation time. When the number of applied prefix-filters increased, the latter decreased because the number of similarity calculations also decreased, but the former increased instead. Note that we did not parallelize the filtering process here. Total processing time decreased most sharply in the case of applying prefix-filters 4 times on both the company name (to be 43%) and address fields (to be 49%).

Fig.4 (d) shows the reduction rate of the number of similarity calculations and processing time when prefix-filter was applied 4 times and the size of target document sets varied. Here, the reduction rate denotes the proportion of the number of similarity calculations or processing time of multi-level prefix-filter, applying prefix-filter 4 times, to those of the current prefix-filter, applying prefix-filter once. This result reveals the effectiveness of multi-level prefix-filter does not change for the size of the target document set.

4.3 Discussion

The experimental results indicated that multi-level prefix-filter could reduce the number of similarity calculations up to 1/4, and that this effectiveness was not lost by changing the size of the target database. In addition, it showed that the optimal number of applied prefix-filters did not depend on the target field or the size of the target database. Therefore, multi-level prefix-filter proved to be more effective than the current prefix-filter without losing the advantages of the current prefix-filter (no detection loss, no extra parameter).

The experimental results also indicated that the company name field was more effective than the address field. As mentioned, the address field was longer than that of the company name field on average, and it contained more duplicated documents. Therefore, we expect that the proposed method is effective in the following situation: (i) the length of each document (record) is short, (ii) the number of duplicate documents has been reduced beforehand by simple filtering methods such as deleting exact match documents or documents different only in space, and (iii) detecting the remaining duplicate documents by using a deep similarity function such as edit-similarity.

5 Related work

Duplicate Document Detection for databases has been researched for a long time(Elmagarmid et al., 2007). The current techniques apply the two-stage approach: (i) Reduce document pairs using shallow filtering methods, and then (ii) calculate similarity between the remaining document pairs. Multi-level prefix-filter belongs to the first step (i).

Current filtering methods were independent of the similarity function. Jaro(Jaro, 1989) proposed Standard Blocking, which created many record blocks in which each record shared the same first n terms, and calculated the similarity of document-pairs included in the same record block. Hernandez(Hernandez and Stolfo, 1995) proposed the Sorted Neighborhood Method (SNM), which first sorted records by a given key function, and then grouped adjacent records within the given window size as a block. McCallum(McCallum et al., 2000) improved them by allowing a record to locate in plural blocks in order to avoid detection loss.

However, the problems of these filtering methods using blocking are that the user needs trial and error parameters such as first n terms for Standard Blocking, and that these incur detection loss in spite of improvements being attempted, caused by two documents of a correct document pair existing in different blocks. Prefix-filter solved these problems: (i) all document pairs equal or more than similarity threshold (ST) are obtained without any detection loss, and (ii) any extra parameter for filtering is not required other than ST. As we clarified in Section 4, multi-level prefix-filter proved to be more effective than the current prefix-filter without losing these advantages.

Another filtering method without any detection loss, called PARTENUM, has been proposed recently(Arasu et al., 2006). However, it needs to adjust two kinds of parameters ($n1$, $n2$) for obtaining optimal processing time according to the size of target document set or the similarity threshold.

6 Conclusion

In this paper, we proposed multi-level prefix-filter, which reduces the number of similarity calculations more efficiently and maintains the advantage of the current prefix-filter by applying multiple different

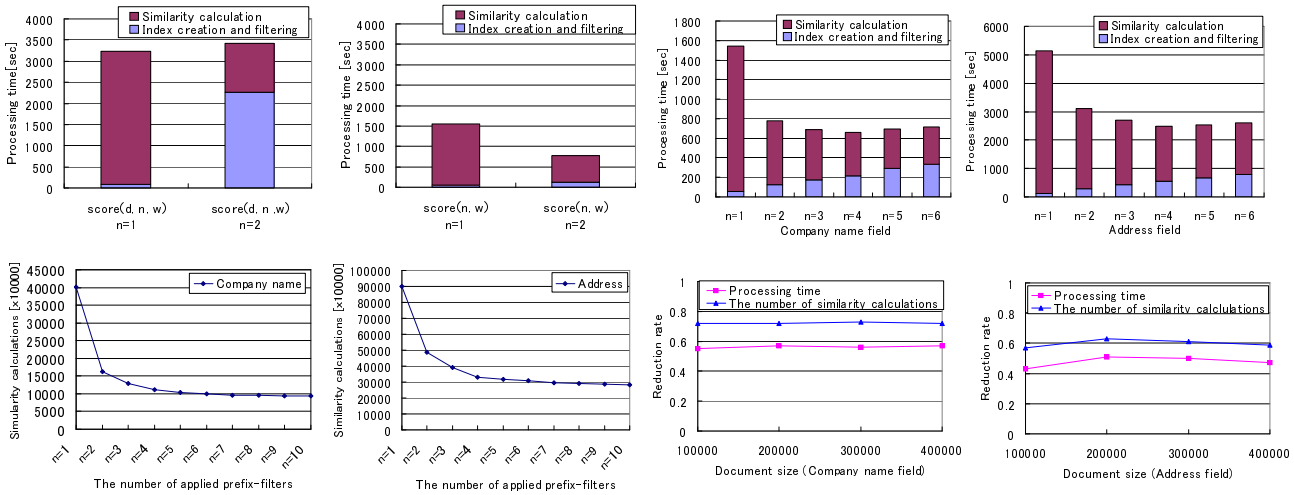

Figure 4: Experimental result.

prefix-filters. Experiments with a customer database composed of 200,000 documents and edit-distance for similarity calculation showed that it could reduce the number of similarity calculations to 1/4 compared with the current prefix-filter.

References

Arvind Arasu, Venkatesh Ganti, and Raghav Kaushik. 2006. Efficient exact set-similarity joins. *Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 918–929.

Roberto J. Bayardo, Yiming Ma, and Ramakrishnan Srikant. 2007. Scaling up all pairs similarity search. *Proceedings of the 16th International Conference on World Wide Web*, pages 131–140.

Surajit Chaudhuri, Venkatesh Ganti, and Raghav Kaushik. 2006. A primitive operator for similarity joins in data cleaning. *Proceedings of the 22nd International Conference on Data Engineering(ICDE'06)*, pages 5–16.

Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. 2007. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering, vol.19, no.1*, pages 1–15.

Mauricio A. Hernandez and Salvatore J. Stolfo. 1995. The merge/purge problem for large databases. *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 127–138.

M. A. Jaro. 1989. Advances in record linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Society, 84 (406)*, pages 414–420.

Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. 2000. Efficient clustering of high-dimensional data sets with application to reference matching. *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 169–178.

Sunita Sarawagi and Alok Kirpal. 2004. Efficient set joins on similarity predicates. *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 743–754.

A The minimum proportion of common terms

Here, we explain how to obtain x of edit-similarity. First,

$$edit\_distance(d1, d2) \geq max(|d1|, |d2|) - |d1 \cap d2|$$

($|d1 \cap d2|$ denotes the number of common terms in both $d1$ and $d2$), and

$$\begin{aligned} ST &\leq edit\_sim(d1, d2) \leq \frac{|d1 \cap d2|}{max(|d1|, |d2|)} \\ &\leq \frac{|d1 \cap d2|}{|d1|}. \end{aligned}$$

Therefore,

$$x = min\{\frac{|d1 \cap d2|}{|d1|}\} = ST.$$