An Efficient Filtering Approaches For Recognizing and Suppression of Duplicate Web Documents
K.SUNEETHA¹, K.MUNIDHANALAKSHMI², C.BHUvana³
¹Assistant Professor(SL), Dept of MCA, SVEC, A. Rangampet, Tirupati.
²Assistant Professor, Dept of MCA, SVEC, A. Rangampet, Tirupati.
³Assistant Professor, Dept of CSSE, SVEC, A. Rangampet, Tirupati.
¹umasuni.k@gmail.com, ²kmd.dhana@gmail.com, ³bhuvanachintha@gmail.com

ABSTRACT
With the increasing amount of data and the need to integrate data from multiple data sources, a challenging issue is to find near duplicate records efficiently. In this paper, we focus on efficient algorithms to find pairs of records such that their similarities are above a given threshold. Several existing algorithms rely on the prefix filtering principle to avoid computing similarity values for all possible pairs of records. We discussed new filtering techniques by exploiting the ordering information by integrating into the existing methods and drastically reduce the candidate sizes, hence improve the efficiency. In this paper, we discussed 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.

Keywords: Duplicate pages, Near Duplicates, Prefix Filtering, Positional Filtering, Suffix Filtering, Multi-Level Prefix-Filter

1. INTRODUCTION
Today, Internet marks the era of information revolution and people rely on search engines for getting relevant information without replicas. There are many identical documents across the Web. As the duplicated web pages increase the indexing space and time complexity, finding and removing these pages becomes significant for search engines and other likely system. The effective duplicate removal has become one of the most important techniques to improve search engines [2].

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 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. 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[3].

In this paper, we discussed 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 shown that multi-level prefix-filter could reduce the number of similarity calculations to 1/4 compared with the current prefix-filter.

2. RELATED WORK
Near Duplicate Object Detection Near duplicate object detection has been studied under different names in several areas, including record linkage, merge-purge, data duplication, name matching. Similarity functions are the key to the near duplicate detection task. For text documents, edit distance and Jaccard similarity on q-grams are commonly used. Due to the huge size of Web documents, similarity among documents is evaluated by Jaccard or overlap similarity on small or fix sized sketches. Exact Near Duplicate Detection Algorithm Existing methods for exact near duplicate detection usually convert constraints defined using one similarity function into equivalent or weaker constraints defined on another similarity measure, converts edit distance constraints to overlap constraints on q-grams. Jaccard similarity constraints and 1/2-sided normalized overlap constraints can be converted to overlap constraints. Constraints on overlap, dice and Jaccard similarity measures can be
coverted to constraints on cosine similarity transforms Jaccard and edit distance constraints to Hamming distance constraints. The techniques proposed in previous work fall into two categories. In the first category, exact near duplicate detection problems are addressed by inverted list based approaches, as discussed above. The second category of work is based on the pigeon hole principle. The records are carefully divided into partitions and then hashed into signatures, with which candidate pairs are generated, followed by a post-filtering step to eliminate false positives[3].

Approximate Near Duplicate Object Detection Several previous work has concentrated on the problem of retrieving approximate answers to similarity functions. LSH (Locality Sensitive Hashing) is a well-known approximate algorithm for the problem. It regards each record as a vector and generates signatures for each record with random projections on the set of dimensions. Broder et al. [1] addressed the problem of identifying near duplicate Web pages approximately by compressing document records with a sketching function based on min-wise independent permutations. The near duplicate object detection problem is also a generalization of the well-known nearest neighbor problem, which is studied by a wide body of work, with many approximation techniques considered by recent work.

Duplicate Document Detection for databases has been researched for a long time(Elmagarmid et al., 2007). The 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 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 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.

2.1 PREFIX FILTERING BASED METHODS

A naive algorithm[4] to compute t-similarity join result is to enumerate and compare every pair of records. This method is obviously prohibitively expensive for large datasets, as the total number of comparisons is O(n^2).

Efficient algorithms exist by converting the Jaccard similarity constraint into an equivalent overlap constraint due to

\[ J(x, y) \geq t \iff O(x, y) \geq \alpha = \frac{t}{1+t} \cdot (|x| + |y|) \quad \text{Equation (1)} \]

An efficient way to find records that overlap with a given record is to use inverted indices. An inverted index maps a token w to a set of identifiers of records that contain w. After inverted indices for all tokens in the record set are built, we can scan each record x, probe the indices using every token in x, and obtain a set of candidates; merging these candidates together gives us their actual overlap with the current record x; final results can be extracted by removing records whose overlap with x is less than \( t \) / \( 1+t \cdot (|x| + |y|) \). The main problem of this approach is that the inverted lists of some tokens, often known as “stop words”, can be very long. These long inverted lists incur significant overhead for building and accessing them. In addition, computing the actual overlap by probing indices essentially requires memorizing all pairs of records that share at least one token, a number that is often prohibitively large. Several existing work takes this approach with optimization by pushing the overlap constraint into the similarity value calculation phase. For example employs sequential access on short inverted lists but switches to binary search on the \( \alpha \) - 1 longest inverted lists. Another approach is based on the intuition that if two canonicalized records are similar, some fragments of them should overlap with each other, as otherwise the two records won’t have enough overlap. This intuition can be formally captured by the prefix-filtering principle rephrased below. Lemma 1 (Prefix Filtering Principle). Consider an ordering O of the token universe U and a set of records, each with tokens sorted in the order of O. Let the-position of a record x be the first \( p \) tokens of x. If O(x, y) \( \geq \alpha \), then the \((|x| - \alpha + 1)\)-prefix of x and the \((|y| - \alpha + 1)\)-prefix of y must share at least one token[4]. Since prefix filtering is a necessary but not sufficient condition for the corresponding
overlap constraint, we can design an algorithm accordingly as: we first build inverted indices on tokens that appear in the prefix of each record in an indexing phase. We then generate a set of candidate pairs by merging record identifiers returned by probing the inverted indices for tokens in the prefix of each record in a candidate generation phase. The candidate pairs are those that have the potential of meeting the similarity threshold and are guaranteed to be a superset of the final answer due to the prefix filtering principle. Finally, in a verification phase, we evaluate the similarity of each candidate pair and add it to the final result if it meets the similarity threshold. A subtle technical issue is that the prefix of a record depends on the sizes of the other record to be compared and thus cannot be determined before hand. The solution is to index the longest possible prefixes for a record x. It can be shown that we only need to index a prefix of length $|x| - t$: $|x| + 1$ for every record x to ensure the prefix filtering based method does not miss any similarity join result. The major benefit of this approach is that only smaller inverted indices need to be built and accessed (by a approximately $(1 - t)$ reduction). Of course, if the filtering is not effective and a large number of candidates are generated, the efficiency of this approach might be diluted. Later there is a need to propose additional filtering methods to alleviate this problem. There are several enhancements on the basic prefix-filtering scheme. Also by considering the prefix filtering method on top of a commercial database system, further improves the method by utilizing several other filtering techniques in candidate generation phase and verification phase[4].

2.2 POSITIONAL FILTERING:

Positional Filtering Principle: Consider an ordering O of the token universe U and a set of records, each with tokens sorted in the order of O. Let token $w = x[i]$, $w$ partitions the record into the left partition $x_l(w) = x[1 \ldots (i - 1)]$ and the right partition $x_r(w) = x[i \ldots |x|]$. If O(x, y) $\geq \alpha$, then for every token $w \in x \cap y$, $O(x_l(w), y_l(w)) + \min(|x_r(w)|, |y_r(w)|) \geq \alpha$.

Positional Filtering-Based Algorithm

A natural idea to utilize the positional filtering principle is to combine it with the existing prefix filtering method, which already keeps tracks of the current overlap of candidate pairs and thus gives us $O(x_l(w), y_l(w))$.

**Statement 1**: ppjoin (R, t)

Input : R is a multiset of records sorted by the increasing order of their sizes; each record has been canonicalized by a global ordering O; a Jaccard similarity threshold $t$

Output : All pairs of records $x_h$, $y_i$, such that $\text{sim}(x, y) \geq t$; Statement 1 Describes the ppjoin algorithm, an extension to the All-Pairs contains following

**statement 2**: SuffixFilter(x, y, Hmax, d)

Input : Two sets of tokens x and y, the maximum allowable hamming distance Hmax between x and y, and current recursive depth d

Output : The lower bound of hamming distance between x and y .

To combine positional filtering and prefix-filtering. Like the All-Pairs algorithm, ppjoin algorithm takes as input a collection of canonicalized record already sorted in the ascending ordered of their sizes. It then sequentially scans each record x, finds candidates that intersect x’s prefix $x[1 \ldots p]$, and accumulates the overlap in a hash map. The generated candidates are further verified against the similarity threshold to return the correct join result. Note that the internal threshold used in the algorithm is an equivalent overlap threshold $\alpha$ computed from the given Jaccard similarity threshold $t$. The document frequency ordering $\text{Odf}$ is often used to canonicalize the records. It favors rare tokens in the prefixes and hence results in a small candidate size and fast execution speed. Readers are referred to [3] for further details on the All-Pairs Statement.

Several novel aspects can be elaborated such as: (i) the inverted indices used in Statement 1 ii) the use of positional filtering (iii) the optimized verification algorithm. By indexing both tokens and their positions for tokens in the prefixes, positional filtering can utilize the positional information. An upper bound of the overlap between x and y can be computed, and only admit this pair as a candidate pair if its upper bound is no less than the threshold $\alpha$. Specifically, $\alpha$ is computed according to Equation (1); $\text{ubound}$ is an upper bound of the overlap between right partitions of x and y with respect to the current token w, which is derived from the number of unseen tokens in x and y with the help of the positional information in the index Iw; $A[y]$ is the current overlap for left partitions of x and y. It is then obvious that if $A[y] + \text{ubound}$ is smaller than $\alpha$, then prune the current candidate y.

**Algorithm 1: Verify(x, A, $\alpha$)**

Input : px is the prefix length of x and py is the prefix length of y
1. for each $y$ such that $A[y] > 0$ do
   2. $wx \leftarrow$ the last token in the prefix of x;
   3. $wy \leftarrow$ the last token in the prefix of y;
4. O ← A[y];
5. if wx < wy then
6. ubound ← A[y] + |x| − px;
7. if ubound ≥ _ then
8. O ← O + |x|((px + 1) . . . |x|) \( \cap y\[(A[y] + 1) . . . |y|]\]
9. Else
10. ubound ← A[y] + |y| − py;
11. if ubound ≥ _ then
12. O ← O + |x|((A[y] + 1) . . . |x|) \( \cap y\[(py + 1) . . . |y|]\]
13. if O ≥ _ then
14. S ← S ∪ (x, y);

Algorithm 1 is designed to verify whether the actual overlap between x and candidates y in the current candidate set \([5.1, y \ A[y] > 0}\), meets the threshold \( \alpha \). An optimization is to first compare the last token in both prefixes, and only the suffix of the record with the smaller token (denoted the record as u) needs to be intersected with the entire other record (denoted as v). This is because the prefix of u consists of tokens that are smaller thanwu (the last token in u’s prefix) in the global ordering and v’s suffix consists of tokens that are larger than wv. Since wu < wv, u’s prefix won’t intersect with v’s suffix. In fact, the workload can still be reduced by skipping the first A[y] number of tokens in v since at least A[y] tokens have overlapped with u’s prefix and hence won’t contribute to any overlap with u’s suffix. The above method is implemented through Lines 4, 5, 8, and 12 in Algorithm 1. This optimization in calculating the actual overlap immediately gives rise to a pruning method. Later the upper bound of the overlap as the length of the suffix of u (which is either \(|x| − px\) or \(|y| − py\)) can be computed. Lines 6 and 10 in the algorithm perform the estimation and the subsequent lines test whether the upper bound will meet the threshold \( \alpha \) and prune away unpromising candidate pairs directly [5].

2.3 SUFFIX FILTERING

In this section, we first describe the need to looking for further filtering method, and then introduce a divide-and-conquer based suffix filtering method, which is a generalization of the positional filtering to the suffixes of the records.

**Quadratic Growth of the Candidate Size**

Let’s consider the asymptotic behavior of the size of the candidate size generated by the prefix filtering-base methods. The candidate size is \( O(n^2) \) in the worst case. An empirical evidence on several real datasets suggests that the growth is indeed quadratic. For example, the square root of query result size and candidate sizes of the All-Pairs algorithm and our ppjoin algorithm is shown in Figure 1. It can be observed that while positional filtering helps to further reduce the size of the candidates, it is still growing quadratically (albeit with a much slower rate than All-Pairs) [3].

3. EXTENSION TO OTHER SIMILARITY MEASURES

The prefix length for a record x will be \( x−\alpha+1 \). The size filtering threshold is \( \alpha \). It can be shown that positional filtering will not help pruning candidates, but suffix filtering is still useful. The Hamming distance threshold, Hmax, for suffix filtering will be \( |x| + |y| − 2\alpha − (i + j − 2) \).

**Edit Distance**: An Edit distance is a common distance measure for strings. An edit distance constraint can be converted into weaker constraints on the overlap between the qgram sets of the two strings. Specifically, let \(|u|\) be the length of the string u, a necessary condition for two strings to have less than \( \delta \) edit distance is that their corresponding q-gram sets must have overlap no less than \( \alpha = (\max(|u|, |v|) + |q − 1| − q\delta) \). The prefix length of a record x (which is now a set of q-grams) is \( q\delta + 1 \). The size filtering threshold is \( |x| − \delta \). Positional filtering will use an overlap threshold \( \alpha = |x| − q\delta \). The Hamming distance threshold, Hmax, for suffix filtering will be \( |y| − |x| + 2q\delta − (i + j − 2)/4 \).

4. FAST DUPLICATE DOCUMENT DETECTION USING MULTI-LEVEL PREFIX-FILTER

**Multi-level prefix-filter**: A 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.1 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].

**Priority decision using** \(\text{Score}(n,w)\): \(\text{Score}(n,w)\) can be defined as the score of a term \(w\) on \(n\)-th prefix-filter, as follows, and a higher priority to a smaller value of \(\text{Score}(n,w)\) can be given.

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

Where \(\text{df}(w)\) is the document frequency of \(w\) over the target documents, and \(\text{sdf}(i,w)\) 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 and which was not selected by previous prefix-filters.

**Priority decision using** \(\text{Score}(d,n,w)\): \(\text{Score}(d,n,w)\) can be defined as the score of a term \(w\) contained in document \(d\) on \(n\)-th prefix-filter, as follows, and give a higher priority to a smaller value of \(\text{Score}(d,n,w)\)[3].

\[
\text{Score}(d,n,w) = \begin{cases} 
\text{df}(w) & n = 1 \\
\text{ds}(d) \cap \text{DSS}_w & n \geq 2 \\
\end{cases}
\]

Where \(\text{ds}(d)\) is target documents of similarity calculation of \(d\) left after the \(n-1\)-th prefix-filter, and \(\text{DSS}_w\) 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.

**CONCLUSION**
In this paper, we discussed 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 prefix-filters.
Experiments with a customer database (no of 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. 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, multilevel 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).

REFERENCES:

AUTHORS BIBLIOGRAPHY

Ms. K. Suneetha obtained Bachelor’s degree in Sciences from S.V.University, Tirupathi. Then she obtained her Master’s degree in Computer Applications from S.V.University. She is working as Assistant Professor in the Department of Master of Computer Applications at Sree Vidyanikethan Engineering College, A. Rangampet, Tirupati. She is pursuing her Ph.D. in Computer Science in the area of Data Warehousing and Data Mining. She is in teaching since 2001. She presented many papers at National and International Conferences and published articles in National & International journals.

Ms. K. Munidhanalakshmi is working as Assistant Professor of Master of Computer Applications of Sree Vidyanikethan Engineering College, A. Rangampet, Tirupati, India. She is having more than 6 years of teaching experience and 1 years of research experience. Her areas of interests include Software Testing, Cloud Computing, Databases, etc., She has published several national and international papers in conferences and journals.

Ms. C. Bhuvana working as an Assistant Professor in the Department of Computer Science and System Engineering of Sree Vidyanikethan Engineering College, A. Rangampet, Tirupati, India. She is having more than 6 years of teaching experience. Her areas of interests include Networking, Databases, Cloud Computing, and Software Testing. She has published several national and international papers in conferences.