An Incremental Prefix Filtering Approach for the All Pairs Similarity Search Problem

Hoang Thanh Lam, Dinh Viet Dung
University of Pisa Italy
Email: {lam,dinh}@di.unipi.it

Raffaele Perego, Fabrizio Silvestri
ISTI, CNR Pisa Italy
Email: {raffaele.perego,fabrizio.silvestri}@isti.cnr.it

Abstract—**Given a set of records, a threshold value t and a similarity function, we investigate the problem of finding all pairs of records such that similarity between each pair is above t. We propose several optimizations on the existing approaches to solve the problem. Our algorithm outperforms the state-of-the-art algorithms in the case with large and high-dimensional datasets. The speedup we achieved varied from 30% to 4-x depending on the similarity threshold and the dataset properties.**

I. Introduction

There are plenty of real-world applications of solving the similarity search problem (formal definition is given in the following section). For instance, consider some examples listed in the paper [2]: *Near duplicate document detection and elimination* [3], [5] or *Collaborative filtering* and *Query recommendation*. Although all the aforementioned problems are well-known in the database community but solving them under online contexts with large corpus is very challenging. A brute force algorithm with quadratic complexity is infeasible in practice. Many approaches proposed in the past were focusing on the approximation techniques [3], [4]. On this manner, non-trivial amount of error is allowed. However, recent work from the database community [1], [8], [2], [6], [7] has considered exact solutions for this problem.

In this paper, we proposed two optimization techniques based on prefix extension and early termination of overlap computing. The proposed optimizations can be integrated into the existing prefix filtering-based approaches to form hybrid solutions that solve the all-pairs similarity search problem efficiently. We have validated our algorithm against the most recently proposed algorithms with 5 different data-sets. The preliminary results showed that the proposed algorithm outperformed the other algorithms in the case with large data-sets and high dimensionality.

Paper outline: The paper is organized as follows. The formal definition of the all pairs similarity search is given in section 2. Section 3 revisits some techniques used to solve the problem. Section 4 discusses our proposals for solving the problem. Results of the experiments conducted with the 5 collections are discussed in Section 5. Some concluding remarks and directions for future work are finally outlined.

II. Problem Statement

Given a set of records $R = \{r_1, r_2, \cdots, r_n\}$, where each record is a set of tokens taken from a finite universe $U = \{t_1, t_2, \cdots, t_m\}$. A function $sim(\cdot,\cdot)$ measures the similarity of two records. Given a threshold value $t \in \Re$, the all pairs similarity search problem is aimed at finding all pairs of records $x, y \in R$ such that $sim(x, y) \geq t$.

Prior work [8] considered each record as a multiset[1] allowing duplicated tokens . We denote $|x|$ as the number of tokens in the record x - or record length, the following functions can be used to measure the similarity between two non-empty records x and y:

- **Cosine similarity**: $C(x, y) = \frac{|x \cap y|}{\sqrt{|x|.|y|}}$
- **Jaccard similarity**: $J(x, y) = \frac{|x \cap y|}{|x \cup y|}$
- **Overlap similarity**: $O(x, y) = |x \cap y|$
- **Dice similarity**: $D(x, y) = \frac{2.|x \cap y|}{|x|+|y|}$

All the aforementioned functions are commutative and can be transformed to the Overlap Similarity easily. For example, we can present the Jaccard similarity as a function of the overlap $O(x, y)$ and the record lengths as follows: $J(x, y) = \frac{O(x,y)}{|x|+|y|-O(x,y)}$. As a results, $J(x, y) \geq t$ if and only if $O(x, y) \geq \alpha$, where $\alpha = \frac{t}{1+t} * (|x| + |y|)$.

Similarly, from the equation $C(x, y) = \frac{O(x,y)}{\sqrt{|x|.|y|}}$ we can imply that $C(x, y) \geq t$ if and only if $O(x, y) \geq \alpha$, where $\alpha = t.\sqrt{|x|.|y|}$. In this paper, we are focusing on the commonly used Jaccard similarity and the Cosine similarity, however, it is easy to extend our approach for the rest similarity functions.

Moreover, from the aforementioned facts, both Jaccard and Cosine similarity functions can be presented as a function of overlap similarity and the records lengths. In other words, our interested similarity functions is able to be represented with a general form : $sim(x, y) = f(O(x, y), |x|, |y|)$, where f is a real-value function admitting three arguments.

III. Preliminary Background

In this section, we try our best to cover the recently proposed techniques for solving the all pairs similarity search problem. However, due to limit space the readers can refer to the proper references [2], [8] for more details and explanations.

A. A Naive Approach

A brute force solution of the all pairs similarity search problem is shown in Algorithm 1. The NaivePair function

[1]A multiset treats different occurrences of a word in a record as different tokens. For example, the record $\{to\ be\ or\ not\ to\ be\}$ can be considered as a multiset $\{to_1, be_1, or, not, to_2, be_2\}$ by re-enumerating the duplicates

Algorithm 1 $NaivePair(R, t)$

```
1: Input: a set of records $R = \{r_1, r_2, \cdots, r_n\}$, a similarity function $sim(\cdot,\cdot)$ and a similarity threshold $t \in [0, 1]$
2: Output: All pairs of records (x,y), such that $sim(x, y) \geq t$
3: $S \leftarrow \varnothing$
4: for all $i = 2 \rightarrow n$ do
5:     for all $j = 1 \rightarrow i - 1$ do
6:         $sim(r_i, r_j) \leftarrow f(Overlap0(r_i, r_j), |r_i|, |r_j|)$
7:         if $sim(r_i, r_j) \geq t$ then
8:             $S \leftarrow S \cup \{(r_i, r_j)\}$
9:         end if
10:    end for
11: end for
12: return $S$
```

Algorithm 2 $Overlap0(r_1, r_2)$

```
1: Input: two records $r_1$ and $r_2$; each record has been sorted by a global ordering $\Omega$
2: Output: the overlap $O(x, y)$
3: $overlap \leftarrow 0$
4: $i \leftarrow j \leftarrow 0$
5: while $i \leq |r_1|$ AND $j \leq |r_2|$ do
6:     if $r_1[i] == r_2[j]$ then
7:         $overlap \leftarrow overlap + 1$
8:         $i \leftarrow i + 1$
9:         $j \leftarrow j + 1$
10:    else
11:        if $r_1[i] < r_2[j]$ then
12:            $i \leftarrow i + 1$
13:        else
14:            $j \leftarrow j + 1$
15:        end if
16:    end if
17: end while
18: return $overlap$
```

subsequently calls the $Overlap0(\cdot, \cdot)$ function described in Algorithm 2 to calculate the overlap between every possible pairs of records (line 6) and check against the threshold value (line 7) to return the feasible pairs in S.

The pseudo-code of the $Overlap0(r_1, r_2)$ function computing the intersection between two lists is shown in Algorithm 2 in which r_1 and r_2 are presumed to be sorted by a global ordering. The typical procedure for computing intersection of two sorted lists is very well-known and its complexity is as many as the record lengths: $\Theta(|r_1| + |r_2|)$. Therefore, the complexity of the $NaivePair$ function is quadratic, i.e. $\Theta(C.n^2)$, where C is a constant and as big as the average length of the records. The quadratic complexity of the NaivePair algorithm makes it infeasible for applications with large amount of data. Thus, in the subsequent sections we will consider some approaches avoiding exhaustively computing pairwise similarity.

B. A Basic Inverted Index-based Approach

An *Inverted Index* is a well-known data structure adopted in large-scale information retrieval systems to answer full-text queries quickly. Several work [1] has adopted this data structure to solve the all pairs similarity search problems. According to these approaches, the algorithm is generally divided into three phases:

- *Indexing Phase:* incrementally constructs the inverted index.
- *Candidate Generation Phase:* by traversing the index it generates candidate pairs that have potential of meeting the similarity threshold.
- *Verification Phase:* the generated candidate pairs are checked against the threshold similarity to form a result set in a process called the

In practice, several heuristic techniques have been proposed to filter out the pairs that we know for sure never meet the similarity threshold. Hence, the number of pairwise similarity comparisons can be reduced significantly compared with the Naive approach. A simple prototype of the inverted index based methods is presented in the paper by Bayardo et. al. [2], the readers can refer to the All-Pairs-0 algorithm in their paper for more details.

C. A Prefix Filtering Based Approach

The inverted index-based methods reduce the total number of promising candidates sent to the final verification phase. Even so, in practice this number is still high, for example, [8] empirically showed that the candidate set was quadratic growth along with the collection size. The reason is that the inverted lists of the stop-words (very frequent words) are very long contributing a lot of promising candidates. This leads to expensive index construction and maintenance [8] . Some work has been proposed to deal with this issue, we will discuss a technique that reduces the number of candidates significantly.

Concretely, we discuss the prefix filtering technique utilized in the most of the recent work on the all pairs similarity search problem. This technique is based on the observation that if all the records are sorted by a global ordering, some fragments of them must share several common tokens with each other in order to meet the threshold similarity. The following lemma (proof will be given in the following sections) formally states the idea behind the approach:

Lemma 1: Assume that all the tokens in each record are ordered by a global ordering Ω . Let p-prefix of a record x be the first p tokens of the record. 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.

The positive effect of lemma 1 is that when we consider a pair of records for meeting a predefined threshold similarity it is not necessary to consider the entire records but only the prefixes of them. If their prefixes with the given lengths are not intersected we can safely prune them out from the promising candidate set. Besides, in practice, instead of indexing the entire corpus we just need to index the prefixes resulting in a very compacted inverted index.

The lemma is formulated for the overlap similarity $O(x, y)$, however, it is able to extend the lemma for the other functions as well. Concretely, for the Jaccard similarity, if $|x| \geq |y|$ and $J(x, y) \geq t$ then the $(|x| - \lceil \frac{2t}{1+t}|x| \rceil + 1)$-prefix of x and the $(|y| - \lceil t|y| \rceil + 1)$-prefix of y must share at least one token. Besides, similar proposition can be made for Cosine function: the $(|x| - \lceil t^2|x| \rceil + 1)$-prefix of x and the $(|y| - \lceil t|y| \rceil + 1)$-prefix of y share at least one token if $C(x, y) \geq t$ [2].

D. A Prototype of Prefix Filtering Approach

Bayardo et. al. [2] adopts prefix filtering based technique to conduct an algorithm called the AllPairs algorithm[2]. The pseudo-code of the Allpairs algorithm for Cosine similarity is shown in Algorithm 3. In the *Indexing Phase*, the Allpairs function subsequently loads the records from disk and incrementally builds every posting list I_i corresponding to each token t_i occurring in the record prefixes (line 5-11). Every variable p_i (line 7) is assigned an appropriate record prefix length. Only tokens in the prefixes are indexed (line 8-9). Besides, Allpairs calls the $FindMatches(r, I_1, \cdots, I_m, t)$ to generate candidate pairs having high potential of meeting the similarity threshold. Further more, in this function, the generated candidates are checked against the similarity threshold.

In order to generate candidate set the $FindMatches$ function (Algorithm 4) subsequently scans every posting list of the tokens occurring in the prefix of r(line 6). It adopts a heuristics known as the *size filtering* [2], [8] to filter out candidates by considering the candidates in increasing orders of their lengths and filter the records whose lengths are shorter than the $minsize$ (line 7). In the next step, partial similarity scores of the candidates are accumulated in a hashmap A. The accumulated similarity scores are the actual similarity of the prefixes. Based on the partial similarity and information about suffix length, an upper bound on the similarity of the records is checked against the threshold similarity (line 14). Only candidates passing this check are sent to the verification phase in which similarity value is finally evaluated and checked against the similarity threshold again.(line 15-18).

E. A Positional Filtering and Suffix Filtering Approach

Prefix filtering technique helps prune out infeasible pairs, however, in practice the number of pairs surviving after this filtering phase is still quadratic growth [8]. Following the prefix filtering technique, the positional filtering (PPJOIN) and the suffix filtering approaches (PPJOIN+) [8] were proposed to prune out further the infeasible pairs.

The crucial idea behind the former approach is that information about token's position in every record is used to define an overlap upper-bound. This upper-bound is then compared against the similarity threshold to make pruning decision. On the other hands, the latter approach sets up an overlap upper-bound by exploiting recursive binary lookups for positions of elements of one record in another record. In practice, prefix-based method, position-based method and suffix-based method are not alternative but can be used together. Since the suffix-based method adopts recursive binary look-ups it is very computational demanding. So it usually follows after other filters when the number of candidates have already been reduced substantially.

Algorithm 3 $AllPairs(R, t)$

```
1: Input: a set of records $R = \{r_1, r_2, \cdots, r_n\}$, which has been ordered increasingly by record lengths, a similarity function cosine $C(\cdot,\cdot)$ and a threshold value $t \in [0, 1]$
2: Output: All pairs of records (x,y), such that $C(x, y) \geq t$
3: $S \leftarrow \varnothing$
4: $I_1, I_2, \cdots, I_m \leftarrow \varnothing$   //Posting lists
5: for all $i = 1 \to n$ do
6:     $S \leftarrow FindMatches(r_i, I_1, \cdots, I_m, t)$
7:     $p_i \leftarrow |r_i| - \lceil t.|r_i| \rceil + 1$   //Prefix length
8:     for all $j = 1 \to |p_i|$ do
9:         $I_{r_i[j]} \leftarrow S \cup \{i\}$
10:    end for
11: end for
12: return $S$
```

Algorithm 4 $FindMatches(r, I_1, \cdots, I_m, t)$

```
1: Input: a record $r$, currently built inverted lists $I_1, \cdots, I_m$, Cosine similarity function $C(\cdot,\cdot)$ and a threshold value $t \in [0, 1]$
2: Output: All the of records $y$, such that $sim(r, y) \geq t$
3: $M \leftarrow \varnothing$
4: $A \leftarrow$ empty map from vector id to accumulation score
5: $minsize \leftarrow |r|t^2$
6: for all $i = 1 \to |r|$ do
7:     Remove all y from $I_{r[i]}$ s.t $|y| < minsize$ //Size filtering
8:     for all $y \in I_{r[i]}$ do
9:         $A[y] \leftarrow A[y] + 1$
10:    end for
11: end for
12: for all $y \in A$ do
13:    $y_s \leftarrow$ the suffix of y
14:    if $\frac{A[y]+|y_s|}{\sqrt{|r|.|y|}} \geq t$ then
15:        $d \leftarrow \frac{A[y]+Overlap0(r,y_s)}{\sqrt{|r|.|y|}}$
16:        if $d \geq t$ then
17:            $M \leftarrow M \cup \{r, y\}$
18:        end if
19:    end if
20: end for
21: return $M$
```

IV. Our Approach

A trade-off should be taken into account when we design a filter-based approach is that the effective filtering algorithm usually requires more computational efforts. In fact, the suffix-based filter was shown to be very effective in terms of the number of pairs it prunes. However, it is also very inefficient because of the recursive binary lookup it adopts [8]. A good filter should be both effective (prune out a lot of infeasible pairs) and efficient as well. In this paper, we propose some optimizations for the existing approaches taking into account this trade-off.

A. Optimization from the Prefix Extension

Lemma 1 states that the prefixes of two records must share at least one token in order to satisfy the prefix filtering principle. Nevertheless, in practice this condition is easily satisfied, especially with long prefixes. In this section, we will consider an extended version of lemma 1 which tightens the lower bound, thus make it more difficult to be satisfied.

Lemma 2: Assume that all the tokens in each record are ordered by a global ordering Ω. Let p-prefix of a record x be the first p tokens of the record. Let k be a constant. If $O(x, y) \geq \alpha$ then the $(|x| - \alpha + k)$-prefix of x and the $(|y| - \alpha + k)$-prefix of y must share at least k tokens

[2]http://code.google.com/p/google-all-pairs-similarity-search/

Proof: Assume that w_x, w_y are the last tokens in the $(|x|-\alpha+k)$-prefix of x and the $(|y|-\alpha+k)$-prefix of y respectively. We denote $(|x|-\alpha+k)$-prefix of x as x_l and the suffix as x_r. Similarly, we denote y_l and y_r with the same meanings. Assume without loss of generality that $w_x \leq_\Omega w_y$, we have :

$$\begin{aligned} O(x,y) &= O(x_l,y) + O(x_r,y) & (1)\\ &= O(x_l,y_l) + O(x_l,y_r) + O(x_r,y) & (2)\\ &\leq O(x_l,y_l) + |x_r| & (3)\\ &= O(x_l,y_l) + \alpha - k & (4) \end{aligned}$$

From the right hand of equation (2), we can omit $O(x_l, y_r)$ because we have assumed that $w_x \leq w_y$ resulting in $x_l \cap y_r = \emptyset$. A direct consequence of the last inequality is $O(x_l, y_l) + \alpha - k \geq O(x,y)$. On the other hands, since $O(x,y) \geq \alpha$ we can imply that $O(x_l, y_l) \geq k$. The lemma is proved. ■

The extended lemma tightens the lower bound on the overlap of the prefixes. However, the prefixes are obviously longer. In practice, it is likely that sharing at least k ($k > 1$) tokens is more difficult to be satisfied than sharing only one token. This hypothesis is theoretically supported by the following lemma:

Lemma 3: Given two constants k, p and two records x and y. We assume that all the tokens in each record are ordered by a global ordering. Let x_p and y_p be the p-prefixes of x and y respectively. Let x_{p+k} and y_{p+k} be the extended prefixes, in other words we assume that they are the p+k-prefixes of x and y respectively. If $O(x_{p+k}, y_{p+k}) \geq k+1$ *then* $O(x_p, y_p) \geq 1$

Proof: This lemma is a direct consequence of lemma 1. In fact, since $O(x_{p+k}, y_{p+k}) \geq \alpha$, where $\alpha = k+1$, the $|x_{p+k}| - \alpha + 1$-prefix of x and the $|y_{p+k}| - \alpha + 1$-prefix of y must share at least one token. In other words, $O(x_p, y_p) \geq 1$. The lemma is proved. ■

The consequence of lemma 3 is that, if two records satisfy the condition $O(x_{p+k}, y_{p+k}) \geq k+1$ they also satisfy the condition $O(x_p, y_p) \geq 1$. As a result, the set of candidate pairs satisfying the former condition is just a subset of the set of candidate pairs satisfying the latter condition. Hence, checking against the the former condition might reduce more candidates than the latter one.

We empirically show in the experiments that by extending the prefixes with several tokens we reduce the number of candidate pairs significantly. The proper choice of the constant k depends on the average length of the records. For example, when the average length of the records are long leading to long prefixes (for instance about 100) indexing 2 or 3 extra tokens ($k = 2, 3$) is effective because the size of the extended inverted index does not increase much compared to the original index. Reversely, when the records are short, the extension of the prefixes, even with only 2 or 3 tokens is useless due to the domination of extra overhead. We will study this issue carefully in the experiments.

B. Early Termination of Overlap Computing

Our interesting observation on the results of the state-of-the-art approaches is that even with a very clever filter, the number of candidate pairs surviving after filtering is still very large. Especially, when the similarity threshold is low, the gap between the number of candidate pairs and the number of actual results pairs is enlarged. For example, table I shows the candidate size generated by the state-of-the-art algorithms and the number of actual result pairs over various values of Cosine similarity threshold. In this table, the candidates set generated by the PPJoin+ algorithm with the similarity threshold 0.5 is 3450 times larger that the results set. Since all of these candidate pairs are then sent to the verification phase the overhead of verification is high.

TABLE I
CANDIDATE SIZES AND RESULT SIZE OF THE TREC-9 MEDLINE DATASET OVER DIFFERENT COSINE SIMILARITY THRESHOLDS

Algorithms	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**
AllPairs	416M	138M	41M	9M	969709
PPJoin	382M	117M	32M	6M	561032
PPJoin+	345M	96M	22M	3M	92483
Results	100241	32494	8732	4095	2330

In this paper, we propose a method that continues filtering out further candidate pairs even in the verification phase. In doing so , we can reduce the overhead of the similarity verification. This can be done with the help of the following operation called the *probe and check.*

Assume that we want to check if two records x and y have the overlap satisfying the condition $O(x,y) \geq \alpha$. We also assume that we are at the moment that the records were sent to the verification phase for full similarity evaluation. Let w be a token of the record x, assume that w divides x into two partitions: the left partition denoted by x_l containing all the tokens that less than w and the right partition x_r containing all the tokens greater than w and including w. Similarly, assume with the same meanings that the token w divides y into two partitions denoted by y_l and y_r . We denote $Ubound(w,x,y)$ as the upper bound on the overlap of x and y estimated by the result of the probing for token w in y. Thanks to the certain property of partitions that the left partition of one record does not intersect with the right partition of the other record we can estimate the upper bound as : $Ubound(w,x,y) = O(x_l,y_l) + min(|x_r|,|y_r|)$.

Having the upper bound $Ubound$ we check it against the similarity threshold α. We say that the probe and check operation is successful if $Ubound(w,x,y) \geq \alpha$ and failed otherwise. Thus, the probe and check operation defined in this way is very similar to the probe operation defined in [8]. However, our approach does not utilize any binary search to perform probing. In this paper we show that by using the certain property of the algorithm computing the overlap between two lists (Algorithm 2), we can do probe without necessary binary search. In fact, algorithm 5 is a modified version of algorithm 2 which computes the overlap between two records. Consider the point where $r_1[i] < r_2[j]$ (or $r_1[i] > r_2[j]$), the probe for position of token $r_1[i]$ ($r_2[j]$) in y (x) will return exactly the position number j (i). It turns out that the certain property of algorithm 2 allows us to perform probing without necessary binary search. At this point a similarity upper bound is estimated and checked against the similarity threshold. As soon as the check is failed we can stop the overlap computing early.

It is more likely that when the algorithm scans two records from the start toward the end, the probe and check will become more effective. This hypothesis is supported theoretically by the following lemma:

Lemma 4: Given two records x and y, Assume that, the tokens in two records are ordered by a global ordering Ω. Let w_1 and w_2 be two tokens of the record x, If $w_1 <_\Omega w_2$ then $Ubound(w_2, x, y) \leq Ubound(w_1, x, y)$

Proof: Let x_{l_1}, x_{r_1} and x_{l_2}, x_{r_2} are the partitions of x separated by the token w_1 and w_2 respectively. Similarly, Let y_{l_1}, y_{r_1} and y_{l_2}, y_{r_2} are the partitions of y separated by the token w_1 and w_2 respectively.

Since $w_1 <_\Omega w_2$, we have :

$$\begin{aligned}
&Ubound(w_2, x, y) \\
= \ & O(x_{l_2}, y_{l_2}) + min(|x_{r_2}|, |y_{r_2}|) \\
= \ & O(x_{l_1}, y_{l_1}) + O(x_{l_2} - x_{l_1}, y_{l_2} - y_{l_1}) + min(|x_{r_2}|, |y_{r_2}|) \\
\leq \ & O(x_{l_1}, y_{l_1}) + min(|x_{l_2} - x_{l_1}|, |y_{l_2} - y_{l_1}|) + min(|x_{r_2}|, |y_{r_2}|) \\
\leq \ & O(x_{l_1}, y_{l_1}) + min(|x_{r_1}|, |y_{r_1}|) \\
= \ & Ubound(w_1, x, y)
\end{aligned}$$

Thus $Ubound(w_2, x, y) \leq Ubound(w_1, x, y)$, the lemma is proved. ■

A consequence of the lemma is that as we scan the lists from the beginning to the end, the upper bound on the overlap monotonically decreases. Hence, if we do the probe and check subsequently, sooner or latter, we will reach the point where either the probe and check is failed resulting in early termination of the overlap evaluation or we successfully finish computing the overlap.

C. Incremental Prefix Filtering Algorithm

We have already seen two optimization techniques proposed in the prior sections. These two techniques are not alternative to the other filtering approaches but can be integrated into existing algorithms to conduct a hybrid solution. In this paper, we choose the Allpairs algorithm as the basic platform to which we integrate our techniques introducing a hybrid algorithm called the Incremental Prefix Filtering Algorithm or IPPair for short. The reason we choose Allpairs as the basic platform is that it consumes less memory compared to the other techniques, the implementation can scale up to 20M of records. Moreover, its source code is freely available making it convenient to be compared with and modified.

The pseudo-code of the IPPair (Algorithm 6) is the same as the Allpairs with some minor changes. Concretely, we extend the prefix with k tokens leading to a new check condition. Moreover, to evaluate similarity in verification phase we call the $Overlap$ function instead of $Overlap0$ to enable the early termination technique.

Algorithm 5 $Overlap(r_1, r_2, \alpha)$

```
Input: two records $r_1$ and $r_2$; each record has been sorted by a global ordering $\Omega$ and a threshold value $\alpha$ on the overlap
Output: the overlap $O(x, y)$
Replace lines 11-15 of Algorithm 2 with the following code
if $r_1[i] < r_2[j]$ then
    $Ubound(r_1[i], r_1, r_2) = overlap + min(|r_1| - i, |r_2| - j + 1)$
    if $Ubound(r_1[i], r_1, r_2) < \alpha$ then
        return $-\infty$
    end if
    $i \leftarrow i + 1$
else
    $Ubound(r_1[i], r_2, r_1) = overlap + min(|r_1| - i + 1, |r_2| - j)$
    if $Ubound(r_2[j], r_2, r_1) < \alpha$ then
        return $-\infty$
    end if
    $j \leftarrow j + 1$
end if
```

Algorithm 6 $IPPair(R, t, k)$

```
Input: a set of records $R = \{r_1, r_2, \cdots, r_n\}$, which has been ordered increasingly by their lengths, a similarity function cosine $C(\cdot, \cdot)$ and a threshold value $t \in [0, 1]$, and a constant $k$
Output: All pairs of records ¡x,y¿ s.t. $C(x, y) \geq t$
Replace line 7 in Algorithm 3 with
• $p_i \leftarrow |r_i| - \lceil t.|r_i| \rceil + 1 + k$
Replace line 14 in Algorithm 4 with
• If $A[y] \geq k + 1$ AND $\frac{A[y] + |y_s|}{\sqrt{|r|.|y|}} \geq t$
Replace line 15 in Algorithm 4 with
• $\alpha \leftarrow t.\sqrt{|r|.|y|} - A[y]$
• $d \leftarrow \frac{A[y] + Overlap(r, y_s, \alpha)}{\sqrt{|r|.|y|}}$
```

V. Experiments and Results

In order to do the experiments and compare the results, we obtained the source code of the Allpairs algorithm written in C++. We keep the source code of the Allpairs as original as possible and integrate our modification to implement the IPPair algorithm. The PPJoin and PPJoin+ are available with the binary code, so to make it neutrally fair for comparison we compiled the Allpairs and the IPPair in an Ubuntu machine with 2GB of RAM, 160GBs hard disk, AMD Turion Dual Core 2GHz. Consequently, all the binary programs were executed in another Fedora Linux Machine with 4GB of RAM, 500GBs hard disk, 3GHz CPU. For comparison, all the parameters of PPJoin and PPJoin+ are set up as the default settings.

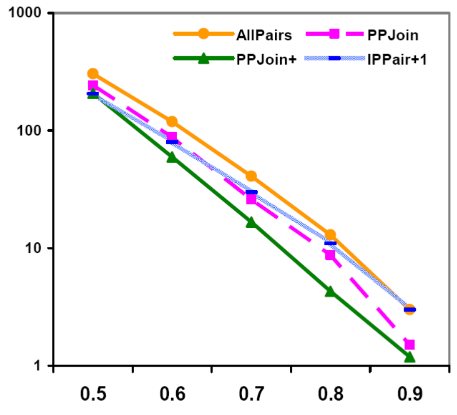
A. The Datasets

We used 5 different datasets with different properties described in table II. These datasets are those that were used in the paper [8]. Among the datasets DBLP contains a large number of documents but the average size of documents is very small. The other datasets like and the MedLine-4gram has the largest size and very long documents in average. The reader can refer to [8] to understand how these datasets are obtained from raw text.

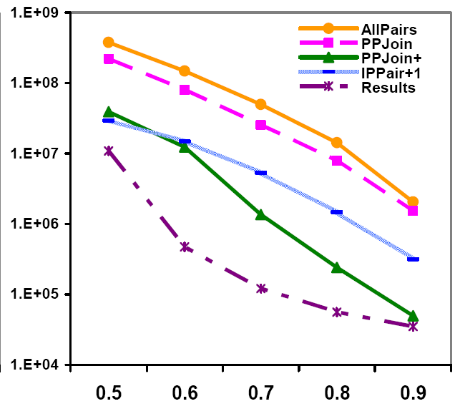
B. Running Time and Candidate Size

Experiments with Cosine Similarity

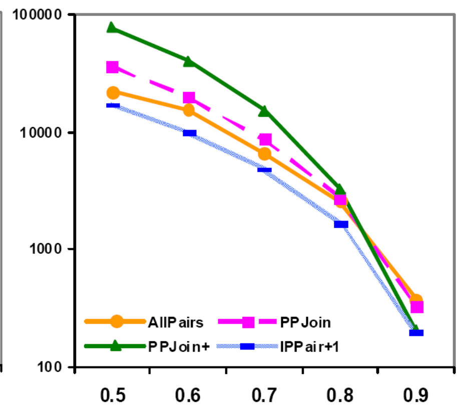
First of all we compare the running time and the candidate size generated by the IPPair with the other algorithms on figure 1 (due to limit space we include only the results with Cosine similarity, however, the results with Jaccard similarity are almost the same). On this figure, the IPPair+1 stands for



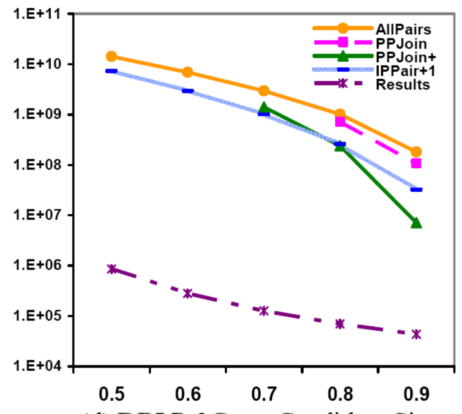
(a) DBLP Running Time



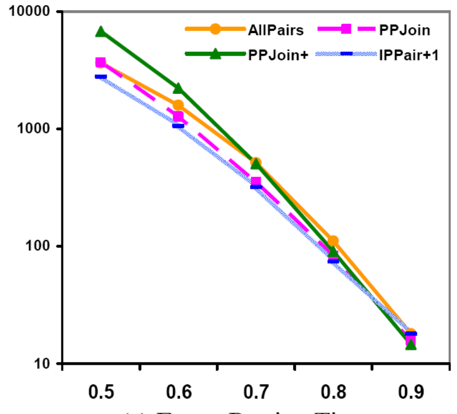
(b) DBLP Candidate Size



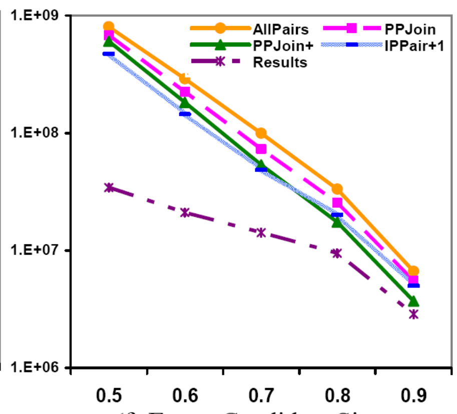
(c) DBLP-3Gram Running Time



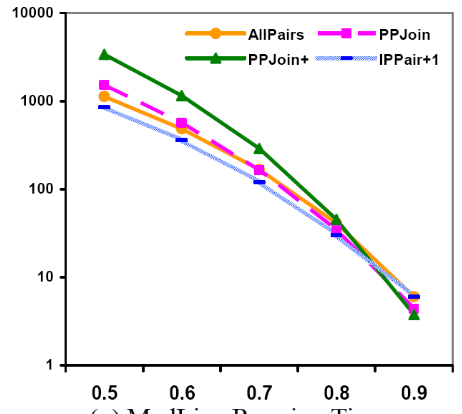
(d) DBLP-3Gram Candidate Size



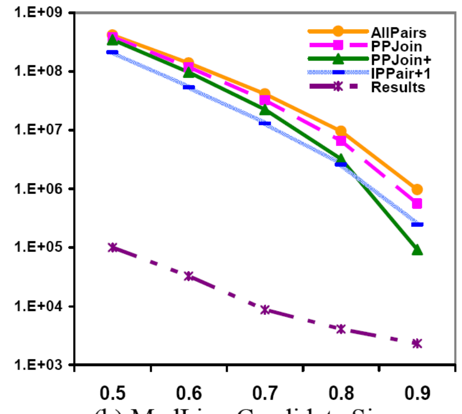
(e) Enron Runing Time



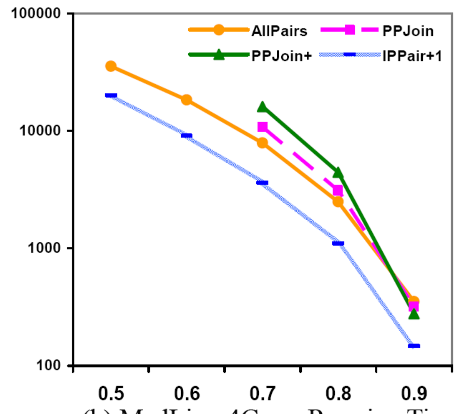
(f) Enron Candidate Size



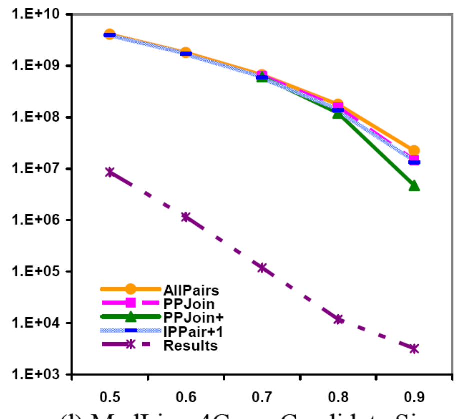
(g) MedLine Running Time



(h) MedLine Candidate Size



(k) MedLine-4Gram Running Time



(l) MedLine-4Gram Candidate Size

Fig. 1. The logarithm scale running time (in seconds) and the candidate size generated by the IPPair and the state-of-the-art algorithms over various Cosine similarity thresholds.

TABLE II
THE DATASET STATISTICS

Datasets	n	avg_len	Size
DBLP	873524	14.2	53MB
DBLP-3Gram	873524	102	365MB
Enron	504680	274	559MB
MedLine	339217	133	183MB
MedLine-4Gram	339217	866	1.2GB

IPPair with one token prefix extension. In some cases, the PPJoin and the PPJoin+ were failed to finish or they finished with wrong estimation of the candidate size due to some bugs in their implementation, we ignore the errors and leave the result empty in such cases.

Observing the obtained result, we divide the experimental results into three groups with different properties. First, consider the results on small dataset DBLP on Figure 1.a and Figure 1.b. According to these figures, PPJoin+ seems the best in terms of running time and also the candidate size (the smaller the better). The obtained results are correlated with the report by the paper [8]. However, when the similarity threshold drops down from 0.7 to 0.5, the running time of the IPPair and the PPJoin+ becomes closer. The behavior of the candidate size on Figure 1.b confirms again this fact. Overall, for this dataset, PPJoin+ is the best, our IPPair is slightly better PPJoin and outperforms Allpairs.

In the next step, consider another group of datasets including the DBLP-3Gram, MedLine and the Medline-4Gram which are much larger than the DBLP. All of these datasets appear with big size and long records. A first conclusion can be made: IPPair outperforms all the other algorithms. Concretely, it can be from 40% to 2 times faster than the AllPairs and it can reduce much candidates than the AllPairs algorithm. Besides, it's interesting that although still performs well in the cases with high similarity threshold ($t \geq 0.7$), PPjoin+ is the worst algorithm even compared with her brother PPJoin when $t \leq 0.7$ (Figure 1.c.g.k). The results are surprising because the number of candidates that the PPJoin+ can filter is still much larger than the other algorithms can (Figure 1.d.h.l). A possible explanation is that the overhead of the filtering technique used in PPJoin+ is too expensive for long-records collection.

Another conclusion on the results on these datasets is that the speed-up of IPPair seems higher in the cases with low similarity threshold. A possible explanation for this behavior is that when the similarity threshold drops down from 0.7 the prefix becomes longer. As a result, the prefix extension technique adopted in IPPair becomes more efficient. Moreover, on these datasets the candidate set generated by all the algorithms is far bigger than the results set (hundreds or thousands times bigger). Under this circumstance, early termination technique is very effective.

Finally, consider the result reported on the Enron dataset. Although in this case, the IPPair still outperforms the other algorithms but the speed-up degrades. Concretely, IPPair steadily 30% faster than the AllPairs algorithm (Figure 1.e). If we look at the candidate size figure (Figure 1.f), we can guess the reason of the speed-up degradation. That is, the generated candidate size remains bigger than the result size, but it is just tens times bigger instead of hundreds or thousands times as in the other tests. In other words, the filtering phase is very effective thus the verification overhead no longer dominates the overall running time. Since the IPPair algorithm benefits from improving the verification phase, the speed-up achieved in this case is not much.

VI. CONCLUSIONS AND FUTURE WORK

We introduced some optimization techniques integrated into the existing approaches to solve the all pairs similarity search problem. The validation with 5 different datasets shows that the proposed techniques speed up the existing approaches from 30%. Especially, in the case with large, high-dimensional datasets and low similarity threshold our approaches can achieve 2x-4x speedup compared to existing approaches. Although all the state-of-the-art algorithms are proven to be efficient for small datasets, there are lack of evidences for them to scale up to bigger collections. All of algorithms are supposed to run without considering the limit of memory size. In the future, we propose to consider many optimizations such as data compression, cache-oblivious algorithm, distributed and parallel solutions to scale up the algorithm to larger datasets.

REFERENCES

[1] A. Arasu, V. Ganti, and R. Kaushik. Efficient exact set-similarity joins. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 918–929. VLDB Endowment, 2006.

[2] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 131–140, New York, NY, USA, 2007. ACM.

[3] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. *Comput. Netw. ISDN Syst.*, 29(8-13):1157–1166, 1997.

[4] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388, New York, NY, USA, 2002. ACM.

[5] G. S. Manku, A. Jain, and A. Das Sarma. Detecting near-duplicates for web crawling. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 141–150, New York, NY, USA, 2007. ACM.

[6] S. Sarawagi and A. Kirpal. Efficient set joins on similarity predicates. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 743–754, New York, NY, USA, 2004. ACM.

[7] E. Spertus, M. Sahami, and O. Buyukkokten. Evaluating similarity measures: a large-scale study in the orkut social network. In *KDD '05: The eleventh ACM conference on Knowledge discovery in data mining*, pages 678–684, New York, NY, USA, 2005. ACM Press.

[8] C. Xiao, W. Wang, X. Lin, and J. X. Yu. Efficient similarity joins for near duplicate detection. In *WWW '08: The 17th international conference on World Wide Web*, pages 131–140, New York, NY, USA, 2008. ACM.