ZigZag: Supporting Similarity Queries on Vector Space Models

Wenhai Li[1*] Lingfeng Deng[1] Yang Li[1] Chen Li[2]

[1]Computer School, Wuhan University, China. *Email: lwh@whu.edu.cn [2]University of California, Irvine, USA

ABSTRACT

In this paper we study the problem of supporting similarity queries on a large number of records using a vector space model, where each record is a bag of tokens. We consider similarity functions that incorporate non-negative global token weights as well as record-specific token degrees. We develop a family of algorithms based on an inverted index for large data sets, especially for the case of using external storage such as hard disks or flash drives, and present pruning techniques based on various bounds to improve their performance. We formally prove the correctness of these techniques, and show how to achieve better pruning power by iteratively tightening these bounds to exactly filter dissimilar records. We conduct an extensive experimental study using real, large-scale data sets based on different storage platforms, including memory, hard disks, and flash drives. The results show that these algorithms and techniques can efficiently support similarity queries on large data sets.

KEYWORDS

Similarity query, Vector space models, Elastic bounds, ZigZag

ACM Reference Format:
Wenhai Li[1*] Lingfeng Deng[1] Yang Li[1] Chen Li[2]. 2018. ZigZag: Supporting Similarity Queries on Vector Space Models. In *SIGMOD'18: 2018 International Conference on Management of Data, June 10–15, 2018, Houston, TX, USA*. ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3183713.3196936

1 INTRODUCTION

In recent years, we see an increasing number of applications that need to process large collections of records that are sets, bags, or strings. Many of them need to compute the similarity between two records, where each record consists of elements such as keywords. Depending on the application domain, a record can be a publication title, an abstract, a product review, a set of grams of a string, a vector of bits for a molecule, or a set of features of a multimedia object (e.g., image, video, or audio) [2, 4, 27, 35, 37, 41]. The following are two illustrative examples.

The human resource (HR) department of a large company has a database of its employees, where each employee has a list of professional skills such as marketing, negotiation, project management, python, C++, and Javascript. An HR manager often wants to find employees who have skills similar to a particular employee, or all pairs of employees with similar skills. In this case, two employees are considered to be relevant as long as they have enough common skills, even if their skill sets are not exactly the same. As another example, consider a collection of publications, where each publication has a title. We want to find publications with a title similar to a given title even if their titles are not exactly the same.

As shown in the examples, one general problem in these applications is to find those entities that are similar to each other. The concept of "similarity" can be very domain specific and semantically rich. In the first example, if employees have different proficiencies on the same skill, how does the HR manager decide which employees are more "similar" to a particular employee? More interestingly, if the manager prefers some skills and cares less about others, how can we take his/her preference into consideration when doing the query? Similar problems exist in other applications, which require a flexible and powerful similarity function. Many of these applications such as information retrieval and bioinformatics [24] need the support of similarity queries with semantically rich functions that consider token weights, such as term weights [17] and entity probabilities [41]. Many of them are using a tf-idf weighting method [36].

In this paper, we study the problem of similarity queries on vector space models. Compared to commonly used functions such as Jaccard and cosine, a vector space model is general in two aspects. First, it allows a token to have a weight, and different tokens can have different weights. Most existing functions assume all the tokens have the same weight. This flexibility is particularly important for applications where we want to assign different importance values to different tokens, such as employee skills in the above example, and inverse document frequencies (idf) in document search. Second, it can also allow a record-specific value per token, such as token frequencies (tf) in document search, and this flexiblity is particular important for long records with many elements. In Section 2.3 we present a detailed analysis of the benefits of vector space models compared to existing similarity functions.

We first formally define the problem of similarity queries on collections of records using a vector space model (Section 2). We then address how to support similarity queries, and focus on efficiency issues for large data sets, especially

SIGMOD'18, June 10–15, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-4703-7/18/06…$15.00
https://doi.org/10.1145/3183713.3196936

when using external storage such as hard disks or flash drives. We give a naive algorithm on top of an inverted index, and its main idea is to process the tokens in the query one by one, and scan their inverted lists to compute the similarity of each candidate record (Section 3.1). To improve its performance, we first present a technique to prune candidate records based on a bound using the remaining tokens for each candidate (Section 3.2). Next we develop two novel techniques that can effectively prune candidates based on bounds on the records and query tokens (Section 4). In Section 5 we further improve these techniques by iteratively tightening these bounds to achieve better pruning power. We conducted extensive experiments using real, large data sets on various storage platforms, including memory, hard disks, and SSD. The results showed that these algorithms can support similarity queries efficiently on large data sets (Section 6).

1.1 Related Work

The idea of discovering and exploiting term weights is a fundamental problem in the field of information retrieval. Besides the two key concepts of term frequency and inverse document frequency [26], recent methods tend to employ probabilistic models [37], frequent item mining [4], and other specified weighting strategies [27]. Together with advanced applications [22, 35] of emerging Web-scale content mining [41], many weighting methods for effective crowd extraction [2] have introduced new challenges across various domains. Chemical weights [24] in molecular and elements also benefit from the efficient processing of weighted similarity retrieval.

There are various kinds of similarity functions on strings and sets. For similarity search, many algorithms used a gram-based approach (e.g., [6, 29, 30, 34, 40, 51]). For similarity join, filtering techniques are widely used. For instance, an algorithm called gram-count [18] utilizes the fact that for two strings to be similar based on a threshold δ, their length difference should be within δ. Length filtering uses the length of a string to reduce the number of candidates.

Pruning techniques are generally used in the context of similarity queries, particularly joins, and many algorithms were developed by assuming the data can fit in main memory. Prefix filtering [5, 9, 12, 31, 38, 39, 44, 46, 49, 50] utilizes the fact that two strings are similar only if they share some commonality in their prefixes. Many algorithms have been proposed based on prefix, such as AllPair [5], PPJoin+ [50], GroupJoin [9], MPJoin [39], PartitionJoin [14], SkippingScheme [47], MPJoin-PEL [31], ED-Join [49], AdaptJoin [44], QChunk [38], VChunk [46], and Pivotal prefix [12]. Other related algorithms exist such as M-Tree [11], trieJoin [16], and PartEnum [3]. Besides the experimental studies [23, 32], there is a recent survey about string similarity queries [52]. For parallel similarity join, there are a number of studies that utilize the MapReduce framework [1, 13, 15, 25, 33, 42, 43, 48], in which many studies use the prefix to prune dissimilar records.

Many of these studies also addressed the pruning effectiveness in string similarity search [52]. In large-scale similarity search, a package called Flamingo [8] supports indexes on external storage to deal with the case of limited memory. It supports several count-based similarity functions based on set and bag semantics. Partition and merging optimizations have been introduced solve the so-called T-occurrence problem [29]. The research in [7] presented a general list-merging strategy for count-based selections using external storage.

By utilizing information retrieval factors [21], hybrid measurements have recently emerged in the literature [20, 45]. The effectiveness of a variety of similarity functions have been measured in [10] based on different data sets. The research in [19] incorporated token weights into an inverted index in a manner to guarantee the quality of subsequent query answers. It can be seen as a special case of our p-norm similarity function. Wang et al. [45] integrated weighted grams into functions such as Jaccard, Cosine, and Dice. To our best knowledge, there are few studies in the literature that address large-scale similarity selections by considering both token weights and their record-specific preferences.

2 PROBLEM FORMULATION

In this section, we formally define the problem of weighted similarity selection on vector spaces.

2.1 p-Norm Similarity

For a real number $p \geq 1$, the "p-norm" of an m-dimensional vector $\mathbf{R} = \langle r_1, \ldots r_m \rangle$ is defined as:

$$l_p(\mathbf{R}) = \sqrt[p]{\sum\nolimits_{i \in [1,m]} r_i^p}.$$

Definition 1. *(p-Norm Similarity) The p-norm similarity of two non-zero vectors $\mathbf{R} = \langle r_1, \ldots r_m \rangle, \mathbf{Q} = \langle q_1, \ldots q_m \rangle$ is*

$$W_p(\mathbf{R}, \mathbf{Q}) = \frac{\sqrt[p]{\left(\sum_{i \in [1,m]} r_i^{\frac{p}{2}} \times q_i^{\frac{p}{2}}\right)^2}}{\sqrt[p]{\sum_{i \in [1,m]} r_i^p} \times \sqrt[p]{\sum_{i \in [1,m]} q_i^p}}. \quad (1)$$

Definition 2. *(p-Norm Similarity Selection) Given a collection of records $\mathbb{S}$, a p-norm selection includes a record $\mathbf{Q}$ and a threshold τ. It is to find all the records $\mathbf{R} \in \mathbb{S}$ such that*

$$W_p(\mathbf{R}, \mathbf{Q}) \geq \tau. \quad (2)$$

A common use case of p-norm similarity is document search, where a document (record) is modeled as a bag of words. We consider the corresponding similarity defined as follows.

Definition 3. *(Bag-based p-Norm Similarity) Let R and Q be two bags of tokens. Their bag-based p-norm similarity is*

$$W_p(R, Q) = \frac{\left(\sum_{t \in R \cap Q} (f(t,R)w(t))^{\frac{p}{2}} (f(t,Q)w(t))^{\frac{p}{2}}\right)^{\frac{2}{p}}}{\left(\sum\limits_{t \in R} (f(t,R)w(t))^p \sum\limits_{t \in Q} (f(t,Q)w(t))^p\right)^{\frac{1}{p}}}. \quad (3)$$

In the formula, $w(t)$ is the weight (a non-negative real number) of a token t. Equation 1 becomes Equation 3 when the elements in a vector have a one-to-one mapping to the tokens in the domain $\mathbb{S}$. In this case, each element $r_i \in \mathbf{R}$ is denoted by a global weight $w(t)$ times a record-specific

degree $f(t, R)$ if token t_i is included by R. For simplicity, we call $c(t, R) = f(t, R)w(t)$ as the *factor* of a token t in a record R. While we focus on 2-norm similarity (defined below), the results can be applied to general p-norm similarity, especially when the dimension of vector space is extremely large.

$$W_2(R, Q) = \frac{\sum_{t \in R \cap Q} f(t, R)w(t)f(t, Q)w(t)}{\sqrt{\sum_{t \in R} (f(t, R)w(t))^2} \sqrt{\sum_{t \in Q} (f(t, Q)w(t))^2}}. \quad (4)$$

For simplicity, we define the *length* of a record R as

$$\mathsf{len}(R) = \sqrt{\sum\nolimits_{t \in R} c^2(t, R)}.$$

Then Equation 4 becomes

$$W_2(R, Q) = \frac{\sum_{t \in R \cap Q} c(t, R)c(t, Q)}{\mathsf{len}(R)\mathsf{len}(Q)}. \quad (5)$$

This similarity function generalizes the commonly used tf-idf cosine function [10]. In this model, the *degree* $\mathrm{tf}(t, R)$ is the number of occurrences of a token t in a record R, and the *speciality* $\mathrm{idf}(t)$ is generally defined as the logarithmically scaled inverse fraction of the records that contain the token t. In this paper, we illustrate how to efficiently perform similarity selection using Equation 2 (when $p = 2$) and Equation 4, and the results can be extended to the general case of other p values. One can integrate any token weights and degrees into the framework. (See Appendix B.3.)

2.2 An Example

Next we use an example to illustrate these concepts. Table 1 shows a set of tokens with associated weights.

Table 1: Some tokens with their weights (as "w" fields).

token	w	token	w	token	w	token	w	token	w
know	10	strong	8	many	6	look	7	new	6
christs	11	world	6	come	6	good	6	old	8
incident	10	jordan	7	regard	10	will	4	us	5
nba	10	believe	7	larger	10	end	7	days	8
treating	12	people	5	really	6	still	7	want	6

Table 2 shows a collection of records and a query record Q, each consisting of a bag of case-insensitive tokens, with a length and the similarity to the query record Q (see Equation 4). For instance:

$$\mathsf{len}(Q) = \sqrt{\sum\nolimits_{t \in Q} (\mathrm{tf}(t, Q)w(t))^2} = 28.1,$$

$$\mathsf{len}(R_8) = \sqrt{\sum\nolimits_{t \in R_8} (\mathrm{tf}(t, Q)w(t))^2} = 26.1.$$

In addition,

$$l_2(R_8, Q) = \sum\nolimits_{t \in Q \cap R_8} \mathrm{tf}(t, R_8)\mathrm{tf}(t, Q)w^2(t) = 520,$$

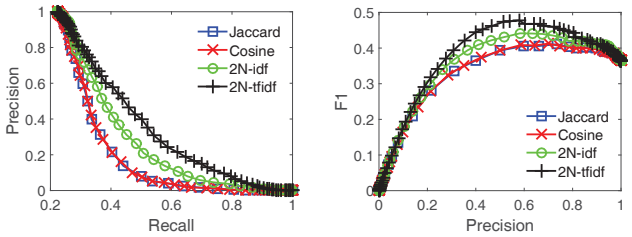
and their similarity is $520/(28.1 \times 26.1) \simeq 0.7$. Therefore, if the similarity threshold τ is 0.7, then R_8 should be an answer. We can show that R_3 is an answer as well.

Table 2: Sample records $\mathbb{S}$, a similarity query record Q, and records' lengths and similarities with respect to Q.

id	Record	Len	Sim
R_1	many christs believe world will come end	18.5	0.5
R_2	larger us christs still believe christs	26.6	0.6
R_3	children want good christs	13.9	0.7
R_4	new people know incident treating old people	23.3	0.5
R_5	people regard jordan nba really good	18.6	0.1
R_6	believe strong us good days new world	17.6	0.1
R_7	good look jordan nba treating people regard	22.4	0.3
R_8	old christs really good christs days	26.1	0.7
Q	:christs people treating incident good christs	28.1	

2.3 Comparison with Existing Functions

The p-norm similarity function is a generalized form of the cosine function by allowing a token to have a record-specific weight in each record. The cosine function can be viewed as a special p-norm similarity function where $f(t, R)$'s are the same for all tokens t in all records R, and the tokens carry the same weight $w(t) = 1$. The generalization is especially important for a large collection of long records, since in such a case both the global weight and the frequency of a token in a record are valuable in the similarity function.



(a) Precision versus recall. **(b) F1 versus precision.**

Figure 1: Effectiveness comparison of different functions.

We conducted an experiment to evaluate the effectiveness of a few common similarity functions. One challenge in such an experiment is that the same similarity threshold has different semantic meanings for different functions. To overcome this problem, we conducted the experiment as follows. Given the ground truth about similar pairs of records, for each function f, for each threshold τ, we did a similarity search using f and τ, and computed the precision and recall of this search. Figure 1a shows the results on a data set called "AskUbuntu" for Jaccard, Cosine, idf 2-norm (where all token degrees are the constant 1), and tf-idf 2-norm. Given a recall, the higher the precision of a function, the better the function is. The figure shows that the tf-idf 2-norm function gave the best accuracy results. More details of the experiments are in Appendix B. Notice that the purpose of this experiment was *not* to show that the p-norm function is the best for all applications. Instead, it was to show that it is a better similarity in certain applications.

In this paper we focus on queries when a threshold is given, as in previous studies in the literature. A good threshold

largely depends on the application domain and query-specific requirements, and how to decide this threshold itself needs a separate study. Our techniques can be used to help answer nearest neighbor queries, which can be supported by running selection queries with a gradually decreasing threshold.

3 BASIC SEARCH METHODS

In this section we first present a naive search algorithm based on inverted index, then develop an improved algorithm by doing effective pruning of candidates.

3.1 Naive Algorithm

The following lemma is based on Equation 5.

LEMMA 1. *Let $\mathbb{S}$ be a collection of records. A record $R \in \mathbb{S}$ satisfies $W_p(\mathbf{R}, \mathbf{Q}) \geq \tau$ if and only if we have*

$$\sum\nolimits_{t \in R \cap Q} c(t,R)c(t,Q) \geq \tau \mathsf{len}(R)\mathsf{len}(Q). \quad (6)$$

Clearly an answer record should share at least one token with the query record Q. Thus we can develop a naive algorithm as follows. We first build an inverted index, where the record list of each token contains the ids of all the records including this token. Given a query record Q, we scan the tokens in Q and compute the union $\mathbb{C}$ of their inverted lists, which is a candidate set of answers. We verify each record in $\mathbb{C}$ using Equation 6, and return those that satisfy the equation as answers to the query. Next we use new concepts to illustrate how to improve the performance of the algorithm.

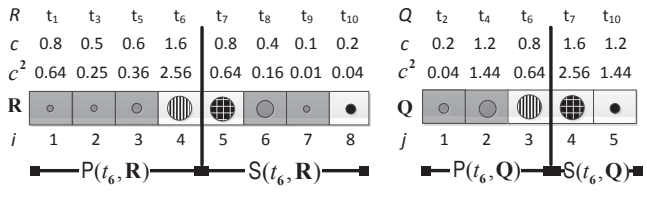
DEFINITION 4. *(Processed versus Remaining Tokens) For a total order $\mathbb{O}$ of tokens, we sort the tokens in a record R following $\mathbb{O}$. The* processed tokens *of R w.r.t. a position i are the list of tokens in $\mathbf{R}$ with a position no larger than i, while the* remaining tokens *of R w.r.t. i are the list of tokens in $\mathbf{R}$ with a position larger than i.*

Figure 2 shows an example. Consider a total order $\mathbb{O} = \{t_1, \ldots, t_{10}\}$, using which we scan the tokens in the query record Q. At token t_6, we have processed tokens $\mathsf{P}(\mathbf{R}, t_6) = \langle t_1, t_3, t_5, t_6 \rangle$, and their length $\mathsf{len}(\mathsf{P}(\mathbf{R}, t_6))$ is

$$\sqrt{(0.1 \times 8)^2 + (0.1 \times 5)^2 + (0.2 \times 3)^2 + (0.8 \times 2)^2} = 1.95.$$

The remaining tokens are $\mathsf{S}(\mathbf{R}, t_6) = \langle t_7, t_8, t_9, t_{10} \rangle$ with the length of $\mathsf{len}(\mathsf{S}(\mathbf{R}, t_6)) = 0.92$. Also, we have $\mathsf{len}(\mathsf{P}(\mathbf{Q}, t_6)) = 1.46$ and $\mathsf{len}(\mathsf{S}(\mathbf{Q}, t_6)) = 2$.

In the figure, the factor of each token in a record can be computed as the product of the degree $f(t, R)$ and the global weight $w(t)$. If we store the degree $f(t, R)$ in each record entry in the inverted index, it can be used to compute the product efficiently during the scan of each list. If we further store the length $\mathsf{len}(R)$ in each record entry, the verification can be performed more efficiently. In our running example, given a token t in Fig. 2, suppose the record length $\mathsf{len}(R)$ and the degree $f(t, R)$ are stored in the entry of R on the inverted list of t. Based on the total order $\mathbb{O}$, we first scan the record list of the first token t_2 in $\mathbf{Q}$. The record R does not contain this token, and R will not appear in the candidate $\mathbb{C}$. When we scan the record list of t_6, we find R with its degree



$\mathbb{O}$	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
$w(t)$	0.1	0.2	0.1	0.4	0.2	0.8	0.8	0.4	0.1	0.2
$f(t,R)$	8	0	5	0	3	**2**	**1**	1	1	**1**
$f(t,Q)$	0	1	0	3	0	**1**	**2**	0	0	**6**

Figure 2: Processed and remaining tokens of a record R and a query record Q based on a total token order.

$f(t_6, R) = 2$ and length $\mathsf{len}(R) = 2.16$, and the summation of the token-wise factor products before (including) token t_6 can be computed as $aw(R, Q) = f(t_6, R)f(t_6, Q)w^2(t_6)$. We next consider t_7 and t_{10} in $\mathbf{Q}$, each of which contributes a factor to the summation $aw(R, Q)$. After processing all the tokens of $\mathbf{Q}$, we have the summation $aw(R, Q)$. Also, we have the length $\mathsf{len}(R)$, thus can use Inequality 6 to determine whether R is an answer.

3.2 Using Bound on Remaining Tokens

We next study how to prune records in the naive algorithm using a bound on the factor products of remaining tokens. In particular, if we organize the tokens in each record following the total order, we can prune records based on the common tokens processed so far and the length of remaining tokens. The pruning is based on the following lemmas.

LEMMA 2. *Given a total order $\mathbb{O}$ and a threshold τ, each token t_k shared by an answer record $\mathbf{R}$ at its position i and the query record $\mathbf{Q}$ at its position j should satisfy*

$$\sum_{l=1}^{k} (c(t_l, R)c(t_l, Q)) + \mathsf{len}(\mathsf{S}(t_k, \mathbf{R}))\mathsf{len}(\mathsf{S}(t_k, \mathbf{Q})) \geq \delta_{R,Q}, \quad (7)$$

where $c(t, R)c(t, Q) = 0$ if $t \notin R \cap Q$.

LEMMA 3. *Given a total order $\mathbb{O}$ and a threshold τ, we scan the tokens in Q following the order $\mathbb{O}$, and accumulate the factor products of the common tokens shared by R and Q. The following equality holds when we completely scan all the tokens in Q:*

$$\sum\nolimits_{t=1}^{|Q|} (c(\mathbf{Q}[l], R)c(\mathbf{Q}[l], Q)) = \sum\nolimits_{t \in R \cap Q} c(t, R)c(t, Q).$$

To utilize these results to do pruning, we need to have the factor of each token in a record, as well as the remaining length after each position of a record $\mathbf{R}$. We can achieve the goal by storing both the factor and the related weights in the inverted index, as shown in Figure 3. We store the token weight $w(t)$ (or specialty idf(t) in the tf-idf case) in the token entry. The number of tokens is generally limited, so such information can be maintained in memory. Each entry on an inverted list includes four elements: $\langle rid, L, P, f \rangle$, where rid is a record identifier, L is the total length $\mathsf{len}(R)$, P is the length of tokens of R up to this token $\mathbf{R}[i] = t$, i.e.,

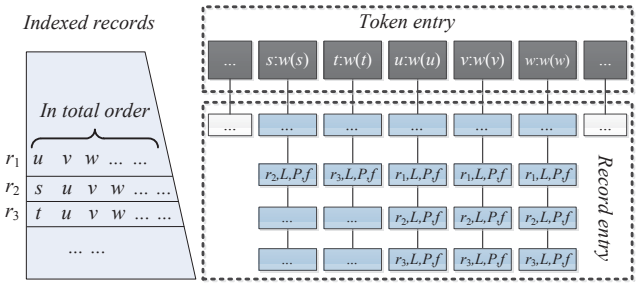


Figure 3: Inverted index where each entry has a total length L, processed-token length P, and f of the record.

$P = \mathsf{len}(\mathsf{P}(t, \mathbf{R}))$. We can derive the length of the remaining tokens after position i using the following equation:

$$\mathsf{len}(\mathsf{S}(\mathsf{t}, \mathbf{R})) = \sqrt[p]{\mathsf{len}^p(R) - \mathsf{len}^p(\mathsf{P}(t, \mathbf{R}))}. \quad (8)$$

The value f is the degree of token t in the record R. It guarantees that we can compute the factor $c(t, R) = f(t, R)w(t)$ when we incrementally measure the common tokens between R and Q.

Algorithm 1 is a baseline algorithm that uses the additional information to do effective pruning. We use the example given in Figure 2 to show the benefit of the algorithm. Suppose the tokens of record R have been maintained in the inverted index as shown in Figure 3. We use a query record Q to probe each of its ordered tokens $\{t_2, t_4, t_6, t_7, t_{10}\}$. Given a threshold 0.8, when processing token t_6, we know that the current common token produces the pair-wise factor $1.6 \times 0.8 = 1.28$ while their remaining-token lengths are 0.92 and 2, respectively. Since their total lengths can derive a lower bound $\delta_{R,Q} = 0.8 \times 1.89 \times 2.55 = 3.37$, we have $1.28 + 0.92 \times 2 = 3.12 < \delta_{R,Q}$. Based on Lemma 2, we remove R from the candidate set without any verification. This pruning step makes this algorithm more efficient than the naive algorithm.

Algorithm 1: Baseline: Selection using an inverted index.

Input : I : Inverted index; $\mathbb{O}$: Token order; $\langle Q, \tau\rangle$: Query.
Output : A set of similar records O.

```
Scan the inverted lists of the tokens in Q following order 𝕆;
Build a map ℂ : ⟨R, aw(R,Q)⟩ for each candidate record in 𝕊;
Read the inverted list of each token t in Q from I;
Do the following steps for each entry ⟨rid, L, P, f⟩ in a list;
if Inequality 6 holds then
    Add record R to O, and remove R from ℂ if R ∈ ℂ;
else if Inequality 7 holds and rid ∉ ℂ then
    Add ⟨R, aw(R,Q) = c(t,R)c(t,Q), L⟩ to ℂ;
else if Inequality 7 holds and rid ∈ ℂ then
    Increase aw(R,Q) of R in ℂ by c(t,R)c(t,Q);
else
    Remove R from ℂ and no longer consider R;
end
Check the candidates in ℂ with aw and L, and output O;
```

4 BOUNDS ON LENGTH AND PREFIX

Next we develop two advanced pruning techniques to improve the baseline algorithm. The firs technique is based on length bounds of the records on an inverted list, and the second one is based on a bound on the prefix of the query record.

4.1 Pruning Using Record-Length Bounds

We first improve the baseline algorithm by utilizing the following bounds on the lengths of candidate records.

LEMMA 4. *Given a similarity query with a record Q and a threshold τ, we have the following lower bound ($L_{\tau,Q}$) and upper bound ($U_{\tau,Q}$) on the length for an answer record R:*

$$\mathsf{len}(R) \geq L_{\tau,Q} = \tau\mathsf{len}(Q)\left(\max_{t\in Q}\left(\frac{f(t,Q)}{\min_{S\in\mathbb{S}} f(t,S)}\right)\right)^{-1}, \quad (9a)$$

$$\mathsf{len}(R) \leq U_{\tau,Q} = \max_{t\in Q}\left(\frac{\max_{S\in\mathbb{S}} f(t,S)}{f(t,Q)}\right)\frac{\mathsf{len}(Q)}{\tau}. \quad (9b)$$

The proof of the lemma is in Appendix A.3. Since only the records that have non-zero degrees are maintained on the inverted list of a token, we have $\min_{S\in\mathbb{S}}(f(t,S)) > 0$. In tf-idf, we can assume $\min_{S\in\mathbb{S}}(f(t,S)) = 1$ for $\forall t \in R \cap Q$ without loss of generality.

As the tokens in Q have different maximal factors in $\mathbb{S}$, we can derive different lower/upper bounds for these records. Thus, we give a tighter bound than that in Inequality 9b.

LEMMA 5. *Given a query record Q and a threshold τ, each answer record R satisfies the following inequality:*

$$\mathsf{len}(R) \leq U_{\tau,Q} = \frac{\sum_{t\in Q} w^2(t)\max_{S\in\mathbb{S}} f(t,S)f(t,Q)}{\tau\mathsf{len}(Q)}. \quad (10)$$

4.1.1 Storing Length Bounds on Inverted Lists. To use these bounds, we revise the inverted index in Figure 3 by storing length pointers for each inverted list, and sorting the entries of each list based on the increasing order of their lengths. To efficiently locate an entry for a length, we can partition a list into a sequence of groups, and maintain an array of length pointers *lenptrs* (in each token entry) to point to the length of the first record within each group, as shown in Figure 4. In addition to the specialty, a maximal degree and an array of length pointers are stored in each token entry. We maintain the maximal degree $\max_{S\in\mathbb{S}} f(t,S)$ for a token t to store the maximal degree of all the records on the inverted list of t. For example, there is an entry r_8 on the inverted list of token s, of which the degree $f(s, r_8) = 8$ is the maximal value of all the records including s. When we derive the two bounds in Inequalities 9a and 10, this maximal degree can be obtained by probing a token entry in Q. Notice that this structure does not apply in the traditional set-based similarity selection as all tokens in that setting have the same degree [29]. In the vector space model, however, the variant degrees have a significant effect on the length bounds. As shown in Figure 4, we scan the tokens of Q in the increasing order of their frequencies. Such an order allows us to process

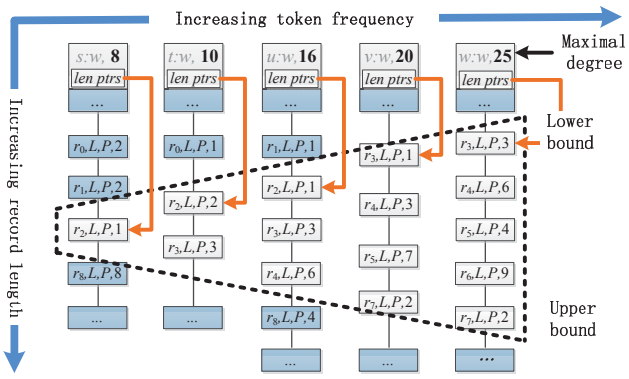


Figure 4: Scanning record entries to perform Vertical Filter.

the least frequent tokens first. To efficiently skip irrelevant records, we sort each inverted list in the increasing order of their record lengths. One can probe the length pointers in a token entry to obtain the last group that has its first record length no larger than the lower bound given in Inequality 9a. As shown in Figure 4, the record entry could be pointed by a length pointer. While one can provide a length pointer for each record entry, this approach may require a lot of space in the token entry. To solve this problem, we split inverted lists into multiple groups, each is pointed to by a pointer $\langle len, ptr\rangle$. We take the length of the first record in a group as its len field, and maintain the location of this entry as ptr. We can locate the last group that satisfies the lower bound directly. Following the inverted lists from this group, we can stop by the first record with its length larger the upper bound. By storing each list in a sequence of fixed-size blocks, we can focus on the inverted lists within length bounds.

4.1.2 Algorithm Zig*: Using Length Bounds.* Algorithm 2 shows algorithm Zig that uses the length bounds to improve the baseline algorithm. The two bounds are used in lines 7 and 8. To discard the records that have their total lengths smaller than $L_{\tau,Q}$, line 7 searches the length pointers of a token t, which comes from the ordered tokens in the query record $\mathbf{Q}$ (sorted by line 1). Thus the first entry of the group satisfying Inequality 9a is processed. By iteratively scanning each record entry bounded by line 8, we accumulate the factor products of tokens shared by a candidate record and the query in line 11. For the record in $\mathbb{C}$, line 13 verifies whether its remaining length and accumulated factor products with Q satisfy Inequality 7. It checks the lower bound $\delta_{Q,re.rid}$ against aw and the remaining lengths as follows. For the new record (with its factor product initialized by line 9) and the existing candidate records in $\mathbb{C}$ (probed in the hash map $\mathbb{C}$ in line 10), a record $re.rid$ has all its information $\langle rid, L, P, f\rangle$ associated in its record entry re. We compute the remaining length $\mathsf{slen}(re)$ (in line 13) using its total length L and processed-token length P based on Equation 8. In line 6, we reduce the remaining length of Q by the factor of t, such that line 13 can use Inequality 7 to either remove a candidate in line 14 or update $\mathbb{C}$ by the latest record in line 16.

Algorithm 2: Zig: Selection using length bounds.

Input : I : Inverted index; $\mathbb{O}$: Token order; $\langle Q, \tau\rangle$: Query.
Output : A set of similar records O.

```
1  Sorting tokens in Q using 𝕆;        // Selection based on 𝕆
2  sq = len(Q); // Initialize remaining-token length of Q
3  ℂ ← ∅ ; // Init the candidates with an empty hash map
4  for t in Q do
5      w ← I[t].w;
6      sq = √(sq² − w² × f²(t, Q));        // Remaining-token
       length of Q
7      re ← I[t].lenptr by len < L_{τ,Q};   // Begin from Eq. 9a
8      while re.L ≤ U_{τ,Q} do     // End iteration by Eq. 10
9          aw = w² × re.f × f(t, Q);
10         if ℂ contains re.rid then          // Accumulating
11             aw = ℂ[re.rid].aw + aw
12         end
13         if aw + slen(re) × sq < δ_{Q,re.rid} then
14             Remove re.rid from ℂ;                // By Eq. 7
15         else
16             ℂ ← ⟨re.rid, ⟨aw, re⟩⟩;                // Update
17         end
18         re ← next(re);
19     end
20 end
21 O ← Check(ℂ); // Directly check ℂ using Inequality 6
```

By Lemma 3, as we have maintained the accumulated factor products of Q and each record in $\mathbb{C}$, we directly check the candidates in $\mathbb{C}$ in line 21. We do not extract the remaining tokens in each record in $\mathbb{C}$ as the summation of their factor products will not increase after we scan all the tokens in Q.

4.2 Pruning Using Prefix Bounds

Our second pruning technique is based on the observersation that we can find all answers by only considering a subset (prefix) of tokens in the query record Q, as shown in the following lemma.

Lemma 6. *Based on a total token order $\mathbb{O}$, a record $\mathbf{Q}$ should share at least one token with any of its answer records before the following token position*

$$j \left| \sum_{t\in\mathbf{Q}(j:]} \left(w^2(t) f(t,Q) \max_{S\in\mathbb{S}} f(t,S) \right) < \eta_{\tau,Q} \right. , \tag{11}$$

where $\eta_{\tau,Q} = \tau L_{\tau,Q} \mathsf{len}(Q)$.

Appendix A.5 gives a proof. This lemma shows that we can compute all the answers by only considering a subset of the tokens in Q. Based on Equation 11 we introduce the following defintion.

Definition 5. *(Prefix) Given a total token order $\mathbb{O}$, the* prefix *of a query record $\mathbf{Q}$ for a threshold τ can be defined as $P_\tau(Q) = \mathbf{Q}[:p]$ where*

$$p = \min_{j\le|Q|} \left(j \left| \sum_{t\in\mathbf{Q}(j:]} \left(w^2(t) f(t,Q) \max_{S\in\mathbb{S}} f(t,S) \right) < \eta_{\tau,Q} \right. \right) . \tag{12}$$

Zag: Pruning Using Prefix. We can integrate the prefix idea into the Zig algorithm as follows. We modify the vector $\mathbf{Q}$ in line 4 to $P_\tau(Q)$. From Definition 5, we can compute the prefix of the query record independently from records in the data set. We denote the following improved algorithm as Zag to highlight its idea of choosing a prefix "horizontally."

Algorithm 3: Zag: Selection using prefix based on Zig.

```
   Input : RI : Record index; ...
   Output : A set of similar records O.
   ...
4  for t in P_τ(Q) do   // Compute and use prefix by Def. 5
       ...
20     for ⟨re.rid, ⟨aw, re⟩⟩ ∈ ℂ do     // Additional pruning
21         if aw ≥ δ_{Q,re.rid} then   // Already similar to Q
22             Move re.rid from ℂ to O;
23         else if re.aw + slen(re) × sq < δ_{Q,re.rid} then
24             Remove re.rid from ℂ;              // By Eq. 7
25         end
26     end
27 end
28 O ←verify(ℂ, RI) by probing RI;     // Using Inequality 6
```

Tradeoff between prefix length and verification cost. Definition 12 shows a minimal number of tokens in the query record Q using which we can find all similar answers. Clearly we can still find all these answers by considering more tokens, i.e., by considering a "longer prefix." As we increase the prefix length, we have to pay a higher cost by accessing more inverted lists. At the same time, these additional lists can also help us prune more candidates, thus reduce the verification cost. In our experiments we will evaluate the tradeoff between the prefix length and the verification cost.

5 PRUNING WITH ELASTIC BOUNDS

We next study how to improve the performance by iteratively tightening the bounds to achieve better pruning power.

5.1 Tightening Length Bounds Iteratively

In Inequalities 9a and 10, the bounds are computed by the minimal and maximal degrees of all the records of a token. One observation is that when we scan the inverted lists along with the ordered tokens, the degree estimations for both bounds come from $\mathbb{S}$ in its entirety. They can be tightened by only using records that could be similar to the query, as illustrated in Figure 5. If we want to scan the ordered tokens $\mathbf{Q} = \{s, t, u, v, w\}$, we first use the entire collection $\mathbb{S}$ to estimate $\max_{S\in\mathbb{S}} f(s, S)$. When considering t, we find that its maximal degree within the dashed area is $f(t, r_3) = 3$, which is much smaller than the estimation using $\mathbb{S}$, i.e., $\max_{S\in\mathbb{S}} = 10$. The upper bound within the dashed area is tighter thus can be used to do better pruning. Similarly, we also have a tighter lower bound within the dashed area than Inequality 9a. In addition, if we re-check the candidate set $\mathbb{C}$ when considering a new token in Q, some candidates generated by previous tokens may no longer be similar to Q.

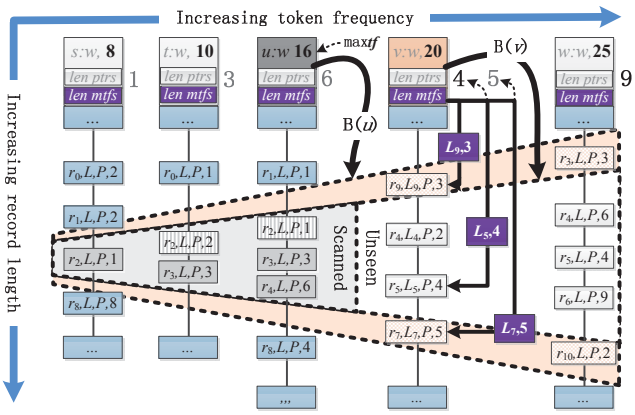


Figure 5: Bounding lengths by elastic maximal degrees.

We can iteratively tighten the bound in Inequality 10 as follows. We focus on the records that are potentially similar to Q, say, the similar records not in the output. We call them "undecided records," while the records already in the output are called "decided records." Instead of estimating the maximal degree in the entire collection, i.e., $\max_{S\in\mathbb{S}} f(t, S)$, we next show how to tighten the degree bounds based on the record set $\mathbb{B}$ that satisfies the latest length bounds. When we want to process the $(j+1)^{\text{st}}$ token of $\mathbf{Q}$, as the first j tokens $\mathbf{Q}[:j]$ have been processed, the records bounded by $\mathbb{B}$ will be used to estimate the bounds of degree of $\mathbf{Q}[j+1]$.

Lemma 7. *Given a query record $\mathbf{Q}$ with its tokens sorted by a total order $\mathbb{O}$, if we have a candidate set $\mathbb{C}$ and a bounded set $\mathbb{B}$ after processing its first j tokens $\mathbf{Q}[:j]$, any undecided record that is similar to Q should have the upper bound of*

$$U_{\tau,Q,j} = U_l + U_r, \qquad \textit{where} \tag{13}$$

$$U_l = \frac{\sum_{t\preceq\mathbf{Q}[j]} w^2(t) f(t, \max_{S\in\mathbb{C}} f(t,S)) f(t,Q)}{\tau \mathsf{len}(Q)}, \tag{14a}$$

$$U_r = \frac{\sum_{t\succ\mathbf{Q}[j]} w^2(t) f(t, \max_{S\in\mathbb{B}} f(t,S)) f(t,Q)}{\tau \mathsf{len}(Q)}. \tag{14b}$$

In this lemma, the symbol "$t \preceq s$" denotes that the total order of token t is no larger than that of s, i.e., $\mathbb{O}(t) \le \mathbb{O}(s)$. Also, the symbol "$t \succ s$" means $\mathbb{O}(t) > \mathbb{O}(s)$. Appendix A.6 proves this lemma. It says that the undecided candidates can be rechecked using the tighter upper bound given in Equation 13, which can provide a better pruning power. Similarly we can derive a tighter lower bound on the length.

Lemma 8. *Suppose we have a candidate set $\mathbb{C}$ and a record set $\mathbb{B}$ after we have processed the first j tokens $\mathbf{Q}[:j]$, any undecided record that is similar to $\mathbf{Q}$ has the lower bound of*

$$L_{\tau,Q,j} = L_l + L_r, \qquad \textit{where} \tag{15}$$

$$L_l = \tau \mathsf{len}(Q) \left(\max_{t\in Q} \left(\frac{f(t,Q)}{\min_{S\in\mathbb{C}} f(t,S)} \right) \right)^{-1}, \tag{16a}$$

$$L_r = \tau \mathsf{len}(Q) \left(\max_{t\in Q} \left(\frac{f(t,Q)}{\min_{S\in\mathbb{B}} f(t,S)} \right) \right)^{-1}. \tag{16b}$$

Similar to Lemma 7, this lemma can be used to provide a tighter bound than Inequality 9a. From the proof of Lemma 7, we can see that the undecided records that could be similar to Q determine the maximal degrees of the tokens before $\mathbf{Q}[j]$, and the records within the bounds that are derived from the latest maximal degrees produce the maximal degrees in the next step. The following theorem can be used to iteratively tighten the two bounds.

THEOREM 9. *If we have a candidate set $\mathbb{C}$ and a record set $\mathbb{B}$ after we have processed the first j tokens $\mathbf{Q}[:j]$, we have the following bounds when we scan the remaining tokens.*

$$L_{\tau,Q} = \min\left(\min_{S\in\mathbb{C}}\left(\mathsf{len}(S)\right), L_r\right), \tag{17a}$$

$$U_{\tau,Q} = \max\left(\max_{S\in\mathbb{C}}\left(\mathsf{len}(S)\right), U_r\right). \tag{17b}$$

5.1.1 Zig+: Tightening Bounds Iteratively. The following is an algorithm called Zig+ that uses Theorem 9 to tighten length bounds iteratively. Based on the inverted index given in Figure 4, we introduce another vector for maximal degrees in each token entry for these bounds. It is used for purposes. 1) We want to use the bounds given in Theorem 9 to probe the token entries; 2) We want the degree mappers to provide the maximal degree w.r.t. the length bounds.

Algorithm 4: Zig+: Selection with elastic bounds in Zig.

```
   Input : I : Inverted index; ℂ: Token order; ⟨Q, τ⟩ : Query.
   Output : A set of similar records O.
   ...
   j = 1
   new ub[|Q|]          // Init the maximal degrees for the
   tokens in Q
 4 for t in Q do
   |  ...
20 |  for ⟨re.rid, ⟨aw, re⟩⟩ ∈ ℂ do     // Additional pruning
21 |  |  if aw ≥ δ_{Q,re.rid} then   // Already similar to Q
22 |  |  |  Move re.rid from ℂ to O;
23 |  |  else if re.aw + slen(re) × sq < δ_{Q,re.rid} then
24 |  |  |  Remove re.rid from ℂ;              // By Eq. 7
25 |  |  end
26 |  end
27 |  while true do
28 |  |  Prune ℂ by Eq. 13 with the latest ub;
29 |  |  Update U_{τ,Q} by Eq. 17b with the latest ub;
30 |  |  if ℂ or U_{τ,Q} changed then
31 |  |  |  Update ub;    // Update the maximal degrees
32 |  |  else
33 |  |  |  break;
34 |  |  end
35 |  end
36 |  j ← j + 1;
37 end
38 O ← ℂ                  // Directly output all keys in ℂ
```

Figure 5 shows the corresponding index structure, where an array $lenmtfs$ is maintained in each token entry. Its element is $\langle len, mtf\rangle$, which implies that the length len of a record has a stepping degree mtf on the inverted list. We denote a stepping degree $mtf = f(t, R)$ as the degree associated with a record R on an inverted list of token t, in which any record with a length no larger than R always has its degree (of t) smaller than that of R, i.e., $\exists_{mtf=f(t,R)} : \forall_{t\in S}, \mathsf{len}(S) \leq \mathsf{len}(R) \Rightarrow f(t,S) < f(t,R)$. Take token v as an example. Its inverted list contains four records $\{r_9, r_4, r_5, r_7\}$ (we omit the other entries denoted by ellipsis), which respectively have the degrees $\{3, 2, 4, 5\}$. Thus, $v.lenmtfs$ includes three elements, namely, $\langle l_9, 3\rangle, \langle l_5, 4\rangle, \langle l_7, 5\rangle$, where $l_\star$ gives the length of this record. Given an upper bound, e.g., $l_5 \leq \mathbb{B}_u < l_7$, we directly obtain the maximal degree of t bounded by $\mathbb{B}$, i.e., $\max_{S\in\mathbb{B}} f(t,S) = 4$. In each inverted list, we incrementally maintain its degree mappers as follows. We scan its entries, and add an element to the mappers if its degree in a record is larger than the mtf value of the last element. In the above example, record r_4 does not generate an element as its degree 2 is no larger than that of the last element $\langle l_9, 3\rangle$.

5.1.2 Analysis on Space Usage. So far, we have introduced two important structures, namely, the length pointers and the degree mappers. As discussed in Section 4.1.1, we group the length pointers, such that each group of record entries has only one element. For the degree mappers, similarly, we maintain an element for each stepping degree on an inverted list. The memory overhead of these structures can be configured by different grouping/stepping size. It is efficient for the tf-idf weighting scheme, as in this case the degrees (tf) are small integers even in very long inverted lists. We remove the length pointer (or degree mapper) that has only one element, and directly probe their token entries if necessary.

We introduce the above elastic bounds into Algorithm 2 by using two additional phases at the end of each step. In Algorithm 4, we add lines 20∼37 before line 20 of Algorithm 2 by iteratively applying two phases after each token in Q has been processed. Before the selection from line 4, we initialize a cursor j and a vector of maximal degrees for the tokens in Q. Using the vector **ub**, we tighten the bounds $U_{\tau,Q}$ (lines 27∼35). After each inverted list has been scanned, line 28 first uses Equation 13 to recheck the candidates, where the checking bound $U_{\tau,Q,j}$ comes from both the current candidates $\mathbb{C}$ (for $\max_{S\in\mathbb{C}} f(t,S)$) and the latest **ub** (for $\max_{S\in\mathbb{B}} f(t,S)$). As shown earlier, the maximal degrees of the tokens in Q can be efficiently obtained by probing the latest upper bound $U_{\tau,Q}$ against the degree mappers. Also, line 29 uses Equation 17b to shrink the upper bound based on the latest **ub**. The key structure **ub** will be updated by line 31 depending on the updated $\mathbb{C}$ and $U_{\tau,Q}$. We maintain the degrees of all the scanned tokens in each candidate to update **ub** by $\mathbb{C}$, and use the degree mappers to update **ub** by the latest $U_{\tau,Q}$. The three components will iterate until no more changes are observed (as shown in line 33). We use a pruning phase (lines 20∼26 in Algorithm 4) to shrink $\mathbb{C}$ for better supporting the shrinking phase shown above. Notice that different from line 16 in Algorithm 2, Zig+ will not add a new record into $\mathbb{C}$ if its total length is larger than U_τ (in Equation 14b). Compared to Zig, the elastic bounds in Zig+ can improve the pruning efficiency.

5.2 Tightening Prefix Bounds Iteratively

We next propose a technique to iteratively tighten the query prefix and present two algorithms. Based on the idea that the maximal degree of a token should be determined by the records that satisfy the bounds, we develop a method that exploits the interdependency between the bounds and maximal degrees to iteratively tighten them. As shown in Equation 12, the prefix also relies on the maximal degrees of all the tokens in the query. So we can apply the maximal degrees to determine a new prefix in each step. As shown in Appendix A.5, the prefix bound depends on the accumulated factor products of the query and the maximal degrees. Since the maximal degrees could gradually decrease, we can obtain a shorter prefix by utilizing the latest maximal degrees after we update them in line 31 of Algorithm 4.

Definition 6. *(Elastic Prefix) If a query record* $\mathbf{Q}$ *has its tokens sorted by a total order* $\mathbb{O}$*, and we have a bounded set* $\mathbb{B}$ *after processing the* j^{th} *token in* $\mathbf{Q}[j]$*, the* j^{th} *elastic prefix of* $\mathbf{Q}$ *on the threshold* τ *is defined as* $P_{\tau,j}(Q) = \mathbf{Q}[:p]$*, where*

$$p = \min_{i \geq j} \left(i \left| \sum_{t \in \mathbf{Q}(i:]} \left(w^2(t) f(t,Q) \max_{S \in \mathbb{B}} f(t,S) \right) < \eta_{\tau,Q} \right. \right), \quad (18)$$

and its bound $\eta_{\tau,Q} = \tau L_{\tau,Q} \mathsf{len}(Q)$ *depends on Equation 17a.*

Slightly different from Definition 5, this new definition formulates a prefix with elastic bounds. After we have scanned the j^{th} token in $\mathbf{Q}$, we can use this definition to dynamically update the prefix using the latest lower bound (which is used to compute the bound of the remaining-token length $\eta_{\tau,Q}$) and maximal degrees. We continuously adjust the prefix bound i, and terminate the process when the length of the remaining tokens $\mathbf{Q}(i:]$ is less than the bound $\eta_{\tau,Q}$. The condition $i \geq j$ ensures that the updated prefix always includes the tokens scanned so far. We claim that as the upper bound of the length never increases, this elastic prefix is always no longer than the one determined by Definition 5. We integrate both of the prefixes into the algorithm Zig+, as given Algorithms 5 and 6. We highlight their modifications on Zig+ in the related lines.

Algorithm 5: Zag+: Applying elastic prefix to Zig+.

```
   Input : RI : Record index; ...
   Output : A set of similar records O.
   ...
 4 for t in P_τ(Q) do   // Compute and use prefix by Def. 5
   |  ...
37 end
38 O ←verify(ℂ, RI) by probing RI;     // Using Inequality 6
```

Different strategies to use the prefix. We have three strategies to terminate the scanning process in Zig+. 1) *Not using a prefix*: we can perform the iterative filter-and-shrink process in Zig+ without early termination. When the elastic bounds become tighter progressively, Zig+ can access short inverted lists in the remaining tokens due to the effects of length bounds. 2) *Using a prefix pessimistically*: we can use

Algorithm 6: ZigZag: Integrating elastic prefix in Zig+.

```
   Input : RI : Record index; ...
   Output : A set of similar records O.
   ...
 4 for t in P_τ,j(Q) do                  // Use elastic prefix
   |  ...
37 |  Update P_τ,j(Q); // Compute elastic prefix by Eq. 18
38 end
39 O ←verify(ℂ, RI) by probing RI;     // Using Inequality 6
```

Zag+ to conduct the filter-and-verify framework, in which the prefix of each query is computed only once (as shown in line 4 of Algorithm 5). It will terminate Zig+ after processing the tokens in the prefix. 3) *Using a prefix optimistically*: we can also employ the elastic prefix given in ZigZag to make use of the filter-and-verify framework with a shorter prefix. In this case, as shown in line 37 of Algorithm 6, it can shorten the prefix after each step. The algorithm will use the latest prefix in the following steps by line 4. The pessimistic approach of using a prefix is no worse than the optimistic approach in terms of the verification cost, but may require a higher cost to scan the inverted lists.

6 EXPERIMENTS

In this section we present the experimental results of the proposed techniques. We used three real datasets. The first one, denoted by Pubmed, included 26 million titles of Pubmed publication records. The second one, denoted by MT, was extracted from the social media dataset Memetracker [28], which included 64 million quotes and phases appearing in 2009. The last one, denoted by Twitter, which comes from the text field in a large collection of tweets. In each dataset, we removed the stop words, and treated a record as a bag of case-insensitive keywords.

Table 3: Datasets.

Dataset	Record#	Field	h(tf)	v(tf)	AvgT#	MaxT#	Token#
Pubmed	26m	Title	0.024	0.004	7.8	132	3.2m
MT	64m	Phrase	0.089	0.033	9.8	40671	3.1m
Twitter	50m	Text	0.048	0.006	7.25	132	6.6m

In the table, "AvgT#" means the average token number in a record, and "MaxT#" means the maximal token number in a record. The table includes two commonly used statistical measures of the degree distributions of the datasets. The first one, denoted by horizontal average variance h(tf), is defined as

$$h(\text{tf}) = \frac{1}{|\mathbb{S}|} \sum_{R \in \mathbb{S}} \sqrt{\frac{1}{|R|} \sum_{t \in R} (\text{tf}(t,R) - \text{avg}(\text{tf}(\cdot,R)))^2}, \quad (19)$$

where $\text{avg}(\text{tf}(\cdot,R)) = \frac{1}{|R|} \sum_{t \in R} \text{tf}(t,R)$. It averages all records by the standard deviations of their token degrees. Another one is

$$v(\text{tf}) = \frac{1}{|\mathbb{T}|} \sum_{t \in \mathbb{T}} \sqrt{\frac{1}{|I[t]|} \sum_{R \in I[t]} (\text{tf}(t,R) - \text{avg}(\text{tf}(t,\cdot)))^2}, \quad (20)$$

where $\text{avg}(\text{tf}(t,\cdot)) = \frac{1}{|I[t]|}\sum_{R\in I[t]}\text{tf}(t,R)$, named vertical average variance, averages all the tokens by the standard deviations of the degrees along with their inverted lists $I[t] = \{R \in \mathbb{S} | t \in R\}$. We use them to measure the effects of the elastic scheme w.r.t. both of the variances based on the intuition that larger variance may produce more remarkably change of the degrees.

We conducted the experiments on three storage platforms (given in Table 4), respectively using memory, hard disk (HDD), and flash drive (SSD). To measure the efficiency of the proposed methods in IO-intensive environments, we use very small physical memories in the two disk-based platform. The CPU on the SSD platform was slower than others, and was relatively slower in the scan-intensive workloads. We implemented two indexes based on page-oriented B-Tree to manage records and inverted lists, where the token weights and record identifiers were maintained in a certain number of intermediate nodes. In the following table, "II" (and "RI") stands for the Inverted Index (and the Record Index), and "*BS*" denotes the *B*lock *S*ize in each index.

Table 4: Three storage platforms in the experiments.

Storage	CPU	Used/Total Mem	II *BS*	RI *BS*
Memory	E5-2640 2.6GHz	250GB/320GB	4KB	4KB
HDD	E5300 2.6GHz	1GB/2GB	4KB	4KB
SSD	T6600 2.2GHz	1GB/2GB	4KB	4KB

The experiments were conducted on a Ubuntu14.04LTS-4.4SMP linux server. The techniques were implemented in Java using JDK 1.8. In each experiment, we randomly selected 100 records from a dataset as a query record, and ran each query six times and measured the average time. Before each round, we cleared the system caches. For comparison purposes, we adopted the Flamingo package [8] and made some changes to support similarity search based on tf-idf, where tf is the token frequency in a record, and idf(t) is computed by $\log_2(1 + |\mathbb{S}|/|\{R \in \mathbb{S} : t \in R\}|)$.

6.1 Efficiency on Different Platforms

We give the results of five algorithms on three storage platforms as shown in Figure 6. The algorithms performed differently on the data sets due to their different distributions. In general, the running time was the smallest on Pubmed and the largest on Twitter, mainly because of their data sizes. The ZigZag algorithm performed the best among the algorithms on Twitter on all the three storage platforms. On average it was 4 times faster than the worst algorithm Zig in all the datasets using $\tau \geq 0.8$. Significantly, by introducing elastic bounds, Zig+, Zag+, and ZigZag reduced at least half of the running time of Zig and Zag on the MT data set.

Figure 6 shows that the algorithms ran much faster in memory than HDD and SSD. An interesting result is that the running time Zig and Zig+ on SSD was not significantly better than that on HDD. As mentioned above, the CPU in the SSD platform was slower than that of the HDD platform, which resulted in longer running time in the scan-intensive

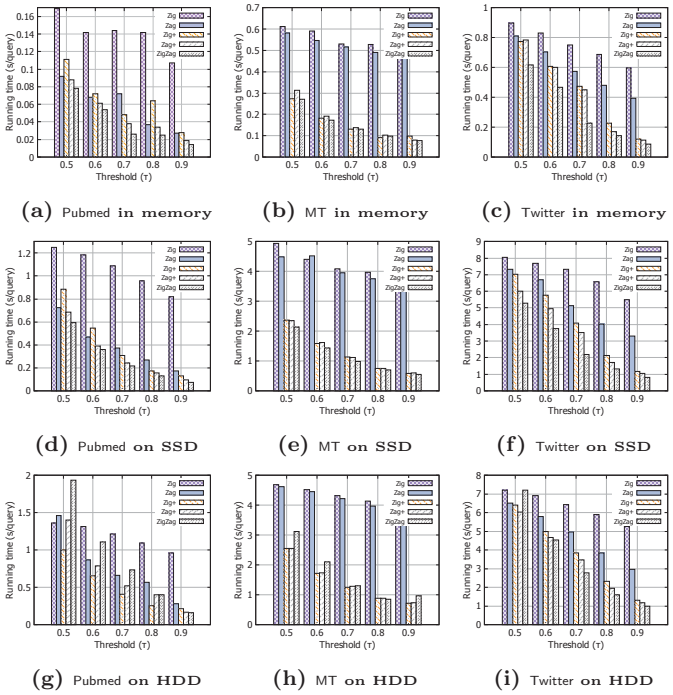


Figure 6: Running time of the proposed algorithms with different thresholds on three storage platforms.

workloads in Zig and Zig+. In contrast, due to the random IO's, the other algorithms ran much faster when SSD became more efficient than HDD. As for ZigZag, as mentioned in Section 5.2, the shorter prefix generally produced more candidates due to the fact that many unseen tokens in the remaining set made it difficult to decide a similar record. Thus, its running time on HDD was significantly longer than that of SSD. It is clear that the algorithms that utilized the elastic bounds, i.e., Zig+, Zag+, and ZigZag, performed 2 times faster than other algorithm on all the storage platforms.

6.2 Effect of Bounds

We conducted experiments to evaluate the effect of different bounds in different algorithm in terms of the number of IOs per query. The results are shown in Figure 7. The results from the datasets showed different IO behaviors of the algorithms, which depended on the effect of the two elastic bounds.

Length Bounds. The results in Figure 6 showed the performance improvement of Zig+ compared to Zig (by 62%). Also, Zag+ was always better than Zag (an average improvement of 59%). These benefits came from the elastic length bounds. However, this improvement was not much on the Twitter data set. As shown in Equation 20, we have a looser distribution of degrees for a token with a larger v(tf) value. In other words, an inverted list with a larger v(tf) value has record degrees less uniformly distributed. In all the three datasets, the degrees were roughly proportional to the record lengths on an inverted list, and more scattered degrees can

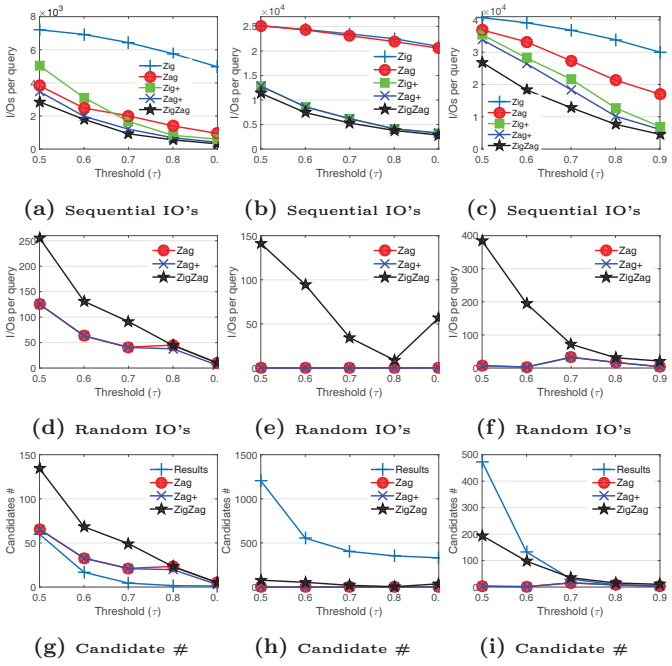


Figure 7: Sequential IO number, random IO number, and candidate number of two algorithms, with the sub-figures in each column respectively run on Pubmed, MT, Twitter.

better benefit the proposed elastic length bounds given in Theorem 9. Also notice the effect of the average length of the inverted lists, and a longer inverted list can better support the elastic strategy. The Twitter data set had a large token vocabulary, and the average size of its inverted lists was relatively small. The small improvement of elastic bounds on this data set was due to its moderate v(tf) and large vocabulary, as shown in Table 3.

Query Prefix. Figures 7a∼7c show that ZigZag always outperformed other algorithms in terms of sequential IOs. This result was not surprising since the algorithm scanned the smallest number of inverted lists due to its shortest prefix. In addition, compared to Zig+ and Zag+, this algorithm had the same scanning costs on the same tokens. In these figures, Zigzig was substantially better than Zig+ in terms of sequential IOs. Based on Equation 19, a record with a larger h(tf) tends to have a larger variance of the token degrees, which will result in some tokens associated with larger degrees. In many real datasets, a token that appears multiple times in some records tends to have a higher global frequency, and be associated with a smaller weight. In our experiments, records had their tokens sorted by a weight-decreasing order. Thus, the count of the prefix given in Definitions 5 and 6 more likely decreased if their remaining tokens had larger degrees. In the Pubmed data set, its h(tf) was much smaller than others. Its records had larger degrees in their remaining tokens. Thus, the prefix-based method had a shorter prefix on this data set. As analyzed above, it could be hard to remove the candidates with many unseen tokens. In Figures 7g and 7d, in the prefix-based methods, both the candidates and random IOs on Pubmed were on average 20 times larger than that of the other datasets.

Recall that Zag has a candidate-pruning phase in its lines 20∼26. Therefore, as shown in Figures 7h and 7i, the size of candidates in Zag was significantly smaller than the count of the results. This improvement validates the pruning power of this additional phase. We also witness similar improvements in Zag+ and ZigZag in the figures. It is more difficult to accept a record in ZigZag as an answer given a very large threshold, e.g., 0.9, due to the following reasons. (1) In general, $\delta_{R,Q}$ with a larger τ may be larger; and (2) The prefix is shorter for a larger threshold, and the accumulated factor products of the prefix of a record and Q is generally smaller. Since ZigZag shortens the prefix greedily, it may generate the shortest prefix.

6.3 Scalability

We evaluated the scalability of two representative methods, namely, Zig+ and ZigZag, by increasing the number of records for the MT data set. As shown above, these two algorithms respectively had the smallest number of random IO's (which is zero in Zig+) and the smallest number of sequential IO's (in ZigZag). We sampled 100 records from the data set, and used them to generate queries. We also increased the number of records from 1 million to 64 millions. Figure 8 shows the running time on the two storage platforms. In addition, Figure 9 shows the selection costs in terms of the number of sequential IO's and random IO's. As shown in Figure 8, the running time was proximately consistent with Figures 6h and 6e. The running time increased linearly as we increased the number of records. The cross point in Figure 7h also appears in Figure 9c.

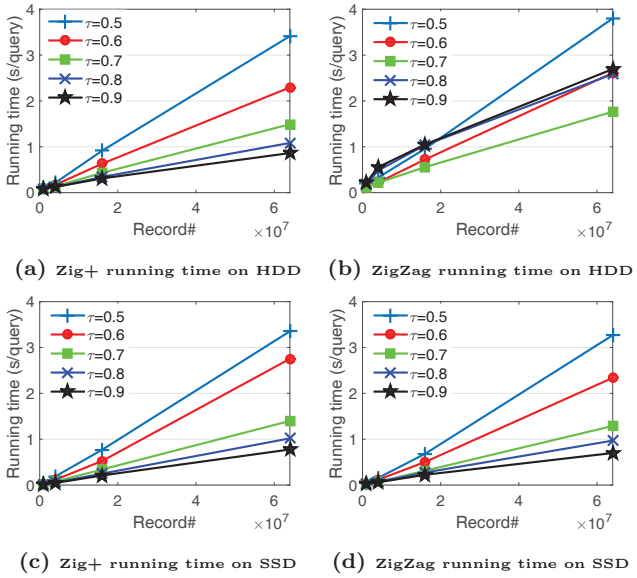
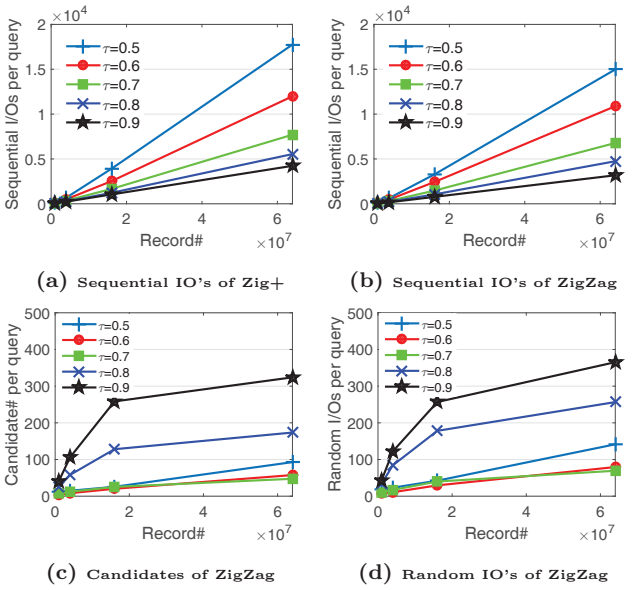


Figure 8: Scalability comparison using external storage.



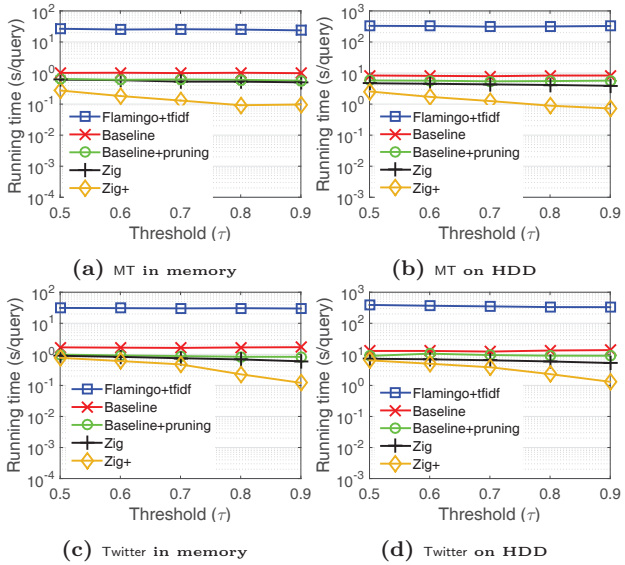
(a) Sequential IO's of Zig+ (b) Sequential IO's of ZigZag

(c) Candidates of ZigZag (d) Random IO's of ZigZag

Figure 9: IO's comparison of two algorithms on MT.

6.4 Comparison with Existing Methods

We compared the proposed algorithms with three existing methods in the literature. The first naive method comes from the Flamingo package [8], which supports similarity search queries on large datasets using external storages. We performed the tf-idf 2-norm selection queries by verifying the list-union results using Flamingo, where the similarity function was computed per record. We denote this method as "Flamingo+tfidf". We also implemented Algorithm 1 using



(a) MT in memory (b) MT on HDD

(c) Twitter in memory (d) Twitter on HDD

Figure 10: Comparison of flamingo, baseline and Zig+.

the proposed index, in which lines 7∼10 were respectively disabled (denoted by Baseline) or enabled (denoted by Baseline+pruning) to measure their effectiveness. We allocated a small amount of memory (less than 100MB) for the weights and pointers based on the grouping strategy shown in Section 5.1.2. The associated record weights and degrees with a size linear to the cardinality of record entries introduced 1X∼2X disk overhead compared to Flamingo. These weights could help reduce the candidate number, which may remarkably reduce the memory usage in large-scale datasets.

Figure 10 shows the running time of the baseline implementations, Zig, and Zig+. It shows that, due to high costs of verification and the on-demand computation, Flamingo+tfidf was slower than Baseline by an order of magnitude. In the figures, the three baseline algorithms, namely Flamingo+tfidf, Baseline, and Baseline+pruning, required the same running time for all the thresholds. The running time of Zig slightly decreased when we increased the threshold, and it was at most 2X faster than Baseline+pruning (in Figure 10d). In the figures, Zig+ was always better than Zig, which is consistent with what we observed in Figure 6. It outperformed Zig by at least 2X on MT, and was much faster with larger thresholds (e.g. $\tau \geq 0.6$.)

Summary: The experimental results showed that the proposed algorithms can efficiently support similarity queries using p-norm functions. The length and prefix bounds provided pruning power to remove dissimilar candidates, and the elastic scheme significantly reduced the running time on large data sets. ZigZag outperformed other algorithms on the SSD platform by reducing the scan cost. This algorithm did not have this advantage for the case of hard disks due to its larger number of random IO's and the corresponding higher cost of candidate verification.

7 CONCLUSIONS

In this paper we studied the problem of supporting similarity queries on a large number of records using a vector space model, where each record is a bag of tokens. We considered similarity functions that incorporate global token weights as well as record-specific token degrees. We developed a family of algorithms based on an inverted index, and presented various pruning techniques to improve their performance. We conducted an extensive experimental study using real, large data sets on various storage platforms, including memory, herd drive, and SSD. The results showed that these algorithms can support similarity queries efficiently on large data sets.

ACKNOWLEDGMENT

This research was sponsored by the National Science Foundation of China, grant 61572373, 61472290 and 60903035, the National High Technology Research and Development Program of China, grant 2017YFC08038. Chen Li has been partially supported by NSF award 1305430. Besides the reviewers, we would also thank High Performance Computing Center of Wuhan University.

REFERENCES

[1] F. N. Afrati, A. D. Sarma, D. Menestrina, A. Parameswaran, and J. D. Ullman. Fuzzy joins using mapreduce. In *IEEE ICDE*, pages 498–509. IEEE, 2012.

[2] Y. Amsterdamer, S. B. Davidson, T. Milo, and et al. Oassis: Query driven crowd mining. In *ACM SIGMOD*, pages 589–600, June 2014.

[3] A. Arasu, V. Ganti, and R. Kaushik. Efficient exact set-similarity joins. In *VLDB*, pages 918–929, 2006.

[4] E. Baralis, L. Cagliero, A. Fiori, and P. Garza. Mwi-sum: A multilingual summarizer based on frequent weighted itemsets. *ACM TOIS*, 34(1):5–35, 2015.

[5] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *WWW*, pages 131–140, 2007.

[6] A. Behm, S. Ji, C. Li, and J. Lu. Space-constrained gram-based indexing for efficient approximate string search. In *IEEE ICDE*, 2009.

[7] A. Behm, C. Li, and M. J. Carey. Answering approximate string queries on large data sets using external memory. In *IEEE ICDE*, pages 888–899, 2011.

[8] A. Behm, R. Vernica, S. Ji, J. Lu, L. Jin, Y. Lu, and C. Li. UCI Flamingo Package 3.0, 2010.

[9] P. Bouros, S. Ge, and N. Mamoulis. Spatio-textual similarity joins. *PVLDB*, 6(1):1–12, 2012.

[10] A. Chandel, O. Hassanzadeh, N. Koudas, M. Sadoghi, and D. Srivastava. Benchmarking declarative approximate selection predicates. In *ACM SIGMOD*, pages 353–364, 2007.

[11] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An e cient access method for similarity search in metric spaces. In *VLDB*, pages 426–435. Citeseer, 1997.

[12] D. Deng, G. Li, and J. Feng. A pivotal prefix based filtering algorithm for string similarity search. In *ACM SIGMOD*, pages 673–684. ACM, 2014.

[13] D. Deng, G. Li, S. Hao, J. Wang, and J. Feng. Massjoin: A mapreduce-based method for scalable string similarity joins. In *IEEE ICDE*, pages 340–351. IEEE, 2014.

[14] D. Deng, G. Li, H. Wen, and J. Feng. An efficient partition based method for exact set similarity joins. *PVLDB*, 9(4):360–371, 2015.

[15] C. Doulkeridis and K. Nørvåg. A survey of large-scale analytical query processing in mapreduce. *The VLDB Journal*, 23(3):355–380, 2014.

[16] J. Feng, J. Wang, and G. Li. Trie-join: a trie-based method for efficient string similarity joins. *The VLDB Journal*, 21(4):437–461, 2012.

[17] S. Gokavarapu, N. Tandon, and V. Varma. A weighted tag similarity measure based on a collaborative weight model. In *Proceedings of the 2nd international workshop on Search and mining user-generated contents*, pages 75–86. ACM, October 2010.

[18] 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.

[19] M. Hadjieleftheriou, A. Chandel, N. Koudas, and D. Srivastava. Fast indexes and algorithms for set similarity selection queries. In *IEEE ICDE*, pages 267–276, 2008.

[20] M. Hadjieleftheriou, N. Koudas, and D. Srivastava. Incremental maintenance of length normalized indexes for approximate string matching. In *ACM SIGMOD*, pages 429–440, June 2009.

[21] M. Hadjieleftheriou and D. Srivastava. Weighted set-based string similarity. *IEEE Data Eng. Bull.*, 33(1):25–36, 2010.

[22] J. Jia, C. Li, X. Zhang, C. Li, M. J. Carey, and S. su. Towards interactive analytics and visualization on one billion tweets. In *ACM SIGSPATIAL*, pages 85:1–85:4. ACM, 2016.

[23] Y. Jiang, G. Li, J. Feng, and W.-S. Li. String similarity joins: An experimental evaluation. *PVLDB*, 7(8):625–636, 2014.

[24] K.-T. Kim, L.-E. Rioux, , and S. L. Turgeon. Molecular weight and sulfate content modulate the inhibition of α-amylase by fucoidan relevant for type 2 diabetes management. *PharmaNutrition*, 3(3):108–114, 2015.

[25] Y. Kim and K. Shim. Parallel top-k similarity join algorithms using mapreduce. In *IEEE ICDE*, pages 510–521. IEEE, 2012.

[26] R. Konow, G. Navarro, C. L. A. Clarke, and A. López-Ortíz. Inverted treaps. *ACM TOIS*, 35(3):22:1–22:45, 2017.

[27] B. Koopman and G. Zuccon. Understanding negation and family history to improve clinical information retrieval. In *ACM SIGIR*, pages 971–974, July 2014.

[28] J. Leskovec, L. Backstrom, and J. Kleinberg. Meme-tracking and the dynamics of the news cycle. In *ACM SIGKDD*, pages 497–506, June 2009.

[29] C. Li, J. Lu, and Y. Lu. Efficient merging and filtering algorithms for approximate string searches. In *IEEE ICDE*, pages 257–266, 2008.

[30] C. Li, B. Wang, and X. Yang. VGRAM: improving performance of approximate queries on string collections using variable-length grams. In *VLDB*, pages 303–314, 2007.

[31] W. Mann and N. Augsten. Pel: Position-enhanced length filter for set similarity joins. In *Grundlagen von Datenbanken*, pages 89–94, 2014.

[32] W. Mann, N. Augsten, and P. Bouros. An empirical evaluation of set similarity join techniques. *PVLDB*, 9(9):636–647, 2016.

[33] A. Metwally and C. Faloutsos. V-smart-join: A scalable mapreduce framework for all-pair similarity joins of multisets and vectors. *PVLDB*, 5(8):704–715, 2012.

[34] G. Navarro and R. Baeza-yates. A practical q-gram index for text retrieval allowing errors. *CLEI Electronic Journal*, 1:http://www.clei.cl., 1998.

[35] D. Odijk, E. Meij, I. Sijaranamual, and M. de Rijke. Dynamic query modeling for related content finding. In *ACM SIGIR*, pages 33–42, Agust 2015.

[36] J. H. Paik. A novel tf-idf weighting scheme for effective ranking. In *ACM SIGIR*, pages 343–352, July 2013.

[37] J. H. Paik. A probabilistic model for information retrieval based on maximum value distribution. In *ACM SIGIR*, pages 585–594, Agust 2015.

[38] J. Qin, W. Wang, Y. Lu, C. Xiao, and X. Lin. Efficient exact edit similarity query processing with the asymmetric signature scheme. In *ACM SIGMOD*, pages 1033–1044, 2011.

[39] L. A. Ribeiro and T. Härder. Generalizing prefix filtering to improve set similarity joins. *Information Systems*, 36(1):62–78, 2011.

[40] S. Sarawagi and A. Kirpal. Efficient set joins on similarity predicates. In *ACM SIGMOD*, pages 743–754, 2004.

[41] W. Shen, J. Han, and J. Wang. A probabilistic model for linking named entities in web text with heterogeneous information networks. In *ACM SIGMOD*, pages 1199–1210, June 2014.

[42] Y. N. Silva and J. M. Reed. Exploiting mapreduce-based similarity joins. In *ACM SIGMOD*, pages 693–696. ACM, 2012.

[43] R. Vernica, M. Carey, and C. Li. Efficient parallel set-similarity joins using MapReduce. In *ACM SIGMOD*, 2010.

[44] J. Wang, G. Li, and J. Feng. Can we beat the prefix filtering?: an adaptive framework for similarity join and search. In *ACM SIGMOD*, pages 85–96. ACM, 2012.

[45] J. Wang, G. Li, and J. Feng. Extending string similarity join to tolerant fuzzy token matching. *ACM Transaction on Database Systems*, 39(1):7–45, 2014.

[46] W. Wang, J. Qin, C. Xiao, X. Lin, and H. T. Shen. Vchunkjoin: An efficient algorithm for edit similarity joins. *IEEE TKDE*, 25(8):1916–1929, 2013.

[47] X. Wang, L. Qin, X. Lin, Y. Zhang, and L. Chang. Leveraging set relations in exact set similarity join. *PVLDB*, 10(9):925–936, 2017.

[48] Y. Wang, A. Metwally, and S. Parthasarathy. Scalable all-pairs similarity search in metric spaces. In *ACM SIGKDD*, pages 829–837. ACM, 2013.

[49] C. Xiao, W. Wang, and X. Lin. Ed-join: An efficient algorithm for similarity joins with edit distance constraints. In *VLDB*, 2008.

[50] C. Xiao, W. Wang, X. Lin, and J. X. Yu. Efficient similarity joins for near duplicate detection. In *WWW*, pages 131–140, 2008.

[51] X. Yang, B. Wang, and C. Li. Cost-based variable-length-gram selection for string collections to support approximate queries efficiently. In *ACM SIGMOD*, 2008.

[52] M. Yu, G. Li, D. Deng, and J. Feng. String similarity search and join: a survey. *FOCS*, 10(3):399–417, 2016.

A RELATED PROOFS

A.1 Proof of Lemma 1

Proof. In the bag-based similarity on 2-norm, the condition given in Inequality 2 can be directly applied to Equation 5 and we obtain

$$W_2(R,Q) \geq \tau \Leftrightarrow \frac{\sum_{t \in R \cap Q} c(t,R)c(t,Q)}{\mathsf{len}(R)\mathsf{len}(Q)} \geq \tau.$$

A.2 Proof of Lemma 2

Proof. Suppose $\mathbb{T}$ is the vocabulary that has been sorted by a total order $\mathbb{O}$ on all the tokens in a domain $\mathbb{S}$. If a token t_k is shared by R and S respectively at the positions i and j in their ordered sets $\mathbf{R}$ and $\mathbf{Q}$, the remaining tokens $\mathsf{S}(t_k, \mathbf{R})$ of $\mathbf{R}$ after position i (or $\mathsf{S}(t_k, \mathbf{Q})$ of $\mathbf{Q}$ after its position j) can be characterized by a vector of factors $\langle c(t_{k+1}, R) \dots c(t_{|\mathbb{T}|}, R)\rangle$ with $c(t, R) = 0$ if $t \notin R$ (for Q respectively). Thus, the summation of the factor products should have an upper bound. Based on the Cauchy-Schwarz inequality, for any two vectors $\langle u_1 \dots u_{|\mathbb{T}|}\rangle$ and $\langle v_1 \dots v_{|\mathbb{T}|}\rangle$, we have

$$\sum\nolimits_{i=1}^{|\mathbb{T}|} u_i v_i \le \sqrt{\sum\nolimits_{i=1}^{|\mathbb{T}|} u_i^2}\sqrt{\sum\nolimits_{i=1}^{|\mathbb{T}|} v_i^2}.$$

Naturally one can decide an upper bound using two vectors $\langle c(t_{k+1}, R) \dots c(t_{|\mathbb{T}|}, R)\rangle$ and $\langle c(t_{k+1}, Q) \dots c(t_{|\mathbb{T}|}, Q)\rangle$, i.e.,

$$\sum_{l=k+1}^{|\mathbb{T}|} c(t_l, R)c(t_l, Q) \le \sqrt{\sum_{l=k+1}^{|\mathbb{T}|} c^2(t_l, R)}\sqrt{\sum_{l=k+1}^{|\mathbb{T}|} c^2(t_l, Q)}.$$

As $\mathsf{len}(\mathsf{S}(t_k, \mathbf{R})) = \mathsf{len}(\mathbf{R}(i:]) = \sqrt{\sum_{l=k+1}^{|\mathbb{T}|} c^2(t_l, R)}$, we have

$$\sum\nolimits_{l=k+1}^{|\mathbb{T}|} c(t_l, R)c(t_l, Q) \le \mathsf{len}(\mathbf{R}(i:])\mathsf{len}(\mathbf{Q}(j:]). \quad (21)$$

Also, the summation of the factor products of all their common tokens before (including) t_k can be formulated as

$$\sum_{l=1}^{k} c(t_l, R)c(t_l, Q).$$

Based on Inequality 6, the summation of the factor products of all the tokens in R and Q should be no less than $\delta_{R,Q}$, i.e.,

$$\sum_{l=1}^{k} c(t_l, R)c(t_l, Q) + \sum_{l=k+1}^{|\mathbb{T}|} c(t_l, R)c(t_l, Q) \ge \delta_{R,Q}. \quad (22)$$

From Inequalities 21 and 22, we can prove Lemma 2.

A.3 Proof of Lemma 4

Proof. We next prove Inequalities 9a and 9b, respectively.

1) We first prove the lower bound given in Inequality 9a.

Suppose R is similar to Q on a given threshold $\tau > 0$, i.e., $W_2(R, Q) \ge \tau$. Then we have

$$\sum\nolimits_{t\in R\cap Q} \left(f(t, R)w(t)\right)\left(f(t, Q)w(t)\right) \ge \tau\mathsf{len}(R)\mathsf{len}(Q),$$

or equivalently $\sum_{t\in R\cap Q} c(t, R)c(t, Q) \ge \tau\mathsf{len}(R)\mathsf{len}(Q)$. Recall $c(t, Q) = (c(t, R)f(t, Q))/f(t, R)$, so we have

$$\sum_{t\in R\cap Q} c(t, R)c(t, Q) = \sum_{t\in R\cap Q}\left(c(t, R)\frac{c(t, R)f(t, Q)}{f(t, R)}\right).$$

We relax $f(t, R) \le \min_{S\in\mathbb{S}}(f(t, S))$ to obtain

$$\sum_{t\in R\cap Q}\left(c(t, R)\frac{c(t, R)f(t, Q)}{\min\limits_{S\in\mathbb{S}}(f(t, S))}\right) \ge \sum_{t\in R\cap Q}\left(c(t, R)\frac{c(t, R)f(t, Q)}{f(t, R)}\right).$$

Based on Inequality 6, we also know that

$$\sum_{t\in R\cap Q}\left(c(t, R)\frac{c(t, R)f(t, Q)}{f(t, R)}\right) \ge \tau\mathsf{len}(R)\mathsf{len}(Q).$$

Thus, the following inequality is true.

$$\sum_{t\in R\cap Q}\left(c(t, R)\frac{c(t, R)f(t, Q)}{\min_{S\in\mathbb{S}}(f(t, S))}\right) \ge \tau\mathsf{len}(R)\mathsf{len}(Q).$$

Relaxing its left-hand side by all the tokens in $R \cap Q$, we get

$$\max_{t\in R\cap Q}\left(\frac{f(t, Q)}{\min\limits_{S\in\mathbb{S}} f(t, S)}\right)\sum_{t\in R} c^2(t, R) \ge \sum_{t\in R\cap Q}\frac{c^2(t, R)f(t, Q)}{\min\limits_{S\in\mathbb{S}} f(t, S)}.$$

We get the following inequality due to $\sum\limits_{t\in R} c^2(t, R) = \mathsf{len}^2(R)$.

$$\max_{t\in R\cap Q}\left(\frac{f(t, Q)}{\min\limits_{S\in\mathbb{S}} f(t, S)}\right)\mathsf{len}^2(R) \ge \tau\mathsf{len}(R)\mathsf{len}(Q).$$

The lower bound can be derived by relaxing $R \cap Q$ to Q as

$$\max_{t\in Q}\left(\frac{f(t, Q)}{\min\limits_{S\in\mathbb{S}} f(t, S)}\right)\mathsf{len}^2(R) \ge \max_{t\in R\cap Q}\left(\frac{f(t, Q)}{\min\limits_{S\in\mathbb{S}} f(t, S)}\right)\mathsf{len}^2(R).$$

Combining the above two inequalities, we have

$$\max_{t\in Q}\left(\frac{f(t, Q)}{\min\limits_{S\in\mathbb{S}} f(t, S)}\right)\mathsf{len}^2(R) \ge \tau\mathsf{len}(R)\mathsf{len}(Q).$$

Thus Inequality 9a is proved.

2) We prove the upper bound $U_{\tau,Q}(R)$ of Inequality 9b.

Let us relax the left-hand side of Inequality 6 by Q, then

$$\sum\nolimits_{t\in Q} c(t, R)c(t, Q) \ge \tau\mathsf{len}(R)\mathsf{len}(Q).$$

Suppose each $t \in Q$ has a maximal degree in all the records in $\mathbb{S}$, i.e., $\max_{S\in\mathbb{S}} f(t, S)$. Then

$$\sum_{t\in Q}\left(\frac{\max\limits_{S\in\mathbb{S}} f(t, S)}{f(t, Q)}c(t, Q)c(t, Q)\right) \ge \sum_{t\in Q}\left(\frac{f(t, R)}{f(t, Q)}c(t, Q)c(t, Q)\right)$$

holds since $\max_{S\in\mathbb{S}} f(t, S) \ge f(t, R)$. Its right-hand side can be translated as follows because of $w(t) = c(t, R)/f(Q) = c(t, Q)/f(Q)$:

$$\sum\nolimits_{t\in Q}\left(\frac{f(t, R)}{f(t, Q)}c(t, Q)c(t, Q)\right) \ge \sum\nolimits_{t\in Q}\left(c(t, R)c(t, Q)\right).$$

Combining both inequalities, we can derive the inequality

$$\sum\nolimits_{t\in Q}\left(\frac{\max_{S\in\mathbb{S}} f(t, S)}{f(t, Q)}c(t, Q)c(t, Q)\right) \ge \sum\nolimits_{t\in Q}\left(c(t, R)c(t, Q)\right).$$

By further relaxing its left-hand side by all the tokens in $t \in Q$, we obtain the following inequality:

$$\max_{t\in Q}\left(\frac{\max_{S\in\mathbb{S}} f(t, S)}{f(t, Q)}\right)\sum_{t\in Q} c(t, Q)c(t, Q) \ge \sum_{t\in Q} c(t, R)c(t, Q).$$

Also, from Inequality 6, we have

$$\sum_{t\in Q} c(t, R)c(t, Q) \ge \sum_{t\in R\cap Q} c(t, R)c(t, Q) \ge \tau\mathsf{len}(R)\mathsf{len}(Q).$$

Since $\sum\limits_{t\in Q} c(t, Q)c(t, Q) = \mathsf{len}^2(Q)$, we have

$$\max_{t\in Q}\left(\frac{\max_{S\in\mathbb{S}} f(t, S)}{f(t, Q)}\right)\mathsf{len}^2(Q) \ge \tau\mathsf{len}(R)\mathsf{len}(Q).$$

Inequality 9b is proved. Thus, we have proved Lemma 4.

A.4 Proof of Lemma 5

Proof. Suppose R is similar to Q with $\tau > 0$, then we have

$$\sum_{t \in R \cap Q} \left(f(t,R)w(t)\right)\left(f(t,Q)w(t)\right) \geq \tau \mathsf{len}(R)\mathsf{len}(Q).$$

We introduce $\max_{S \in \mathbb{S}} f(t,S)$ over all the records in $\mathbb{S}$, then

$$\sum_{t \in R \cap Q} w^2(t) \max_{S \in \mathbb{S}} f(t,R) f(t,Q) \geq \tau \mathsf{len}(R)\mathsf{len}(Q).$$

Recall $\sum_{t \in Q}(\cdot) \geq \sum_{t \in R \cap Q}(\cdot)$ always holds, so we have

$$\sum_{t \in Q} w^2(t) \max_{S \in \mathbb{S}} f(t,R) f(t,Q) \geq \tau \mathsf{len}(R)\mathsf{len}(Q).$$

Thus, Inequality 10 is proved.

A.5 Proof of Lemma 6

Proof. We should guarantee it will not result in false negatives if we discard the one that has no common token shared with the prefix of a query record. In the vector space model, the motivation of the absolute prefix is to decide a minimal length on the assumption that all the tokens from both remaining sets (of any record $R \in \mathbb{S}$ and Q) contribute their factors with a maximum degree. In other words, given each position j in a query document $\mathbf{Q}$ that has its tokens sorted by $\mathbb{O}$, if we generate an upper bound for its remaining length $\mathsf{len}(\mathbf{Q}(j:])$, we can decide whether we need more contribution from the tokens before j to satisfy Inequality 6.

As each token $t \in \mathbb{T}$ has a maximal degree $\max_{R \in \mathbb{S}}(f(t,R))$ in $\mathbb{S}$, the summation of the factor products between $\mathbf{Q}(j:]$ and a record R is bounded by these maximal degrees (of the tokens in $\mathbf{Q}(j:]$) as

$$\sum_{t \in \mathbf{Q}(j:]} c(t,R)c(t,Q) \leq \sum_{t \in \mathbf{Q}(j:]} \left(w^2(t) f(t,Q) \max_{S \in \mathbb{S}} f(t,S)\right). \quad (23)$$

Notice that generally we cannot find such a record that has all of its remaining tokens with the maximal degrees in the domain. This upper bound dominates the factor products of the remaining tokens of any record and Q. Suppose there is no common token between the prefix $\mathbf{Q}[:j]$ and R, then the summation of the factor products is at most the right-hand side of Inequality 23. Thus, we can safely prune R if

$$\sum_{t \in \mathbf{Q}(j:]} \left(w^2(t) f(t,Q) \max_{S \in \mathbb{S}} f(t,S)\right) < \delta_{R,Q}.$$

Based on Inequality 9a, all the records that are similar to Q w.r.t. τ have a lower bound $L_{\tau,Q}$. If the maximal summation of factor products that can be derived from $\mathbf{Q}((j:])$ is less than $\tau L_{\tau,Q}\mathsf{len}(Q)$, it will also be less than $\delta_{\tau,Q}$ due to $\tau L_{\tau,Q}\mathsf{len}(Q) \leq \delta_{\tau,Q}$. Thus, a record can be safely pruned if it has no common token with $\mathbf{Q}([:j])$, where position j satisfies the following constraint:

$$j \left| \sum_{t \in \mathbf{Q}(j:]} \left(w^2(t) f(t,Q) \max_{S \in \mathbb{S}} f(t,S)\right) < \tau L_{\tau,Q}\mathsf{len}(Q)\right. .$$

Lemma 6 is proved.

A.6 Proof of Lemma 7

Proof. Equation 14 divides the upper bound of all records into two parts, with one part bounded by token $\mathbf{Q}[j]$, and the other including the remaining tokens after $\mathbf{Q}[j]$. From the total order definition, $\mathbf{Q}[j]$ has a unique global order $\mathbb{O}(\mathbf{Q}[j])$, such that we can always divide a record by comparing their ordered tokens with $\mathbf{Q}[j]$.

From Lemma 2, we know that except for the decided records already in the output, the similar records that include at least one token in $\mathbf{Q}[:j]$ have been included in the candidates $\mathbb{C}$. Thus, the maximal degrees of these tokens only rely on the candidates in $\mathbb{C}$, as the other records have been safely pruned. The summation of the factor products of $\mathbf{Q}[:j]$ and any undecided similar records is no larger than

$$\sum_{t \preceq \mathbf{Q}[j]} w^2(t) f(t, \max_{S \in \mathbb{C}} f(t,S)) f(t,Q). \quad (24)$$

As the maximal degrees of the tokens in $\mathbf{Q}(j:]$ only rest with the records that satisfy the bounds $\mathbb{B}$, the contribution of $\mathbf{Q}(j:]$ and any undecided similar record is no larger than

$$\sum_{t \succ \mathbf{Q}[j]} w^2(t) f(t, \max_{S \in \mathbb{B}} f(t,S)) f(t,Q). \quad (25)$$

Combining Equations 24∼25, we obtain the following maximal contribution of any undecided similar record and Q:

$$\sum_{t \in S \cap Q} w^2(t) f(t, \max_{S \in \mathbb{S}} f(t,S)) f(t,Q) \quad (26a)$$

$$\leq \sum_{t \preceq \mathbf{Q}[j]} w^2(t) f(t, \max_{S \in \mathbb{C}} f(t,S)) f(t,Q) \quad (26b)$$

$$+ \sum_{t \succ \mathbf{Q}[j]} w^2(t) f(t, \max_{S \in \mathbb{B}} f(t,S)) f(t,Q). \quad (26c)$$

Considering the similar condition presented in Inequality 6, we can directly generate the following inequality

$$W_2(S,Q) \geq \tau \Rightarrow \sum_{t \in S \cap Q} w^2(t) f(t, \max_{S \in \mathbb{S}} f(t,S)) f(t,Q) \geq \delta_{R,Q},$$

and the bounding condition on both sides can be derived by

$$W_2(S,Q) \geq \tau \Rightarrow \delta_{R,Q} = \tau \mathsf{len}(R)\mathsf{len}(Q) \quad (27a)$$

$$\leq \sum_{t \preceq \mathbf{Q}[j]} w^2(t) f(t, \max_{S \in \mathbb{C}} f(t,S)) f(t,Q) \quad (27b)$$

$$+ \sum_{t \succ \mathbf{Q}[j]} w^2(t) f(t, \max_{S \in \mathbb{B}} f(t,S)) f(t,Q). \quad (27c)$$

Thus, we have the upper bound given in Equation 14 with its two parts formulated by Equations 14a and 14b, respectively.

A.7 Proof of Theorem 9

Proof. Let us first prove Equation 17b as follows. At the beginning, we produce two bounds by Inequalities 9a and 10, which can determine a set of records if we scan the tokens in $\mathbf{Q}(1:]$. When considering token $\mathbf{Q}[2]$, there are three cases in all the records similar to Q. For each $R \in \mathbb{S} \wedge W_2(R,Q) \geq \tau$:

(1) $R \in \mathbb{C} \Rightarrow \mathsf{len}(R) \leq \max_{S \in \mathbb{C}} \mathsf{len}(S)$;
(2) $R \in O$ has been claimed as a similar record, which will never be missed even if its length is beyond the updated bounds when we scan the upcoming tokens;
(3) Otherwise, we have $\mathsf{len}(R) \leq U_r$.

As the first two cases are trivial, we focus on case 3. In Inequalities 9a and 10, we will not miss records by bounding a record length when we consider $\mathbf{Q}[1]$. In other words, all the similar records that contain token $\mathbf{Q}[1]$ will be included in $\mathbb{C}$ or O, thus the potentially similar records outside $\mathbb{C} \cup O$ do not include $\mathbf{Q}[1]$, i.e., for $j = 1$ we have

$$W_2(R,Q) \geq \tau \wedge R \notin \mathbb{C} \cup O \Rightarrow \mathbf{Q}[j] \notin R \quad (28a)$$

$$\Leftrightarrow f(\mathbf{Q}[j], R) = 0. \quad (28b)$$

Similar to Inequalities 27a∼27c in Appendix A.6, we have

$$W_2(R,Q) \geq \tau \Rightarrow \tau \mathsf{len}(R)\mathsf{len}(Q) \quad (29a)$$

$$\leq \sum_{t \preceq \mathbf{Q}[j]} w^2(t) f(t,R) f(t,Q) \quad (29b)$$

$$+ \sum_{t \succ \mathbf{Q}[j]} w^2(t) f(t, \max_{S \in \mathbb{B}} f(t,S)) f(t,Q). \quad (29c)$$

From Equation 28b we know Equation 29b is zero. Then we have $\mathsf{len}(R) \leq U_{\tau,Q,1} = U_\tau$ in case 3.

We continue this process by increasing j. In case 3 of each step, all the previous steps do not miss similar records, then all the similar records that contain at least one token in $\mathbf{Q}[:j]$ will be included in $\mathbb{C}$ or O. Any upcoming record that is similar to Q must not include any token in $\mathbf{Q}[:j]$ unless it has been included by $\mathbb{C}$ or O. Thus, Equation 28b always holds, and the expression given in Equation 29b is zero. We can claim $\mathsf{len}(R) \leq U_\tau$ is always true.

By maximizing the bounds of cases 1 and 3 in each step, we obtain Equation 17b. Similarly, we can prove Equation 17a. Thus Theorem 9 is proved.

B USABILITY OF SIMILARITY FUNCTIONS

This paper focuses on p-norm similarity given in Definition 3 with $p = 2$. To measure the relative similarity of two records, we formalize the correlation of two records by the summation of factor products based on their degrees and token weights, and generalize the tf-idf cosine similarity [10] with variant token weights and degrees.

B.1 Accuracy Comparisons

As for the precision comparison given in Section 2, we collected the data set from https://github.com/taolei87/askubuntu, which contained a preprocessed collection of questions taken from the AskUbuntu.com 2014 corpus dump. All its records had been tokenized and divided to two sets, in which each record had two fields "Title" and "Question body." Its training set had 167,765 records. In the testing set, each record had been annotated with several similar training records by domain experts. Based on the ground truth about similar pairs of these records, we did similarity search using different functions, and compared them using F1 score, as follows.

$$F1 = \frac{2 \times precision \times recall}{precision + recall}. \quad (30)$$

The following table shows the F1 cores of different functions based on "Title" and the union of both fields ("Full"). It is clear that the maximal F1 score of $2N_{\text{tfidf}}$ was comparable to $2N_{\text{idf}}$ (with a constant degree) based on the short "Title" field. When we used longer records ("Full"), the maximal F1 score of $2N_{\text{tfidf}}$ was better than that of $2N_{\text{idf}}$ by 5%. Notice that both of the weighted functions had their maximal F1 scores higher than Jaccard and Cosine.

Table 5: F1 of various functions with different thresholds.

	Title				Full			
τ	Jac	Cos	$2N_{\text{idf}}$	$2N_{\text{tfidf}}$	Jac	Cos	$2N_{\text{idf}}$	$2N_{\text{tfidf}}$
0.1	0.00	0.00	0.01	0.00	0.00	0.00	0.05	0.01
0.2	0.04	0.01	0.03	0.03	0.14	0.00	0.37	0.12
0.3	0.20	0.03	0.12	0.11	**0.40**	0.07	**0.42**	0.37
0.4	0.34	0.09	0.28	0.27	0.38	**0.39**	0.40	**0.47**
0.5	**0.38**	0.24	0.40	0.39	0.37	0.39	0.39	0.42
0.6	0.37	0.36	**0.42**	**0.42**	0.37	0.37	0.37	0.39
0.7	0.37	**0.38**	0.40	0.40	0.37	0.37	0.37	0.37
0.8	0.37	0.37	0.38	0.38	0.37	0.37	0.37	0.37
0.9	0.37	0.37	0.37	0.37	0.37	0.37	0.37	0.37

B.2 Discussions about Application Domains

As shown above, the token weights and degrees integrated in tf-idf 2-norm similarity show certain superiority in longer records. It becomes the Cosine function when we use the weight $w(t) = 1$ and use a boolean function for the degree. It is different from existing set-based functions such as Jaccard, and has different application domains compared to gram-based methods (e.g. edit distance).

In the context of massive datasets consisting of many long records, if users want to search the records that are semantically similar to a query record, they can integrate the semantical token weights and record-specific degrees into the 2-norm similarity, and model the user preference by placing different degrees in the query record. In this case, the degrees and token weights could be more valuable for both the underlying records and the user interests.

B.3 Extensions Beyond tf-idf

Although our experiments used idf-weighting functions to feed the 2-norm similarity, all our optimization methods do not depend on such special weights. For use-defined token weights, one can define a token order and integrate the weights of processed tokens into the entries on the inverted lists.

In the analysis of inverted index, we suppose that each degree $f(t,R)$ is an integer to denote the frequency of token t in a record R. In many applications, users want to use functional token degrees, e.g., the logistic token frequency [4]. Thus, the degree mappers that have been given in Section 5.1.1 need to be revised to accommodate these data types, such that the functional degrees can also be used in the proposed strategy. While it is easy to revise the mappers for these domain-specific applications, the topics regarding how to update their stepping ranges for best pruning power needs future work.