mathematics

Article

Efficient List Intersection Algorithm for Short Documents by Document Reordering

Lianyin Jia [1,2], Dongyang Li [1], Haihe Zhou [1] and Fengling Xia [3,*]

[1] Faculty of Information Engineering and Automation, Kunming University of Science and Technology, Kunming 650500, China; lianyinjia@kust.edu.cn (L.J.); 20212204204@stu.kust.edu.cn (D.L.); 18908715777@189.cn (H.Z.)
[2] Yunnan Key Lab of Computer Technology Applications, Kunming University of Science and Technology, Kunming 650500, China
[3] Faculty of Civil Aviation and Aeronautics, Kunming University of Science and Technology, Kunming 650500, China
* Correspondence: xiafengling@kust.edu.cn

Abstract: List intersection plays a pivotal role in various domains such as search engines, database systems, and social networks. Efficient indexes and query strategies can significantly enhance the efficiency of list intersection. Existing inverted index-based algorithms fail to utilize the length information of documents and require excessive list intersections, resulting in lower efficiency. To address this issue, in this paper, we propose the LDRpV (Length-based Document Reordering plus Verification) algorithm. LDRpV filters out documents that are unlikely to satisfy the intersection results by reordering documents based on their length, thereby reducing the number of candidates. Additionally, to minimize the number of list intersection operations, an intersection and verification strategy is designed, where only the first m lists are intersected, and the resulting candidate set is directly verified. This approach effectively improves the efficiency of list intersection. Experimental results on four real datasets demonstrate that LDRpV can achieve a maximum efficiency improvement of 46.69% compared to the most competitive counterparts.

Keywords: LDRpV; list intersection; inverted index; document reordering; verification

MSC: 68P20

check for updates

Citation: Jia, L.; Li, D.; Zhou, H.; Xia, F. Efficient List Intersection Algorithm for Short Documents by Document Reordering. *Mathematics* **2024**, *12*, 1328. https://doi.org/10.3390/math12091328

Academic Editors: Jou-Ming Chang, Ling-Ju Hung and Chia-Wei Lee

Received: 7 April 2024
Revised: 24 April 2024
Accepted: 25 April 2024
Published: 26 April 2024

1. Introduction

An inverted index is a key data structure that underpins most information retrieval (IR) systems. List intersection is one of the most important operations in an inverted index [1,2], playing a pivotal role in various fields such as search engines, database systems, and social networks. In database systems, join operations require finding common document identifiers (docIDs) through list intersection [3,4]. In social networks, list-intersection-based triangle counting is often used to discover closely knit communities [5,6].

In this paper, we focus on the study of multi-way list intersection (MWLI) algorithms, i.e., given a query document q and a document dataset D, searching all documents containing all words in q. Even though this issue has been studied both early and extensively [7–10], its significance keeps it a hot research topic, with many methods being proposed.

While there are numerous research works related to MWLI, most are suitable for long documents or lists and are primarily based on external storage, emphasizing reducing input/output (IO) operations through list compression. In this paper, we focus on in-memory MWLI algorithms for short documents (e.g., documents consisting of several tens of words). Short documents, such as user comments and shopping records in malls, are common in reality. Unlike long documents, short documents have shorter lengths, making them suitable for residing in memory. Thus, this paves the way for introducing efficient verification mechanisms in subsequent discussions.

Document reordering, also referred to as document ID redistribution, was proposed by Blandford et al. in 2002 [11]. The core idea lies in assigning consecutive docIDs to closely related documents, thus making neighboring docIDs in the index more concentrated, while those not related have greater gaps. This can lead to more efficient compression of inverted indexes or improved query performance. Although document reordering has been widely applied in fields like information retrieval, it is mostly used for index compression rather than enhancing query efficiency. Moreover, existing document reordering techniques primarily adopt clustering and traveling salesman strategies [12,13] for document ID redistribution, making the reordering process relatively costly.

Despite the effectiveness of current MWLI algorithms in enhancing list intersection efficiency, they have several limitations: (1) Existing algorithms typically require intersections on all lists, resulting in significant overheads, especially when there are many long lists. (2) These algorithms have to consider a large number of docIDs in the lists during intersections, even if they might skip some of them later on, leading to excessive docID comparisons. (3) Existing document reordering methods are either overly complex or not specifically designed to optimize query performance, making them ill-suited for efficient adaptation to the techniques presented in this paper.

These limitations prompted the authors to contemplate the following questions: (1) Is it necessary to intersect all lists involved in a query? (2) Can we design a straightforward and efficient document reordering method to avoid accessing certain docIDs in inverted lists?

To address these concerns, we introduce Length-based Document Reordering (LDR) and LDR-Filtering, filtering out a significant number of candidates unlikely to meet the intersection query in a straightforward manner. Furthermore, this method adopts an "intersection + verification" approach to reduce excessive list intersections, further enhancing query efficiency. Based on LDR-Filtering and verification, LDRpV, an efficient list intersection algorithm for short documents, is proposed. LDRpV is not only simple to implement but also highly efficient. Extended experimental results demonstrate that it can improve efficiency by 46.69% compared to the most competitive existing algorithms.

We make the following contributions in this paper:

(1) We propose a simple heuristic LDR and LDR-Filtering, filtering out a large number of docIDs that cannot enter the final results. To the best of the authors' knowledge, this paper is the first work that leverages document length reordering to enhance the list intersection efficiency.

(2) We introduce the "intersection + verification" approach, which only intersects a few short lists and fully harnesses the advantages of both intersection and verification.

(3) Experimental results on multiple datasets demonstrate that LDRpV significantly outperforms existing methods in terms of intersection efficiency for short documents.

The remainder of this paper is organized as follows: Related work is presented in Section 2. Our core index and list intersection algorithm are designed in Section 3. Section 4 discusses the experiments and results. In Section 5, we conclude the paper.

2. Related Work

2.1. List Intersection Algorithm

There is a long stream of research on list intersection. Broadly speaking, list intersection algorithms can be categorized into in-memory intersection algorithms and external-memory-based intersection algorithms. External-memory-based intersection algorithms store inverted lists in external memory. As a result, they prioritize reducing the number of IOs required to access the lists. For this reason, lists are typically stored in a compressed format in external memory [14]. In contrast, in-memory algorithms assume that the lists are stored in memory, and their main focus is on enhancing the efficiency of list intersections.

Additionally, list intersection algorithms can be further divided into two-way intersection algorithms and multi-way intersection algorithms. Two-way intersection algorithms focus on the intersection operations between two lists (especially long lists), with classical algorithms including divide-and-conquer, skip-lists, and the Glomb method [8]. Multi-way intersection algorithms extend the two-way intersection to $|q|$-way intersections (where $|q|$ represents the number of words in query q), often utilizing the relationships in lengths between multiple lists to improve intersection efficiency [8]. Among the MWLI algorithms, Helmer et al. [7] proposed the intersection algorithm based on inverted indexes; this algorithm obtains the final intersected results by intersecting the inverted lists corresponding to q sequentially. Culpepper et al. [8] introduced the small versus small (SVS) algorithm, which employs heuristic rules to sort the lists from shortest to longest before conducting intersections, ensuring the intermediate results of the list intersection are as small as possible. Similar to the classic document-at-a-time (DAAT) algorithm [9], Deng et al. [10] proposed LCSearch, a list intersection algorithm based on crosscuts, which conducts intersections on all lists simultaneously. Additionally, LCSearch uses the maximum gap between adjacent IDs to enhance the efficiency of the intersection.

Moreover, recent years have also seen research in parallel list intersections [1], privacy-preserving list intersections [15], and many other related domains.

2.2. Document Reordering

The primary idea behind document reordering [11] is to assign consecutive docIDs to closely related documents. This ensures that related documents in the index are more concentrated, while unrelated documents are spaced further apart. This approach can lead to more efficient compression of inverted indexes or enhance query performance.

Within the realm of document reordering methods, Ding et al. [12] proposed a bottom-up approach. It organizes similar documents into a graph and reassigns docIDs to the documents through a graph traversal algorithm similar to the traveling salesman problem (TSP). Differing from the aforementioned work, Dhulipala et al. [13] proposed a top-down iterative binary partitioning method to recursively divide documents into multiple similar subsets, assigning consecutive docIDs to each subset. Building upon the foundation of [13], Wang et al. [16] started with the intersection of two lists, employing document reordering to minimize expected runs and thereby enhancing the efficiency of list intersections. Similarly, also based on [13], Mackenzie et al. [17] further enhanced the efficiency of document reordering by reducing iterative counts, heuristic swapping costs, and discarding sort operations. Their experimental results demonstrate that this can reduce the reordering time from 95 min to 26 min without sacrificing compression precision. Furthermore, Zhao et al. [18] employed document reordering for news summarization, Yafay et al. [19] utilized document reordering based on access counts to enhance the efficiency of dynamic pruning algorithms, and Ramaswamy et al. [20] posited that aside from promoting list intersections, document reordering also boosts disjunctive queries.

As is evident from the above, document reordering has gradually become a research hotspot in recent years. However, current methods often rely on complex algorithms such as graph traversal or clustering for document reordering, necessitating intricate offline operations. In practice, many simple and effective document reordering methods can yield superior results. Silvestri et al. [21], by simply assigning identifiers to documents according to the lexicographical ordering of the URLs, achieved a 40% improvement in compression rate. Yan et al. [22] further used URL-based document reordering to optimize the compression method. Length-based reordering is researched in [23]; however, it uses length to distribute documents to different nodes and is mainly used to improve the effects of list compression.

3. Indexing and Algorithm

Most document reordering methods focus on finding the optimal document sequence to compress the index size, giving less consideration to enhancing the efficiency of intersections through document reordering. Therefore, we introduce a simple and efficient length-based document reordering, which substantially reduces the number of intersections required, thereby improving the efficiency of intersections.

3.1. Necessary Definitions and General Framework

In this section, we first present the necessary definitions and then provide the general framework of our indexing and algorithm.

3.1.1. Necessary Definitions

Consider a dataset D consisting of a series of documents d, where each document d is composed of a series of words w and each document d has a unique docID. Considering that this paper focuses on short documents, we model the document as a set and remove any duplicate elements within it. $|d|$ and $|D|$ denote the length of document d and the size of the dataset D, respectively, while $d[i]$ represents the i-th element in the document. It is assumed in this paper that the words in a document are sorted in ascending order by default.

Based on the aforementioned descriptions, the definition of MWLI is as follows:

Definition 1. *Multi-way list intersection (MWLI) query: Given a query q and an inverted index constructed from D, MWLI searches the $|q|$ lists corresponding to q for all documents $d \in D$ containing all the words in q, that is to say, $q \subseteq d$.*

3.1.2. General Framework

For ease of understanding, we provide the general framework of our index and algorithm, as shown in Figure 1.

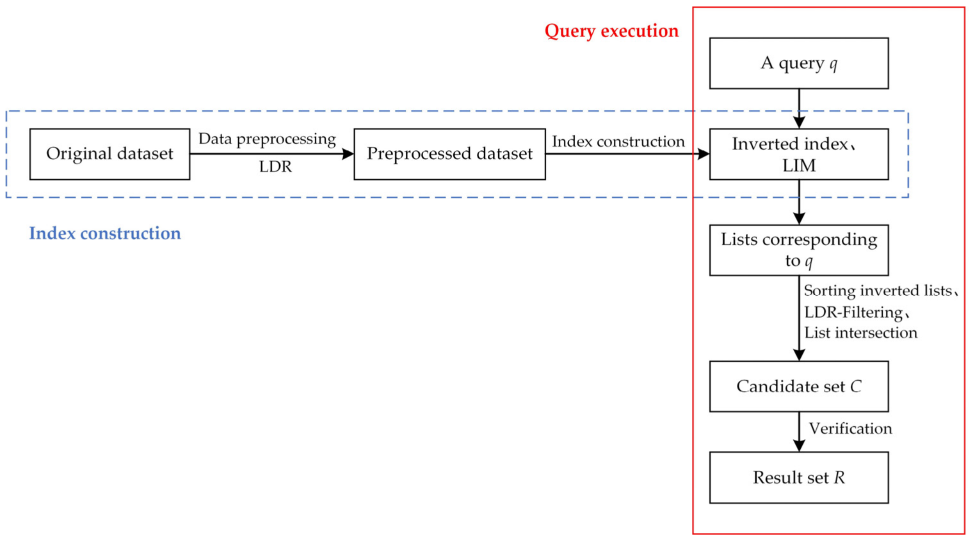


Figure 1. General framework.

As shown in Figure 1, our framework is generally divided into two parts: index construction (in blue dashed box) and query execution (in red solid box). Specifically, the index construction part includes data preprocessing and the construction of our core index, while the query execution part demonstrates the execution process of our LDRpV.

3.2. Length-Based Document Reordering (LDR)

To effectively utilize LDR in subsequent algorithms, we preprocess the original dataset D by sorting the documents in the dataset in ascending order of their lengths and reallocating docIDs to the sorted documents. For documents of the same length, each document is treated as a string and sorted in the way of string comparison. Given the example documents and their original docIDs shown in Table 1, their new docIDs after LDR are displayed in the right column of Table 1. In the description below, we use new docIDs as the default docIDs when there is no ambiguity.

Table 1. Document reordering.

Document	Original docID	New docID
$\{e_1, e_3, e_4, e_5\}$	0	5
$\{e_1, e_3\}$	1	2
$\{e_1, e_3, e_4, e_5, e_6\}$	2	6
$\{e_1, e_3, e_5, e_7\}$	3	4
$\{e_3, e_4, e_5, e_6, e_7\}$	4	8
$\{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$	5	9
$\{e_1, e_2, e_3, e_7\}$	6	3
$\{e_2, e_3, e_4, e_5, e_7\}$	7	7
$\{e_1, e_2\}$	8	1
$\{e_2\}$	9	0

3.3. Indexing

3.3.1. Inverted Index

The inverted index is widely used in fields such as information retrieval [14,24,25] and set similarity query and join [26–28]. The inverted index is composed of two parts: a directory and inverted lists. The directory consists of all distinct words from dataset D. Each word w in the directory corresponds to an inverted list composed of the docIDs of all documents that contain w. The list corresponding to w is denoted as Iw in this paper, with w also referred to as the source word (SW) of Iw. For the dataset D reordered as shown in Table 1, the constructed inverted index is illustrated in Figure 2.

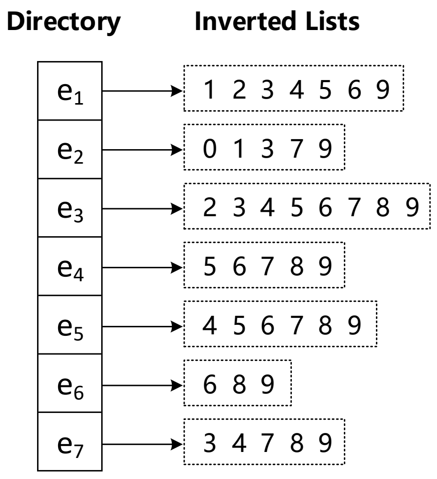


Figure 2. Inverted index.

In a typical search engine, inverted lists store not only the docIDs but also information such as the frequency and positions of term w in each document. However, since this article models documents as sets and focuses on list intersections, we only store docIDs in each list.

3.3.2. Length and docID Mapping (LIM)

A simple inverted index finds it challenging to directly utilize the information from LDR. As a result, an auxiliary length and docID mapping (LIM) table has been further designed. LIM is implemented based on an array and stores the starting docID (sID) corresponding to each length to facilitate subsequent filtering and list intersection queries. Taking the dataset reordered from Table 1 as an example, the constructed LIM is illustrated in Figure 3.

length	1	2	3	4	5	6	7
sID	0	1	3	3	6	9	9

Figure 3. LIM.

3.4. List Intersection Algorithm

3.4.1. LDR Based on Filtering (LDR-Filtering)

Based on LDR, we propose an efficient filtering, LDR-Filtering, which is the core filtering technique of this paper, and its fundamental principle is based on the following theorem.

Lemma 1. *For a given query q, any document d with a length less than $|q|$ cannot be a result of the list intersection query.*

Proof. The proof of Lemma 1 is quite intuitive. If $|d|<|q|$, then there must be words in q that do not appear in d; hence, d cannot be a result of the list intersection query. So, Lemma 1 holds. □

Given a query document $q = \{e_1, e_2, e_3, e_5, e_7\}$ and the inverted index in Figure 2, the schematic diagram of LDR-Filtering is shown in Figure 4.

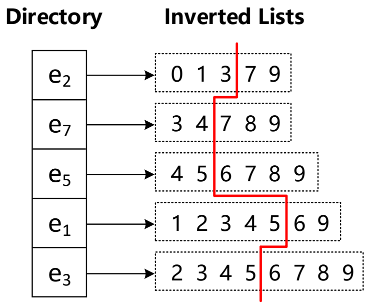


Figure 4. Schematic diagram of LDR-Filtering.

As shown in Figure 4, for a candidate document d, if $|d| < |q| = 5$, then d cannot be a result of the list intersection query. From the LIM in Figure 3, we know that the sID of length 5 is $LIM[5] = 6$. Based on Lemma 1, documents with a docID less than 6 cannot be query results. Therefore, in Figure 4, the docIDs to the left of the red dividing line can be safely filtered out, reducing the number of candidates from 10 to 4, thereby lowering the query cost. As seen from this, the idea behind LDR-Filtering is straightforward and easy to implement, significantly reducing the number of candidate documents.

3.4.2. Algorithm

An intuitive approach is to apply LDR-Filtering on all inverted lists corresponding to q to exclude docIDs that do not meet the length requirements and then perform list intersections on the filtered lists. However, this approach has several shortcomings:

(1) Applying LDR-Filtering on the inverted list requires additional overhead (e.g., binary search). Applying LDR-Filtering to all lists introduces additional costs.

(2) Considering that when the intersection result list of two lists is short, continuing to intersect the result list with the remaining lists cannot further significantly reduce the number of intersection results and incurs additional intersection costs. Therefore, performing intersections on all lists requires significant operation costs, affecting query performance.

To address these shortcomings, drawing inspiration from the verification idea in TTJoin [29] and Limit [30], we propose the LDRpV algorithm. LDRpV introduces a list intersection + verification approach based on LDR-Filtering. It consists of the following four main steps:

(1) Sorting the inverted lists:

We define I as the list collection of $|q|$ inverted lists corresponding to q and $I[i]$ as the i-th inverted list in the collection. Considering that $|I[i] \cap I[j]| \leq \min(|I[i]|, |I[j]|)$, we sort the $|q|$ inverted lists in I in ascending order by list length. This ensures that subsequent list intersections can be performed following a length heuristic optimization strategy, prioritizing intersections of shorter lists. When there is no ambiguity, we also use I to denote the sorted inverted lists.

(2) LDR-Filtering:

To avoid the shortcomings of applying LDR-Filtering to all lists, LDRpV only applies LDR-Filtering to the shortest list (SL, which is also $I[0]$ in the sorted lists I). The details of LDR-Filtering include the following: (a) obtaining the sID corresponding to $|q|$ using $LIM[|q|]$; (b) finding the first position p in SL where the docID is greater than or equal to sID; (c) obtaining the partial lists $SL' = SL[p, \ldots, |SL|-1]$ in SL, where $SL[p, \ldots, |SL|-1]$ denotes the right part of SL starting from position p.

(3) List intersection:

To address the inefficiencies of intersecting all lists, LDRpV only performs intersections on a limited number of the first m lists. Specifically, the candidate C of intersections is obtained by $C = SL' \cap I[1] \cap \ldots \cap I[m-1]$. It is important to note that SL' is the partial list obtained after applying LDR-Filtering on SL; hence, C does not contain documents with a length less than $|q|$. For simplicity, in our proposed algorithm, it is assumed that the list intersection operation is conducted using a binary search approach.

It should be noted that when $|q| <= m$, the results in $C = SL' \cap I[1] \cap \ldots \cap I[|q|-1]$ are directly the final results, obviating the need for further verification steps.

(4) Verification:

Let q' be a document composed of the remaining words in q excluding the words that have already participated in the intersection, that is, $q' = q - \{I[i].SW | i \in [0, \min(m, |q|) - 1]\}$, where $I[i].SW$ indicates the source word of $I[i]$. For each $d \in C$, we directly verify whether $d \supseteq q'$ holds. If it does, then d is a result that meets the list intersection query. During verification, a direct binary search is performed in d to find the position of $q'[0]$ in d, determining the starting position for verification. It is important to note that both d and q' are ordered in word ascending order. For any $0 \leq i < |q'|$, if $q'[i]$ does not appear in d, then d cannot be a query result of q', and the verification terminates prematurely, making the actual verification process fast and efficient.

Based on the aforementioned introduction, the final implemented LDRpV algorithm is shown in Algorithm 1.

Algorithm 1. LDRpV

```
//Input: $q$: a query document
        $m$: number of lists need to be intersected
//Output: $R$: list intersection results of $q$
1. $I \leftarrow$ sorted lists corresponding to $q[0] \sim [|q|-1]$ in list length ascending order
2. $SL \leftarrow I[0]$
3. $sID \leftarrow LIM[|q|]$
4. $p \leftarrow$ the first position with $docID \geq sID$ in $SL$
5. $SL' \leftarrow SL[p, \ldots, |SL|-1]$
6. **if** $|q| <= m$ **then**
7.         $R = SL' \cap I[1] \cap \ldots \cap I[|q|-1]$
8. **else**
9.         $C = SL' \cap I[1] \cap \ldots \cap I[|m|-1]$
10.        $q' = q - \{I[i].SW | i \in [0, \min(m, |q|) - 1]\}$
11.        **for each** $d \in C$ **do**
12.            **if** $d \supseteq q'$ **then**
13.                $R \leftarrow R \cup \{d\}$
```

For LDRpV, the main time cost comes from the expense of intersecting and verifying m lists, with the time cost of these two parts being $p * (m) * (m-1) * \bar{l} * \log_2 \bar{l} + (\left|q\right| - m) + (|\bar{d}| - m)$, where p represents the probability that the length of a document is greater than or equal to $|q|$ in the entire dataset, $\bar{l}$ represents the average length of inverted lists, and $\left|\bar{d}\right|$ represents the average length of documents.

For LDRpV, the total space overhead consists of the space occupied by the inverted index and LIM, with the space overhead of these two parts being $(d_e * \bar{l} + L) * B$, where d_e is the number of distinct elements, L is the max document length in D, and B is the number of bytes in an integer.

4. Experiment

4.1. Experimental Environment

We used an Intel (R) Core (TM) i5-6500 CPU @ 3.20 GHz with four cores and four threads. The memory was 16 GB @ 2133 MHz. The operating system was Windows 10, and IntelliJ IDEA 2022.1.1 was the IDE with JDK 18.0.1.1.

4.2. Datasets

In the experiments, we used the following four real datasets:

- ACCIDENTS (http://fimi.uantwerpen.be/data/, accessed on 11 October 2023): This dataset is the Flanders traffic accident dataset collected from 1991 to 2000 and contains 340,184 documents, each composed of different circumstances in which the accident occurred.
- DBLP (https://dblp.uni-trier.de/xml/, accessed on 11 October 2023): This dataset is a snapshot extracted from the DBLP bibliography. The dataset has 781,514 documents, and each document is a title of a certain publication.
- FLICKR (https://www.kaggle.com/datasets/hsankesara/flickr-image-dataset/, accessed on 11 October 2023): This dataset is a sentence-based image description dataset. There are 158,915 documents, and each document represents a comment on a certain image.
- T40 (http://fimi.uantwerpen.be/data/, accessed on 11 October 2023): This dataset was generated using the generator from the IBM Almaden Quest research group. The dataset has 100,000 documents.

The specific information for the four datasets is shown in Table 2.

Table 2. Dataset information.

Dataset	ACCIDENTS	DBLP	FLICKR	T40
#sets	340,183	781,514	158,915	100,000
MinLen	18	3	1	4
MaxLen	51	219	35	77
AvgLen	33.8	14.1	7.0	39.7
#distinct elements	468	517,326	21,989	942
#total elements	11,500,870	11,013,317	1,101,572	3,960,507
AvgListLen	24,574.5	21.3	50.1	4204.4

In the subsequent experiments, we randomly selected 1000 documents from each dataset as the query sets. The average document lengths in each query set for ACCIDENTS, DBLP, FLICKR, and T40 were 33.8, 14.0, 6.9, and 39.4, respectively, which were close to the average lengths of the corresponding datasets. In the following experiments, each query was performed five times, and the average total elapsed times were reported.

4.3. Experimental Results and Analysis

4.3.1. Optimal m Selection

The optimal m has a significant impact on the performance of the LDRpV algorithm. To determine the optimal m, we applied LDR-Filtering and carried out experiments on different m values. The query times for the ACCIDENTS, DBLP, FLICKR, and T40 datasets are shown in Table 3.

Table 3. Optimal m value selection.

m	ACCIDENTS (ms)	DBLP (ms)	FLICKR (ms)	T40 (ms)
1	2337.89	7.85	37.26	318.37
2	986.58	2.84	18.90	106.73
3	779.67	3.77	18.56	72.83
4	727.98	3.88	19.71	70.87
5	721.79	4.93	19.19	68.11
6	721.75	5.63	20.38	71.05
7	727.81	6.95	19.49	70.58
8	733.76	9.24	19.84	69.75
9	734.21	10.31	19.88	69.65
10	738.39	9.64	20.17	71.01
12	746.66	10.79	19.68	78.08
16	754.15	13.63	19.62	78.51
20	771.04	14.02	19.93	73.81
50	842.44	13.05	20.81	76.87
100	809.97	13.66	34.43	79.56
∞	1159.41	12.66	48.33	130.20

From Table 3, it can be observed that the optimal m varies across different datasets. For ACCIDENTS, DBLP, FLICKR, and T40, the optimal m values are 6, 2, 3, and 5, respectively. The intersection times for all four datasets exhibit a distinct U-shaped structure, indicating that they first drop sharply as m increases, then level off, and finally show an increase again. This suggests the following: (1) Direct intersection or direct verification often cannot achieve the best performance. (2) It is crucial for enhancing algorithm performance to first intersect a limited number of the shortest lists and then proceed to verification. This also demonstrates the effectiveness of the "intersection + verification" approach. (3) The optimal m is relatively small, and optimality is often neared when $m = 2$ or $m = 3$. Unless otherwise specified, we used the optimal m for each dataset by default in subsequent experiments.

4.3.2. Analysis of the Filtering Efficiency of LDR-Filtering

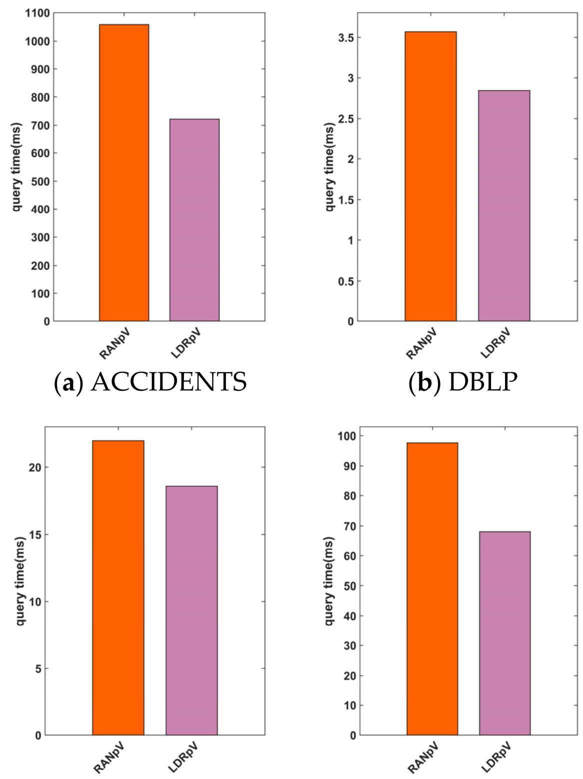
To validate the effectiveness of LDR-Filtering, under the condition of optimal m, a comparison was made between the LDRpV and RANpV (an algorithm that replaces LDR in LDRpV with a random document order). The experimental results are shown in Figure 5a–d.

As can be seen from Figure 5, the performance of LDRpV is significantly better than RANpV, with efficiency improvements ranging between 18.39% and 46.69% across the four datasets. The query times of RANpV and LDRpV are 1058.72 ms and 721.75 ms on ACCIDENTS, respectively, indicating that LDR-Filtering can significantly improve query efficiency. The higher efficiency of LDRpV is due to the superior filtering strength of LDR-Filtering compared to a random document order. To measure the magnitude of LDR-Filtering's filtering power, the LDR-Filtering rate, LFR, is defined as shown in Equation (1).

$$\text{LFR} = 1 - \frac{\sum_{i=1}^{1000} |SL_i'|}{\sum_{i=1}^{1000} |SL_i|} \tag{1}$$

Here, $|SL_i|$ represents the length of the shortest list for the i-th query, while $|SL_i'|$ represents the length of the shortest list for the i-th query after the application of LDR-Filtering.

The LFRs for the four datasets are shown in Figure 6. The filtering capacity of the datasets, ranked from highest to lowest, is as follows: T40 > ACCIDENTS > DBLP > FLICKR. Among them, T40, which has the best performance, can filter out 38.1% of the candidate set, while FLICKR, with the poorest performance, only filters 21.1% of the candidate set. The differences in LFR across various datasets arise from variations in dataset lengths and query length distributions. Intuitively, the more dispersed the dataset length distribution and the longer the query length, the larger the LFR. Overall, LDR-Filtering demonstrates a significant filtering effect.



(**a**) ACCIDENTS (**b**) DBLP

(**c**) FLICKR (**d**) T40

Figure 5. Comparison between random order and length order.

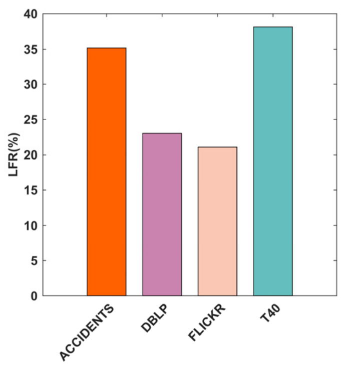


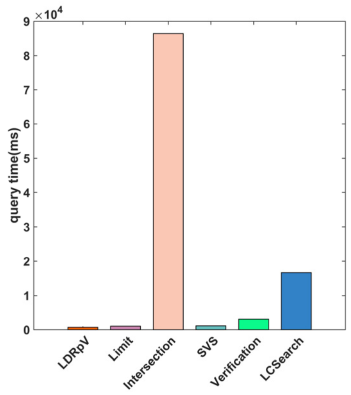
Figure 6. LDR-Filtering rate.

4.3.3. Comparison with Other Algorithms

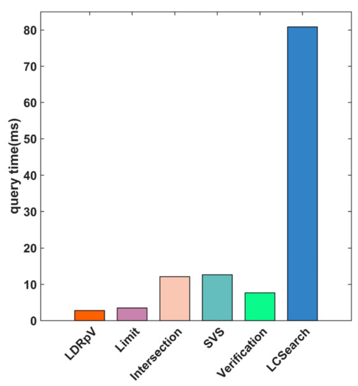
To further investigate the performance of LDRpV compared to other algorithms, we compare it with the following algorithms:

- Intersection [7]: A direct list intersection algorithm.
- SVS [8]: An algorithm that uses heuristic rules to intersect the lists from the shortest to the longest.
- Verification: An algorithm designed in this paper that directly verifies the shortest list.
- LCSearch [10]: An intersection algorithm that simultaneously crosscuts multiple lists.
- Limit [30]: A join algorithm based on Trie implementation using Trie query + verification. For fairness, we modify it into an inverted index-based list intersection algorithm.

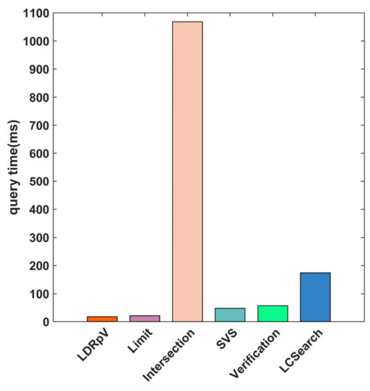
The results of the comparison between different algorithms are shown in Figure 7a–d.



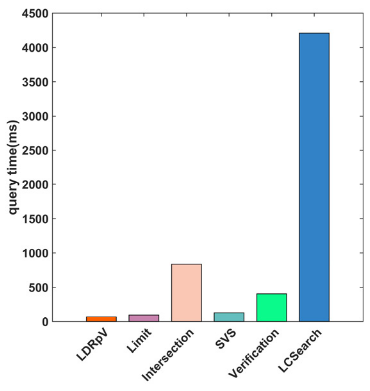
(**a**) ACCIDENTS



(**b**) DBLP



(**c**) FLICKR



(**d**) T40

Figure 7. Comparison of LDRpV with other algorithms.

As shown in Figure 7, among the six algorithms, LDRpV significantly outperforms the other algorithms on all four datasets. The LCSearch and Intersection algorithms, due to their lack of list-length-based heuristic rules, are significantly less efficient than the other algorithms. This indicates that the list-length-based heuristic rules are crucial for enhancing the efficiency of intersection queries. For SVS and Verification, even though they utilize the heuristic rules, their sole intersection or verification method also results in lower efficiency. In contrast, the LDRpV and Limit algorithms perform relatively well. Contributing to LDR-Filtering, LDRpV outperforms Limit across all four datasets. On ACCIDENTS, DBLP, FLICKR, and T40, the query times for LDRpV are 721.75 ms, 2.84 ms, 18.56 ms, and 68.11 ms, respectively, whereas they are 1058.72 ms, 3.57 ms, 21.97 ms, and 97.73 ms for Limit. The efficiency improvements of LDRpV over SVS on these four datasets are 60.64%, 3.45×, 1.60×, and 91.16%, respectively. Compared to the best competitor, Limit, the efficiency improvements of LDRpV are 46.96%, 25.54%, 18.39%, and 43.50%, respectively.

5. Conclusions

In this paper, we designed an efficient list intersection query algorithm, LDRpV, specifically for short documents. Within the algorithm, we employ LDR-Filtering to quickly and efficiently exclude documents with a length smaller than $|q|$. Furthermore, to address the inefficiencies of solely intersection and verification algorithms, we devised an "intersection + verification" strategy, only intersecting the first m lists, thus achieving a high intersection efficiency. Extended experimental results demonstrate that LDRpV significantly outperforms the state-of-the-art algorithms.

A possible limitation of LDRpV is that LDR-Filtering does not work well for very short queries, as fewer unqualified documents can be filtered out. For the most extreme case, $|q|= 1$, no documents can be filtered out. To address this limitation, a potential improvement direction for LDRpV is to adaptively choose whether to use LDR-Filtering for queries with different lengths, which will be our future work.

Author Contributions: Methodology, L.J. and F.X.; Validation, D.L.; Resources, H.Z.; Writing—original draft, L.J. and D.L.; Writing—review & editing, F.X. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China under Grant Nos. 62262035 and 62262034.

Data Availability Statement: The data will be made available by the authors on request.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Bellas, C.; Gounaris, A. Exploiting GPUs for fast intersection of large sets. *Inf. Syst.* **2022**, *108*, 101992. [CrossRef]
2. Jia, L.; Tang, H.; Li, M.; Zhao, B.; Wei, S.; Zhou, H. An Efficient Association Rule Mining-Based Spatial Keyword Index. *Int. J. Data Warehous. Min.* **2023**, *19*, 1–19. [CrossRef]
3. Raman, V.; Qiao, L.; Han, W.; Narang, I.; Chen, Y.-L.; Yang, K.-H.; Ling, F.-L. Lazy, adaptive rid-list intersection, and its application to index anding. In Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, Beijing, China, 11–14 June 2007; pp. 773–784.
4. Shin, Y.; Ahn, J.; Im, D.-H. Join optimization for inverted index technique on relational database management systems. *Expert Syst. Appl.* **2022**, *198*, 116956. [CrossRef]
5. Hu, L.; Zou, L.; Liu, Y. Accelerating triangle counting on GPU. In Proceedings of the 2021 International Conference on Management of Data, Virtual Event China, 20–25 June 2021; pp. 736–748.
6. Xu, Q.; Zhang, F.; Yao, Z.; Lu, L.; Du, X.; Deng, D.; He, B. Efficient load-balanced butterfly counting on GPU. *Proc. VLDB Endow.* **2022**, *15*, 2450–2462. [CrossRef]
7. Helmer, S.; Moerkotte, G. A performance study of four index structures for set-valued attributes of low cardinality. *VLDB J.* **2003**, *12*, 244–261. [CrossRef]
8. Culpepper, J.S.; Moffat, A. Efficient set intersection for inverted indexing. *ACM Trans. Inf. Syst.* **2010**, *29*, 1–25. [CrossRef]
9. Fontoura, M.; Josifovski, V.; Liu, J.; Venkatesan, S.; Zhu, X.; Zien, J. Evaluation strategies for top-k queries over memory-resident inverted indexes. *Proc. VLDB Endow.* **2011**, *4*, 1213–1224. [CrossRef]
10. Deng, D.; Yang, C.; Shang, S.; Zhu, F.; Liu, L.; Shao, L. Lcjoin: Set containment join via list crosscutting. In Proceedings of the 2019 IEEE 35th International Conference on Data Engineering (ICDE), Macao, China, 8–11 April 2019; pp. 362–373.
11. Blandford, D.; Blelloch, G. Index compression through document reordering. In Proceedings of the DCC 2002 Data Compression Conference, Snowbird, UT, USA, 2–4 April 2002; pp. 342–351.
12. Ding, S.; Attenberg, J.; Suel, T. Scalable techniques for document identifier assignment in inverted indexes. In Proceedings of the 19th International Conference on World Wide Web, Raleigh, NC, USA, 26–30 April 2010; pp. 311–320.
13. Dhulipala, L.; Kabiljo, I.; Karrer, B.; Ottaviano, G.; Pupyrev, S.; Shalita, A. Compressing graphs and indexes with recursive graph bisection. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 1535–1544.
14. Moffat, A.; Mackenzie, J. Efficient immediate-access dynamic indexing. *Inf. Process. Manag.* **2023**, *60*, 103248. [CrossRef]
15. Adir, A.; Aharoni, E.; Drucker, N.; Kushnir, E.; Masalha, R.; Mirkin, M.; Soceanu, O. Privacy-preserving record linkage using local sensitive hash and private set intersection. In Proceedings of the International Conference on Applied Cryptography and Network Security, Rome, Italy, 20–23 June 2022; pp. 398–424.
16. Wang, Q.; Suel, T. Document reordering for faster intersection. *Proc. VLDB Endow.* **2019**, *12*, 475–487. [CrossRef]

17. Mackenzie, J.; Petri, M.; Moffat, A. Faster index reordering with bipartite graph partitioning. In Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event Canada, 11–15 July 2021; pp. 1910–1914.
18. Zhao, C.; Huang, T.; Chowdhury, S.B.R.; Chandrasekaran, M.K.; McKeown, K.; Chaturvedi, S. *Read Top News First: A Document Reordering Approach for Multi-Document News Summarization*; Association for Computational Linguistics: Dublin, Ireland, 2022; pp. 613–621.
19. Yafay, E.; Altingovde, I.S. Faster Dynamic Pruning via Reordering of Documents in Inverted Indexes. In Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval, Taipei, Taiwan, 23–27 July 2023; pp. 2001–2005.
20. Ramaswamy, V.; Konow, R.; Trotman, A.; Degenhardt, J.; Whyte, N. Document Reordering is Good, Especially for e-Commerce. In Proceedings of the eCOM@ SIGIR, Tokyo, Japan, 7–11 August 2017.
21. Silvestri, F. Sorting out the document identifier assignment problem. In Proceedings of the European Conference on Information Retrieval, Rome, Italy, 2–5 April 2007; pp. 101–112.
22. Yan, H.; Ding, S.; Suel, T. Inverted index compression and query processing with optimized document ordering. In Proceedings of the 18th International Conference on World Wide Web, Madrid, Spain, 20–24 April 2009; pp. 401–410.
23. Kane, A.; Tompa, F.W. Distribution by document size. In Proceedings of the Workshop on Large-Scale and Distributed Systems for Information Retrieval, New York, NY, USA, 24–28 February 2014.
24. Saini, D.K.J.B.; Patil, P.; Gupta, K.D.; Kumar, S.; Singh, P.; Diwakar, M. Optimized web searching using inverted indexing technique. In Proceedings of the 2022 IEEE 11th International Conference on Communication Systems and Network Technologies (CSNT), Indore, India, 23–24 April 2022; pp. 351–356.
25. Formal, T.; Lassance, C.; Piwowarski, B.; Clinchant, S. From distillation to hard negative sampling: Making sparse neural ir models more effective. In Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, 11–15 July 2022; pp. 2353–2359.
26. Xiao, G.; Wang, J.; Lin, C.; Zaniolo, C. Highly efficient string similarity search and join over compressed indexes. In Proceedings of the 2022 IEEE 38th International Conference on Data Engineering (ICDE), Kuala Lumpur, Malaysia, 9–12 May 2022; pp. 232–244.
27. Luo, J.; Gao, H.; Li, J.; Zhou, Z.; Zhang, T. freshJoin: An Adaptive Algorithm for Set Containment Join. *Proc. VLDB Endow.* **2017**, *11*, 293–308.
28. Jia, L.; Tang, J.; Li, M.; Li, R.; Ding, J.; Chen, Y. A Trie Based Set Similarity Query Algorithm. *Mathematics* **2023**, *11*, 229. [CrossRef]
29. Yang, J.; Zhang, W.; Yang, S.; Zhang, Y.; Lin, X. Tt-join: Efficient set containment join. In Proceedings of the 2017 IEEE 33rd International Conference on Data Engineering (ICDE), San Diego, CA, USA, 19–22 April 2017; pp. 509–520.
30. Bouros, P.; Mamoulis, N.; Ge, S.; Terrovitis, M. Set containment join revisited. *Knowl. Inf. Syst.* **2016**, *49*, 375–402. [CrossRef]