Batch processing of Top-k Spatial-textual Queries

FARHANA M. CHOUDHURY, School of Science, RMIT University, Australia
J. SHANE CULPEPPER, School of Science, RMIT University, Australia
ZHIFENG BAO, School of Science, RMIT University, Australia
TIMOS SELLIS, Data Science Research Institute, Swinburne University of Technology, Australia

Several indexing techniques have been proposed to efficiently answer top-k spatial-textual queries in the last decade. However, all of these approaches focus on answering one query at a time. In contrast, how to design efficient algorithms that can exploit similarities between incoming queries to improve performance has received little attention. In this paper, we study a series of efficient approaches to *batch process* multiple top-k spatial-textual queries concurrently. We carefully design a variety of indexing structures for the problem space by exploring the effect of prioritizing spatial and textual properties on system performance. Specifically, we present an efficient traversal method, SF-SEP over an existing space-prioritized index structure. Then, we propose a new space-prioritized index structure, the MIR-Tree to support a filter-and-refine based technique, SF-GRP. To support the processing of text-intensive data, we propose an augmented, inverted indexing structure that can easily be added into existing text search engine architectures, and a novel traversal method for batch processing of the queries. In all of these approaches, the goal is to improve the overall performance by sharing the I/O costs of similar queries. Finally, we demonstrate significant I/O savings in our algorithms over traditional approaches by extensive experiments on three real datasets, and compare how properties of different datasets affect the performance. Many applications in streaming, micro-batching of continuous queries, and privacy-aware search can benefit from this line of work.

CCS Concepts: • **Information systems** → **Data structures**; *Location based services*;

Additional Key Words and Phrases: Spatial-textual queries, batch queries, spatial-textual queries, efficient query processing.

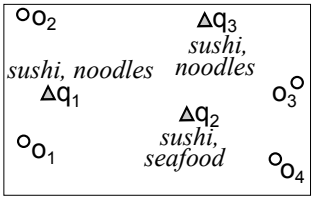
ACM Reference Format:
Farhana M. Choudhury, J. Shane Culpepper, Zhifeng Bao, and Timos Sellis. 2018. Batch processing of Top-k Spatial-textual Queries. *ACM Trans. Spatial Algorithms Syst.* 9, 4, Article 39 (March 2018), 41 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

With the increasing popularity of GPS enabled mobile devices, queries with local intent are quickly becoming one of the most common types of search task on the web. Web contents are available in a variety of forms, including social network posts, points of interest (stores, tourist attractions), news articles, advertisements, and multimedia sharing sites, many of which now depend heavily on geographical information. Common search tasks can include a user location via GPS, and use the information available on the web to improve search effectiveness. Around 53% of Bing searches

Authors' addresses: Farhana M. Choudhury, School of Science, RMIT University, Melbourne, Australia, farhana.choudhury@rmit.edu.au; J. Shane Culpepper, School of Science, RMIT University, Melbourne, Australia, shane.culpepper@rmit.edu.au; Zhifeng Bao, School of Science, RMIT University, Melbourne, Australia, shane.culpepper@rmit.edu.au; Timos Sellis, Data Science Research Institute, Swinburne University of Technology, Hawthorn, Australia, tsellis@swin.edu.au.

2374-0353/2018/3-ART39 $15.00
https://doi.org/0000001.0000001



Objects	Terms and frequency
o_1	$(sushi, 1)$
o_2	$(sushi, 1), noodles, 5)$
o_3	$(sushi, 5), (seafood, 5)$
o_4	$(seafood, 4), (noodles, 1)$

Fig. 1. Example top-k spatial-textual queries

from mobile devices are geographical and have local intent [24]. Location-aware search is not limited to mobile devices. A recent report [26] showed that 84% of all computer users have searched with local intent.

A *Top-k Spatial-Textual Query* is an important search task that has been extensively studied in the literature [8, 15, 22, 28, 31]. Given a set of spatial-textual objects, a top-k spatial-textual query returns the k most similar objects to the query by combining both spatial similarity and textual similarity. For example in Figure 1, o_1, o_2, o_3, o_4 are four restaurants where the locations are shown with circles, along with the term frequencies of the textual description for each restaurant. Let q_1 be a top-1 spatial-textual query with a location shown as a triangle, and its query text is ("`sushi`", "`noodles`"). Although restaurant o_1 is closer to q, o_2 is returned as the result when both spatial and textual similarity are considered.

Many real-life applications require the processing of multiple top-k spatial textual queries over a static or semi-static collection of data, i.e., the collection changes infrequently, in a short period of time. Examples where this might occur include: (i) Multiple-query optimization, where the overall efficiency and throughput can be improved by grouping or partitioning a large set of queries; (ii) Continuous processing of a query stream, where in each time slot, the queries that have arrived can be processed together; (iii) Offline batch processing of top-k queries as a pre-processing step for other data analytical queries. For example, a company might use a set of query logs to detect potential customers who found their products in prior searches. When the set of products of the company is unchanged since the time of the query log, the results for each query in the log can be batch processed offline; and (iv) Supporting privacy-aware search where real top-k spatial-textual queries are masqueraded using artificial queries to hide the real intent or exact location [14, 16].

In this article, given a set of top-k spatial-textual queries Q, our aim is to compute the results for all the queries concurrently and efficiently as a batch. Consider the example in Figure 1 again. Let, q_1, q_2, and q_3 be three different top-k spatial-textual queries to be batch processed. Assume that $k = 1$ for q_1 and q_3, and $k = 2$ for q_2. Both q_1 and q_3 have the same query keywords ("`sushi`", "`noodles`") and the query keywords of q_2 are "`sushi`" and "`seafood`". In order to achieve the goal of efficiently answering multiple top-k spatial-textual queries, we carefully study two fundamental yet indispensable technical challenges.

- Challenge 1: Index Design - How can we extend the index structures utilized in answering individual top-k spatial-textual queries to cater for batch query processing?
- Challenge 2: Shared computation - How can we leverage common computational and I/O costs for many queries in a single batch?

For Challenge 1, several different index structures have been proposed for independent query processing, for example, the IR-tree, CIR-tree, DIR-tree, and CDIR-tree [8, 15, 28], the Spatial Inverted Index (S2I) [22], the Integrated Inverted Index (I^3) [32] and the IL-Quad-tree [30]. Unfortunately, these approaches were developed to process individual queries independently. This can lead to retrieving the same disk pages repeatedly for multiple queries, especially if the queries share

keywords and/or are spatially close to one another. In addition, for spatial-textual data, most of the existing indexing techniques choose a *space-first* model as the primary indexing dimension and augment the textual part into the spatial index [8, 15, 28, 30, 32]. However, there is also a vast amount of location-embedded web data being processed for location-aware search [24, 26], and current web search engines are primarily designed to efficiently query text. Unfortunately, in terms of indexing structures, it can require significant changes to integrate space-first indexes into existing web search engines. Please refer to Table 1 in Section 2 for a systematic comparison of the various indexing approaches to spatial-textual top-k query processing.

For batch processing, there is relatively little prior work on spatial-textual queries. Ding et al. [10] look at batch processing of textual queries in large scale web search engines, but do not consider queries with local intent. Wu et al. [29] have studied a similar problem called *Joint Top-k Spatial Keyword Query Processing*. However, this work addresses the problem only for *Boolean* top-k spatial-textual queries, where the top-k objects are returned based on the spatial proximity from the query location, and each of the results contain *all* of the query keywords without using a weighted textual similarity. The proposed index and pruning rules are applicable to Boolean top-k queries only, and are not easily extensible to queries that consider both spatial and textual similarity. The distinctions between these two related problems are discussed in detail in Section 2.

To process a batch of top-k spatial-textual queries efficiently, in this article we propose a series of carefully designed indexes and traversal algorithms. This article builds on our previous work [5, 6], where (i) the batch processing of top-k spatial-textual queries and the approach SF-SEP is first studied [5], and (ii) a sub-problem of a new query type (MaxBRSTkNN) [6] finds the top-k spatial-textual objects for multiple queries (users) jointly using the approach SF-GRP.

In particular, the SF-SEP approach was first introduced in Choudhury et al. [5], where the I/O costs are shared among the queries by maintaining separate priority queues to track the most promising results. Objects are indexed using an existing structure, which was designed to process queries individually. In later work, we proposed new indexing data structures: the MIR-tree, and a filter-and-refine based technique called SF-GRP [6]. The key idea was to group the queries and traverse the index for the entire group in order to better support shared resource usage, instead of processing queries individually.

The commonality in these two approaches [5, 6] is that, both of them prioritize spatial properties in index construction, which is good for spatial-textual dataset containing a small amount of text information. However, when the text component dominates the dataset, for example, web search queries with local intent, using a space-first index and the corresponding top-k algorithms may not be appropriate, and the space overhead of the space-prioritized index can be an order of magnitude larger than the text-first index (see Sec.6 for details). Moreover, as shown by Chen et al. [3], the text-first index often outperforms a spatial-first index when processing individual top-k spatial-textual queries.

Therefore, to answer the batch top-k spatial-textual queries, we propose a new text-first approach in this article (Section 5), which to our best knowledge, is the first work addressing the batch processing of top-k spatial-textual queries using a text-first model. In summary, this work extends our prior work by making the following contributions:

- We propose a new text-first index structure (SIF index) for text intensive data using a block-based inverted file.
- We present a new traversal algorithm over SIF and several techniques to prune both blocks and objects in the posting lists of the inverted file.

- We propose a novel mechanism to share I/O costs among multiple queries using SIF, where the idea is to maintain a limited number of blocks in a buffer and retrieve the blocks in a specific order so that each block is retrieved at most once.
- We compare the performance of these approaches along with appropriate baselines using three different datasets. We show how the composition of the dataset being used can have a significant impact on performance, and that the choice of algorithm should be data dependent in practice.

The rest of the article is organized as follows. Section 2 presents an overview of related work. Section 3 introduces the preliminaries and defines the problem. Section 4 presents the two space-prioritized approaches: the SF-SEP approach (Section 4.2) and the SF-GRP approach (Section 4.3). In Section 5, we describe our new text-first index SIF, and present algorithms for single and batch processing of top-k spatial-textual queries over SIF. We present our extensive experimental evaluation in Section 6, and conclude in Section 7.

2 RELATED WORK

The problem of processing multiple queries as a batch has been examined in the past in several different contexts. Sellis [25] and Hong et al. [13] studied the multiple-query processing optimization problem in relational databases. The main idea is that if multiple queries have a common sub-expression, then the sub-expressions can be evaluated once, and reused by the associated queries. Although their strategies are not directly applicable to spatial-textual queries, our approach also optimizes query processing by sharing computations across multiple queries.

In spatial databases, researchers have studied how to efficiently process multiple range queries [21] and multiple nearest neighbor queries [4, 33]. For multiple nearest neighbor queries, a few alternative pruning techniques using an R-tree to minimize the number of I/O operations have been explored. Zhang et al. [33] propose different heuristics to group the queries, including an R-tree based method and a spatial hashing method, so that similar queries can be processed together. For multiple range query processing in spatial databases, Papadopoulos and Manolopoulos [21] first sort the queries based on their spatial similarities using a Space Filling Curve (SFC), and then partition them into different groups in order to share the I/O costs. In our work, we consider processing a single group of queries efficiently, where any grouping or partitioning method can be employed prior to applying our approach.

The batch processing of text queries in large document search engines was explored by Ding et al. [10]. While this work has a similar motivation to our current approach, Ding et al. do not consider queries or collections which contain spatial information. Since we focus on batch processing of top-k queries over spatial-textual data, we now review two related problems: (1) top-k spatial-textual queries, and (2) the batch processing of spatial-textual queries.

2.1 Top-k spatial-textual queries

Given a set of spatial-textual objects, a top-k spatial-textual query returns a ranked list of top-k objects, ranked according to a weighted sum of the distance to the query location, and the text relevance to the query keywords. Several different indexing and traversal methods have been proposed to efficiently process a single query [8, 15, 22, 28, 30–32]. Chen et al. [3] performed a comprehensive study comparing most of these techniques for a wide variety of related problems. For top-k queries, they show that S2I [22] is the most efficient approach in most cases, albeit with a significant space overhead. When the number of query keywords is large (greater than 5), the CDIR-tree [8] performs slightly better than S2I in terms of runtime.

Table 1. Related work

	Approach		Text Similarity		Batch
	Space	Text	Boolean	Ranked	
IR-tree [8]	✓		✓	✓	
DIR-tree [8]	✓	✓	✓	✓	
CDIR-tree [8]	✓	✓	✓	✓	
LiIR-tree [15]	✓		✓	✓	
SFC-Skip [7]	✓	✓	✓		
I^3 [32]	✓		✓	△	
S2I [22]		✓	✓	✓	
IL-Quad [30]		✓	✓	△	
Aggregate top-k [31]		✓	✓	✓	
GeoWAND [17]		✓	✓	✓	
WIBR-tree [29]		✓	✓		✓
Batch-SF-SEP [5]	✓		✓	✓	✓
Batch-SF-GRP [6]	✓		✓	✓	✓

In S2I, for each frequent term t, the objects containing that term are indexed using an aggregated R-tree (aR-tree) [20]. Each node n of the aR-tree stores the maximum textual impact of t for the objects of the subtree rooted at n. For each infrequent term t', S2I uses an inverted file to store the objects containing t'; for each object, the object ID, the location, and the impact of t' for the object are stored as a tuple. Finally, S2I organizes the blocks and trees by the vocabulary. Physically, for each distinct term, it stores the number of objects containing the term (the document frequency), a flag indicating the type of storage used by the term (block or tree), and a pointer to the block or the aR-tree that stores the object. The major drawback of the S2I approach is that an object is stored multiple times in different structures, and the space overhead due to the redundancies can become unacceptable in large collections. The IR-tree and its variants, including the CDIR-tree also suffer from large space overheads in the index structures. This issue was also pointed out by Chen et al. [3].

More recently, Mackenzie et al. [17] proposed an in-memory index called GEOWAND that combines spatial information and inverted indexes. The posting list for each term is associated with the minimum bounding rectangle (MBR) of the objects stored in that list. For each term t, the maximum textual impact of t among all objects in the posting list of t is also stored. Using these two features, the approach extends the WAND algorithm (presented in Section 5.1) to prune the objects that cannot be a result of a top-k spatial textual query.

2.2 Batch processing of top-k spatial-textual queries

Wu et al. [29] consider the problem of answering multiple conjunctive Boolean top-k queries, where a query returns k objects according to the distance from the query location, and each result object must contain *all* of the query keywords. Given a set of queries, the queries are processed jointly as a single query. They introduce two indexes, the *W-IR-tree* and the *W-IBR-tree*, where the objects are partitioned based on terms. In this way, the objects that satisfy a Boolean predicate over the terms can be easily identified while traversing the tree. Wu et al. also propose the GROUP algorithm that can be used with an *W-IR-tree* as well as other existing indexes such as the IR-tree or the CDIR-tree [8, 28]. However, most of these pruning strategies depend on Boolean constraints, and are not easily amenable to the more general case of ranked top-k queries.

A categorization of previous work is shown in Table 1. If spatial property is prioritized to construct the index, the index is listed as "index: space", and we call it a space-first index. For example, an

Table 2. Basic notation

Symbol	Description
O	The set of objects
Q	The set of queries
ℓ_o, ℓ_q	The location of object o (query q)
d_o, d_q	The text description of object o (query q)
k_q	The number of objects to be returned as the result of q
α	Preference parameter to weight spatial and textual similarities
$STS(o,q)$	The combined spatial and textual similarity between object o and query q
$SS(\ell_o,\ell_q)$	Spatial similarity between the locations of o and q
$TS(d_o,d_q)$	Textual similarity between the text of object o and query q

IR-tree is essentially an R-tree constructed with MBRs, and the inverted file for the objects stored are associated with that node. Similarly, if text property is prioritized, we refer to this approach as a *text-first index*, e.g., the index described by Mackenzie et al. [17]. If both spatial and textual property is considered while constructing an index, both "space" and "text" are marked. The symbol $\triangle$ means that the index was not originally designed to answer the corresponding query type (Boolean or ranked), but can be extended to process that query type. Here, a ranked top-k query returns the k most similar objects based on both spatial and textual similarities.

Although there are many approaches that explore the processing of individual top-k spatial-textual queries using spatial-first and/or text-first indexes, there are only other three works [5, 6, 29] which focus on batch processing of queries. Wu et al. [29] address the problem for the Boolean case. Finally, our two prior approaches ([5, 6]) also consider batch processing of top-k spatial-textual queries, and are listed as "Batch-SF-SEP" and "Batch-SF-GRP", respectively. Both of these works prioritize spatial property when constructing the index, whose performance suffers when using text-intensive datasets in terms of index size and query processing time (as highlighted in Section 1). From Section 5 onward, we propose a text-first approach, which to the best of our knowledge is the first work that addresses the batch processing of top-k spatial-textual queries in a text-first manner.

3 PROBLEM FORMULATION

Table 2 presents the basic notation used in the remainder of the paper. Let O be a set of objects in a spatial-textual database. Each object $o \in O$ is defined as a pair (ℓ_o, d_o), where ℓ_o is a location in Euclidean space and d_o is a text document. Give a query location ℓ_q, a set of query keywords d_q, and the number of results to return k_q, a **Top-k Spatial-Textual Query**, $q(\ell_q, d_q, k_q)$, returns a ranked list of the k_q most relevant spatial-textual objects from O according to a similarity metric, $\hat{S}$. **Batch Processing of Top-k Spatial-Textual Queries** concurrently processes a set Q of top-k spatial-textual queries. A widely adopted similarity metric for top-k spatial-textual queries is the linear combination of spatial similarity and textual similarity [3].

$$STS(o,q) = \alpha \cdot SS(\ell_o,\ell_q) + (1-\alpha) \cdot TS(d_o,d_q)\,, \tag{1}$$

where $SS(\ell_o,\ell_q)$ is the spatial similarity between the query location and the object's location, $TS(d_o,d_q)$ is the textual similarity, and the preference parameter $\alpha \in [0,1]$ defines the importance of one similarity measure relative to the other. Both measures are normalized within $[0,1]$.

Spatial similarity. The spatial similarity of an object o with respect to a query q is:

$$SS(\ell_o, \ell_q) = 1 - \frac{dist(\ell_o, \ell_q)}{d_{max}}, \tag{2}$$

where $dist(\ell_o, \ell_q)$ is the Euclidean distance, and d_{max} is the maximum Euclidean distance between any two points in O.

Text similarity. An object o is considered similar to a query q iff d_o contains at least one term $t \in d_q$. Several measures can be used to compute the similarity between any two text descriptions [18]. We use the TF·IDF metric [23] for illustration purpose in this work, but our approach is applicable to any text-based similarity measure.

The TF (Term Frequency), $TF(t, d_o)$, counts how many times the term t appears in a document object d_o, and the IDF (Inverse Document Frequency) $IDF(t, O) = \log \frac{|O|}{|o \in O, TF(t,d_o)>0|}$ measures the importance of t w.r.t. all of the documents in an object collection O. The text similarity of an object d_o with respect to a query q is

$$TS(d_o, q) = \sum_{t \in d_q \cap d_o} TF(t, d_o) \times IDF(t, O) \tag{3}$$

4 SPACE-PRIORITIZED APPROACHES

In this section, we present two approaches where the spatial property of the data is prioritized in index building and the processing of a batch of top-k spatial-textual queries. Specifically, the approaches: (i) process the batch by maintaining a separate priority queue for each query during the index traversal, and (ii) group the queries and do a single index traversal to create a candidate set of top-k objects, and then verify the results w.r.t. each query. In Section 4.1 we first describe how to use an IR-tree [8] as a preliminary indexing structure for batch processing. Then we present two new space-prioritized approaches, SF-SEP in Section 4.2 and SF-GRP in Section 4.3.

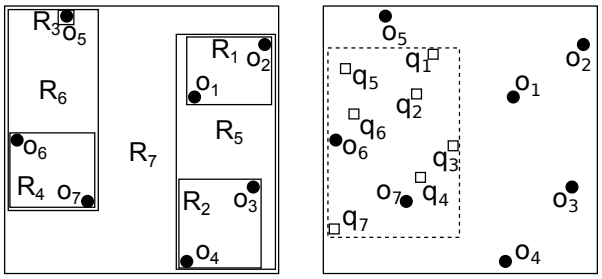
4.1 Preliminaries: the IR-tree

An IR-tree is an R-tree [12] where each node is augmented with a reference to an inverted file [34] for the documents in the subtree. Each node R contains a number of entries, and consists of a reference to a child node of R, the MBR of all entries of the child node, and an identifier of a text description. If R is a leaf node, this is the identifier of the text description of an object. Otherwise, the text identifier is used for a pseudo-text description. The pseudo-text description is the union of all text descriptions in the entries of the child node. The weight of a term t in the pseudo-document is the maximum weight of the weights of this term in the documents contained in the subtree. Each node has a reference to an inverted file for the entries stored in the node. A posting list of a term t in the inverted file is a sequence of pairs $\langle d, \boldsymbol{w}(d,t)\rangle$, where d is the document id containing t, and $\boldsymbol{w}(d,t)$ is the weight of term t in d.

Figure 2a shows the locations and the text descriptions of an example dataset $O = \{o_1, o_2, \ldots, o_7\}$ and Figure 3 illustrates the IR-tree for O. Table 4 and Table 5 present the inverted files of the leaf nodes (InvFile 1 - InvFile 4) and the non-leaf nodes (InvFile 5 - InvFile 7), respectively. In this example, the weights are shown as term-frequencies to simplify the presentation, but the actual weights stored in the inverted files are pre-computed using Equation 3.

4.2 Batch processing using separate priority queues (SF-SEP)

We now present a best-first algorithm where the key idea is to process all of the queries $q \in Q$ concurrently by maintaining separate priority queues to track the possible result objects (or nodes of the index) for each query. In this approach, if multiple queries share a result object, or require the



(a) Objects and MBRs (b) Queries

Objects	Terms and frequency
o_1	$(t_1,1),(t_2,4)$
o_2	$(t_4,1)$
o_3	$(t_1,5),(t_3,5)$
o_4	$(t_4,2)$
o_5	$(t_1,4),(t_2,1)$
o_6	$(t_1,1),(t_3,1)$
o_7	$(t_1,2),(t_4,3)$

Fig. 2. Spatial-textual objects and queries

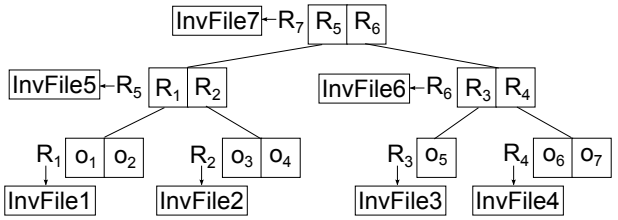


Fig. 3. The example IR-tree

Table 3. Text description of the queries

Term \ Query	q_1	q_2	q_3	q_4	q_5	q_6	q_7
t_1	1	1	1	1	1	1	1
t_2	0	0	0	1	1	0	0
t_3	1	1	0	1	0	0	0
t_4	1	1	0	0	0	1	1

Table 4. Posting lists of the leaf nodes of example IR-tree

Term	**InvFile 1**	**InvFile 2**	**InvFile 3**	**InvFile 4**
t_1	$(o_1,1)$	$(o_3,5)$	$(o_5,4)$	$(o_6,1),(o_7,2)$
t_2	$(o_1,4)$		$(o_5,1)$	
t_3		$(o_3,5)$		$(o_6,1)$
t_4	$(o_2,1)$	$(o_4,2)$		$(o_7,3)$

Table 5. Posting lists of the non-leaf nodes of example IR-tree

Term	**InvFile 5**	**InvFile 6**	**InvFile 7**
t_1	$(R_1,1)$, $(R_2,5)$	$(R_3,4)$, $(R_4,2)$	$(R_5,5),(R_6,4)$
t_2	$(R_1,4)$	$(R_3,1)$	$(R_5,4),(R_6,1)$
t_3	$(R_2,5)$	$(R_4,1)$	$(R_5,5),(R_6,1)$
t_4	$(R_1,1),(R_2,2)$	$(R_4,3)$	$(R_5,2),(R_6,3)$

ALGORITHM 1: SF-SEP (Q, IRtree)

```
1.1  Initialize an array H of |Q| min-priority queues to order the top-k results for each query.
1.2  Initialize an array PQ of |Q| min-priority queues to track node traversal for each query.
1.3  for each q ∈ Q do
1.4      PQ_q ← ENQUEUE(IRtree(root), 0)
1.5  while Q ≠ ∅ do
1.6      Select PQ_r randomly.
1.7      E ← TOP(PQ_r)
1.8      if E is an object then
1.9          for each q ∈ Q do
1.10             while PQ_q ≠ ∅ and SIZEOF(H_q) < k_q and TOP(PQ_q) is an object do
1.11                 o ← DEQUEUE(PQ_q)
1.12                 H_q ← ENQUEUE(o, STS(o, q))
1.13                 if PQ_q = ∅ or SIZEOF(H_q) ≥ k_q then
1.14                     Mark q finished.
1.15     else
1.16         READ(E)
1.17         for each q ∈ Q do
1.18             if E ∈ PQ_q then
1.19                 for each e ∈ E do
1.20                     if d_q ∩ d_e ≠ ∅ then
1.21                         PQ_q ← ENQUEUE(e, STS(e, q))
1.22                 Remove E from PQ_q.
1.23 Return H
```

same node to be retrieved, that node is guaranteed to be retrieved from disk at most once during the process. The pseudo-code of the process is shown in Algorithm 1. We assume that all objects $o \in O$ are stored on disk and indexed using an IR-tree. The algorithm is generic, and applicable to other space-first index structures as well, as described in Section 4.2.1.

In order to process all of the queries in a single pass, two arrays of priority queues of size $|Q|$ are necessary. First, a max-priority queue H_q is maintained for each query $q \in Q$ to store the top-k results (Line 1.1). For each query $q \in Q$, we also maintain a max-priority queue PQ_q to track the relevant nodes and the objects, where the key is the corresponding relevance score for q, computed according to Equation 1. The algorithm will continue to execute as long as the set Q is non-empty.

In each iteration, a priority queue PQ_r is selected at random and the top element E of that queue is processed (Lines 1.6-1.7). We explain the reasoning behind this selection method and discuss the effect of other selection orders in Section 4.2.2. If E is an object, the top element o of PQ_q is dequeued and inserted into H_q as long as o is an object and H_q has less than k_q elements for each PQ_q (Lines 1.8-1.12). In this manner, all the queries that have any object including E in the top of their queues are considered. If PQ_q is empty (there is no object left that can be a result of q), or H_q has k_q elements, then q is marked as finished, and discarded from further computation (Lines 1.13-1.14).

If E is not an object, then the elements of E are read from disk. For each query $q \in Q$, if the corresponding PQ_q contains E, the elements of E that have at least one keyword of d_q are enqueued in PQ_q. The node E is then removed from these queues (Lines 1.16-1.22). The process terminates

when the results for all of the queries in Q are found. Finally, the results represented as an array of priority queues H is returned.

Example 4.1. Table 6 demonstrates the execution of Algorithm 1 for the data from Figure 2a. The location of queries $Q = q_1, q_2, \ldots, q_7$ are shown in Figure 2b, and the query keywords are shown in Table 3. We show the steps for the queries q_1, q_2 and q_3 from Q as an example. Let $\alpha = 0.5$ and $k = 1$. As shown in the table, the priority queues of the queries are initialized with the root node of the tree. Consider iteration 2 as an instance where node R_5 is selected. Node R_5 is present in the priority queues of the queries q_1, q_2 and q_3, where R_5 is the top element of the priority queue of q_3. The elements of R_5 (R_1 and R_2) are retrieved from disk. Node R_1 and R_2 are enqueued in the priority queues along with the corresponding relevance scores. Then, node R_5 is removed from these queues. Note that, although the objects in R_5 do not contribute to the final result of the queries q_1 and q_2, but R_5 needs to be retrieved for q_3 anyway. The total number of iterations will be the same for any selection order imposed on E. The process continues until the results for all of the queries are found.

Space complexity of the SF-SEP approach. Although in the SF-SEP approach, a separate priority queue is maintained for each query, only those nodes and objects that are also required for individual processing are retrieved. Moreover, we ensure that a node or an object is retrieved only once. If a node is included in multiple priority queues, a pointer to the actual content of the node will suffice rather than copying the actual content of those nodes to each of those priority queues. Thus the worst case space complexity of the SF-SEP approach using the IR-tree is the same as processing a single query in the approach presented by Cong et al. [8] plus the space required to maintain $p \times \sum_{q \in Q}$ the size of the priority queue of q, where p is the size of a pointer.

4.2.1 Using other index structures. The key idea of the algorithm is to share the I/O costs, and the processing among queries. When an object is retrieved from the disk for a query, the score of the object is updated for all the queries that share the object in their corresponding queues. The other processing steps of the Algorithm 1 are same as that of processing a single query. Therefore, this algorithm is easy to be extended to other spatial-first index structures.

4.2.2 Node selection order. In this section we discuss the selection order of the priority queue and the node to process in each iteration of our algorithm.

Random selection. Recall that in each iteration, a priority queue is selected at random and the top element E of that queue is processed. For each query q, the corresponding priority queue PQ_q is maintained according to the maximum relevance of the nodes. Although in each iteration, the top element E is dequeued for a random query, all of the queries that have any object including E in the top of their queues are considered in lines 1.10-1.14 in the same iteration. Thus, we are applying a best-first approach not only for the query q_r, but also for other queries concurrently in each iteration. Moreover, only those nodes and objects that are also required for individual processing are retrieved, and a node or an object is retrieved only once.

In the best-first approach, the computation for a query q can be safely terminated iff k_q objects are found, or PQ_q is empty. The lines 1.13-1.14 ensure the terminating conditions for all queries regardless of the selection of E. Therefore, the algorithm is actually *independent of the retrieval order of the nodes*. We now explain this further by contrasting with other possible retrieval orderings.

Selecting the top node for the maximum number of queues. Let the node E be the top element of the maximum number of queues, which is selected in each iteration. According to lines 1.10-1.12, if E is an object, then any object including E that is in the top of any queue is checked for being a

result object of the corresponding query. If E is not an object, then the elements of E are enqueued in PQ_q if they have at least one keyword of d_q and PQ_q contains E.

In both conditions, all of the computations are the same as when selecting E randomly. In summary, the SF-SEP approach extends the best-first approach presented by Cong et al. [8] that computes the top-k objects of a single query. In the SF-SEP approach, a separate priority queue is maintained for each query in the same order, and traversed in a best-first manner [8]. In each iteration, the top element E is dequeued from the priority queue for a random query, and its score is updated for each query that has E in its corresponding queue in the same iteration. Therefore, only those nodes and objects that are also required for individual processing are retrieved, and our contribution is in the techniques to ensure that a node or an object is retrieved only once. So, although the random selection of the priority queue is applied, the total number of nodes retrieved for the batch of queries will be the same for any other selection order. Moreover, since keeping track of the node that is at the top of the maximum number of queues (or any other ordering) in each iteration requires additional computation, random node selection is the preferred approach.

Table 6. The example steps for the SF-SEP approach (using q_1, q_2, q_3)

Iteration	E	q	$\boldsymbol{PQ}_q$	H_q
		q_1	$(R_7, 1.0)$	–
		q_2	$(R_7, 1.0)$	–
		q_3	$(R_7, 1.0)$	–
1	R_7	q_1	$(R_6, 0.8), (R_5, 0.1)$	–
		q_2	$(R_6, 0.7), (R_5, 0.2)$	–
		q_3	$(R_5, 0.5), (R_6, 0.5)$	–
2	R_5	q_1	$(R_6, 0.8), (R_1, 0.1), (R_2, 0.1)$	–
		q_2	$(R_6, 0.7), (R_2, 0.2), (R_1, 0.1)$	–
		q_3	$(R_6, 0.5), (R_1, 0.2), (R_2, 0.2)$	–
3	R_6	q_1	$(R_3, 0.7), (R_4, 0.6), (R_1, 0.1), (R_2, 0.1)$	–
		q_2	$(R_3, 0.6), (R_4, 0.5), (R_2, 0.2), (R_1, 0.1)$	–
		q_3	$(R_4, 0.5), (R_3, 0.4), (R_1, 0.2), (R_2, 0.2)$	–
5	R_3	q_1	$(o_5, 0.7), (R_4, 0.6), (R_1, 0.1), (R_2, 0.1)$	–
		q_2	$(o_5, 0.6), (R_4, 0.5), (R_2, 0.2), (R_1, 0.1)$	–
		q_3	$(R_4, 0.5), (o_5, 0.4), (R_1, 0.2), (R_2, 0.2)$	–
5	o_5	q_1	$(R_4, 0.6), (R_1, 0.1), (R_2, 0.1)$	o_5
		q_2	$(R_4, 0.5), (R_2, 0.2), (R_1, 0.1)$	o_5
		q_3	$(R_4, 0.5), (o_5, 0.4), (R_1, 0.2), (R_2, 0.2)$	–
5	R_4	q_1	–	o_5
		q_2	–	o_5
		q_3	$(o_7, 0.5), (o_6, 0.4), (o_5, 0.4), (R_1, 0.2), (R_2, 0.2)$	–
5	o_7	q_1	–	o_5
		q_2	–	o_5
		q_3		o_7

4.3 Batch processing by grouping queries (SF-GRP)

In this approach, we first present a new index structure, the Min-max IR-tree (MIR-tree). We propose a filter-and-refine approach, where we group the queries into a batch, and traverse the MIR-tree for the group. We apply several pruning strategies and ensure that a node or an object is accessed at most once. Thus, we can reduce the overall computations by sharing processing and I/O costs. Unlike the approach described in Section 4.2, we use a shared priority queue of objects and nodes during tree

Table 7. Notations used in Section 4.3

Symbol	Description
$\boldsymbol{w}^{\downarrow}(d,t)$	Minimum text weight of a term t in a text document d
$\boldsymbol{w}^{\uparrow}(d,t)$	Maximum text weight of a term t in a text document d
$\mathcal{S}$	Super-query, constructed in Section 4.3.2
$STS^{\uparrow}(E,\mathcal{S})$	Maximum spatial-textual similarity between a node E of the MIR-tree and $\mathcal{S}$
$SS^{\uparrow}(E,\mathcal{S})$	Maximum spatial similarity between E and $\mathcal{S}$
$TS^{\uparrow}(E,\mathcal{S})$	Maximum textual similarity between E and $\mathcal{S}$
$\mathfrak{R}_k(\mathcal{S})$	k-th best lower bound score for any query $q \in \mathcal{S}$
$\mathfrak{R}_k(q)$	k-th ranked object of a query q

Table 8. Posting lists of the leaf nodes of MIR-tree

Term	InvFile 1	InvFile 2	InvFile 3	InvFile 4
t_1	$(o_1,1,1)$	$(o_3,5,5)$	$(o_5,4,4)$	$(o_6,1,1),(o_7,2,2)$
t_2	$(o_1,4,4)$	-	$(o_5,1,1)$	-
t_3	-	$(o_3,5,5)$	-	$(o_6,1,1)$
t_4	$(o_2,1,1)$	$(o_4,2,2)$	-	$(o_7,3,3)$

Table 9. Posting lists of the non-leaf nodes of MIR-tree

Term	InvFile 5	InvFile 6	InvFile 7
t_1	$(R_1,1,0)$, $(R_2,5,0)$	$(R_3,4,4)$, $(R_4,2,1)$	$(R_5,5,0),(R_6,4,1)$
t_2	$(R_1,4,0)$	$(R_3,1,1)$	$(R_5,4,0),(R_6,1,0)$
t_3	$(R_2,5,0)$	$(R_4,1,0)$	$(R_5,5,0),(R_6,1,0)$
t_4	$(R_1,1,0),(R_2,2,0)$	$(R_4,3,0)$	$(R_5,2,0),(R_6,3,0)$

traversal. Finally, the top-k objects of the individual queries are verified and returned as the result from the retrieved objects in the tree traversal step. Table 7 presents the notation used in this Section.

4.3.1 Index: Min-max IR-tree (MIR-tree). We propose the Min-max IR-tree (MIR-tree) to index the objects. The objects are inserted in the same manner as in IR-tree. However, unlike an IR-tree, each term is associated with both the maximum $\boldsymbol{w}^{\uparrow}(d,t)$ and minimum $\boldsymbol{w}^{\downarrow}(d,t)$ weights in each document. The posting list of a term t is a sequence of tuples $\langle d, \boldsymbol{w}^{\uparrow}(d,t), \boldsymbol{w}^{\downarrow}(d,t)\rangle$, where d is the document identifier containing t, $\boldsymbol{w}^{\uparrow}(d,t)$ is the maximum, and $\boldsymbol{w}^{\downarrow}(d,t)$ is the minimum weight of term t in d, respectively. If R is a leaf node, both weights are the same as the actual weight of the term t, $\boldsymbol{w}(d,t)$ in the IR-tree. If R is a non-leaf node, the pseudo-document of R is the union of all text descriptions in the entries of the child node. The maximum (minimum) weight of a term t in the pseudo-document is the maximum (minimum) weight in the union (intersection) of the documents contained in the subtree. If a term is not in the intersection, $\boldsymbol{w}^{\downarrow}(d,t)$ is set to 0.

Table 8 presents the inverted files of the leaf nodes (InvFile 1, InvFile 2, InvFile 3, and InvFile 4) and the non-leaf nodes (InvFile 5, InvFile 6, and InvFile 7) of the MIR-tree for the example objects in Figure 2a. The tree structure of the MIR-tree is same as the IR-tree (Figure 3. As a specific example, the maximum (minimum) weight of term t_1 in entry R_4 of InvFile 6 is 2 (1), which is the maximum (minimum) weight of the term in the union (intersection) of documents (o_6, o_7) of the node R_4.

Construction cost and space complexity of MIR-tree**.** We explain the additional disk storage required by the MIR-tree by comparing with the original IR-tree [8]. In contrast to the IR-tree, the space requirement of the MIR-tree includes an additional weight stored for the minimum text relevance for each term in each node. Specifically, for a node E, let the number of unique terms in the pseudo-document of E be M. Then additional $|\sum_{i=1}^{M} d_i|$ minimum weights need to be stored in the inverted file of node E, where d_i is the number of entries in the posting list of term i in node E. The number of nodes and the number of postings in each inverted list of an MIR-tree are the same as that of the corresponding IR-tree. The construction process of the MIR-tree is almost identical to the original IR-tree. During tree construction, when determining the maximum weight of each term in a node, the minimum weight of that term can be determined concurrently.

Update cost of MIR-tree**.** The insertion and deletion operations of the MIR-tree are adapted from the corresponding operations of the IR-tree [8] operations. The algorithms use a standard implementation of the R-tree [12] with two operations, ChooseLeaf and Split. In both algorithms, the only difference between the IR-tree and the MIR-tree is that the minimum weight of a term (additional requirement of an MIR-tree) needs be adjusted in the inverted lists of the necessary nodes. Please refer to Algorithm 1 of Cong et al. [8] for the pseudocode of the insertion operation on IR-tree (deletion is done in a similar manner using the operations of an R-tree).

As the split and merge of the nodes are executed in the same manner as the IR-tree, and the minimum weight of a term (additional requirement of an MIR-tree) can be adjusted in the same iteration required to adjust the maximum text weight of the IR-tree, the time complexity of both operations for MIR-tree are the same as that of the IR-tree.

Variants of MIR-tree**.** In this work, the proposed MIR-tree is an extension of the original IR-tree presented by Cong et al. [8], who also proposed other variants of the IR-tree, such as the DIR-tree and the CIR-tree, where both spatial and textual criteria are considered to construct the nodes of the tree. The same structures can be used during the construction our proposed extension. For example, the nodes of the MIR-tree can be constructed in the same manner as the DIR-tree, and the posting lists of each node will contain both the minimum and maximum weights of the terms.

4.3.2 Query grouping. Our goal is to access the necessary objects from disk, and avoid duplicate retrieval of objects for different queries. We form a group of queries for this purpose, denoted as a "super-query" ($\mathcal{S}$), and the objects are accessed using this group instead of individual queries.

Super-query construction. The "super-query" ($\mathcal{S}$) is constructed such that $\ell_{\mathcal{S}}$ is the MBR enclosing the locations of all queries, $dUni_{\mathcal{S}}$ is the union, and $dInt_{\mathcal{S}}$ is the intersection of the keywords of all queries, respectively.

As an example, Figure 2b shows the locations of the queries $Q = q_1, q_2, \ldots, q_7$ and the corresponding text descriptions are presented in Table 3. The location of the "super-query", $\ell_{\mathcal{S}}$ is the MBR enclosing the locations of all the queries, shown with a dotted rectangle. Here, the intersection of the keywords of all the queries, $dInt_{\mathcal{S}}$ is '1000' and the union, $dUni_{\mathcal{S}}$ is '1111'.

We now present the notion for upper and lower bound estimations for spatial-textual relevance scores between any query q, and any object node of the MIR-tree using this super-query.

4.3.3 Upper and Lower Bound Estimation. The maximum spatial-textual similarity between any node E of the MIR-tree and the super-query $\mathcal{S}$ is computed as:

$$STS^{\uparrow}(E, \mathcal{S}) = \alpha \cdot SS^{\uparrow}(\ell_E, \ell_{\mathcal{S}}) + (1 - \alpha) \cdot TS^{\uparrow}(d_E, dUni_{\mathcal{S}}),$$

ALGORITHM 2: TREE TRAVERSAL OF THE SF-GRP APPROACH(MIR-tree, Q, k)

Output: The top-k objects of all queries
$\ell_{\mathcal{S}} \leftarrow \forall q \in Q, MBR(\ell_q)$; $dUni_{\mathcal{S}} \leftarrow \forall q \in Q, \cup(d_q)$; $dInt_{\mathcal{S}} \leftarrow \forall q \in Q, \cap(d_q)$
Initialize max-priority queue PQ, RO, min-priority queue LO
$E \leftarrow$ MIR-tree($root$)
ENQUEUE($PQ, E, STS^{\downarrow}(E, \mathcal{S})$)
while $PQ \neq \emptyset$ **do**
  $E \leftarrow$ DEQUEUE(PQ)
  if E *is leaf* **then**
    if $|LO| < k$ **then**
      ENQUEUE($LO, E, STS^{\downarrow}(E, \mathcal{S})$)
      if $|LO| = k$ **then**
        $\mathfrak{R}_k(\mathcal{S}) \leftarrow STS^{\downarrow}(\text{TOP}(LO), \mathcal{S})$
    else if $STS^{\uparrow}(E, \mathcal{S}) \geq \mathfrak{R}_k(\mathcal{S})$ **then**
      ENQUEUE($LO, E, STS^{\downarrow}(E, \mathcal{S})$)
      $Obj \leftarrow$ DEQUEUE(LO)
      $\mathfrak{R}_k(\mathcal{S}) \leftarrow STS^{\downarrow}(Obj, \mathcal{S})$
      if $STS^{\uparrow}(Obj, \mathcal{S}) \geq \mathfrak{R}_k(\mathcal{S})$ **then**
        ENQUEUE($RO, Obj, STS^{\uparrow}(Obj, \mathcal{S})$)
  else
    if $|LO| < k$ **or** $STS^{\uparrow}(E, \mathcal{S}) \geq \mathfrak{R}_k(\mathcal{S})$ **then**
      for *each* $e \in E$ **do**
        ENQUEUE($PQ, e, STS^{\downarrow}(e, \mathcal{S})$)
return INDIVIDUAL_TOPK($Q, \mathcal{S}, LO, RO$)

where $SS^{\uparrow}$ is the maximum spatial similarity computed from the minimum Euclidean distance between the two MBRs using Equation 2, and $TS^{\uparrow}$ is the maximum textual similarity between d_E and the union of the keywords of the queries, $dUni_{\mathcal{S}}$ computed as:

$$TS^{\uparrow}(d_E, dUni_{\mathcal{S}}) = \sum_{t \in dUni_{\mathcal{S}} \cap d_E} \boldsymbol{w}^{\uparrow}(d_E, t)$$

where $\boldsymbol{w}^{\uparrow}(d_E, t)$ is the maximum weight of the term t in the associated document of node E. As described in Section 4.1, if E is a non-leaf node, $\boldsymbol{w}^{\uparrow}(d_E, t)$ is the maximum weight in the union of the documents contained in the subtree of E. Otherwise, $\boldsymbol{w}^{\uparrow}(d_E, t)$ is the weight of term t in document d_E computed using Equation 3.

We now present a lemma that enables us to estimate an upper bound on the relevance between any query $q \in Q$, and any object node E using the super-query $\mathcal{S}$, where E is a node of the MIR-tree.

LEMMA 4.2. *$\forall q \in Q$, $STS^{\uparrow}(E, \mathcal{S})$ is an upper bound estimation of $STS(E, q)$, such that, for any object node E, $STS(E, q) \leq STS^{\uparrow}(E, \mathcal{S})$.*

PROOF. Recall that the $\ell_{\mathcal{S}}$ of the super query is the MBR of the locations for all of the queries in Q. For an object node E of the MIR-tree, $SS^{\uparrow}(E, \mathcal{S})$ is the maximum spatial similarity computed from the minimum Euclidean distance between the two MBRs of E and $\mathcal{S}$ using Equation 2. As the location ℓ_q of any query $q \in Q$ is inside the rectangle $\ell_{\mathcal{S}}$, the value $SS(E, q)$ must be less than or equal to $SS^{\uparrow}(E, \mathcal{S})$. For textual similarity, as $dUni_{\mathcal{S}} = \cup_{q \in Q} d_q$, the maximum textual similarity score

between any query $q \in Q$, and any object node E that can be achieved is $TS^{\uparrow}(E, \mathcal{S})$ from Equation 3. Since the spatial-textual score $STS(E, q)$ is the weighted sum of the corresponding spatial and textual scores, $\forall q \in Q$, the score $STS(E, q)$ must also be less than or equal to $STS^{\uparrow}(E, \mathcal{S})$. □

Lemma 4.2 guarantees that $STS^{\uparrow}(E, \mathcal{S})$ is a correct upper bound estimation of the relevance between a node E of the MIR-tree and any $q \in Q$, as the relevance $STS(E, q)$ is always less than $STS^{\uparrow}(E, \mathcal{S})$. Similarly, a lower bound relevance can be computed as:

$$STS^{\downarrow}(E, \mathcal{S}) = \alpha \cdot SS^{\downarrow}(\ell_E, \ell_{\mathcal{S}}) + (1 - \alpha) \cdot TS^{\downarrow}(d_E, dInt_{\mathcal{S}})$$

where $SS^{\downarrow}$ is the minimum spatial similarity computed from the maximum Euclidean distance between the two MBRs, $TS^{\downarrow}$ is the minimum textual relevance between E and $dInt_{\mathcal{S}}$ computed using the minimum weights of the terms in E. Similar to the upper bound estimation, we can prove that the property $\forall q \in Q, STS(E, q) \geq STS^{\downarrow}(E, \mathcal{S})$ always holds.

4.4 Algorithm

We assume that the set of objects O resides on disk and is indexed using an MIR-tree. The goal of the batch top-k processing is to reduce the number of I/O operations by sharing the I/O costs among queries, and accessing the necessary objects and tree nodes only once. This is achieved by: a) a careful tree traversal; and b) an efficient top-k object computation of the individual queries. We utilize $\mathcal{S}$ to access the MIR-tree and share I/O costs, and the relevance bounds to prune nodes that do not contain a top-k object for any query. The pseudocode of the tree traversal step of the SF-GRP approach is shown in Algorithm 2. Finally, the top-k results of the individual queries are verified by applying several pruning strategies as presented in Algorithm 3.

Filtering step: tree traversal. The pseudocode is presented in Algorithm 2. Here, the MIR-tree is traversed for the super-query $\mathcal{S}$ instead of the individual queries. Line 2.2 shows the construction of $\mathcal{S}$ from Q. Initially, a max-priority queue PQ is created (Line 2.3) to keep track of the nodes that are yet to be visited, where the key is the lower bound similarity $STS^{\downarrow}$ w.r.t. $\mathcal{S}$. We also maintain a min-priority queue LO (Line 2.3) to keep the k objects with the best lower bounds found so far. Lines 2.9-2.12 show how LO is initially filled up with k objects according to their lower bounds. We use actual objects instead of object nodes in LO for a better estimation of relevance. We also store the k·th best lower bound relevance score, $\mathfrak{R}_k(\mathcal{S})$ found so far.

Since the score $\mathfrak{R}_k(\mathcal{S})$ is the k·th best lower bound score for any query $q \in Q$, and any unseen object o, the similarity score must be greater than or equal to $\mathfrak{R}_k(\mathcal{S})$ for o to be one of the top-k objects of q. Therefore, we need to consider only those nodes E, where $STS^{\uparrow}(E, \mathcal{S}) \geq \mathfrak{R}_k(\mathcal{S})$. If E is an object satisfying this condition, then LO is adjusted such that it contains k objects with the best lower bounds. The score $\mathfrak{R}_k(\mathcal{S})$ is also updated accordingly. If Obj dequeued from LO in this adjustment process has a better upper bound than the updated $\mathfrak{R}_k(\mathcal{S})$, Obj is stored in a priority queue RO (shown is Lines 2.13-2.18). Here, RO is a max-priority queue where the key is the upper bound similarity score w.r.t. $\mathcal{S}$. If a non-leaf node E cannot be pruned, the entries of E are retrieved from disk and enqueued in PQ as shown in Lines 2.20-2.22. Finally, the objects in LO and RO are used to compute the top-k objects for individual queries in a later step (Line 2.23).

We traverse the MIR-tree according to the lower bound in descending order so that the objects with the best lower bounds will be retrieved early, thereby enabling better pruning. Next, we present an example to demonstrate the procedure of the tree traversal step.

Example 4.3. The objects $O = o_1, o_2, \ldots, o_7$ in Figure 2a are indexed with an MIR-tree as shown in Figure 3. The queries $Q = q_1, q_2, \ldots, q_7$ are shown in Figure 2b, where the dotted box is the MBR

of the queries ($\ell_{\mathcal{S}}$). Table 8 and Table 3 present the text descriptions. Let $k = 1$, the tree traversal step starts by enqueuing the root node R_7 in PQ, and then performing the following steps:

(1) Dequeue R_7, $PQ : (R_6, 0.6), (R_5, 0.3)$
(2) Dequeue R_6, $PQ : (R_4, 0.6), (R_3, 0.5), (R_5, 0.3)$
(3) Dequeue R_4, $PQ : (o_7, 0.7), (R_3, 0.5), (o_6, 0.5), (R_5, 0.3)$
(4) Dequeue o_7, as $|LO| < k$, $LO : o_7$, $\mathfrak{R}_k(\mathcal{S}) = 0.7$
(5) Dequeue R_3, as $STS^{\uparrow}(R_3, \mathcal{S}) = 0.8$, enqueue o_5 in PQ
$PQ : (o_5, 0.7), (o_6, 0.5), (R_5, 0.3)$
(6) Dequeue o_5, as $STS^{\uparrow}(o_5, \mathcal{S}) = 0.8$, enqueue o_5 in RO
$LO : o_7$, $RO : o_5$, $\mathfrak{R}_k(\mathcal{S}) = 0.7$; $PQ : (o_6, 0.5), (R_5, 0.3)$
(7) Dequeue o_6, as $STS^{\uparrow}(o_6, \mathcal{S}) = 0.9$, enqueue o_5 in RO
$LO : o_7$, $RO : o_6, o_5$, $\mathfrak{R}_k(\mathcal{S}) = 0.7$; $PQ : (R_5, 0.3)$
(8) Dequeue R_5, as $STS^{\uparrow}(R_5, \mathcal{S}) = 0.6 < \mathfrak{R}_k(\mathcal{S})$, discard.

ALGORITHM 3: VERIFICATION STEP OF SF-GRP APPROACH($Q, \mathcal{S}$, LO, RO)

```
3.1  Initialize an array H of |Q| min-priority queues for each q ∈ Q.
3.2  for each q ∈ Q do
3.3      for each o ∈ LO do
3.4          ENQUEUE(H_q, o, STS(o, q))
3.5      ℜ_k(q) ← STS(TOP(H_q), q)
3.6      for each o ∈ RO do
3.7          if STS↑(o, S) < ℜ_k(S) then break
3.8          else if STS(o, q) ≥ ℜ_k(q) then
3.9              ENQUEUE(H_q, o, STS(o, q))
3.10             DEQUEUE(H_q)
3.11             ℜ_k(q) ← STS(TOP(H_q), q)
3.12 return H
```

Verification step. In the tree traversal step, the priority queues LO and RO store all the objects that can be a top-k object of at least one query in Q. Therefore, it is sufficient to consider only the objects in LO and RO to obtain the top-k objects for all $q \in Q$. Algorithm 3 summarizes this process. For each $q \in Q$, a min-priority queue H_q of objects is initialized (Line 3.1) where the key is the total relevance score of the object w.r.t. q. For each q, the relevance score between each element $o \in LO$ and q is computed and inserted in H_q. The score of the k·th ranked object of a query q, $\mathfrak{R}_k(q)$ computed so far is also stored (Line 3.2-3.5).

The objects of RO can be pruned in two steps. First, if the upper bound score of an object $o \in RO$ w.r.t. $\mathcal{S}$ is less than $\mathfrak{R}_k(q)$, then o cannot be a top-k object of q. The subsequent objects of o in RO can also be pruned from consideration (Lines 3.7) as RO is maintained in ascending order of the upper bound scores w.r.t. $\mathcal{S}$. Otherwise, if $STS(o, q) \geq \mathfrak{R}_k(q)$, o is inserted into H_q. H_q is adjusted such that it contains k objects with the highest relevance scores, and $\mathfrak{R}_k(q)$ is updated accordingly as shown in Lines 3.9-3.11. Finally, the priority queue H_q for each query $q \in Q$ contains its top-k objects in reverse order, and $\mathfrak{R}_k(q)$ is the spatial-textual similarity score of the k·th ranked object of the corresponding query.

Example 4.4. Continuing the example of the previous step, let us take q_6 for example. Initially, $LO : (o_7, 0.7, 0.9)$, $RO : (o_6, 0.5, 0.9)$, $(o_5, 0.7, 0.8)$, where the entries are presented as $(o_i, STS^{\downarrow}(o, \mathcal{S})$,

Table 10. Notation used in Section 5

Symbol	Description
B	Block size of a posting list in a SIF index
$\boldsymbol{w}^{\uparrow}(O,t)$	Maximum text weight of a term t in a set objects O
$\boldsymbol{w}^{\uparrow}(b,t)$	Maximum text weight of a term t in a block b in a SIF index
MBR_t	Minimum bounding rectangle of the object locations stored in the posting list of a term t
$MBR_{b,t}$	Minimum bounding rectangle of the object locations stored in a block of the posting list of t
$STS_b^{\uparrow}$	Block level upper bound of the spatial-textual similarity
$STS_\ell^{\uparrow}$	Block level upper bound of the spatial-textual similarity using a table of locations
θ	The current k-th best spatial-textual similarity

$STS^{\uparrow}(o,\mathcal{S})$). First, object o_7 from LO is considered. As $STS(o_7,q_6)=0.75$, so, $\mathfrak{R}_k(q_6)$ becomes 0.75 and $H_{q_6}: o_7$. Then the objects in RO are considered. Here, $STS(o_6,q_6)=0.85$, so $\mathfrak{R}_k(q_6)$ becomes 0.85 and $H_{q_6}: o_6$. As the upper bound of the next object of RO, $STS^{\uparrow}(o_5,\mathcal{S}) < \mathfrak{R}_k(q_6)$, we stop processing for q_6. The top-1 object of q_6 is o_6, where $\mathfrak{R}_k(q_6)=0.85$. The process is repeated for all $q \in Q$.

Space complexity. In the SF-GRP approach, the objects and the nodes that can be a top-k of any of the queries are retrieved in a single pass over the MIR-tree using a single priority queue, so the worst case space complexity of the SF-GRP approach is the same as processing a single query using the approach presented by Cong et al. [8].

5 TEXT PRIORITIZATION APPROACH

As discussed in Section 2, index construction is a fundamental problem for spatial-textual data. Instead of augmenting a spatial-only index using textual information like an IR-tree, in this section, we choose to start with a text-only inverted file and augment it using spatial information. Since the WAND (Weak AND) algorithm is a state-of-the-art document-at-a-time, top-k algorithm commonly used in inverted indexes, we will first review how a WAND traversal works. In particular, we first give an overview of WAND in Sec. 5.1, then we illustrate the mapping from textual-inverted-file to our proposed *Spatial Inverted File* (SIF) (Sec. 5.2), and the mapping from a textual based upper bound as used in a WAND traversal to a spatial-textual based upper bound in our case (Sec. 5.3). Table 10 shows the notations used to present our approach in this Section.

5.1 Preliminaries: the WAND Algorithm

Index structure. The index used is a standard text-only inverted file that is adopted in many web search engines. In particular, for each term t, there is a posting list. A posting list of a term t is a sequence of pairs $\langle o, tf_{o,t}\rangle$, where o is the object ID containing t, and $tf_{o,t}$ is the term frequency of t in o.

Background. Broadly speaking, Document-At-A-Time (DAAT) and Term-At-A-Time (TAAT) processing are the two most commonly used traversal techniques [27]. In DAAT, each list has a pointer that points to the "current" posting in the list. A cursor maintaining the current position in each list is moved forward as a query is being processed. In TAAT, the entire inverted list for the query term that is the rarest in the collection is processed first, and then the next rarest term is processed. An accumulator data structure is used to keep track of the highest scoring candidates. When all of the lists have been processed, the final top-k scoring documents are returned.

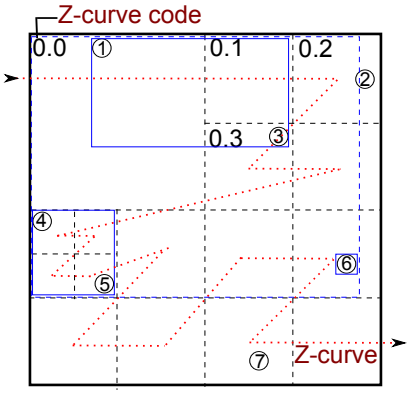


Fig. 4. Z-curve ordering of objects

Previous ID	new ID	Terms and freq.
o_1	o_3	$(t_1,1),(t_2,4)$
o_2	o_2	$(t_4,1)$
o_3	o_6	$(t_1,5),(t_3,5)$
o_4	o_7	$(t_4,2)$
o_5	o_1	$(t_1,4),(t_2,1)$
o_6	o_4	$(t_1,1),(t_3,1)$
o_7	o_5	$(t_1,2),(t_4,3)$

Table 11. Z-order IDs and term frequencies

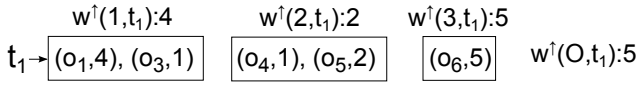


Fig. 5. Posting list for the term 't_1' in SIF index (block size = 2)

While there are advantages and disadvantages to both processing regimes, DAAT processing tends to be favoured in current IR systems because non-textual features can be more easily integrated into the scoring process. For example, the WAND [1] algorithm was proposed to leverage the DAAT traversal process by "skipping" over parts of the posting lists [2, 9, 11]. Therefore, leveraging the intuition behind WAND-like traversals to improve spatial-textual database algorithms is sensible.

We now review how the WAND algorithm can be used to answer individual top-k queries based on text similarity before introducing our solution for batch processing of spatial-textual data later in Sec. 5.4.

Upper bounding index traversal for top-k query processing. In WAND, objects (text documents) are sorted in ascending order of their IDs in each posting list. For each query term, the algorithm maintains a cursor to identify the next candidate document that might need to be scored. In each iteration, the maximum textual similarity score for each posting list is summed in an ascending order of the document ID cursors, until the sum becomes greater than or equal to a threshold, θ. Here, θ is the lowest score of the current top-k documents found so far. The term where this happens is called the "pivot term", and the document ID of the corresponding pointer is called a "pivot".

The crucial observation is that since the pivot is determined using the maximum textual similarity score which represents the *upper bound* score that any query - document pair can achieve, the pivot is the smallest document ID that might be a candidate. Thus, no unscored document with an ID smaller than the current pivot can be a top-k result, and can be safely skipped. As a query is being processed, WAND applies a block skipping pointer movement strategy based on the "pivot". At any point, it is guaranteed that the documents to the left of the pointers have been processed.

5.2 Proposed Index: Spatial Inverted File

Recall the inverted file described in Sec. 5.1. Our proposed *Spatial Inverted File* (SIF) has the same index structure, except that we assign each object an ID based on the location in a space filling curve, and each posting list is augmented with spatial information to facilitate scoring of both components

during traversal. The posting lists are partitioned as blocks of fixed length (typically 64 or 128 postings).

We describe SIF using the example in Figure 4. Figure 4 shows the locations of the same 7 objects $O = \{o_1, o_2, \ldots, o_7\}$ from Figure 2a, where a Space Filling Curve (SFC) is used to assign each object a new object ID corresponding to the position of its location on the SFC. In particular, we use a Z-curve [19] in this paper, highlighted as the red dotted lines in Figure 4. The object IDs are assigned based on their position on this curve. Table 11 shows the mapping of the previous ID with the new ID assigned, and the corresponding text descriptions. The inverted file is constructed using these new IDs where the objects are stored in ascending order of their object IDs in the posting lists. The idea is that if objects are close to each other spatially, they will be closer to each other in the posting list.

In addition, a separate lookup table referred to as *LocTable* is maintained to store the location (latitude and longitude) for each object ID. For each posting list, we maintain the smallest object ID of each block in the same sequence as the posting list in a lookup table, *BlockTable*. The size of the lookup table is negligible when compared to the total size of the inverted file.

For a term t, the posting list is augmented with two pieces of information: (i) the *maximum textual weight* that can be achieved from the postings of that list (the maximum weight of t from the set of objects O), denoted as $\boldsymbol{w}^{\uparrow}(O,t)$; and (ii) the posting-list-level MBR – the minimum bounding rectangle (MBR) that encloses all of the locations for the objects stored in that posting list, denoted as MBR_t. In this paper, the textual similarity score for each term is computed using Equation 3. Similarly, for each block b of the posting list of term t, we maintain (i) the maximum textual weight of t that can be achieved from the objects stored in b, denoted as $\boldsymbol{w}^{\uparrow}(b,t)$; and (ii) the block-level MBR – $MBR_{b,t}$ which is the minimum bounding rectangle that encloses all of the locations for the objects stored in block b.

Example 5.1. Figure 5 shows a sample posting list of term t_1. Let the size of each block be 2. To simplify our illustration, we use the term frequencies instead of the actual weights (computed using Equation 3). In this example, the entries of the *BlockTable* for t_1 are o_1, o_4, and o_6, that are the first object ID of each block. For t_1, its posting-list-level MBR is highlighted by the dotted blue rectangle in Figure 4, and the block-level MBR for each of the three blocks is highlighted by the three blue rectangles of solid borders respectively.

In this paper, the posting lists are stored on disk as a sequence of blocks. The blocks are assumed to be page aligned and can be retrieved from disk individually. The augmented information and the lookup tables are stored separately from the lists and are maintained in main memory.

Space complexity of SIF index. Let the number of entries for a term t_i be M_i. Then the total number of blocks stored on disk is $\sum_i^{|T|} \lceil M_i/B \rceil$, where T is the set of unique terms in the dataset. The augmented information stored in memory consists of the maximum text weight and the MBR for $|\sum_i^{|T|} \lceil M_i/B \rceil|$ blocks and $|T|$ posting lists. The LocTable stores the latitude and longitude of $|O|$ objects in memory.

Comparison with SFC-Skip. Christoforaki et al. proposed an index called SFC-Skip [7] that also reorders object IDs using a Z-curve. The objects are then stored in a blockwise inverted file and the MBRs of each block is maintained. Although their approach is similar in spirit to SIF, the index does not store the textual maximum score for spatial and textual components for each block. Christoforaki et al. also only consider *Boolean Range Queries*, where the inverted file is traversed to find the objects containing all of the query terms, and fall within a fixed distance from the query location. The MBRs are used to skip the blocks if the corresponding MBRs do not intersect with a given query

range, while we use the MBR at the block-level and posting-list-level to quantify the upper bound on the spatial relevance for a top-k KNN query.

To summarize, inverted files are the most widely used index in information retrieval systems, and the SIF methodology can be easily incorporated into existing search platforms for textual document retrieval to support both spatial-textual query and textual query processing using the same index.

5.3 Computing bounds for spatial-textual similarity

Similar to WAND, we also find a pivot object which can help prune objects from the scoring process, so that we can minimize computational costs. Therefore, we now explain how to compute the upper bounds using our spatial-textual similarity formulation before describing the index traversal methods in Section 5.4.

Posting-list-level Upper Bound. We compute an upper bound of the spatial-textual similarity between an object o and a query q using values stored in the posting lists. An upper bound $UB(L,q)$, is computed as

$$STS^{\uparrow}(L,q) = \alpha \cdot SS^{\uparrow}(L,\ell_q) + (1-\alpha) \cdot TS^{\uparrow}(L,d_q)\,, \tag{4}$$

where L is a set of terms such that $L \subseteq d_q$. In Section 5.4.1 and Algorithm 5, we will describe how L is obtained. The upper bound of spatial similarity, $SS^{\uparrow}(L,\ell_q)$ is computed as

$$SS^{\uparrow}(L,\ell_q) = \max_{t \in L} SS(MBR_t,\ell_q)$$

For each term $t \in L$, $SS(MBR_t,\ell_q)$ is computed in the same way as Equation 2:

$$SS(MBR_t,\ell_q) = 1 - \frac{dist(MBR_t,\ell_q)}{d_{max}}\,, \tag{5}$$

where the notations $dist$ and d_{max} carry the same meaning as Equation 2. As $dist(MBR_t,\ell_q)$ is the minimum Euclidean distance between the rectangle MBR_t and the query location, $SS(MBR_t,\ell_q)$ is an upper bound estimation of the spatial similarity for the objects enclosed in MBR_t, with respect to q. The upper bound of text similarity $TS^{\uparrow}(L,d_q)$ is computed as the summation of maximum weights $\boldsymbol{w}^{\uparrow}(O,t)$ contributed by the posting list of each terms $t \in L$. So,

$$TS^{\uparrow}(L,d_q) = \sum_{t \in L} \boldsymbol{w}^{\uparrow}(O,t)$$

Block-level Upper Bound. If the inverted file is stored as a sequence of blocks on disk, we can achieve a tighter upper bound on the similarity by leveraging the values associated with each block of the posting lists.

$$STS_b^{\uparrow}(L,q) = \alpha \cdot SS_b^{\uparrow}(L,\ell_q) + (1-\alpha) \cdot TS_b^{\uparrow}(L,d_q) \tag{6}$$

Here, b is a block in the inverted list of a term $t \in L$ that is currently being traversed, and $L \subseteq d_q$. Let BL be the set of such blocks, one for each term of L. For a block b, $SS(MBR_{b,t},\ell_q)$ is computed from the minimum Euclidean distance between the $MBR_{b,t}$ and ℓ_q using Equation 5. $SS_b^{\uparrow}(L,\ell_q)$ is the maximum spatial similarity contributed by all $b \in BL$. So,

$$SS_b^{\uparrow}(L,\ell_q) = \max_{b \in BL} \{SS(MBR_{b,t},\ell_q)\}$$

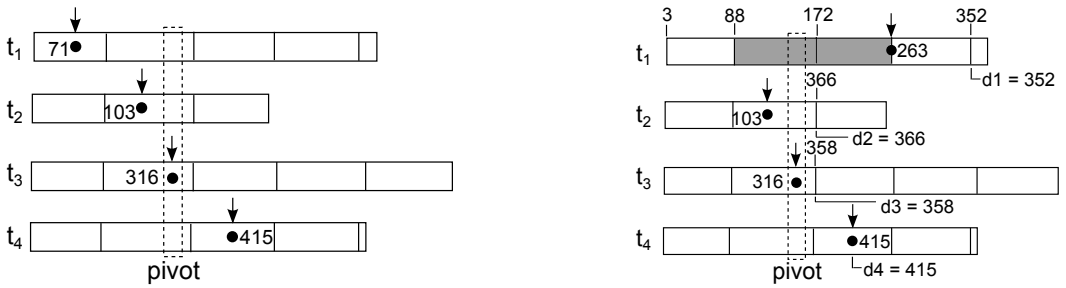


Fig. 6. List traversal for a single query.

The upper bound of text similarity $TS_b^{\uparrow}(L, d_q)$ is computed as the sum of the maximum weights $\boldsymbol{w}^{\uparrow}(b,t)$ contributed by each block $b \in BL$ of the corresponding term $t \in L$. So,

$$TS_b^{\uparrow}(L, d_q) = \sum_{b \in BL} \boldsymbol{w}^{\uparrow}(b,t)$$

Table-based Block-level Upper Bound. As the locations of objects are stored separately in a lookup table *LocTable*, we can also use another upper bound for spatial-textual similarity for an object o, using the actual location of o and the maximum textual similarity scores of the corresponding blocks. Let b be a block of the inverted list of a term $t \in d_q$, where object o is stored. Let BO be the set of such blocks, one for each $t \in d_q$. The bound $STS_{\ell}^{\uparrow}(o, q)$ is computed as

$$STS_{\ell}^{\uparrow}(o, q) = \alpha \cdot SS(\ell_o, \ell_q) + (1 - \alpha) \cdot TS_{\ell}^{\uparrow}(d_o, d_q)$$

where $STS_{\ell}^{\uparrow}(d_o, d_q) = \sum_{b \in BO} \boldsymbol{w}^{\uparrow}(b,t)$. Here, the text component of both $STS_{\ell}^{\uparrow}(o, q)$ and $STS_b^{\uparrow}(L, q)$ are computed in the same way, but $STS_{\ell}^{\uparrow}(o, q)$ uses the exact location of o instead of the MBRs. Therefore, for any object o, the bound $STS_{\ell}^{\uparrow}(o, q)$ is tighter than the corresponding bound $STS_b^{\uparrow}(L, q)$, and inherently tighter than the original BLOCK-MAX WAND approach that uses only the text scores to calculate an upper bound.

5.4 Index Traversal Methods

In this section, we first describe a basic index traversal method using SIF, and show how unnecessary blocks can be skipped using the upper bounds in Section 5.3. Then we present an advanced index traversal method designed to answer multiple queries in a batch.

5.4.1 GEOBW Traversal. Before presenting the algorithm to process a batch of queries, we first introduce our basic index traversal algorithm, GEOBW. In this paper, we use the concept of the BLOCK-MAX WAND algorithm [9], which improves the performance of WAND by enabling skipping blocks of the posting lists. Along with the description of our approach, we also compare it with BLOCK-MAX WAND in different steps of the algorithm. The pseudocode is presented in Algorithm 4.

Given a query q (in form of (ℓ_q, d_q, k_q)), where ℓ_q is a location, d_q is the text description, and k_q is the number of objects to be returned, we maintain a min-priority queue PQ of objects of size at most k_q (Line 4.1). The key is the spatial-textual similarity score with respect to q, $STS(o, q)$, computed using Equation 1. Let θ be the spatial-textual similarity score of the current k-ranked object in PQ.

ALGORITHM 4: *GEOBW*(ℓ_q, d_q, k_q)

```
4.1  Initialize a min-priority queue PQ for the current top-k objects
4.2  θ ← 0
4.3  for each t ∈ d_q do
4.4      curDoc_t ← first(t)
4.5  repeat
4.6      Sort posting lists by curDoc_t
4.7      pivotTerm ← FindPivotTerm(θ, q)
4.8      if pivotTerm = ∅ then Break
4.9      pivot ← curDoc_pivotTerm
4.10     if pivot is the Largest object ID then Break
4.11     for each t ∈ d_q, where curDoc_t < pivot do
4.12         Forward curDoc_t, skip blocks containing only object ID < pivot
4.13     if STS_ℓ^↑(pivot, q) ≥ θ then
4.14         firstT ← First term of the sorted posting lists.
4.15         if curDoc_firstT = pivot then
4.16             /* All the preceding lists contain the pivot object */
4.17             Retrieve blocks pointed by curDoc_t ≤ pivot
4.18             Enqueue(PQ, STS(pivot, q))
4.19             if size(PQ) > k_q then
4.20                 Dequeue(PQ)
4.21             θ ← Top(PQ)
4.22             Move all curDoc_t to the next object ID > pivot.
4.23         else
4.24             Choose term t with the largest IDF, and curDoc_t < pivot. Move curDoc_t to the next object
                   ID ≥ pivot.
4.25     else
4.26         L ← all posting lists where curDoc_t ≤ pivot
4.27         if STS_b^↑(L, q) < θ then
4.28             d' ← object ID for next block
4.29             Choose term t with the largest IDF, where curDoc_t ≤ pivot. Move curDoc_t to the next object
                   ID ≥ d'
4.30         else
4.31             Choose term t with the largest IDF, where curDoc_t ≤ pivot. Move curDoc_t to the next object
                   ID > pivot.
4.32 until stop
4.33 Return PQ
```

Similar to both WAND and BLOCK-MAX WAND, for each query term $t \in d_q$, a pointer to the current posting in the list, $curDoc_t$ is maintained. Each pointer $curDoc_t$ is initialized by pointing to the first element of its corresponding posting list (Lines 4.3-4.4). After initialization, the candidate object with the smallest ID that can be a top-k object, called the *pivot*, is determined. As we need to consider both spatial and textual scores, the computations involved in determining the pivot and skipping in the posting-lists, are different from both WAND and BLOCK-MAX WAND. The steps are demonstrated with the example in Figure 6, where the query terms $d_q = \{t_1, t_2, t_3, t_4\}$. Note that,

ALGORITHM 5: *FindPivotTerm*(θ, q)

```
5.1 L ← ∅
5.2 /* Posting lists are sorted by curDoc_t. */
5.3 for each t ∈ d_q do
5.4     L ← L ∪ t
5.5     if STS↑(L, q) ≥ θ then
5.6         Return t
5.7 Return ∅
```

in this example we use different object IDs and scores than the previous example for the ease of demonstration.

Pivot selection. In each iteration, the posting lists of the query terms are accessed in ascending order of their currently pointed object ID, denoted as $curDoc_t$. An example using Figure 6a is presented below.

Example 5.2. Figure 6a shows the pointer for each of the four (blocked) posting lists, which are arranged in ascending order of their curDoc$_t$. To determine the pivot object, we start computing the $STS^{\uparrow}(L,q)$ for the lists from top to bottom, until we reach a score no less than θ. This computation is shown in Algorithm 5. If no such condition occurs, then the algorithm terminates as no object can be better than the current top-k objects. In Figure 6a, suppose that $STS^{\uparrow}(L,q) \geq \theta$ happens for the third list from the top, the posting list of term t_3. The object ID 316 pointed by the pointer curDoc$_t$ is called the pivot object.

Skipping blocks up to the pivot. According to the WAND algorithm, the pivot object is the object with the smallest ID that can be a top-k object. So, all of the pointers can skip blocks that contain only the objects whose IDs are less than the current pivot. Recall that we maintain a lookup table, *BlockTable*, which maintains the first object ID for each block in the same sequence as they are stored in the posting lists. So the *BlockTable* can be used to skip unused blocks. For example in Figure 6b, the shaded blocks are skipped for the first list. All of the pointers are forwarded accordingly, and now point to blocks where the pivot ID may be found (Lines 4.11-4.12). Here, the pointer of the term t_2 is not forwarded, as the first object ID of the next block is greater than the pivot 316.

Verifying candidates. Unlike WAND, BLOCK-MAX WAND computes a second (and also tighter) upper bound using the maximum impact scores of the blocks to check whether the selected pivot is a valid candidate. In this paper, we adopt a tighter bound than BLOCK-MAX WAND, by using the object locations stored in the *LocTable*. Recall the *LocTable* augmented block-level upper bound introduced at the end of Section 5.3, where o is the pivot object. This can be used to compute the bound $STS_{\ell}^{\uparrow}(pivot,q)$ based on the exact location of the pivot from *LocTable*, and the $\boldsymbol{w}^{\uparrow}(b,t)$ of the blocks for which the current pointers $curDoc_t$ are less than or equal to the pivot. Note that the exact spatial distance between the pivot and q (which we can easily compute) further tightens the overall bound. If $STS_{\ell}^{\uparrow}(pivot,q) \geq \theta$, and all the lists above the pivot term contain the pivot object as well, then we need to compute the exact score between the pivot object and the query (Lines 4.13-4.15). As shown in Lines 4.16-4.17, the corresponding blocks are retrieved from disk, and the score of the pivot object, $STS(pivot,q)$, is computed using Equation 1. The priority queue PQ and the value θ is updated accordingly (Lines 4.18-4.21). All the pointers with $curDoc_t \leq$ pivot are moved to the next object with ID greater than this pivot (Line 4.22).

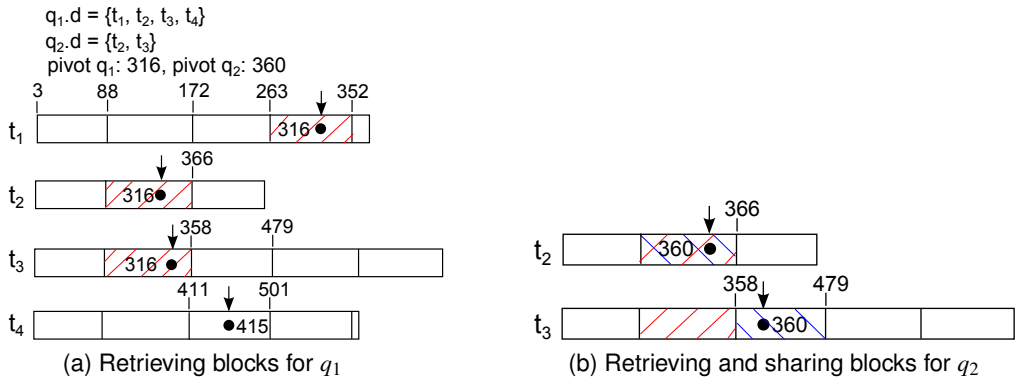


Fig. 7. List Traversal for multiple queries.

If any of the lists above the pivot term does not contain the pivot ID, a pointer above the pivot term is selected and moved to the next smallest ID greater than pivot (Lines 4.23-4.24). Selecting the pointer of the term with the highest IDF is shown in [1] to have a better gain in skipping. If $STS_{\ell}^{\uparrow}(pivot, q) < \theta$, then the pivot cannot be a top-k object. So we need to set the pointers to the first object with an ID greater than the pivot.

Skipping beyond the pivot. Unlike WAND, we can now compute $STS_{b}^{\uparrow}(L, q)$ using the MBRs and the $\boldsymbol{w}^{\uparrow}(b,t)$ for the set of corresponding blocks. If the condition $STS_{b}^{\uparrow}(L, q) < \theta$ is true, we can skip beyond the end of one of the current blocks since the current pivot was discarded based on the sum of the maximum scores of the blocks currently being evaluated. This concept is similar to BLOCK-MAX WAND, where the skipping is improved when compared to moving the pointers to the next object. We illustrate this skipping (Lines 4.27-4.29) in the following example.

Example 5.3. In Figure 6b, suppose that after computing $STS_{b}^{\uparrow}(L, q)$ using the maximum values of the blocks, we find that pivot 316 cannot be a top-k object. Let d_1 be the first object ID of the successor block of t_1. Similarly, the other block boundaries for the posting lists appearing before the pivot is determined. Here, d_4 is the current object ID in the fourth list. Pointers in postings lists beyond the pivot match the document cursor for that list as they are already greater than or equal to the current pivot ID. According to BLOCK-MAX WAND, we can safely skip the objects with an ID less than $d' = \min(d_1, d_2, d_3, d_4)$, which is d_1 in this case.

Recall from Section 5.3 that $STS_{\ell}^{\uparrow}(pivot, q)$ is a tighter upper bound than $STS_{b}^{\uparrow}(L, q)$ for the pivot object, so $STS_{b}^{\uparrow}(L, q) \geq STS_{\ell}^{\uparrow}(pivot, q) \geq STS(pivot, q)$. So, if $STS_{\ell}^{\uparrow}(pivot, q) \geq \theta$ holds, then $STS_{b}^{\uparrow}(L, q) \geq \theta$ also holds. Therefore, we do not need to repeat this check to determine whether we need to compute the score of the pivot object. The process terminates when either (i) no object is left to be processed that can be a top-k object based on the upper bound (Line 4.8); or (ii) all of the objects up to the maximum object ID in the dataset have been either processed or skipped (Line 4.10).

5.4.2 TF-MBW: Multiple-Query Traversal. Given a set of queries Q, our goal is to minimize the I/O cost where the queries share keywords and/or have close locations spatially. Algorithm 6 shows the pseudocode of our proposed approach, TF-MBW, to answer multiple queries as a batch.

ALGORITHM 6: *TF-MBW(Q)*

6.1 Initialize an array PQ of $|Q|$ min-priority queues
6.2 $TU \leftarrow \bigcup_{q \in Q} d_q$
6.3 Initialize an array *BlockBuffer* of TU blocks
6.4 **for** *each* $q \in Q$ **do**
6.5 Execute Lines 4.2-4.4 of Algorithm 4
6.6 Sort posting lists by $curDoc_t[q]$
6.7 $pivotTerm[q] \leftarrow$ FindPivotTerm($\theta[q], q$)
6.8 **if** $pivotTerm[q] = \emptyset$ **then**
6.9 $Q \leftarrow Q \backslash q$
6.10 $pivot[q] \leftarrow curDoc_{pivotTerm}[q]$
6.11 **if** $pivot[q] < ID_{max}$ **then**
6.12 $Q \leftarrow Q \backslash q$
6.13 **while** $Q \neq \emptyset$ **do**
6.14 $pivot_{min} \leftarrow \min_{q \in Q}(pivot[q])$
6.15 $q_{min} \leftarrow$ the query for which minpivot is selected.
6.16 Execute Lines 4.11-4.16 of Algorithm 4 for q_{min}
6.17 **for** *each* $curDoc_t[q_{min}] \leq pivot[q_{min}]$ **do**
6.18 **if** *Block pointed by* $curDoc_t[q_{min}]$ *NOT retrieved before* **then**
6.19 $b \leftarrow$ Retrieve block pointed by $curDoc_t[q_{min}]$
6.20 Mark b as retrieved
6.21 $BlockBuffer[t] \leftarrow b$
6.22 Execute Lines 4.18-4.29 of Algorithm 4 for q_{min}
6.23 Execute Lines 6.6-6.12 of Algorithm 6 for q_{min}
6.24 Return PQ

For each query $q \in Q$, we maintain a separate priority queue PQ_q that stores the current top-k objects of that query (Line 6.1).

Let TU be the union of the terms of all the queries in Q (Line 6.2). We maintain a buffer of size $|TU|$, denoted as *BlockBuffer* to keep the last accessed block for each term $t \in TU$ by any query, or more specifically, only one block for each $t \in TU$. For each query $q \in Q$, the spatial-textual similarity score $\theta[q]$ of the currently k-ranked object, and the pointers $curDoc_t[q]$ are also maintained (Line 6.5). We initialize the pivot object for each individual query $pivot[q]$ once, in the same way as described in Section 5.4.1. If we reach the termination condition for a query q', there is no object left that can be a kNN of that query, so we can exclude it from Q (Lines 6.6-6.12).

Let $pivot_{min}$ be the smallest pivot ID among the current pivots for any of the queries $q \in Q$, and q_{min} be the corresponding query for which $pivot_{min}$ is selected. In each iteration, we take $pivot_{min}$ and process the query q_{min} for that pivot (Lines 6.14- 6.16). While retrieving any block that is required to compute the total score of $pivot_{min}$, one of two conditions can occur: (i) the block that contains the current $pivot_{min}$ was retrieved in a previous iteration; (ii) the block was never retrieved in any prior step.

If condition (ii) holds, the block must be retrieved from disk, do the computation, and then keep the block in *BlockBuffer* for the corresponding term. If condition (i) holds, then the block must be in the corresponding block buffer. This is true since WAND guarantees that the pivot selected in each step is always greater than or equal to all of the pivot IDs selected in the previous step. For multiple

queries, as we process the minimum of all the pivot IDs in each iteration, the $pivot_{min}$ ID of a step is also greater than or equal to the $pivot_{min}$ ID of any previous step. Thus, the objects less than the $pivot_{min}$ ID are guaranteed to be processed already for all of the queries in each step.

Recall that the blocks are stored in a sorted order by object ID, and the objects are sorted in the blocks as well. So if a block containing the current $pivot_{min}$ is retrieved for any prior $pivot_{min}$, that block is guaranteed to be found in the block buffer in this approach. As we maintain the pointers of the terms for all of the individual queries, forwarding the pointers is achieved in the same way for q_{min} as described for GEOBW (Line 6.22). The pivot of the q_{min} is computed again (Line 3.23). We now illustrate I/O sharing among queries with the following example.

Example 5.4. Figure 7 shows two queries q_1 and q_2, where $d_1 = \{t_1, t_2, t_3, t_4\}$ and $d_2 = \{t_2, t_3\}$. Let the starting pivot of q_1 be 316 and the pivot of q_2 be 360 in this example. In this case, we take 316 as the $\text{pivot}_{\text{min}}$ and process 316 for q_1. Suppose we need to compute the score for object 316 for q_1, then the blocks shown with red stripes (Figure 7a) are retrieved from disk to compute this score, and these blocks are stored in the block buffer for the corresponding terms.

The pivot of q_1 is computed again. Let, the $\text{pivot}_{\text{min}}$ selected in the next iteration be 360 for q_2. After checking the conditions, suppose now we find that we need to compute the score of 360 for q_2. The blocks that need to be accessed for q_2 are shown with blue stripes in Figure 7b. As the block for the term t_2 was retrieved for q_1 previously, that block can be found in the BlockBuffer for t_2. Thus the block shown with two color stripes are shared among the queries. The block stored in BlockBuffer for t_3 is not the one that is required by q_2. therefore, we need to retrieve this block, and update BlockBuffer for t_3.

If a block is skipped for all pointers in an inverted list where multiple queries share the same term, that block is not retrieved from disk. The priority queues for each $q \in Q$, and the thresholds θ_q are also updated in this process. If we reach the terminating condition for a query q' such that there is no object left that can be a kNN of the query, we exclude it from Q and traverse the list of the terms in $TU' = \cup_{q \in Q \setminus q'} d_q$. We continue until Q is empty, which indicates that the result for all the queries have been found.

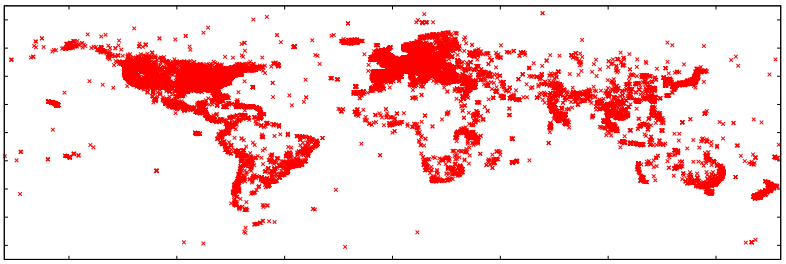
Space complexity. In the TF-MBW approach using the SIF index, a separate priority queue, each of size k is maintained for each query to track the current best objects, totalling $k \times |Q|$ objects for the batch. We also maintain a block buffer to store the most recent retrieved block for each unique query term, so in total $|dUni_S|$ blocks are kept in memory simultaneously.

6 EXPERIMENTAL EVALUATION

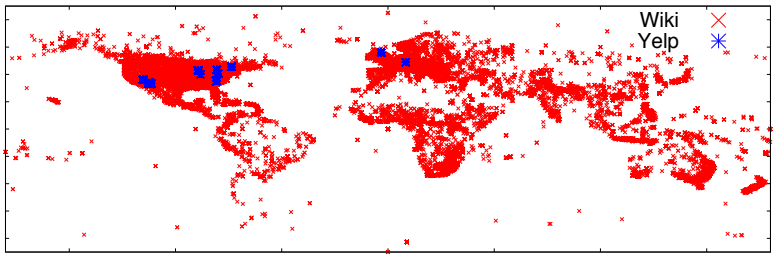
In this section, we present the experimental evaluation for our three proposed algorithms to process multiple spatial-textual queries in a batch: (i) Space-first by maintaining separate priority queues of the queries (SF-SEP), (ii) Space-first by grouping queries (SF-GRP), and (iii) Text-first traversal on the inverted lists TF-MBW. We also compare our approach with an two unbatched baselines, where each query is processed individually using (i) an IR-tree according to the approach described by Cong et al. [8], denoted as the *space-first baseline* (SF-BL) and (ii) a SIF index presented in 5.4, denoted as the *text-first baseline* (TF-BL). We compare the query performance of the algorithms, assuming that a dataset is static throughout the experiment.

Datasets and query generation. All experiments are conducted on three real datasets, (i) Flickr dataset [1], (ii) Wiki dataset, and (iii) Yelp dataset [2].

[1] http://webscope.sandbox.yahoo.com/catalog.php?datatype=i&did=67
[2] http://www.yelp.com.au/dataset_challenge



(a) Flickr dataset



(b) Wiki and Yelp dataset

Fig. 8. Location of the Data objects

For the Flickr dataset, a total of 1 million images that are geotagged and contain at least one user specified tag were extracted from the collection. The locations and tags are used as the location and keywords of the objects. The Wiki dataset was obtained from the authors of [17]. They have generated the dataset from a subset of the TREC[3] ClueWeb09B collection. The documents are geotagged using the Freebase annotations of the ClueWeb Corpora[4]. Finally, the first 1 million documents from the collection, ordered by the spam-score, were selected. This dataset is referred to as Wiki since a large number of Wikipedia articles are in the collection as a result of their low spam score. The Yelp dataset contains information about businesses in 10 cities. For each business, three types of information are available: business location, business attributes, and user reviews on businesses. The attributes and reviews for each business are combined as the text description of that business. Table 12 lists the properties of the datasets. Figure 8 shows the location distribution of the objects, where the red and the blue points in Figure 8b represent the Wiki and the Yelp dataset, respectively. Note that, the distribution of the Flickr and the Wiki datasets are very similar, where for the Yelp dataset, the objects shown with blue points are highly clustered in 10 large cities of USA and Europe.

We used the above datasets to generate the set of queries in the following way. First, an area of a percentage of the dataspace size (here, 4%) was chosen, and the number of queries in a batch, $|Q|$) of objects O_q in that area are taken randomly. The locations of the objects were used as the locations of the queries. Then, QW keywords were randomly selected from O_q as the set of the query keywords. QW was the number of unique query keywords in a batch. These keywords were distributed among the queries such that each query had $|QL|$ number of keywords following the same

[3]https://www.trec.nist.gov
[4]https://lemurproject.org/clueweb09/FACC1

Table 12. Description of dataset

Property	Flickr	Wiki	Yelp
Total objects	1,000,000	1,000,000	61,185
Total unique terms	166,317	2,530,440	266,869
Avg unique terms per object	6.9	269.5	398.7
Total terms in dataset	6,936,385	518,243,837	77,838,026

distribution of keywords of O_q. In this work, we generated 50 such sets of queries and reported the average performance.

Setup. All indexes and algorithms were implemented in Java. The experiments were ran on a 24 core Intel Xeon $E5-2630$ running at 2.3 GHz using 256 GB of RAM, and 1TB 6G SAS 7.2K rpm SFF (2.5-inch) SC Midline disk drives running Red Hat Enterprise Linux Server release 7.2 (Maipo). The Java Virtual Machine Heap size was set to 4 GB. All index structures are disk resident. The number of postings was set to 128 for the inverted lists of the SIF index, the inverted lists associated with the nodes of the IR-tree and the MIR-tree. The page size was fixed at 1 kB for all indexes.

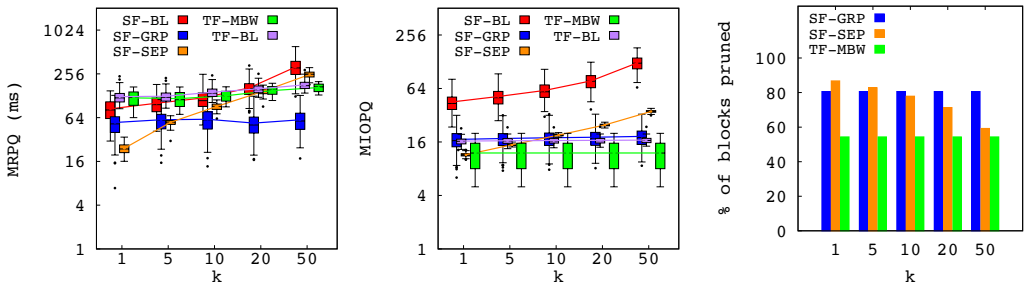
As multiple layers of cache existed between a Java application and the physical disk, we report simulated I/O costs in the experiments instead of physical disk I/O costs. The number of simulated I/O operations is increased by 1 when a node of a tree is visited. When an inverted list is loaded, the number of simulated I/O operations is increased by the number of blocks contained in the list. In the experiments, the performance was evaluated using cold-start queries.

6.1 Performance evaluation

Table 13. Parameters

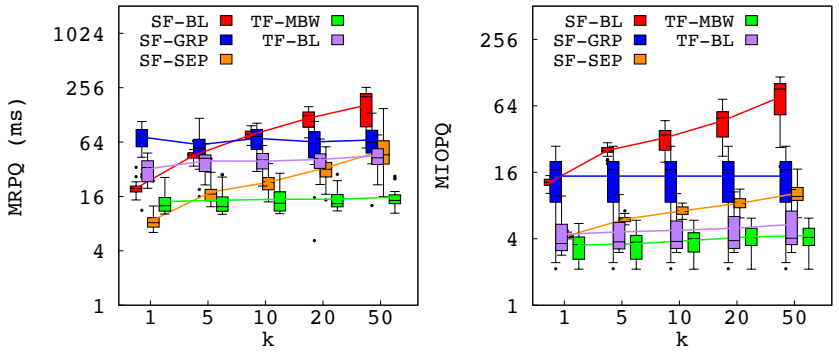
Parameter	Range
k	$1, 5, \mathbf{10}, 20, 50$
α	$0.1, 0.3, \mathbf{0.5}, 0.7, 0.9$
No. of keywords per query, QL	$1, 2, \mathbf{3}, 4, 5, 6$
No. of unique query terms in batch, QW	$5, 10, \mathbf{20}, 30, 40$
MBR of batch as % of dataspace, $Area$	$1, 2, \mathbf{4}, 8, 16$
No. of Queries in a batch, $\|Q\|$	$\mathbf{100}, 200, 400, 800, 1600$

In this section, we evaluate and compare the performance of the approaches by varying several parameters. The parameter ranges are listed in Table 13 where the values in bold represent the default values. In all experiments, we vary a single parameter (while keeping the rest as the default settings) to study the impact on: (i) the Mean Runtime per Query (MRPQ), (ii) the Mean I/O cost per Query (MIOPQ) to compute the top-k objects of all the queries within a batch. We also show the sensitivity of each approach when varying a parameter, by measuring the number of blocks pruned as a percentage of the total number of "relevant" blocks of the batch. Here, the relevant blocks include the blocks of the inverted lists that store the query terms, and the blocks of the tree such that at least one object stored in that subtree contains at least one of the terms. Note that different indexes have a different number of total relevant blocks, and the number varies for different sets of query terms in a batch. The scalability of the algorithms is evaluated by varying the number of queries in a batch,



(a) Runtime (b) I/O cost (c) % of blocks pruned

Fig. 9. Effect of varying k for Flickr dataset

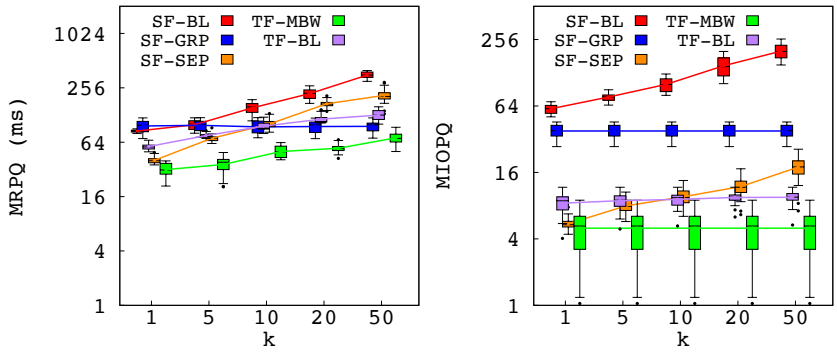


(a) Runtime (b) I/O cost

Fig. 10. Effect of varying k for Yelp dataset

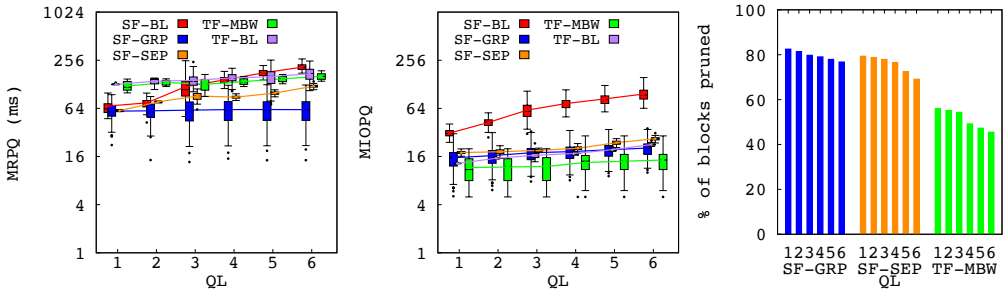
and by reporting (i) the Total Runtime, and (ii) the Total I/O cost for the batch, instead of the mean values. Furthermore, we also evaluate the space requirements and efficiency trade-offs for all of the approaches.

Varying k. The experimental results when varying the top-k are shown in Figure 9, Figure 10, and Figure 11 for the Flickr, Yelp and Wiki dataset, respectively. Here, the major observations are: (1) The costs incurred by the SF-GRP method do not vary much as k increases. Because the queries share the I/O costs, the objects required for more top-k results for a query are often retrieved as results from the other queries. (2) The I/O cost of TF-MBW is less than the other three approaches for all three datasets. The reason is that in the spatial-first approaches, I/O costs are incurred for both the nodes of the tree (the spatial component), and the inverted files associated with each of those nodes. In contrast, there is only one inverted file for TF-MBW, and the necessary blocks of the query terms are retrieved. (3) The MRPQ of TF-MBW is the best among the approaches for the Wiki dataset, worse for the Flickr dataset, and gradually becomes better as k increases for Yelp dataset. The text intensive approach TF-MBW relies on the variation of the scores of the objects to



(a) Runtime (b) I/O cost

Fig. 11. Effect of varying k for Wiki dataset

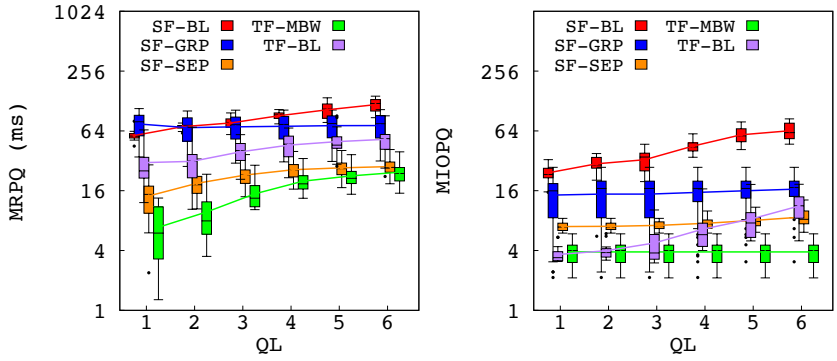


(a) Runtime (b) I/O cost (c) % of blocks pruned

Fig. 12. Effect of varying QL for Flickr dataset

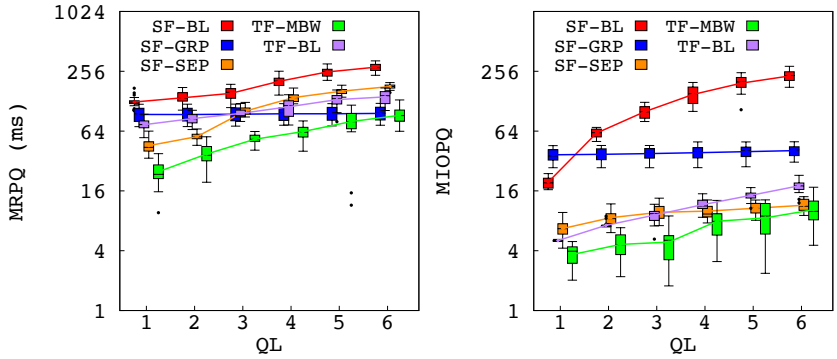
improve the pruning effect. If many objects have a similarity score close to the threshold θ, they will become pivot objects, and be scored even if there is little or no change to the current top-k candidate set. In Flickr, as the number of terms per object is small and the term frequencies do not vary much, TF-MBW does not have much pruning power. As the objects in the Wiki and the Yelp datasets have much more text, greater score differences are seen, and dynamic pruning improves. (4) The text-first baseline, TF-BL is a more competitive baseline than the SF-BL for text intensive datasets. In fact, in the Yelp dataset, where the number of terms per object is much higher than the other two datasets, the TF-BL approach consistently outperforms the SF-GRP approach. One possible reason is that many unnecessary objects may seem promising for the union of the query terms when using the SF-GRP approach.

Varying *QL*. We now vary the number of keywords per query (QL), and present its impact on performance in Figure 12, Figure 13, and Figure 14 for the Flickr, Yelp and the Wiki dataset, respectively. The main observations are: (1) The cost of the baseline increases proportionally with the increase of QL, as more objects become relevant to each query. (2) The I/O cost of the SF-GRP



(a) Runtime (b) I/O cost

Fig. 13. Effect of varying QL for Yelp dataset

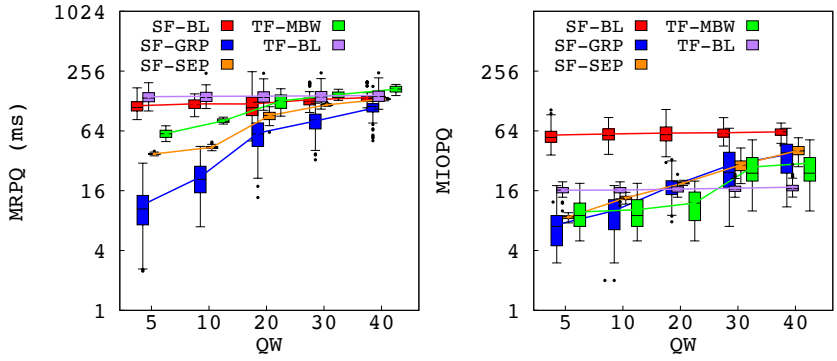


(a) Runtime (b) I/O cost

Fig. 14. Effect of varying QL for Wiki dataset

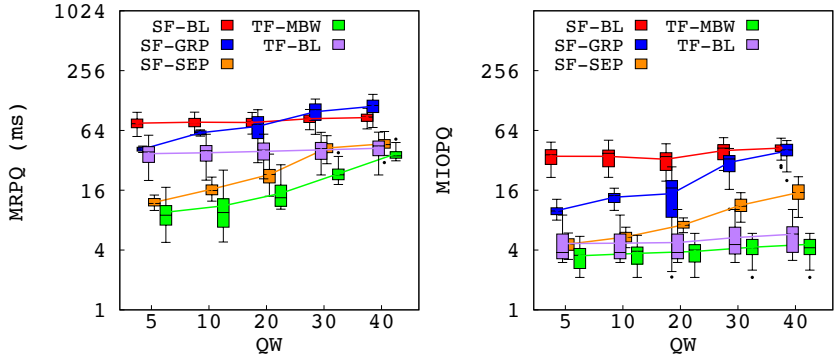
method, where the queries are grouped together as a super-query, remains almost constant. The reason is that, although QL increases, the total number of unique keywords in the group (QW) remains constant, so the number of objects retrieved in the filtering step remains almost the same as well. (3) As more objects become relevant to the queries with the increasing number of keywords per query, the costs of SF-SEP and TF-MBW increase with QL. As these methods share the I/O costs, the increase rate of the costs are much lower than the baseline.

In the text intensive datasets (Yelp and Wiki), more objects contain the query keywords, and must be retrieved than in the Flickr dataset. Moreover, SF-GRP uses the MBR and the union of the query keywords to access the index, which may retrieve unnecessary objects that do not actually score high w.r.t. the query. Therefore, although SF-GRP performs the best in terms of MRPQ for the Flickr dataset, it performs worse than TF-MBW in the Yelp and Wiki datasets. As SF-SEP maintains a priority queue of the relevant nodes for each query individually, only the nodes that are required by



(a) Runtime (b) I/O cost

Fig. 15. Effect of varying QW for Flickr dataset

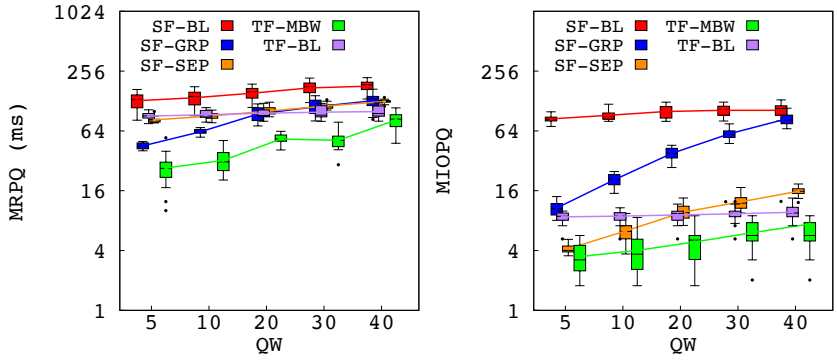


(a) Runtime (b) I/O cost

Fig. 16. Effect of varying QW for Yelp dataset

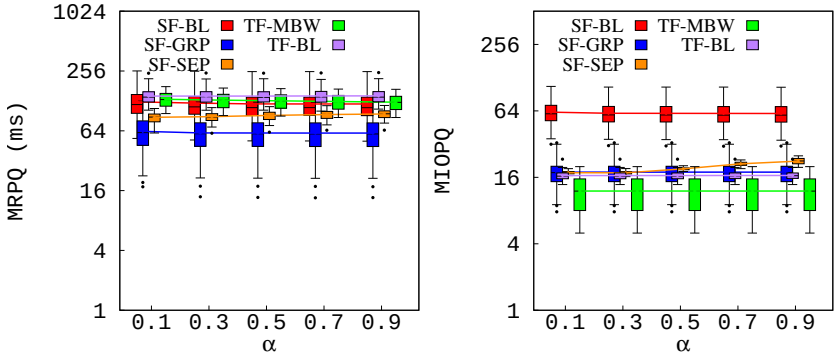
any of the individual queries are retrieved. Therefore, SF-SEP has better performance than SF-GRP in the text intensive datasets.

Varying *QW*. Figure 15, Figure 16, and Figure 17 show the effect on performance when varying the total number of unique keywords, QW, processed in a single batch. Here, a lower value indicates that the queries share more keywords. Although all approaches exploit shared I/O costs across all of the queries, SF-GRP is the best performing algorithm for Flickr, and TF-MBW is better for the Yelp and Wiki dataset. In the text-intensive datasets, the trees (IR-tree and MIR-tree) store an inverted file for each node of the tree. If a node must be retrieved, both the spatial node and the corresponding posting lists for the query must be retrieved. If a node is not shared among queries, multiple blocks must be retrieved. In contrast, a single inverted file is maintained for TF-MBW, and if a block is not shared among queries, only that block must be retrieved. Therefore, the MIOPQ for TF-MBW increases little when compared to the two spatial-first approaches.



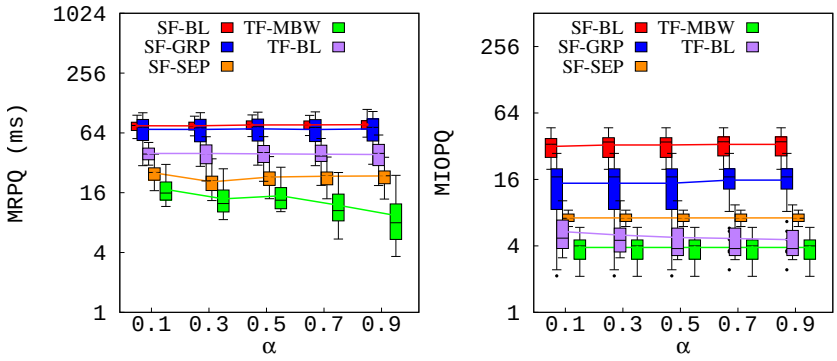
(a) Runtime (b) I/O cost

Fig. 17. Effect of varying QW for Wiki dataset



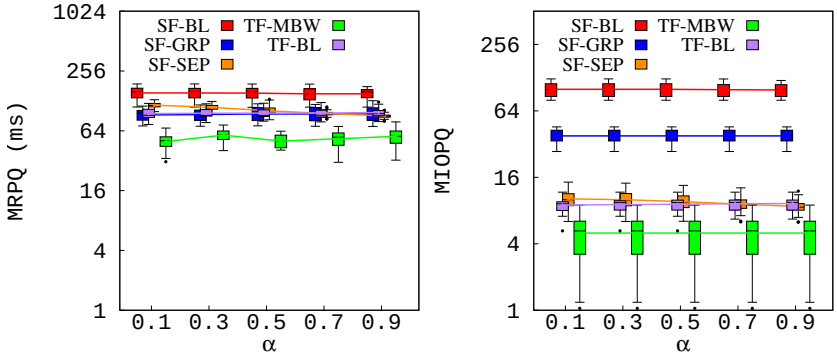
(a) Runtime (b) I/O cost

Fig. 18. Effect of varying α for Flickr dataset



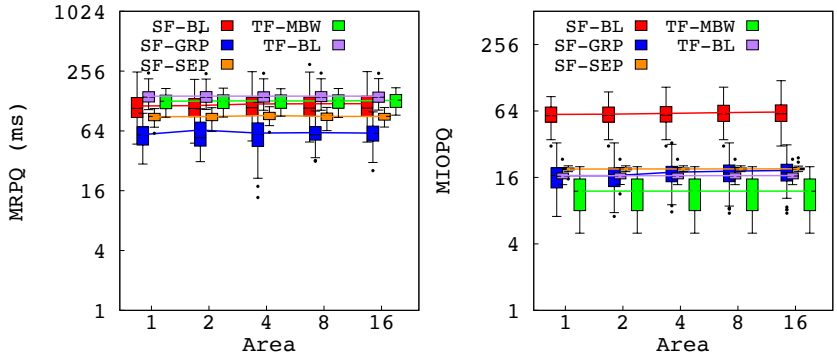
(a) Runtime (b) I/O cost

Fig. 19. Effect of varying α for Yelp dataset



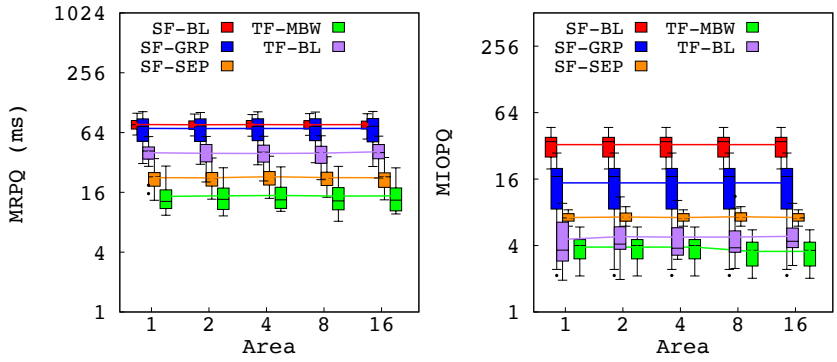
(a) Runtime (b) I/O cost

Fig. 20. Effect of varying α for Wiki dataset



(a) Runtime (b) I/O cost

Fig. 21. Effect of varying *Area* for Flickr dataset



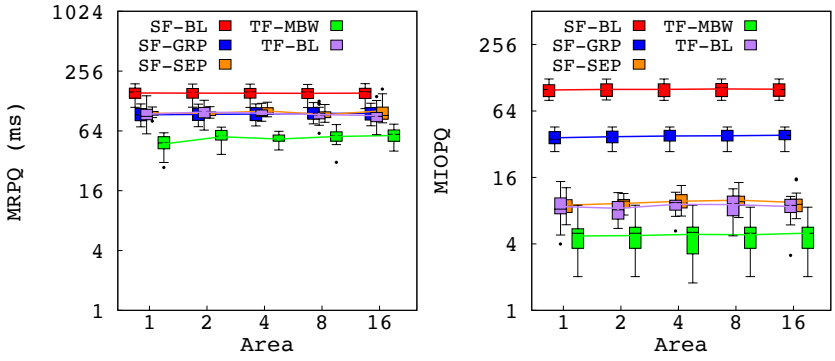
(a) Runtime (b) I/O cost

Fig. 22. Effect of varying *Area* for Yelp dataset

Varying α. Figure 18, Figure 19, and Figure 20 show the results when varying α on each of the three datasets, where a higher value indicates more preference towards spatial similarity. The costs of the spatial-first approaches and the baselines do not vary much with respect to α. The cost of the text-first approach TF-MBW do not vary with α as well, as in each posting list, objects are organized to preserve spatial locality.

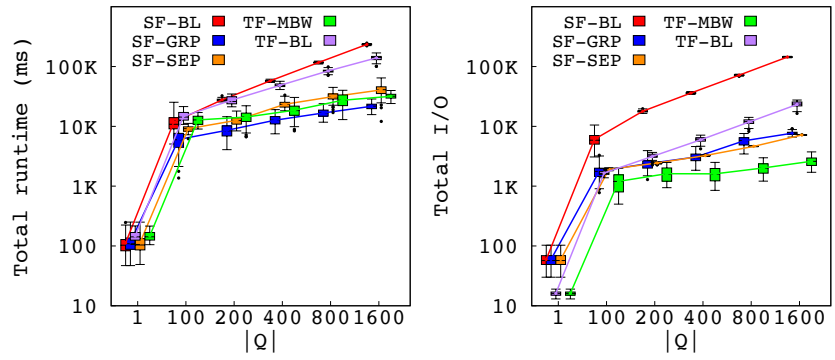
Varying *Area*. Figure 21, Figure 22, and Figure 23 show the impact when varying the area covered by the MBR of the query locations, presented as a percentage of the total dataspace. A higher value indicates that the locations of the queries are more sparse. The performance for all of the methods vary little with respect to the area, as the total number of unique query terms does not change.

Scalability. In one set of experiments, we vary the number of queries $|Q|$ in a batch by keeping the other parameters fixed at their default values, and in another set of experiments we vary both



(a) Runtime (b) I/O cost

Fig. 23. Effect of varying *Area* for Wiki dataset

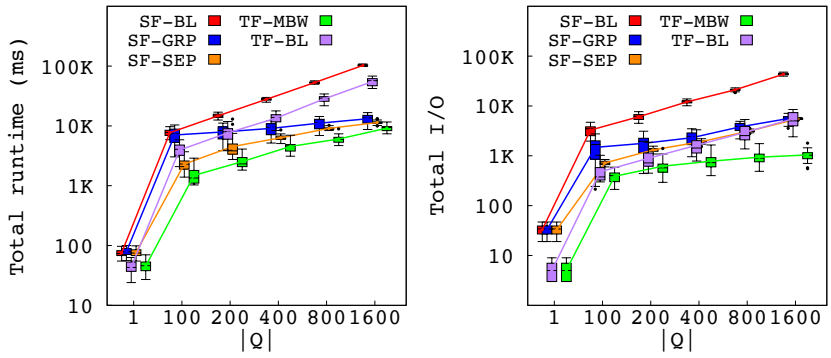


(a) Runtime (b) I/O cost

Fig. 24. Effect of varying $|Q|$ for Flickr dataset

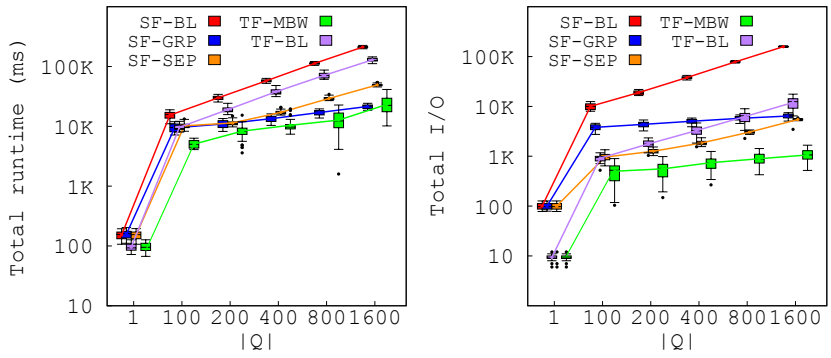
$|QW|$ and the number of unique keywords in a batch, QW in order to evaluate scalability. Figure 24, Figure 25, and Figure 26 show the effect when varying $|Q|$ from 1 to 1600 on the total runtime and the I/O cost of processing a batch of queries for Flickr, Yelp, and Wiki dataset, respectively. As the number of queries increases, the cost of the baseline increases proportionally as it processes the queries one by one. As the I/Os are shared among the queries, the advantage of our proposed approaches becomes more prominent when the number of queries increases. As TF-MBW benefits from sharing only I/O costs for a single inverted file, the total I/O cost of TF-MBW is the lowest among the approaches for all datasets.

When the number of queries in a batch is 1, the I/O cost of the SF-SEP is exactly the same as in SF-BL, as the additional step of sharing retrieved objects in SF-SEP is not required for a single query. Similarly, the cost of the TF-MBW and the TF-BL are the same, as the minimum pivot ID for the queries are not required to be determined for $|Q| = 1$. The cost of the SF-GRP approach is slightly higher than the cost of SF-BL for $|Q| = 1$. The reasons are: (i) Although the SF-GRP approach



(a) Runtime (b) I/O cost

Fig. 25. Effect of varying $|Q|$ for Yelp dataset

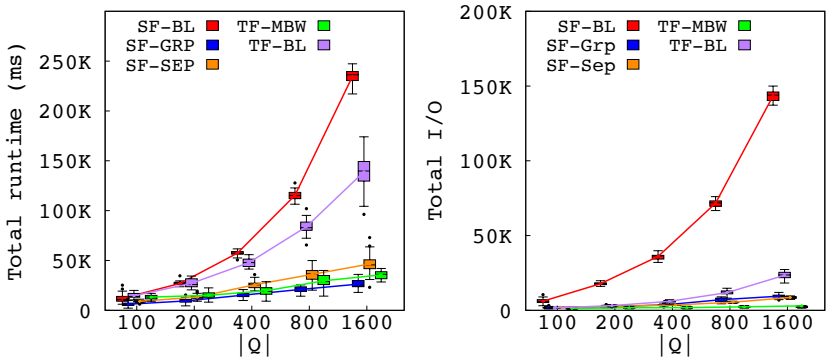


(a) Runtime (b) I/O cost

Fig. 26. Effect of varying $|Q|$ for Wiki dataset

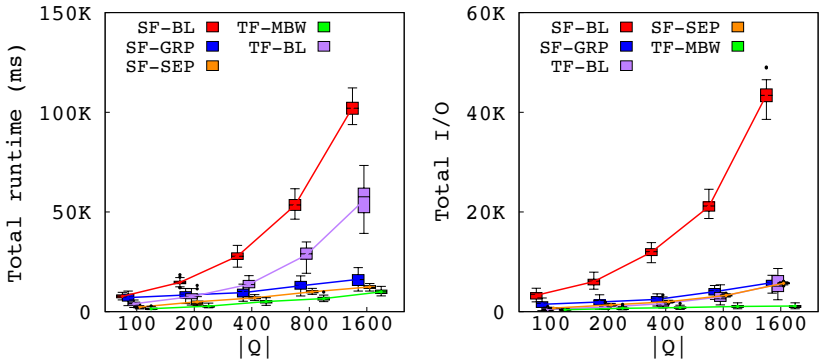
calculates both an upper and a lower bound of relevance, for a single query both bounds are equal. Thus the SF-GRP approach does not provide any additional pruning over SF-BL, but calculates the same value twice. (ii) The SF-GRP approach utilizes the MIR-tree, where both the minimum and maximum text similarity scores are stored, in contrast to the IR-tree, where only the maximum score is stored. Therefore, although the SF-GRP approach outperforms the baseline SF-BL for a higher number of queries in a batch, a higher number of pages are retrieved by SF-GRP to process a single query individually.

Figure 27, Figure 28, and Figure 29 shows the performance for varying both $|Q|$ and QW of the batch of queries. Here, we increase the number of unique keywords in a batch (QW) by 10% at each level of increment of $|Q|$. The performance of the two baselines are similar to the prior case, as increasing QW in a batch do not have much affect on the processing of the queries individually. As a higher QW indicates lower shared keywords among a batch, some additional costs incur for the other approaches than that of varying only $|Q|$.



(a) Runtime (b) I/O cost

Fig. 27. Effect of varying both $|Q|$ and QW for Flickr dataset



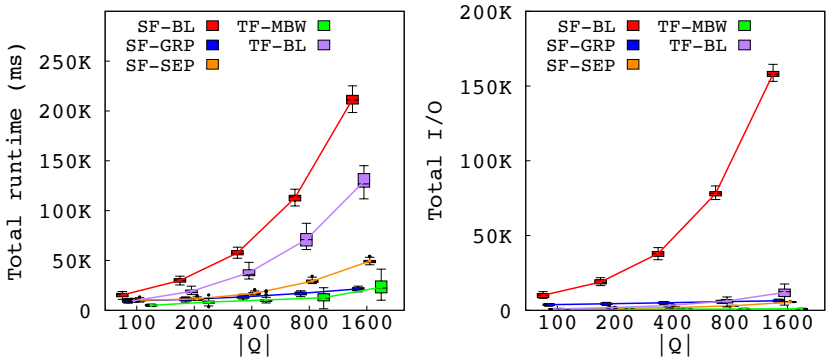
(a) Runtime (b) I/O cost

Fig. 28. Effect of varying both $|Q|$ and QW for Yelp dataset

Table 14. The size of the lookup tables in SIF

Dataset	LocTable	BlockTable
Flickr	31 MB	170 MB
Wiki	31 MB	735 MB
Yelp	1.9 MB	73 MB

Space vs. time efficiency trade-offs. Figure 30 shows the tradeoff between index size of the datasets and the mean runtime per query in the default settings for all approaches. Table 14 shows the size of the lookup tables in SIF index. The index construction time for each index is presented in Table 15. The space-first baseline is omitted as this baseline is consistently outperformed by the other approaches. Clearly, the space-first approaches have a significant space overhead, as we need to



(a) Runtime (b) I/O cost

Fig. 29. Effect of varying both $|Q|$ and QW for Wiki dataset

Table 15. Index construction time in minutes

Index	Flickr	Wiki	Yelp
IR-tree	20	41	15
MIR-tree	28	55	19
TF-MBW	13	35	10

store a separate inverted file for each of the nodes of the tree. The MIR-tree requires more space than the IR-tree, to store both the minimum and the maximum value of the precomputed text scores of each term in each document (or pseudo-document). The advantages of the text-first batch processing techniques become more prominent in both the Wiki and Yelp datasets. The TF-MBW approach benefits from using a single inverted file, as the total number of blocks storing relevant objects is much less than in the space-first structure, and the benefits increase for the datasets with a higher number of keywords.

7 CONCLUSION

In this paper, we have studied how to efficiently answer multiple top-k spatial-textual queries as a batch. In particular, we have carefully studied existing work on top-k spatial-textual queries, and proposed (i) a new traversal method, SF-SEP for batch processing over an existing widely adopted index; (ii) a new index structure, the MIR-tree and a novel traversal approach, SF-GRP based on a query grouping technique. Both of these approaches have adopted a space-first strategy to construct the index, which may have two drawbacks: (1) the index size can be impractically large, (2) the index may not be easily integrated into an existing web search engine architecture. Therefore, we have designed a text-first index, the SIF, which augments a standard inverted file with spatial information. We have proposed a novel block-wise traversal technique in SIF to process multiple top-k spatial textual queries. In all of our proposed methods, the goal is to improve the overall efficiency by sharing the processing and I/O costs of the queries, and avoiding multiple retrievals of the same data. Our approaches are amenable to queries which share a large number of keywords, and/or are in close proximity to each other. Through extensive experiments using three publicly available

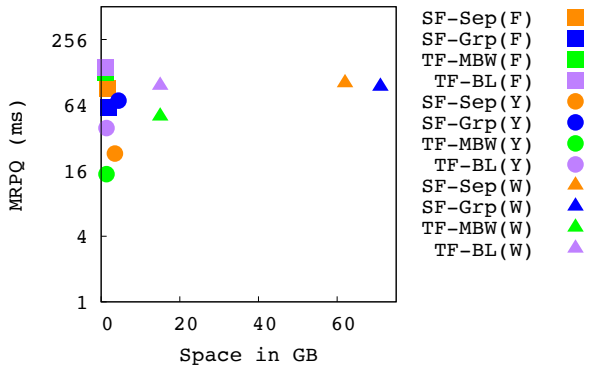


Fig. 30. Space-time efficiency trade-offs for all indexing methods

datasets, we compare the performance and the space requirements of our approaches for different settings. In summary, we find that the text-first approach has significantly lower space requirement and demonstrates better pruning for the text-intensive datasets. However, space-first approaches have better performance for the datasets with few keywords per object, but at the cost of larger index storage requirements.

ACKNOWLEDGMENTS

This work was supported by the Australian Research Council's *Discovery Projects* Scheme (DP140101587). This work was partially supported by ARC DP170102726, DP170102231, Google faculty research award and National Natural Science Foundation of China (NSFC) 91646204. Shane Culpepper is the recipient of an Australian Research Council DECRA Research Fellowship (DE140100275). Farhana Choudhury is the recipient of a scholarship from National ICT Australia. We thank Lisi Chen, Gao Cong, Christian S. Jensen, and Dingming Wu for providing the implementation of the IR-tree in [3].

REFERENCES

[1] Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. 2003. Efficient query evaluation using a two-level retrieval process. In *CIKM*. 426–434.
[2] Kaushik Chakrabarti, Surajit Chaudhuri, and Venkatesh Ganti. 2011. Interval-based pruning for top-k processing over compressed lists. In *ICDE*. 709–720.
[3] Lisi Chen, Gao Cong, Christian S. Jensen, and Dingming Wu. 2013. Spatial keyword query processing: an experimental evaluation. In *VLDB*. 217–228.
[4] Yun Chen and J.M. Patel. 2007. Efficient Evaluation of All-Nearest-Neighbor Queries. In *ICDE*. 1056–1065.
[5] Farhana M. Choudhury, J. Shane Culpepper, and Timos Sellis. 2015. Batch processing of Top-k Spatial-textual Queries. In *GeoRich*. 7–12.
[6] Farhana M. Choudhury, J. Shane Culpepper, Timos Sellis, and Xin Cao. 2016. Maximizing Bichromatic Reverse Spatial and Textual K Nearest Neighbor Queries. *Proc. VLDB Endow.* 9, 6, 456–467.
[7] Maria Christoforaki, Jinru He, Constantinos Dimopoulos, Alexander Markowetz, and Torsten Suel. 2011. Text vs. space: efficient geo-search query processing. In *CIKM*. 423–432.
[8] Gao Cong, Christian S. Jensen, and Dingming Wu. 2009. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB* 2, 1 (2009), 337–348.
[9] Constantinos Dimopoulos, Sergey Nepomnyachiy, and Torsten Suel. 2013. Optimizing top-k document retrieval strategies for block-max indexes. In *WSDM*. 113–122.
[10] Shuai Ding, Josh Attenberg, Ricardo Baeza-Yates, and Torsten Suel. 2011. Batch Query Processing for Web Search Engines. In *WSDM*. 137–146.

[11] Shuai Ding and Torsten Suel. 2011. Faster top-k document retrieval using block-max indexes. In *SIGIR*. 993–1002.
[12] Antomn Guttman. 1984. R-trees: a dynamic index structure for spatial searching. In *SIGMOD*. 47–57.
[13] Mingsheng Hong, Mirek Riedewald, Christoph Koch, Johannes Gehrke, and Alan Demers. 2009. Rule-based Multi-query Optimization. In *EDBT*. 120–131.
[14] H. Kido, Y. Yanagisawa, and T. Satoh. 2005. An anonymous communication technique using dummies for location-based services. In *ICPS*. 88–97.
[15] Zhisheng Li, K. C. K. Lee, Baihua Zheng, Wang-Chien Lee, Dik Lun Lee, and Xufa Wang. 2011. IR-Tree: An Efficient Index for Geographic Document Search. *TKDE* 23, 4 (2011), 585–599.
[16] Hua Lu, Christian S. Jensen, and Man Lung Yiu. 2008. PAD: Privacy-area Aware, Dummy-based Location Privacy in Mobile Services. In *MobiDE*. 16–23.
[17] Joel. Mackenzie, Farhana M. Choudhury, and J. Shane Culpepper. 2015. Efficient Location-Aware Web Search. In *ADCS*. 1–8.
[18] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Sch´utze. 2008. *Introduction to Information Retrieval*. Cambridge University Press.
[19] G. M. Morton. 1966. *A computer oriented geodetic data base and a new technique in file sequencing*. Technical report. IBM Ltd.
[20] Dimitris Papadias, Panos Kalnis, Jun Zhang, and Yufei Tao. 2001. Efficient OLAP Operations in Spatial Data Warehouses. In *SSTD*. 443–459.
[21] Apostolos N. Papadopoulos and Yannis Manolopoulos. 1998. Multiple range query optimization in spatial databases. In *ADBIS*. 71–82.
[22] João B. Rocha-Junior, Orestis Gkorgkas, Simon Jonassen, and Kjetil Nørvåg. 2011. Efficient processing of top-k spatial keyword queries. In *SSTD*. 205–222.
[23] Gerard Salton and Christopher Buckley. 1988. Term-weighting Approaches in Automatic Text Retrieval. *Inf. Process. Manage.* 24, 5 (1988), 513–523.
[24] SearchEngineLand. 2010. Microsoft: 53 Percent Of Mobile Searches Have Local Intent. http://searchengineland.com/microsoft-53-percent-of-mobile-searches-havelocal-intent-55556. (2010). [Online; accessed 18-08-2016].
[25] Timos K. Sellis. 1988. Multiple-query Optimization. *ACM Trans. Database Syst.* 13, 1 (1988), 23–52.
[26] ThinkWithGoogle. 2014. Understanding consumers' local search behavior. https://www.thinkwithgoogle.com/research-studies/how-advertisers-can-extend-their-relevance-with-search.html. (2014). [Online; accessed 18-08-2016].
[27] Howard Turtle and James Flood. 1995. Query Evaluation: Strategies and Optimizations. *Inf. Process. Manage.* 31, 6 (1995), 831–850.
[28] Dingming Wu, Gao Cong, and Christian S. Jensen. 2012. A framework for efficient spatial web object retrieval. *PVLDB* 21, 6 (2012), 797–822.
[29] Dingming Wu, Man Lung Yiu, Gao Cong, and Christian S. Jensen. 2012. Joint Top-K Spatial Keyword Query Processing. *TKDE* 24, 10 (2012), 1889–1903.
[30] Chengyuan Zhang, Ying Zhang Zhang, Wenjie Zhang, and Xuemin Lin. 2013. Inverted linear quadtree: Efficient top k spatial keyword search. In *ICDE*. 901–912.
[31] Dongxiang Zhang, Chee-Yong Chan, and Kian-Lee Tan. 2014. Processing spatial keyword query as a top-k aggregation query. In *SIGIR*. 355–364.
[32] Dongxiang Zhang, Kian-Lee Tan, and Anthony K. H. Tung. 2013. Scalable top-k spatial keyword search. In *EDBT*. 359–370.
[33] Jun Zhang, Nikos Mamoulis, D. Papadias, and Yufei Tao. 2004. All-nearest-neighbors queries in spatial databases. In *SSDBM*. 297–306.
[34] Justin Zobel, Alistair Moffat, and K. Ramamohanarao. 1998. Inverted files versus signature files for text indexing. *ACM Trans. Database Syst.* 23, 4 (1998), 453–490.

Received X X; revised X X; accepted X X