Efficient Query Processing on Spatial and Textual Data: Beyond Individual Queries

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

Farhana Murtaza Choudhury

School of Science

College of Science, Engineering, and Health

RMIT University

Melbourne, VIC, Australia

August, 2017

Declaration

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program; any editorial work, paid or unpaid, carried out by a third party is acknowledged; and, ethics procedures and guidelines have been followed.

Farhana Murtaza Choudhury
School of Science (formerly Computer Science and Information Technology)
RMIT University
Melbourne, Australia
30^{th} August, 2017

For my parents.

Acknowledgement

I am indebted to my supervisors Dr. J. Shane Culpepper and Prof. Timos Sellis for their constant guidance, insightful advices, and endless patience. Their encouragement and support has made this journey truly magnificent. I am grateful to my collaborator Dr. Zhifeng Bao, our co-operation has been a valuable learning experience for me. I would like to thank my colleagues and friends for the inspiring discussions and supportive environment for my research.

It is hard to express my gratitude for my family. I am grateful to my parents, my siblings, and my husband for being by my side.

Abstract

With the increasing popularity of GPS enabled mobile devices, queries with locational intent are quickly becoming the most common type of search task on the web. This development has driven several research work on efficient processing of spatial and spatial-textual queries in the past few decades. While most of the existing work focus on answering queries independently, e.g., one query at a time, many real-life applications require the processing of multiple queries in a short period of time, and can benefit from sharing computations. This thesis focuses on efficient processing of the queries on spatial and spatial-textual data for the applications where multiple queries are of interest. Specifically, the following queries are studied: (i) batch processing of top-k spatial-textual queries; (ii) optimal location and keyword selection queries; and (iii) top-m rank aggregation on streaming spatial queries.

The batch processing of queries is motivated from different application scenarios that require computing the result of multiple queries efficiently, including (i) multiple-query optimization, where the overall efficiency and throughput can be improved by grouping or partitioning a large set of queries; and (ii) continuous processing of a query stream, where in each time slot, the queries that have arrived can be processed together. In this thesis, given a set of top-k spatial-textual queries, the problem of computing the results for all the queries concurrently and efficiently as a batch is addressed.

Some applications require an aggregation over the results of multiple queries. An example application is to identify the optimal value of attributes (e.g., location, text) for a new facility/service, so that the facility will appear in the query result of the maximum number of potential customers. This problem is essentially an aggregation (maximization) over the results of queries issued by multiple potential customers, where each user can be treated as a top-k query. In this thesis, we address this problem for spatial and textual data where the computations for multiple users are shared to find the final result.

Rank aggregation is the problem of combining multiple rank orderings to produce a single ordering of the objects. Thus, aggregating the ranks of spatial objects can provide key insights into the importance of the objects in many different scenarios. This translates into a natural extension of the problem that finds the top-m objects with the highest aggregate rank over multiple queries. As the users issue new queries, clearly the rank aggregations continuously change over time, and recency also play an important role when interpreting the final results. The top-m rank aggregation of spatial objects for streaming queries is studied in this thesis, where the problem is to report the updated top-m objects with the highest aggregate rank over a subset of the most recent queries from a stream.

Contents

List of Figures

List of Tables

Table 0.1: All notation

Symbol	**Description**
O	The set of objects.
Q	The set of queries.
D	The dataspace.
$o.l$, $q.l$	The location of object o (query q).
$o.d$, $q.d$	The text description of object o (query q).
k	The number of objects to be returned as the top-k query result.
α	Preference parameter to weight spatial and textual similarities.
O_q^+	The set of result objects of the top-k query q.
$\boldsymbol{d}^{\downarrow}(o.l, q.l)$, $\boldsymbol{d}^{\uparrow}(o.l, q.l)$	Minimum (maximum) Euclidean distance between object o and query q.
$\boldsymbol{d}^{\Uparrow}$	The maximum distance between any point in D.
$\mathrm{CS}(o, q)$	The combined spatial-textual similarity between o and q.
$\mathrm{SS}(o.l, q.l)$	Spatial similarity between the locations of o and q.
$\mathrm{TS}(o.d, q.d)$	Textual similarity between the text of object o and query q.
$\mathrm{TF}(o.d, t)$	Frequency of the term t in $o.d$.
$\mathrm{IDF}(O, t)$	Inverse Document Frequency of the term t in collection O.
Chapter 3:	
$\boldsymbol{w}^{\downarrow}(d, t)$, $\boldsymbol{w}^{\uparrow}(d, t)$	Minimum (maximum) text weight of a term t in a text document d.
E	A node of a tree index.
$\mathrm{CS}^{\downarrow}(E, q)$, $\mathrm{CS}^{\uparrow}(E, q)$	Minimum (maximum) combined spatial-textual similarity between a node E of an index and q.
$\mathrm{SS}^{\downarrow}(E, q)$, $\mathrm{SS}^{\uparrow}(E, q)$	Minimum (maximum) spatial similarity between E and q.
$\mathrm{TS}^{\downarrow}(E, q)$, $\mathrm{TS}^{\uparrow}(E, q)$	Minimum (maximum) textual similarity between E and q.
PQ	A priority queue to track the relevant objects and the nodes of the index.
H	A priority queue to store the result objects.
Section 3.3.3:	

Table 0.1 – *Continued from previous page*

Symbol	**Description**
q^+	Super-query.
$q^+.l$	Location of super-query.
$q^+.d^\cap$, $q^+.d^\cup$	Intersection (union) of the text of the queries in super-query.
$\mathrm{CS}^\downarrow(E, q^+)$, $\mathrm{CS}^\uparrow(E, q^+)$	Minimum (maximum) combined spatial-textual similarity between a node E of an index and q^+.
$\mathrm{SS}^\downarrow(E, q^+)$, $\mathrm{SS}^\uparrow(E, q^+)$	Minimum (maximum) spatial similarity between E and q^+.
$\mathrm{TS}^\downarrow(E, q^+)$, $\mathrm{TS}^\uparrow(E, q^+)$	Minimum (maximum) textual similarity between E and q^+.
$\mathcal{R}_k(q^+)$	The current k·th best minimum combined spatial-textual similarity of any object for any query $q \in q^+$.
$\mathcal{R}_k(q)$	The k·th best combined spatial-textual similarity of any object for q.
LO	A min-priority queue to keep the k objects with the best lower bounds found so far.
RO	A max-priority queue of objects whose upper bound is better than $\mathcal{R}_k(q^+)$.
Section 3.4:	
B	Block size of a posting list in a SIF index.
$\mathrm{ID}^\Uparrow$	The largest object ID in O.
$\boldsymbol{w}^\uparrow(O, t)$, $\boldsymbol{w}^\uparrow(b, t)$	Maximum text weight of a term t in the set of objects O (objects in block b).
MBR_t, $\mathrm{MBR}_{b,t}$	Minimum bounding rectangle of the objects in the posting list of t (in a block of the posting list of t).
CP_t	A pointer to the current posting of t.
νt	Pivot term, for which the accumulated maximum weight of the terms exceeds the threshold $\mathcal{R}_k(q)$.
ν	Pivot, the object ID of the pointer of the pivot term.
TL	The set of terms preceding νt, where the terms are sorted by the object IDs pointed by the corresponding pointers.
fst	First term of the sorted posting lists.
TU	All unique terms of $q \in Q$.

Table 0.1 – *Continued from previous page*

Symbol	**Description**
$\mathrm{CS}_b^{\uparrow}(\mathrm{TL}, q)$	The maximum combined spatial-textual similarity using the block level upper bounds of the terms in TL.
$\mathrm{SS}_b^{\uparrow}(\mathrm{TL}, q.l)$, $\mathrm{TS}_b^{\uparrow}(\mathrm{TL}, q.d)$	The maximum spatial (textual) similarity using the block level upper bounds of the terms in TL
$\mathrm{CS}_\ell^{\uparrow}(o, q)$, $\mathrm{TS}_\ell^{\uparrow}(o.d, q.d)$	The maximum combined spatial-textual similarity (textual similarity) using the location lookup table and the block level upper bounds where object o is stored.
$\nu^{\Downarrow}$	The minimum pivot ID from the current pivots of $q \in Q$.
$q^{\Downarrow}$	The query for which $\nu^{\Downarrow}$ is selected.
Chapter 4:	
U	The set of users.
L	The set of candidate spatial positions (as point, line, etc.).
W	The set of candidate keywords.
p	A specific object $p \notin O$ for which the optimal location and keyword set needs to be selected as result.
ℓ	The optimal location from L to select as result.
ω	The maximum number of keywords to select in the result.
W'	The best set of ω keywords from W to select as result.
Section 4.3:	
$\overline{\mathrm{SS}}(o.l$,u.l)	The visibility of object o w.r.t. the user u.
$\Delta o.l$	The segment of $o.l$ for which the distances and the orientations of all points can be considered as visually similar.
$\mathrm{VL}(o.l, u.l)$, $\mathrm{VL}_\Delta(\Delta o.l, u.l)$	The perceived length of o (the segment $\Delta o.l$) from $u.l$.
$\angle(o.l, u.l)$, $\angle(\Delta o.l, u.l)$	The angle between $o.l$ (the segment $\Delta o.l$) and the straight line connecting the midpoint of $o.l$ and $u.l$.
$\mathrm{len}(o.l)$, $\mathrm{len}(\Delta o.l)$	The length of $o.l$ (the segment $\Delta o.l$).
Section 4.5:	
$\mathcal{B}_p$	The set of users that are a reverse kNN of an object p.
$\mathcal{B}^{\downarrow}{}_\ell$	The set of users that are definitely a reverse kNN of an object with location ℓ based on a lower bound.

Table 0.1 – *Continued from previous page*

Symbol	**Description**
$\mathcal{B}^{\uparrow}{}_{\ell}$	The set of users that can be a reverse kNN of an object with location ℓ based on an upper bound.
u^{+}	Super-user, constructed in the same way as q^{+} by grouping the users in U.
$u^{+}.l$	Location of super-user.
$u^{+}.d^{\cap}$, $u^{+}.d^{\cup}$	Intersection (union) of the text of the users in super-user.
$\mathcal{R}_k(u^{+})$	The current k·th best minimum combined spatial-textual similarity of any object for any user $u \in u^{+}$.
$\mathcal{R}_k(u)$	The k·th best combined spatial-textual similarity of any object for u.
$\mathrm{CS}^{\downarrow}(\ell, u^{+})$, $\mathrm{CS}^{\uparrow}(\ell, u^{+})$	Minimum (maximum) combined spatial-textual similarity of the object p w.r.t. u^{+} when $p.l = \ell$.
$\mathrm{SS}^{\downarrow}(\ell, u^{+})$, $\mathrm{SS}^{\uparrow}(\ell, u^{+})$	Minimum (maximum) spatial similarity of the object p w.r.t. u^{+} when $p.l = \ell$.
$\mathrm{TS}^{\downarrow}(\ell, u^{+})$, $\mathrm{TS}^{\uparrow}(\ell, u^{+})$	Minimum (maximum) textual similarity of the object p w.r.t. u^{+} when $p.l = \ell$.
$\mathrm{CS}^{\downarrow}(\ell, u)$, $\mathrm{CS}^{\uparrow}(\ell, u)$	Minimum (maximum) combined spatial-textual similarity of the object p w.r.t. u when $p.l = \ell$.
$\mathrm{SS}^{\downarrow}(\ell, u)$, $\mathrm{SS}^{\uparrow}(\ell, u)$	Minimum (maximum) spatial similarity of the object p w.r.t. u when $p.l = \ell$.
$\mathrm{TS}^{\downarrow}(\ell, u)$, $\mathrm{TS}^{\uparrow}(\ell, u)$	Minimum (maximum) textual similarity of the object p w.r.t. u when $p.l = \ell$.
$W^{\uparrow}$	The set of ω number of keywords of the highest weights from $u^{+}.d^{\cup} \cap W$.
$W_u{}^{\uparrow}$	The set of ω number of keywords of the highest weights from $u.d \cap W$.
$W^{\uparrow}{}_{w,u}$	The set of ω number of keywords of the highest weights from $W \cap u.d$ such that $W^{\uparrow}{}_{w,u} \cap w \neq \varnothing$.
M	The set of all possible combinations of ω number of keywords from W.

Table 0.1 – *Continued from previous page*

Symbol	**Description**
Section 4.6:	
LO_u	A min-priority queue to keep the k objects with the best lower bounds w.r.t. u found so far.
Section 4.7:	
EU	A node of an MIUR-tree of the users.
Section 4.8.2:	
c	A cell of the auxiliary Quadtree.
$OR(u)$	The obstructed region of u.
$OL(c)$	A list of the users for which c is completely inside $OR(u)$.
$len^{\downarrow}(E)$, $len^{\uparrow}(E)$	The minimum (maximum) length of an object stored in the subtree rooted at node E.
$o.\theta$	The angle of o w.r.t. the Cartesian X-axis.
Section 4.8.3:	
$\chi_{E,u}$, $\hat{\chi}_{E,u}$	The set of leaf level cells of the Quadtree that are visible (not visible) from u and intersects with at least one object in node E.
χ_{E,u^+}, $\hat{\chi}_{E,u^+}$	The set of leaf level cells of the Quadtree that are visible (not visible) from at least one user u from u^+ and intersects with at least one object in node E.
$\boldsymbol{d}^{\downarrow}{}_{\perp}(o.l, u.l)$	The perpendicular distance of the line $o.l$ from $u.l$.
$\text{VL}^{\downarrow}_{\Delta}(\Delta o.l, u.l)$, $\text{VL}^{\uparrow}_{\Delta}(\Delta o.l, u.l)$	Minimum (maximum) perceived length of segment $\Delta o.l$ from $u.l$.
$\angle^{\downarrow}(\Delta o.l, u.l)$, $\angle^{\uparrow}(\Delta o.l, u.l)$	Minimum (maximum) angle between the segment $\Delta o.l$ and $u.l$.
$\angle^{\downarrow}(\Delta o.l, u^+)$, $\angle^{\uparrow}(\Delta o.l, u^+)$	Minimum (maximum) angle between the segment $\Delta o.l$ and any user u from u^+.
Chapter 5:	
$\mathbb{W}$	Sliding window of $\lvert\mathbb{W}\rvert$ most recent queries.
$\text{Con}(q)$	Spatial constraint (range or kNN) of query q.
$\boldsymbol{r}(o, q)$	Ranked position of o based on $\boldsymbol{d}^{\downarrow}(o, q)$.
O_q^+	The set of objects in O that satisfy $\text{Con}(q)$.

Table 0.1 – *Continued from previous page*

Symbol	**Description**
$\rho(o, \mathbb{W})$	Popularity (aggregated rank) of o for queries in $\mathbb{W}$.
qo	The least recent query, which is excluded from $\mathbb{W}$.
qn	The most recent query, which is added to $\mathbb{W}$.
c	A leaf level cell of a quadtree.
$\boldsymbol{d}^{\downarrow}(o, c_q)$ ($\boldsymbol{d}^{\uparrow}(o, c_q)$)	The minimum (maximum) Euclidean distance between o and any query in cell c_q.
$\boldsymbol{r}^{\downarrow}(o, c_q)$ ($\boldsymbol{r}^{\uparrow}(o, c_q)$)	Lower (upper) bound rank of o for any query q in cell c_q.
B	Block size of the rank lists.
$\boldsymbol{d}^{\downarrow}(b, c_q)$ ($\boldsymbol{d}^{\uparrow}(b, c_q)$)	The minimum (maximum) distance between any object in a block b and any query in cell c_q.
Section 5.5:	
ϵ	Approximation parameter.
$\mathbb{W}_{i-1}$, $\mathbb{W}_i$	Two consecutive windows, where $\mathbb{W}_i$ is derived from $\mathbb{W}_{i-1}$ by excluding qo and adding qn, $\lvert\mathbb{W}_i\rvert=\lvert\mathbb{W}_{i-1}\rvert$.
$\hat{\boldsymbol{r}}(o, q)$ ($\hat{\boldsymbol{\rho}}(o, \mathbb{W})$)	Approximation of $\boldsymbol{r}(o, q)$ ($\rho(o, \mathbb{W})$).
R_i	The set of result objects for a window $\mathbb{W}_i$.
o_m	The m·th object from the set R_{i-1}.
$\hat{\boldsymbol{r}}^{\downarrow}(b, q)$ ($\hat{\boldsymbol{r}}^{\uparrow}(b, q)$)	Lower (upper) bound rank of any object in block b for q.
$\hat{\boldsymbol{\rho}}(o_{m+1}, \mathbb{W}_{i-1})$	The approximate popularity of top $(m+1)$·th object from R_{i-1} in previous window $\mathbb{W}_{i-1}$.
$\hat{\boldsymbol{\rho}}(o_m, \mathbb{W}_{i-1} \backslash qo)$	The approximate popularity of top m·th object from R_{i-1}, updated w.r.t. excluding q_o from Window $\mathbb{W}_{i-1}$.
$\hat{\boldsymbol{\rho}}(o_m, \mathbb{W}_i)$	The approximate popularity of top m·th object from R_i.
$\zeta(\hat{\boldsymbol{r}}(o, q))$	Contribution of q to the approximate popularity of o.
Δ_o	The popularity gain of object o for the current window.
$\Delta_o^{\uparrow}$	The maximum popularity gain among all objects for the current window.
Δ_b	An upper bound of the gain of the objects $o \in b$.
BSR	Block-level safe rank.
OSR	Object-level safe rank.
VO	The set of validation objects.

Chapter 1

Introduction

With the widespread use of mobile devices with geo-positioning system (GPS) receivers, the potential of the location data is evident. Our daily life has been enriched with the location based services: people can precisely know their location, the nearby points of interest (e.g., stores, businesses, tourist attractions), and can find how to reach to their destinations. Other uses of GPS data include providing emergency services effectively in the required areas, tracking endangered species to better understand their movement and behaviour, tracking the movement and spread of oil spills by mounting GPS receivers on buoys, etc. As a result of the proliferation of location data, a geographical web is emerging where the contents are associated with both spatial locations and text descriptions.

1.1 Background

1.1.1 Spatial and Spatial-textual Data

The spatial-textual contents are available in a wide range of forms, including points of interest, public transportation, social network posts, news articles, advertisements, multimedia sharing sites, etc. In Figure 1.1, we see an example of spatial-textual objects in Google Maps. Each red circle in the map represents the location of a building at RMIT University, Melbourne. Each building is associated with an address (e.g., building 16, 336 - 348 Swanston street) and a short description (e.g., auditorium). In social network applications, contents associated with locations, known as the geo-tagged contents, have significantly increased in recent years. Examples of such applications are: the location where a photo was taken can be assigned to

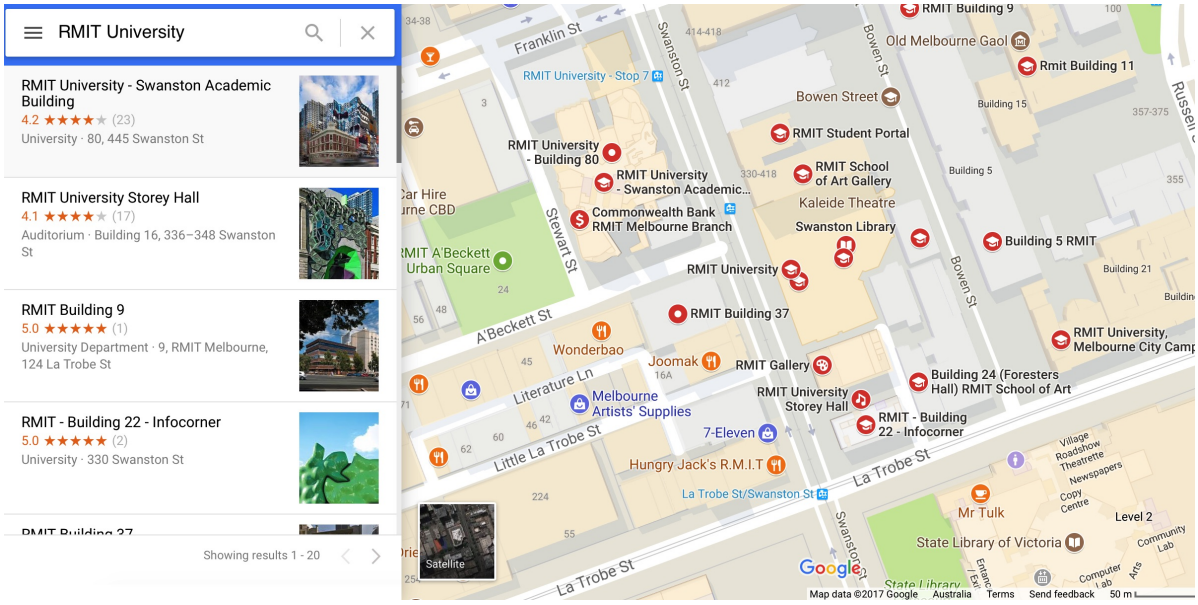


Figure 1.1: A search for "RMIT University" in Google maps. The red circles represent the buildings of RMIT University in the area of focus.

the photo uploaded in Flickr[1], the current location of a user can be combined with the tweets posted in Twitter[2], and Foursquare[3] allows mobile users to "check in" at a location (mark a location when they visit).

The contents can be broadly categorized into two types according to the frequency of their updates. A set of objects, where the locations and the text descriptions of the objects do not update often, is a collection of **static** data. For example, the locations and the descriptive text associated with stores, restaurants, tourist attractions usually do not change frequently. In contrast, where the contents of a collection frequently update, such objects are called **dynamic** objects. For spatial data, if the location of a data object frequently change to an adjacent location, such dynamic objects are called **moving** objects (e.g., a moving car). When new contents are accumulated very frequently in a collection, such dynamic objects are called **streaming** objects. For example, in the year 2016, 51 million new photos were uploaded per month on average in Flickr[4].

[1] www.flickr.com
[2] https://twitter.com
[3] https://foursquare.com
[4] https://www.flickr.com/photos/franckmichel/6855169886

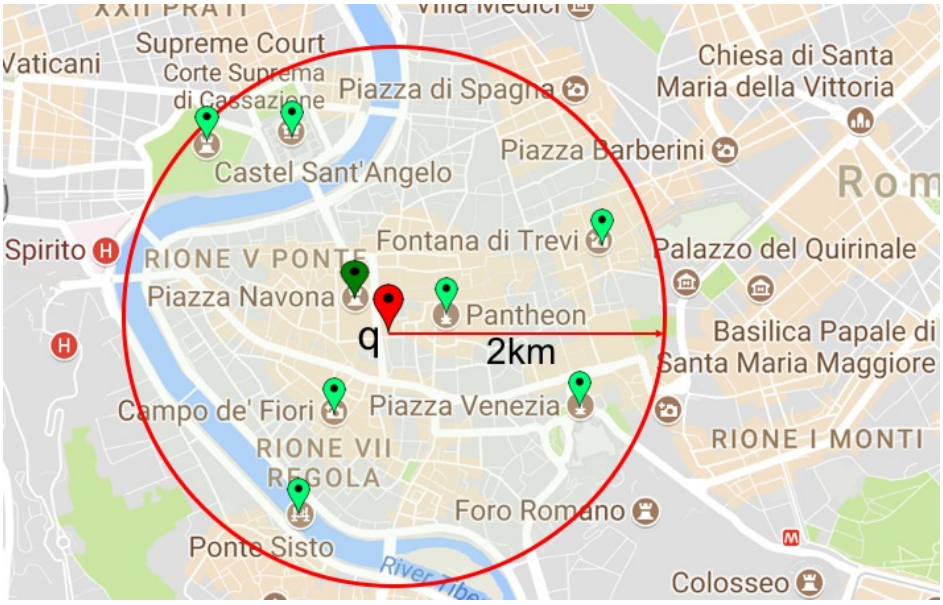


Figure 1.2: An example of a spatial range query. The red pin represents the query location q and the red circle represents the query range. All of the objects inside the query range are results, each represented with a green pin. The nearest neighbor for q is shown with a dark green pin.

1.1.2 Spatial and Spatial-textual Queries

As a result of the proliferation of information with locations, queries with locational intent are quickly becoming the most common type of search task on the web. Around 53% of Bing searches from mobile devices are geographical and have local intent [108]. Location-aware search is not limited to mobile devices. A recent report [117] showed that 84% of all computer users have searched with local intent. This development has driven recent research work on efficient processing of spatial and spatial-textual queries.

In general, a spatial query takes a location as an argument, and the distances between the objects and the query play a vital role in obtaining the results. In spatial databases, the fundamental distance based queries are:

- ***Range queries.*** Consider the example of Figure 1.2 where a query location q is marked with a red pin, and a circle is shown around q with a radius r of 2 kilometers on the map. A range query returns all objects from a dataset that are located within r distance from the query location q. The results of the example range query are shown with green pins in Figure 1.2.

Figure 1.3: An example of reverse k nearest neighbor query. Although the nearest neighbor of q is "Piazza Navona" (shown with dark green pin), it is not the RkNN of q for $k = 1$, as the closest object of "Piazza Navona" is "Pantheon" (shown with light green pin), not q. The set of RkNN objects for q is empty.

- k ***nearest neighbor queries (top-k queries).*** Given a query location q, a nearest neighbor query returns the object with the smallest distance from q. A k nearest neighbor (kNN) query returns k objects that are the nearest to q. In the example of Figure 1.2, the nearest object to the query location q is "Piazza Navona", i.e., the result of a kNN query for q where $k = 1$. The term 'top-k queries' is often used to denote a k nearest neighbor query in literature.

- ***Reverse k nearest neighbor queries.*** Given a query location q, a reverse k nearest neighbor (RkNN) query returns every object o such that q is a k nearest neighbor for o. In the example of Figure 1.2, the query q is the nearest neighbor for "Piazza Navona", "Pantheon", and "Campo de' Fiori", hence these three objects are the RkNN for q when $k = 1$. Consider another example in Figure 1.3 with a different query location. Here, the nearest neighbor for q is also "Piazza Navona", but the nearest neighbor for "Piazza Navona" is "Pantheon", not q. Therefore, "Piazza Navona" is not the RkNN for q when $k = 1$. In this example, the result set of the RkNN query for q is empty.

A textual query consists of a set of keywords and finds the objects that matches with those keywords under some constraints. Some work consider keywords as Boolean conjunctive

Figure 1.4: An example of Boolean range query for the query keyword "Piazza".

predicates where the objects that do not contain all of the query keywords are filtered, while other proposals return a ranked list of objects according to a textual similarity function [88]. Spatial-textual queries are usually associated with the combination of these spatial and textual constraints. For example, given a query location q, a query radius r, and a set of query keywords, a Boolean range query returns the objects that are within r distance from q and contain all of the query keywords. Given the same query location and query radius as Figure 1.2, and the query keyword "Piazza", Figure 1.4 shows the results of the Boolean range query.

The efficient processing of spatial and spatial-textual queries has received considerable attention in the database community. To support different types of queries on spatial and spatial-textual data, several data structures (i.e., indexes) and index access methods have been proposed. The main motivation of the design of the indexes is to support efficient computation (e.g., distance computation, filtering based on keywords) and minimize the I/O operations on the storage devices. The I/O minimization is crucial, especially for the secondary storage devices due to its high latency.

While most of the existing work focuses on answering queries independently and individually (i.e., the processing of a query does not depend on the processing of other queries), many real-life applications require the processing of multiple queries together in a short period of time, and can benefit from sharing computations. This thesis is motivated from these practical

applications and focuses on efficient processing of the queries on spatial and spatial-textual objects where multiple queries are of interest.

1.2 Beyond Individual Query Processing: Motivation and Contribution

In contrast to the traditional query processing techniques that process the queries independent of each other, this thesis focuses on the problems and the challenges that require processing multiple queries on spatial and spatial-textual data. The major challenge of processing multiple queries is the sharing of computations and I/Os, that is, how the common computations and the I/O operations on the storage devices can be leveraged for multiple queries. In this thesis, we have explored the following three types of problems that have important applications, and can benefit from exploiting shared computations among the queries.

1.2.1 Batch Processing of Queries

The motivation comes from different application scenarios that require processing multiple queries over a short period of time, including -

- Multiple-query optimization: the increasing volume of queries in the web (e.g., 19 billion searches per month in Twitter[5], 100 billion searches per month in Google[6]) suggests a need for means of batch query processing.
- Supporting privacy-aware search: An existing strategy to offer search privacy is called the 'fake-query approach', where the real queries are hidden among multiple synthetic queries to obscure the real intent or the exact location. The queries (both real and synthetic) can be partitioned and re-grouped for batch processing to offer further privacy while improving the overall efficiency.

In this thesis, given a set of queries, we address the problem of computing the results for all of the queries concurrently and efficiently as a batch. As there are several different index structures that have been proposed for independent query processing, this thesis investigates the means to extend the existing indexes to utilize common computation and I/Os in a batch, especially if the queries share keywords and/or are spatially close to one another.

[5]http://searchengineland.com/twitter-does-19-billion-searches-per-month-39988
[6]http://www.internetlivestats.com/google-search-statistics

1.2.2 Optimal Location and Keyword Selection Queries

This problem is motivated from an important business task that requires an aggregation over the results of multiple queries. The business task is to identify the best location and the descriptive keywords for a new product or service (e.g., restaurant, store) that can attract the maximum number of customers. A customer is usually interested in a limited number of services that are highly relevant to her preferences, in other words, the services that are ranked high in her query results. Thus, the business task can be translated to a problem of identifying the location and keywords for a new service so that the service will appear in the query result of the maximum number of potential customers. This problem is essentially an aggregation (maximization) over the results of queries issued by multiple potential customers, where each customer can be treated as a top-k query.

In this thesis, we address this problem for spatial-textual data where multiple customers are grouped together and the computations are shared to find the final result of the aggregation.

1.2.3 Top-m Rank Aggregation on Streaming Queries

Rank aggregation is the problem of combining multiple rank orderings to produce a single "best" ordering of the objects. Thus, aggregating the ranks of spatial objects can provide key insights into the importance of the objects in many different scenarios. This translates into a natural extension of the problem that finds the top-m objects with the highest aggregate rank over multiple queries.

As an example scenario, in real estate, each user has a preference on the housing location, and a house is ranked based on the distance from her preferred location (e.g., close to a school or a railway station). A house that is ranked high by many users' queries has a higher aggregate rank, and is therefore more popular in the market. Such information can be used as a starting search point for a new buyer, or to be used as a metric for a potential seller on the best time to enter the market.

As the users issue new queries, clearly the rank aggregations continuously changes over time, and recency can also play an important role when interpreting the final results. In this thesis, we consider the problem of top-m rank aggregation of spatial objects for streaming queries, where, given a set of objects, a stream of distance based spatial queries, the problem is to report the updated top-m objects with the highest aggregate rank over a subset of the most recent queries from the stream.

1.3 Publications

In this section we present the research papers that resulted from this Ph.D. study. For each paper, we refer to the corresponding chapter in which the content of the paper is included.

Paper 1. **Farhana M. Choudhury**, J. Shane Culpepper, and Timos Sellis, "Batch processing of top-k spatial-textual queries", Proc. of ACM SIGMOD Workshop on Managing and Mining Enriched Geo-Spatial Data (GEORICH), pp. 7-12, 2015. The content of this paper is included in Chapter 3.

Paper 2. **Farhana M. Choudhury**, J. Shane Culpepper, Zhifeng Bao, and Timos Sellis, "Batch processing of top-k spatial-textual queries", under review in Transactions on Spatial Algorithms and Systems (TSAS). The content of this article is included in Chapter 3.

Paper 3. **Farhana M. Choudhury**, J. Shane Culpepper, Timos Sellis, and Xin Cao, "Maximizing bichromatic reverse spatial and textual k nearest neighbor queries", Proc. of the Very Large DataBases (VLDB) Endowment (PVLDB), 9(6), pp. 456 - 467, 2016. The content of this paper is included in Chapter 4.

Paper 4. **Farhana M. Choudhury**, J. Shane Culpepper, Zhifeng Bao, and Timos Sellis, "Finding optimal spatial-textual object in obstructed and unobstructed space", review in Very Large DataBases (VLDB) Journal. The content of this article is included in Chapter 4.

Paper 5. **Farhana M. Choudhury**, Zhifeng Bao, J. Shane Culpepper, and Timos Sellis, "Monitoring the top-m rank aggregation of spatial objects in streaming queries", Proc. of the IEEE International Conference on Data Engineering (ICDE), pp. 585 - 596, 2017. The content of this paper is included in Chapter 5.

Additional Publications

During the course of this Ph.D., the following papers has been published, but not included in this thesis as they are not directly connected to the research topic.

- Joel Mackenzie, **Farhana M. Choudhury**, and J. Shane Culpepper, "Efficient location-aware web search", Proc. of the Australasian Document Computing Symposium (ADCS), pp. 1-8, 2015.

- Liangjun Song, Zhifeng Bao, **Farhana M. Choudhury**, and Timos Sellis, "Joint Top-k subscription query processing over microblog threads", in Proc. of Australasian Database Conference (ADC), pp. 381-394, 2016.

- Ch. Md. Rakin Haider, Arif Arman, Mohammed Eunus Ali, and **Farhana M. Choudhury**, "Continuous maximum visibility query for a moving target". in Proc. of Australasian Database Conference (ADC), pp. 82-94, 2016.

1.4 Thesis organization

The rest of the thesis is organized as follows: Chapter 2 presents an overview of the fundamental related work. In Chapter 3, we investigate the processing of batch queries. Chapter 4 presents the optimal location and keyword selection problem and the solutions. The top-m rank aggregation on streaming queries is explored in Chapter 5. Finally, Chapter 6 summarizes the thesis and discusses possible extensions of the current work.

Chapter 2

Literature review

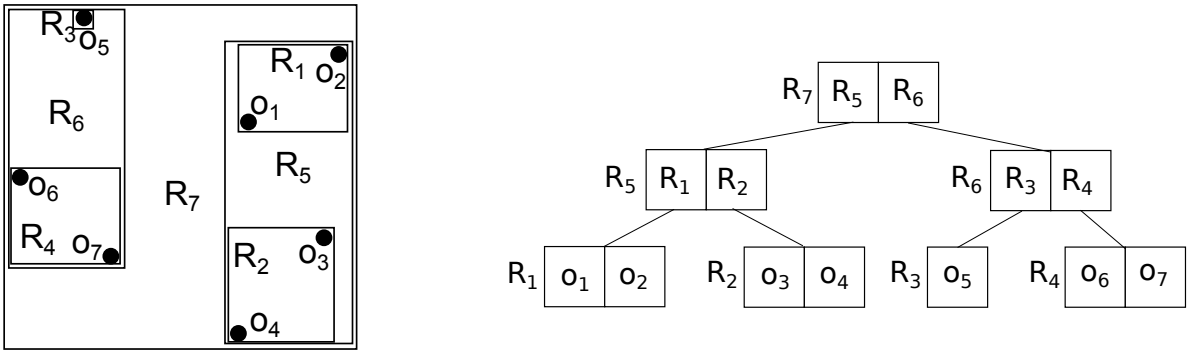
In this thesis, we address problems involving multiple queries on spatial and spatial-textual data. Our work relies on the state-of-the-art spatial and textual indexes and different combinations of the indexes to process the queries. First, we briefly discuss the general index structures and the basic query processing techniques using the indexes. As introduced in Section 1.1.2, the basic spatial query types are: (i) range queries, (ii) k nearest neighbor queries (kNN), and (iii) reverse k nearest neighbor queries (RkNN). The basic textual queries are (i) Boolean and (ii) text similarities based rank queries. The spatial-textual queries are usually associated with the combination of these spatial and textual query constraints. Finally, we present an overview of the research studies that address the processing of individual and multiple queries in this area, and discuss where our work stands in the literature.

2.1 Spatial Databases: Indexes and Query Processing

2.1.1 Spatial Indexes

Space partitioning indexes. A grid-based index [94] partitions a space into multiple non-overlapping regions, denoted as grid cells. Each object is allocated to a grid cell corresponding to its spatial position and a data structure is maintained to access the objects against the grid IDs. The main advantage of the structure is that, the index can be created first, and the spatial objects can be inserted in the cells without changing the structure. Such a structure is called 'space-driven' or data independent structure.

Finkel and Bentley [40] propose the Quadtree, which partitions a two-dimensional space by recursively subdividing it into four quadrants or regions, denoted as Quadtree cells. Each cell



(a) Objects and MBRs

(b) R-tree

Figure 2.1: The example of an R-tree

has a maximum capacity; a cell is split into four lower-level cells if its maximum capacity is reached during insertion. Thus, unlike a grid index, the granularity of the partitions can be varied according to the number and/or the nature of the data to be stored. As shown in the study by Kothuri et al. [69], Quadtrees are suitable for update intensive applications.

Bentley [5] proposed the k-dimensional tree (k-d tree), which is a space-partitioning data structure to index points in a k-dimensional space. A k-d tree is a binary tree where each node is a k-dimensional point. One of the dimensions is chosen for each non-leaf node, and an implicit hyperplane is generated that passes through the point and perpendicular to that dimension's axis. The space is partitioned into two parts by this hyperplane. Points to the left of the hyperplane are represented by the left subtree of that node and points to the right of the hyperplane are represented by the right subtree.

R-tree and its variants. The R-tree [53] is a commonly used spatial index that can be used to store any geometric shape. The idea is to group the spatially close objects in a Minimum Bounding Rectangle (MBR). The R-tree is a hierarchy of nodes, each containing a number of entries. Each entry of a non-leaf node consists of the identifier of a child node and the MBR of all entries of that child node. The entries of a leaf node are the data objects. The number of entries in a node is bounded by a maximum value. The construction of R-trees is 'data-driven', that is, the resulting structure dependents on the dataset being indexed. Figure 2.1a shows the locations of an example set of objects $O = \{o_1, o_2, \ldots, o_7\}$ and Figure 2.1b illustrates the R-tree for O. In this example, the maximum capacity of a node is set as '2'.

The performance of the index depends on how the MBRs are constructed. Two variants of the R-tree, namely the R+-tree [110] and the R*-tree [4] were proposed to improve the

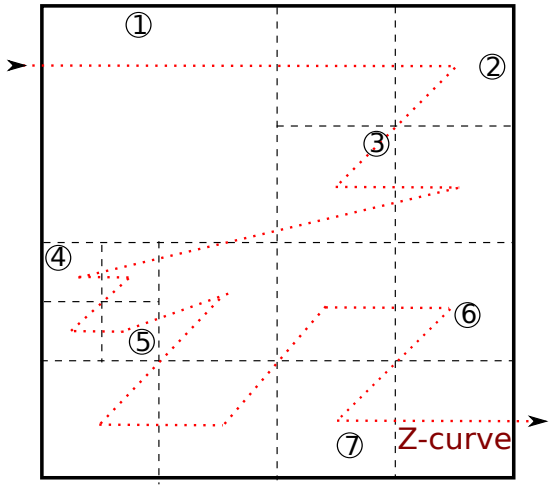


Figure 2.2: An example of the Z-curve ordering

performance by minimizing the 'coverage' and 'overlap'. Coverage refers to the total area that covers the MBRs. Overlap is the total area that is contained in more than one node. By minimizing the coverage, the amount of empty spaces covered by the nodes are reduced. Minimizing the overlap allows a single path to be followed during query processing, while an area is contained in only one node. The R+-tree ensures that the entries of the internal nodes do not overlap by allowing partitions to split rectangles. The R*-tree attempts to reduce both coverage and overlap, using node splitting and reinsertion.

Space filling curves. Space filling curves, such as the Z-order curve [90] and the Hilbert curve [56] are used to impose a linear ordering on spatial objects, such that the objects close to each other in space are also close to each other in the ordering. After the data is ordered, any one-dimensional data structure (e.g., the B-tree [27]) can be used to index them. Figure 2.2 shows an example where the IDs of a set of objects $O = \{o_1, o_2, \ldots, o_7\}$ are assigned according to their position in a Z-curve.

2.1.2 Spatial Query Processing

All of the spatial indexes support efficient processing of spatial range queries. As an example, in R-trees, the nodes are recursively traversed from the root node to the leaf nodes to answer a range query. If a node does not intersect with the query range, the subtree rooted at that node is pruned. When the leaf nodes are reached, each of its entries (data objects) are verified and returned as result if that entry is actually contained in the query range.

Roussopoulos et al. [106] proposed a branch-and-bound traversal algorithm to answer kNN queries. Starting from the root node, the minimum and the maximum distance of each node from the query location are computed as an upper bound (overestimation) and a lower bound (underestimation) of the spatial similarity, respectively. The k best lower bound spatial similarities are maintained and updated as the nodes are traversed. If the upper bound spatial similarity of a node is not equal or better than the k best lower bound similarity scores, the subtree rooted at that node is pruned from consideration. When a leaf node is reached, the actual distance of the objects in that node from the query is calculated. Finally, k objects with the smallest distances are returned as results. Hjaltason and Samet [57] proposed an incremental algorithm to answer kNN queries that is applicable to a large class of hierarchical spatial data structures. In this approach, the nodes of the trees are accessed in a best-first order of their upper bound distances from the query location. Finally, the first k data objects that are accessed in this order are returned as the result. Their experiments show that the algorithm significantly outperforms the branch-and-bound approach.

2.2 Information Retrieval: Models, Indexes, and Query Processing

2.2.1 Information Retrieval Models

Several models have been proposed to address different information retrieval needs. The choice of a retrieval model depends highly on the application. The vector space model [147], the Okapi BM25 model [104], and the language model [100] are all commonly used models.

The representation of a set of documents as vectors in a common vector space is known as the vector space model. Both documents and queries are represented as vectors, where each dimension corresponds to each distinct term. If a term is present in a document, its value along the corresponding dimension, referred to as the term weight, can be computed using different schemes. One of the most commonly used schemes is the TF·IDF metric [147]. The TF (Term Frequency), $\mathrm{TF}(d,t)$ counts how many times the term t appears in a document d, and the IDF (Inverse Document Frequency) $\mathrm{IDF}(O,t) = \log \frac{|O|}{|o \in O, \mathrm{TF}(d,t)>0|}$ measures the importance of t with respect to all of the documents in an object collection O. The text similarity between a document d and a query q is calculated as:

$$\mathrm{TS}(d,q) = \sum_{t \in q \cap d} \mathrm{TF}(d,t) \times \mathrm{IDF}(O,t) \tag{2.1}$$

In information retrieval, the language model is a query likelihood model. Given a query, the documents are ranked based on the probability that the document's language model would generate the terms of the query. Usually, smoothing is necessary to assign probability for unseen terms in a document. Finally, the Okapi BM25 Model [104] takes the document length into account to compute the text relevance. Unlike the TF·IDF metric, the IDF score in BM25 may give negative scores for terms that appear in a large number of documents.

2.2.2 Text Indexing

The inverted file [147] is widely used in many of the state-of-the-art large-scale information retrieval systems, such as web search engines. An inverted file consists of a vocabulary of all distinct terms in a collection of documents. Each term is associated with an inverted list where each inverted list is a sequence of postings. Each posting contains the identifier of such a document d whose description contains the term and usually the frequency of that term in d as well. In general, the postings in each inverted list are sorted by the document ID, and the lists are compressed.

The signature file [37] and the bitmap are two other text indexes that can be used for document indexing. The key idea of signature files is to create a quick filter to obtain all documents that match with the query, though some additional documents that do not match may pass the filter also, as 'false hits'. A signature, typically a hash coded version, is generated for each document for this purpose. A bitmap in which each bit represents the occurrence of a keyword can be viewed as a special type of signature file. As demonstrated by the empirical results in the work by Zobel et al. [148], signature files are not competitive with the inverted file in terms of efficiency and space, specially for large datasets. However, a recent work [48] on signature files, denoted as the BitFunnel uses Bloom filter [6] to represent the set of terms in each document as a fixed sequence of bits to address the limitations of signature files.

2.2.3 Text Query Processing

The two most commonly used traversal techniques in inverted files are: (i) Document-At-A-Time (DAAT) and (ii) Term-At-A-Time (TAAT) processing [119]. In DAAT, each inverted 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. The total score of the current qualifying document for the query terms are computed before proceeding to the next one.

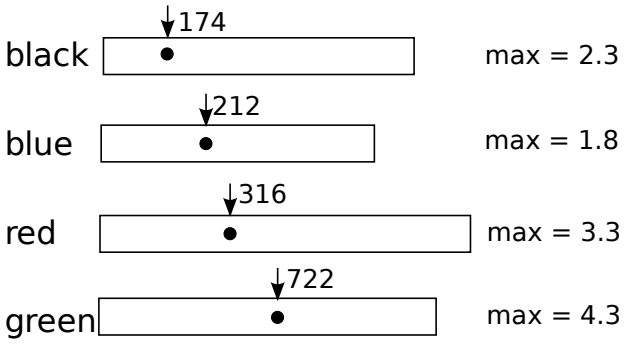


Figure 2.3: An example of the WAND traversal

In TAAT, the scores for all of the qualifying documents are computed concurrently for one query term at a time. The entire inverted list for the query term that is the rarest in the collection is usually processed first, and then the next rarest term is processed. An accumulator data structure is used to keep track of the partial scores of the documents. When the lists of all of the query terms have been processed, k documents with the highest scores are returned.

The inverted lists are required to be stored in order of the document IDs for DAAT traversal; TAAT allows other orderings of the lists. While there are advantages and disadvantages to both processing regimes, DAAT processing tends to be favoured in current IR systems as non-textual features can be more easily integrated into the scoring process. We present a DAAT traversal algorithm, denoted as the WAND [8] next.

WAND algorithm. In WAND, objects (text documents) are sorted in ascending order of their IDs in each posting list. For each query term, the algorithm maintains the pointers 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 corresponding document IDs of the pointers, until the sum becomes greater than or equal to a threshold, $\mathcal{R}_k(q)$ for the query q. Here, $\mathcal{R}_k(q)$ 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". For example, in Figure 2.3, the posting lists are sorted by the corresponding document IDs of the pointers, where the query terms are 'black, blue, red, green'. Let, the current threshold $\mathcal{R}_k(q) = 2.6$. After summing up the maximum textual similarity scores from the top, the summation exceeds the threshold for the term 'blue'. So the pivot term is 'blue', and the pivot is '212'.

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 un-scored 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 document 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. If the "pivot" document is a candidate, the total score of that document is computed for the query. If the total score is greater than the threshold, the current top-k documents and the threshold are updated accordingly. As more documents are processed, more future documents are likely to be skipped based on the refined threshold score.

2.3 Spatial-textual Databases: Indexes and Query Processing

Existing work on spatial-textual indexes combine a spatial index with a text index to answer the queries. A straightforward approach to utilize both types of indexes is described as a baseline method by Zhou et al. [145] and Christoforaki et al. [26]. In this approach, the spatial component of the objects is indexed by a spatial index (e.g., R-tree, Quadtree) and the textual part is indexed separately by a text index (e.g., inverted file, signature file). A query can be answered by either filtering with the spatial index first, then verified by the text index, or vice-versa. The main drawback of this approach is the number of the irrelevant objects retrieved in its filtering step. If the spatial index is used first, the retrieved objects are spatially relevant with the query, but may not be textually relevant. The same is true if the text index is used first for filtering. However, the advantage of two separate indexes is the ease of maintenance and updates.

Several hybrid spatial-textual indexes that combine the two types of indexes have been proposed and studied to answer different types of queries [24, 26, 28, 76, 105, 114, 129, 142]. In these indexes, both the spatial and textual pruning can be applied simultaneously during the query processing steps. Based on the combination scheme, the hybrid indexes can be broadly categorized into two types, namely, spatial-first and textual-first. If spatial (textual) property is prioritized to construct the index, the index is denoted as spatial-first (textual-first). We describe these indexing approaches next.

Spatial-first indexes. The IR-tree is essentially an R-tree, where each node is augmented with a reference to an inverted file for the documents in the sub-tree of the node. Each node

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	$(t_1, 2), (t_4, 3)$

Table 2.1: Text description of the objects shown in Figure 2.1a

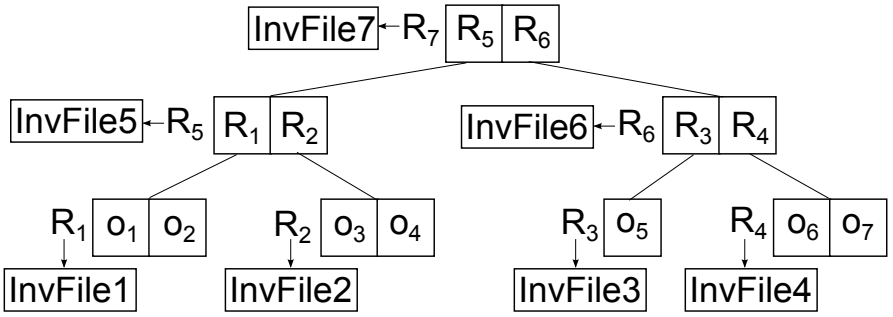


Figure 2.4: The example IR-tree

R contains a number of entries, consisting of a reference to a child node of R, the MBR 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 a data object. Otherwise, the text identifier refers to a pseudo-text description, which is the union of all text descriptions in the entries of its child nodes. 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.

Table 2.1 shows the text descriptions of an example dataset $O = \{o_1, o_2, \ldots, o_7\}$, where the locations of the objects are the same as the example shown in Figure 2.1a. Figure 2.4 illustrates the IR-tree for O. Table 2.2 and Table 2.3 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.

Several variants of the IR-tree, including DIR-tree, CIR-tree, and CDIR-tree were also proposed [28, 129], aimed at optimizing the performance. During the insertion process, a DIR-tree takes both spatial and textual information into account to both minimize the areas of

Table 2.2: 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 2.3: 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	$(R_1, 1), (R_2, 2)$	$(R_4, 3)$	$(R_5, 2), (R_6, 3)$

MBRs and maximize the text similarities between the objects of each node. A parameter β was introduced to balance the weights of the two components. In a CIR-tree, the objects of each node are grouped into a number of clusters based on their text descriptions. Instead of constructing a single pseudo-document for each node, a pseudo-document is constructed for each cluster in each node to achieve a better text relevance estimation in the upper level nodes. A CDIR-tree is a combination of the DIR-tree and the CIR-tree, and showed the best performance in the experimental studies [28, 129].

Li et al. [76] presented an index to process spatial-textual kNN queries, which was also called an IR-tree. We denote this index with LiIR-tree to distinguish between the index proposed by Cong et al. [28]. Unlike the IR-tree, a LiIR-tree stores one integrated inverted file for all non-leaf nodes. Specifically, it stores the weights of a term for the nodes together in the same memory block, which refers to a linked list of pages. The leaf nodes in the LiIR-tree are associated with their individual inverted files. The organization of the inverted files in a LiIR-tree results in a smaller index size than the IR-tree. The LiIR-tree has similar performance to the IR-tree as shown by Chen et al. [21] and Wu et al. [129], but performs worse than the improved variants of the IR-tree.

The IR-tree and its variants can answer both Boolean and top-k queries. The tree is traversed in a best-first manner based on the spatial-textual similarity, similar to the approach described for spatial data by Hjaltason and Samet [57] (Section 2.1.2). If the text constraint is a Boolean conjunction only the nodes containing all of the query keywords in the corresponding inverted file are considered.

Felipe et al. [39] propose the IR^2-tree, which integrates an R-tree and signature files to answer spatial-textual queries with Boolean text constraints. A signature file is stored with each node of the IR^2-tree. The signature file of a node is the union of all signatures of its entries, each representing a child node. A file associated with a node summarizes the presence and absence of the terms in the objects rooted at that node. The drawback of an IR^2-tree is that the same signature length is used for all levels, which is equal to the number of distinct terms in the collection. This approach produces more false positives in higher levels. Hence, the Multi-level IR^2-tree (MIR^2-tree) was proposed in the same paper to overcome this problem. The optimal signature length is derived for each level and the signatures of the entries for a node are superimposed. A multi-level superimposed coding [12] is used, which reduced the number of false positives, particularly in the intermediate nodes. However, signature files can only determine whether a given document contains the query keywords or not, but fail to rank them based on the textual similarity. In addition, the performance of traditional signature files are generally inferior to inverted files [148] (although recent work [48] is challenging this notion).

Lu et al. [83] proposed the Intersection-Union R-tree (IUR-tree) to address spatial-textual reverse kNN queries. The tree is constructed in the same manner as an R-tree, where each node is augmented with two text vectors: an intersection vector and a union vector. The weight of each term in the intersection (union) text vector is the minimum (maximum) weight of the terms in the document objects that are contained in the corresponding subtree. Each non-leaf node is also associated with the number of objects in the subtree rooted at that node. A branch-and-bound algorithm is proposed over the index to answer the RkNN query.

Textual-first indexes. The spatial Inverted Index (S2I) was proposed by Rocha-Junior et al. [105] and employs different strategies for indexing frequent and infrequent terms. For each frequent term t, the objects containing that term are indexed using an aggregated R-tree (aR-tree) [97]. Each node of the aR-tree stores the maximum textual weight of t for the objects of the subtree rooted at that node. For each infrequent term t', the S2I uses an inverted file to store the objects containing t'; for each object, the object ID, the location, and the weight of t' for the object are stored as a tuple. Finally, the S2I organizes the blocks and trees in an alphabetic order. For each distinct term, S2I stores the number of objects containing the term (the document frequency), a flag indicating the type of storage used by the term (tree or block), and a pointer to the aR-tree or the block that stores the object. During query processing, the inverted lists of the query terms are accessed in a term-at-a-time order, and the partial spatial-

textual scores of the qualifying objects are maintained until the lists for all of the query terms are traversed. As shown in a comparative study [21], S2I outperforms the other indexes in terms of both processing time and I/O cost for smaller number of query terms. However, this index requires a large amount of storage space, as an object is stored multiple times in many trees and blocks.

Zhang et al. [143] present an algorithm using an inverted file to answer spatial-textual kNN queries, where the postings in each inverted list are sorted based on their term weights. When a query is issued, a new list is generated where the objects are sorted based on their distances from the query location. The lists of the query terms and the list for distances are traversed simultaneously. A threshold is derived and updated based on the scores of the objects that have been accesses so far. The traversal of the lists can be early terminated based on this threshold, such that no subsequent object can be a result of the query.

Christoforaki et al. [26] proposed different indexes that retrieve the objects within a spatial query region, and then rank them according to a combined ranking function. Two approaches based on space filling curves were proposed. One is SFC-Quad, which maintains an inverted file where the object IDs in each inverted list are assigned and ordered based on their spatial positions on a Z-curve. The idea is that all objects in the query region are likely to be close to each other in this ordering, and thus close to each other in the inverted list. This allows skipping large parts of the list. SFC-Quad uses an in-memory Quadtree to enable skipping based on the spatial component of the query. When a query is processed, the Quadtree is first traversed to obtain m number of ID ranges that contain all objects intersecting the query range. These ranges are merged into a smaller number of ranges to reduce random disk I/O costs, and then the disk-resident inverted lists are swept to fetch the needed parts. In another approach denoted as SFC-SKIP, each block stores a minimum bounding rectangle encompassing all of the objects contained in the block. During query processing, any block whose MBR does not intersect with the query range can be skipped.

Zhang et al. [142] proposed an index denoted as the Integrated Inverted Index (I^3), where a Quadtree is created for each term of the inverted file. For each term, a list of objects and their associated locations in a Quadtree cell are stored. A lookup table is maintained that stores the page information for each term. In I^3, a signature file [37] is also stored, which aggregates the object IDs for each term and also stores the upper bound score of the text similarity weights according to a information retrieval model. Based on the summary information, different pruning strategies are proposed for Boolean and top-k queries. As shown in their experiments, this index outperforms both IR-tree and S2I index.

Another index was proposed that utilize an inverted file and a Quadtree by Chengyuan et al. [24], denoted as the inverted linear Quadtree (IL-Quadtree). Unlike I^3, a linear Quadtree [44] is created for each term where only the non-empty leaf nodes of the Quadtree are stored in an auxiliary disk-based one dimensional structure (e.g., B+-tree). Each node is encoded by a Z-order curve based on the path of the node in the Quadtree. This approach is designed to answer kNN queries with Boolean constraints.

Tao and Sheng [114] proposed the spatial inverted list (SI-index) that uses an R-tree for each inverted list to retrieve the objects based on their distances from the query location. An ID is assigned to each object based on their position in a Z-curve, so that the objects close in space gets close-by IDs. A different technique was proposed to construct the R-trees, where each block of an inverted list becomes a leaf node of the corresponding R-tree. Therefore, the data objects are not duplicated in these two structures. Each inverted list is divided into a number of disjoint blocks such that the number of points in each block is between B and $2B-1$, where B is the block size, and then these blocks can be directly used to construct the R-tree nodes. Thus the SI-index has significantly reduced index size when compared against the S2I index.

More recently, Mackenzie et al. [85] 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 an existing document-at-a-time algorithm [8] to prune the objects that cannot be a result of a top-k spatial textual query.

Summary. A summary of the spatial-textual indexes is shown in Table 2.4. If both spatial and textual properties are prioritized while constructing an index, both 'spatial' and 'textual' are marked under index priority. The symbol $\triangle$ means that the index was not originally designed to answer the corresponding query type, but can be extended to process that type of query.

Chen et al. [21] performed a comprehensive study comparing most of these techniques for a wide variety of related problems. For spatial-textual kNN queries, S2I [105] outperformed the other indexes in most cases, although the index had a significant space overhead. When the number of query keywords is large (greater than 5), the CDIR-tree [28] performs slightly better than S2I in terms of runtime.

Table 2.4: Summary of spatial-textual indexes

Index	**Index priority**		**Spatial constraint**		**Textual constraint**	
	Spatial	Textual	Range	Ranked	Boolean	Ranked
IR^2 [39]	✓		✓		✓	
LiIR-tree [76]	✓		✓	✓	✓	✓
I^3 [142]	✓		✓	✓	✓	△
IR-tree [28]	✓		✓	✓	✓	✓
DIR-tree [28]	✓	✓	✓	✓	✓	✓
CDIR-tree [28]	✓	✓	✓	✓	✓	✓
SFC-Quad [26]	✓	✓	✓		✓	
SFC-Skip [26]	✓	✓	✓		✓	
SI-index [114]		✓	✓	✓	✓	△
S2I [105]		✓	✓	✓	✓	✓
IL-Quad [24]		✓	✓	✓	✓	△
Aggregate top-k [143]		✓	✓	✓	✓	✓
GeoWAND [85]		✓	✓	✓	✓	✓

2.4 Overview of Queries

Table 2.5 presents the studies that address different types of queries for different categories of data. As presented in Section 1.1.2, data can be either static or dynamic based on the frequency of the updates. The first part of the table shows the work where individual queries are processed independently of each other, and the second part presents the summary of literature where multiple queries are of interest. As range queries and kNN queries are often addressed in the same paper over the same index in the literature, we present them together in the table. The references for the spatial-textual queries include both Boolean and textual similarity constraints. The problems involving streaming data are categorized into two types, one where the data objects are streaming, and the other is where the queries are streaming.

2.4.1 Individual Queries

As shown in Table 2.5, efficient processing of the individual queries has received considerable attention in the research community. Different solutions have been proposed to answer the basic query types on static data, in both spatial [1, 16, 30, 53, 57, 106, 116] and spatial-textual [24, 26, 28, 39, 76, 83, 84, 105, 114, 120, 129, 142, 145] databases. The proposed indexes and the solution techniques are presented in the previous sections (Section 2.1-2.3).

The problem of a moving query is to continuously report the updated query results as the location of the query changes. Cheema et al. [15] address the problem for spatial range queries, and there are several other studies that answer the problem for kNN queries [55, 72, 92, 96].

Table 2.5: Overview of queries

	Query \ Data		Static	Dynamic		
				Moving	Streaming	
					Obj	Query
Individual query	Range and kNN	Spatial	[53, 57, 106]	[15, 55, 72, 92, 96]	[7, 70]	
		Spatial-textual	[24, 26, 28, 39, 76, 105, 114, 120, 129, 142, 145]	[50, 51, 61, 128, 131]	[122]	[20]
	RkNN	Spatial	[1, 16, 30, 116]	[14, 17, 18, 52, 65, 118]	[68]	
		Spatial-textual	[83, 84]			
Multiple queries	Range and kNN	Spatial	[22, 98, 144]		[91, 93]	
		Spatial-textual	[130]		[20, 51, 62, 74, 79, 123, 124, 137]	
	RkNN	Spatial	[77, 81, 126, 146]	[45]		
		Spatial-textual	[47]			

Mouratidis et al. [92] presented solutions to update the results where both the query location and the objects' locations can move. For spatial-textual data, the results that best match with a query with respect to location and text similarity are updated as the query location moves [50, 51, 61, 128, 131]. For streaming data, when new objects arrive in a stream, the results of the existing query needs to be updated. Chen et al. [20] addressed the problem where both new objects and queries with Boolean range constraints are streaming, and the objects that satisfy the query constraints need to be updated. The approaches to address the moving and the streaming problems are discussed in detail in Section 5.2.

2.4.2 Multiple Queries

In spatial databases, there are several solutions for batch processing of a set of queries over static dataset [22, 98, 144]. Wu et al. [130] addressed the batch processing of spatial-textual queries, which is applicable to Boolean kNN constraints only (discussed in Section 3.6).

The maximized reverse k nearest neighbor query is an aggregation over multiple kNN queries (details in Chapter 4). In spatial databases, given a set of kNN queries, a maximized RkNN

returns the location for an object so that the object will be a kNN of the maximum number of queries (i.e., the object will be a RkNN of the maximum number of queries) [77, 81, 126, 146]. Ghaemi et al. [45] addressed this problem for $k = 1$, where both the queries and the objects can move. Gkorgkas et al. [47] addressed the problem of finding the text description of an object with a fixed location such that the object will be a kNN of the maximum number of queries based on spatial-textual similarities. More details of the approaches are available in Section 4.2.

There are several studies where multiple queries are considered together against streaming objects [20, 51, 62, 74, 79, 123, 124, 137]. In general, a set of queries are registered to an object stream, and when a new object arrives, the object is reported to the queries if it qualifies as a result for that query. As the queries are known beforehand, the set of queries are often indexed and several pruning techniques are applied on them to reduce the number of queries to be checked against each new object. These approaches are discussed in Section 5.2 in detail.

2.4.3 Summary

This section reviews the existing studies related to spatial and spatial-textual queries. In this thesis, we address three different problems involving multiple queries that are shown with shades in their respective position in Table 2.5. Specifically, (i) we present solutions to batch process spatial-textual kNN queries, where the existing work addresses only the Boolean version of the problem; (ii) we present solutions to find both the optimal location and the set of keywords for the maximized reverse k nearest neighbor problem, where the existing work can find either the optimal location for an object, or the optimal set of keywords for a fixed location; and (iii) we are the first to introduce and address the rank aggregation of spatial streaming queries.

Chapter 3

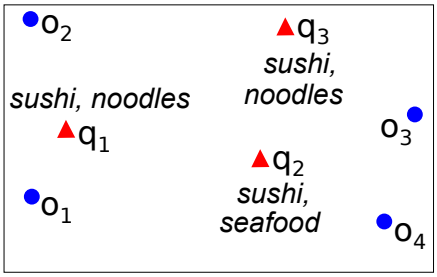
Batch Processing of Top-k Spatial-Texual Queries

A top-k spatial-textual query is an important search task that has been extensively studied in the literature. However, most of these approaches focus on answering one query at a time. In contrast, how to design efficient algorithms that can exploit similarities between multiple queries to improve performance has received little attention. In this chapter, given a set of top-k spatial-textual queries, we present our approaches to compute the results for all the queries concurrently and efficiently as a batch.

3.1 Introduction

As defined in Chapter 1, a top-k spatial-textual query returns the k most similar objects for a query by combining both spatial and textual similarity. For example, in Figure 3.1, o_1, o_2, o_3, and 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_1, restaurant 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 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



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

Figure 3.1: Example top-k spatial-textual queries

time slot, the queries that have arrived can be processed together; (iii) supporting privacy-aware search where real top-k spatial-textual queries are masqueraded using artificial queries to hide the real intent or exact location [66, 82]; and (iv) 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. In this case, the results for each query in the log can be batch processed offline

Given a set of top-k spatial-textual queries Q, we propose approaches to process and find the results for all of the queries concurrently and efficiently as a batch. Consider the example in Figure 3.1 again. Let q_1, q_2, and q_3 be three different top-k spatial-textual queries to be batch processed, and the value of k be 1. Both q_1 and q_3 have the same query keywords ("sushi", "noodles") and the query keywords of q_2 are "sushi" and "seafood". Object o_2 is the result of both q_1 and q_3, and object o_3 is the result of q_2. Our aim is to process these queries together as a batch and find the top-k spatial-textual objects for each of them. In order to achieve the goal of efficiently answering multiple top-k spatial-textual queries, we study the following 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 the 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 [28, 76, 129], the Spatial Inverted Index (S2I) [105], the Integrated Inverted Index (I^3) [142] and the IL-Quad-tree [24]. 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 include the textual component in the spatial index [28, 76, 129, 142]. However, there is also a vast amount of location-embedded web data being processed for location-aware search [108, 117], 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 2.4 in Section 2 for a systematic comparison of the various indexing approaches for processing a spatial-textual top-k query.

For batch processing, there is little prior work on spatial-textual queries. Ding et al. [33] look at batch processing of textual queries in large scale web search engines, but do not consider queries with local intent. Wu et al. [130] study 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, as mentioned in Chapter 1. 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 3.6.

To process a batch of top-k spatial-textual queries efficiently, we propose a series of carefully designed indexes and traversal algorithms in this chapter. In particular, we first propose a new traversal method, SF-Sep that shares the I/O costs among the queries where the objects are indexed using an existing structure (which is designed to process queries individually). Then, we propose a new index structure, the MIR-tree along with a filter-and-refine based technique, SF-Grp over this index. The idea is to group the queries and traverse the index for this group, instead of individual queries to better support the sharing. The commonality in these two approaches is that, both of them prioritize spatial properties in index construction, which is good for spatial-textual datasets 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.3.5 for details). Moreover, as shown by Chen et al. [21], 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*

index structure (SIF index) for text intensive data using an inverted file and a new traversal algorithm, TF-MBW over the SIF index.

In summary, we have included the following contributions in this chapter:

- A new traversal method, SF-SEP to share the I/O costs between the queries are proposed using an existing spatial-first index structure (Section 3.3.2).
- A new index structure, the MIR-tree and a filter-and-refine based technique, SF-GRP over this index is proposed, where the focus is to utilize the shared computations and I/Os among the queries by grouping them together (Section 3.3.3).
- A text-first index, the SIF and a traversal algorithm, TF-MBW are presented in Section 3.4.
- The performance of these approaches is compared along with appropriate baselines to answer batch top-k spatial-textual queries.

The rest of the chapter is organized as follows: Section 3.2 introduces the preliminaries and defines the problem. Section 3.3 presents the a space-prioritized baseline, the SF-BL (Section 3.3.1) and two space-prioritized approaches: the SF-SEP approach (Section 3.3.2) and the SF-GRP approach (Section 3.3.3). In Section 3.4, we describe the 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 3.5. We present an overview of related work in Section 3.6 and conclude in Section 3.7.

3.2 Problem Formulation

Let O be a set of objects in a spatial-textual database D. Each object $o \in O$ is defined as a pair $(o.l, o.d)$, where $o.l$ is a location in Euclidean space and $o.d$ is a text document. Given a query location $q.l$, a set of query keywords $q.d$, and the number of results to return k, a **Top-k Spatial-Textual Query** returns a ranked list of k objects from O that are the most relevant to $q.l$ and $q.d$ according to a spatial-textual similarity metric.

Without loss of generality, the following widely adopted linear weighted combination is used in this chapter as the combined spatial-textual similarity:

$$\mathrm{CS}(o, q) = \alpha \cdot \mathrm{SS}(o.l, q.l) + (1 - \alpha) \cdot \mathrm{TS}(o.d, q.d)\,, \tag{3.1}$$

where $\mathrm{SS}(o.l, q.l)$ is the spatial similarity between the location of the object and the query, $\mathrm{TS}(o.d, q.d)$ 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 computed as:

$$\mathrm{SS}(o.l, q.l) = 1 - \frac{\boldsymbol{d}^{\downarrow}(o.l, q.l)}{\boldsymbol{d}^{\Uparrow}}, \tag{3.2}$$

where $\boldsymbol{d}^{\downarrow}(o.l, q.l)$ is the minimum Euclidean distance between these two locations and $\boldsymbol{d}^{\Uparrow}$ is the maximum Euclidean distance between any two points in D.

Text similarity. An object o is considered similar to a query q if and only if $o.d$ contains at least one term $t \in q.d$. Several measures can be used to compute the similarity between any two text descriptions, as described in Section 2.2.1. We use the TF·IDF metric [107] for illustration purpose in this work, but our approach is applicable to any text-based similarity measure.

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

$$\mathrm{TS}(o.d, q.d) = \sum_{t \in q.d \cap o.d} \mathrm{TF}(o.d, t) \times \mathrm{IDF}(O, t) \tag{3.3}$$

The formal definition of our problem, the batch processing of top-k spatial-textual queries is presented in the following.

Definition 1. ***Batch processing of top-k spatial-textual queries.*** *Given a set of spatial-textual objects O, a set of top-k spatial-textual queries Q, each $q \in Q$ is defined as $(q.l, q.d, k)$, where $q.l$ is the query location, $q.d$ is the set of query keywords and k is the number of objects to return as result, returns a subset $O_q^+ \subseteq O$ for each query q such that $|O_q^+| = k$, $\forall o \in O_q^+$, $\forall o' \in \{O - O_q^+\}$, $CS(o, q) \geq CS(o', q)$.*

Table 3.1 presents the basic notation used in the remainder of this chapter.

Table 3.1: Basic notation

Symbol	Description
O	The set of objects.
Q	The set of queries.
D	The dataspace.
$o.l$, $q.l$	The location of object o (query q).
$o.d$, $q.d$	The text description of object o (query q).
k	The number of objects to be returned as the result of a top-k query.
α	Preference parameter to weight spatial and textual similarities.
O_q^+	The set of result objects of the top-k query q.
$\boldsymbol{d}^{\downarrow}(o.l, q.l)$, $\boldsymbol{d}^{\uparrow}(o.l, q.l)$	Minimum (maximum) Euclidean distance between object o and query q.
$\boldsymbol{d}^{\Uparrow}$	The maximum distance between any point in D.
$\mathrm{CS}(o, q)$	The combined spatial-textual similarity between o and q.
$\mathrm{SS}(o.l, q.l)$	Spatial similarity between the locations of o and q.
$\mathrm{TS}(o.d, q.d)$	Textual similarity between the text of object o and query q.
$\mathrm{TF}(o.d, t)$	Frequency of the term t in $o.d$.
$\mathrm{IDF}(O, t)$	Inverse Document Frequency of the term t in collection O.
$\boldsymbol{w}^{\downarrow}(d, t)$, $\boldsymbol{w}^{\uparrow}(d, t)$	Minimum (maximum) text weight of a term t in a text document d.
$\mathrm{CS}^{\downarrow}(E, q)$, $\mathrm{CS}^{\uparrow}(E, q)$	Minimum (maximum) combined spatial-textual similarity between a node E of an index and q.
$\mathrm{SS}^{\downarrow}(E, q)$, $\mathrm{SS}^{\uparrow}(E, q)$	Minimum (maximum) spatial similarity between E and q.
$\mathrm{TS}^{\downarrow}(E, q)$, $\mathrm{TS}^{\uparrow}(E, q)$	Minimum (maximum) textual similarity between E and q.
PQ	A priority queue to track the relevant objects and the nodes of the index.
H	A priority queue to store the result objects.

3.3 Proposed Space Prioritized Approaches

In this section, we present our 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, (i) In Section 3.3.1, we present a state-of-the-art algorithm as a baseline, where each query is processed individually using a space prioritized index, the IR-tree [28]; (ii) We propose a space prioritized approach, SF-Sep in Section 3.3.2 where the batch is processed by maintaining a separate priority queue for each query during the index traversal; and (iii) We propose the SF-Grp approach in Section 3.3.3, where the queries are grouped and the top-k objects of the queries are retrieved by a single traversal of the index.

3.3.1 Space Prioritized Baseline Approach (SF-BL)

In this baseline approach, each query $q \in Q$ in a batch is processed individually using an existing algorithm presented by Cong et al. [28]. The set of objects are indexed using an IR-tree, as presented in Section 2.3.

Upper Bound Relevance. As shown by Cong et al. [28], the maximum spatial-textual similarity between any node E of the IR-tree and a query q is computed as:

$$\mathrm{CS}^{\uparrow}(E, q) = \alpha \cdot \mathrm{SS}^{\uparrow}(E.l, q.l) + (1 - \alpha) \cdot \mathrm{TS}^{\uparrow}(E.d, q.d)\,,$$

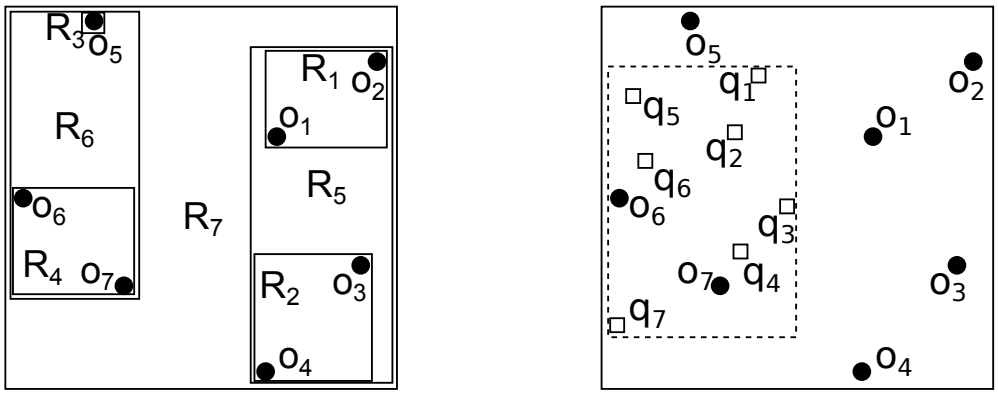
where $\mathrm{SS}^{\uparrow}(E.l, q.l)$ is the maximum spatial similarity computed from the minimum Euclidean distance between the query location and the MBR of E using Equation 3.2, and $\mathrm{TS}^{\uparrow}(E.d, q.d)$ is the maximum textual similarity, computed as:

$$\mathrm{TS}^{\uparrow}(E.d, q.d) = \sum_{t \in q.d \cap E.d} \boldsymbol{w}^{\uparrow}(E.d, t)\,,$$

where $\boldsymbol{w}^{\uparrow}(E.d, t)$ is the maximum weight of the term t in the associated document of node E. As described in Section 2.3, if E is a non-leaf node, $\boldsymbol{w}^{\uparrow}(E.d, t)$ is the maximum weight in the union of the documents contained in the subtree of E. Otherwise, $\boldsymbol{w}^{\uparrow}(E.d, t)$ is the weight of term t in document $E.d$ computed using Equation 3.3.

ALGORITHM 1: Spatial-first baseline (SF-BL)

Input: A set of queries Q, a positive integer k, and an IR-tree over the set of objects O.
Output: The top-k results for each query in Q.
Initialize an array H of $|Q|$ max-priority queues to order the top-k results for each query.
for *each* $q \in Q$ **do**
    Initialize a max-priority queue PQ.
    PQ $\leftarrow$ Enqueue(IR-tree(root), 0)
    while PQ *not empty* **do**
        $E \leftarrow$ Dequeue(PQ)
        if E *is an object* **then**
            $H_q \leftarrow$ Enqueue$(E, \mathrm{CS}^{\uparrow}(E, q))$
            if $|H_q| \geq k$ **then**
                Break
        else
            Read(E)
            for *each element* $e \in E$ **do**
                if $q.d \cap e.d \neq \varnothing$ **then**
                    PQ $\leftarrow$ Enqueue$(e, \mathrm{CS}^{\uparrow}(e, q))$
return H



(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	$(t_1, 2), (t_4, 3)$

Figure 3.2: Spatial-textual objects and queries

3.3.1.1 Algorithm

When processing a query $q \in Q$, a best-first traversal algorithm is used to retrieve the top-k objects. The pseudo-code of the baseline is shown in Algorithm 1. The algorithm uses a priority queue PQ to maintain the objects and the nodes that are yet to be visited. The value of $\text{CS}(o, q)$ is used to order PQ. In each iteration, the top node E of PQ (the most relevance node) is dequeued (line 1.8). If E is an object, E is included in the result for q (Lines 1.9 - 1.10). A priority queue H_q is maintained to order the top-k objects of each query $q \in Q$. The process for a query q terminates when k objects have been found (Lines 1.11 - 1.12). Otherwise, the elements of E are read from disk and the elements that contain at least one query keyword from $q.d$ are added to the queue (Lines 1.14- 1.17). Finally the algorithm returns the results for all of the queries in Q.

A trace of processing required for the baseline is sketched in Table 3.3. Figure 3.2a shows the locations of the objects $O = o_1, \ldots, o_7$ and the corresponding text descriptions. The IR-tree

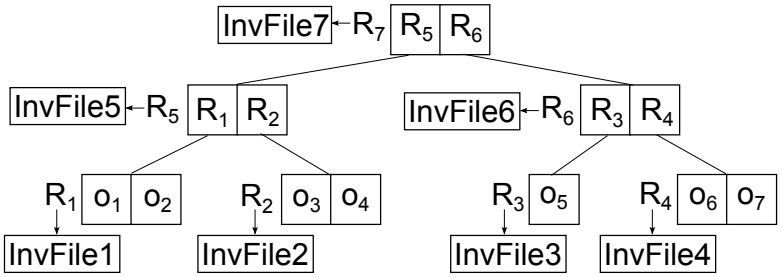


Figure 3.3: The example IR-tree

Table 3.2: 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 3.3: The example steps of the baseline approach

Query	E	**PQ**	H_q
q_1	R_7	$(R_6, 0.8), (R_5, 0.1)$	–
	R_6	$(R_3, 0.7), (R_4, 0.6), (R_5, 0.1)$	–
	R_3	$(o_5, 0.7), (R_4, 0.6), (R_5, 0.1)$	–
	o_5	$(R_4, 0.6), (R_5, 0.1)$	o_5
q_2	R_7	$(R_6, 0.7), (R_5, 0.2)$	–
	R_6	$(R_3, 0.6), (R_4, 0.5), (R_5, 0.2)$	–
	R_3	$(o_5, 0.6), (R_4, 0.5), (R_5, 0.2)$	–
	o_5	$(R_4, 0.5), (R_5, 0.2)$	o_5
q_3	R_7	$(R_5, 0.5), (R_6, 0.5)$	–
	R_5	$(R_6, 0.5), (R_1, 0.2), (R_2, 0.2)$	–
	R_6	$(R_4, 0.5), (R_3, 0.4), (R_1, 0.2), (R_2, 0.2)$	–
	R_4	$(o_7, 0.5), (o_6, 0.4), (R_3, 0.4), (R_1, 0.2), (R_2, 0.2)$	–
	o_7	$(o_6, 0.4), (R_3, 0.4), (R_1, 0.2), (R_2, 0.2)$	o_7

is constructed according to the description presented in Section 2.3, and shown in Figure 3.3. Figure 3.2b shows the locations of the queries $Q = q_1, q_2, \ldots, q_7$. The corresponding text descriptions are presented in Table 3.2. We show the steps for the queries q_1, q_2, and q_3 as example. Here, $\alpha = 0.5$, $q_1.d = \{t_1, t_3, t_4\}$, $q_2.d = \{t_1, t_3, t_4\}$, $q_3.d = \{t_1\}$, and $k = 1$.

In the baseline approach, a node of the tree might be retrieved multiple times for different queries, which can result in high I/O costs. For instance, the nodes R_3, R_4, R_5 and R_6 all are

retrieved three times from the disk, and the objects o_5 is retrieved twice in this example. To overcome this drawback, we propose efficient solutions to batch process the queries where a node is guaranteed to be retrieved at most once.

3.3.2 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 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 2. 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 3.3.2.1.

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. For each query $q \in Q$, a max-priority queue PQ_q is also maintained to track the relevant nodes and the objects, where the key is the corresponding similarity score for q, computed according to Equation 3.1. 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 2.8 - 2.9). The reasoning behind this selection method is explained along with a discussion on the effect of other selection orders in Section 3.3.2.2. 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 elements for each PQ_q (Lines 2.12 - 2.14). In this manner, all of 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 elements, then q is marked as finished, and discarded from further computations (Lines 2.15 - 2.17).

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 $q.d$ are enqueued in PQ_q. The node E is then removed from these queues (Lines 2.19 - 2.25). 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.

ALGORITHM 2: SF-SEP

2.1 **Input:** A set of top-k queries Q, a positive integer k, and an IR-tree over the set of objects O.
2.2 **Output:** The top-k results for each query in Q.
2.3 Initialize an array H of $|Q|$ min-priority queues to order the top-k results for each query.
2.4 Initialize an array PQ of $|Q|$ max-priority queues to track node traversal for each query.
2.5 **for** *each* $q \in Q$ **do**
2.6 $\quad \text{PQ}_q \leftarrow \text{ENQUEUE}(\text{IR-tree(root)}, 0)$
2.7 **while** $Q \neq \varnothing$ **do**
2.8 $\quad$ Select PQ_r randomly.
2.9 $\quad E \leftarrow \text{TOP}(\text{PQ}_r)$
2.10 $\quad$ **if** E *is an object* **then**
2.11 $\quad\quad$ **for** *each* $q \in Q$ **do**
2.12 $\quad\quad\quad$ **while** $\text{PQ}_q \neq \varnothing$ & $|H_q| < k$ & $\text{TOP}(\text{PQ}_q)$ *is an object* **do**
2.13 $\quad\quad\quad\quad o \leftarrow \text{DEQUEUE}(\text{PQ}_q)$
2.14 $\quad\quad\quad\quad H_q \leftarrow \text{ENQUEUE}(o, \text{CS}(o, q))$
2.15 $\quad\quad\quad\quad$ **if** $\text{PQ}_q = \varnothing \parallel |H_q| \geq k$ **then**
2.16 $\quad\quad\quad\quad\quad$ Mark q finished.
2.17 $\quad\quad\quad\quad\quad$ Remove q from Q.
2.18 $\quad$ **else**
2.19 $\quad\quad \text{READ}(E)$
2.20 $\quad\quad$ **for** *each* $q \in Q$ **do**
2.21 $\quad\quad\quad$ **if** $E \in \text{PQ}_q$ **then**
2.22 $\quad\quad\quad\quad$ **for** *each* $e \in E$ **do**
2.23 $\quad\quad\quad\quad\quad$ **if** $q.d \cap e.d \neq \varnothing$ **then**
2.24 $\quad\quad\quad\quad\quad\quad \text{PQ}_q \leftarrow \text{ENQUEUE}(e, \text{CS}^{\uparrow}(e, q))$
2.25 $\quad\quad\quad\quad$ Remove E from PQ_q.
2.26 Return H

3.3.2.1 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 disk for a query, the score of the object is updated for all of the queries that share the object in their corresponding queues. The other processing steps of the Algorithm 2 are the same as that of processing a single query. Therefore, this algorithm is easy to be extended to other spatial-first index structures.

3.3.2.2 Node Selection Order

In this section we discuss the selection order of the tree nodes in each iteration of our algorithm.

Random Selection. Recall that for each query q, the corresponding priority queue PQ_q is maintained according to the maximum similarity 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 2.12 - 2.14 in the same iteration. Thus, a best-first approach is applied not only to the query for which PQ_r is selected, 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 when k objects are found, or PQ_q is empty. The Lines 2.15 - 2.17 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 Line 2.12, 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 $q.d$ and PQ_q contains E. In both conditions, all of the computations are same as when selecting E randomly. The computations do not rely on the selection of E. Moreover, keeping track of the node that is the top of the maximum number of queues in each iteration requires additional computations. Therefore, the random selection of a node is preferred.

Example 1. *Table 3.4 demonstrates the execution of Algorithm 2 for the data from Figure 3.2a. The location of queries $Q = q_1, q_2, \ldots, q_7$ are shown in Figure 3.2b, and the query keywords are shown in Table 3.2. 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 similarity 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.

Table 3.4: The example steps for the SF-SEP approach (for query q_1, q_2, q_3)

Iteration	E	q	$\mathbf{PQ}_q$	H_q
Initialization	-	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	$(R_4, 0.5), (R_3, 0.4), (R_1, 0.2), (R_2, 0.2)$	–
4	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	$(R_4, 0.5), (o_5, 0.4), (R_1, 0.2), (R_2, 0.2)$	–
6	R_4	q_1	–	o_5
		q_2	–	o_5
		q_3	$(o_7, 0.5), (o_6, 0.4), (o_5, 0.4), (R_1, 0.2), (R_2, 0.2)$	–
7	o_7	q_1	–	o_5
		q_2	–	o_5
		q_3		o_7

3.3.3 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 the queries are grouped into a batch, and the MIR-tree is traversed for that group. Several pruning strategies are applied, and a node or an object is accessed at most once. Thus, the overall computation is reduced by sharing the

Table 3.5: Notations used in Section 3.3.3

Symbol	**Description**
q^+	Super-query.
$q^+.l$	Location of super-query.
$q^+.d^\cap$, $q^+.d^\cup$	Intersection (union) of the text of the queries in super-query.
$\text{CS}^\downarrow(E, q^+)$, $\text{CS}^\uparrow(E, q^+)$	Minimum (maximum) combined spatial-textual similarity between a node E of an index and q^+.
$\text{SS}^\downarrow(E, q^+)$, $\text{SS}^\uparrow(E, q^+)$	Minimum (maximum) spatial similarity between E and q^+.
$\text{TS}^\downarrow(E, q^+)$, $\text{TS}^\uparrow(E, q^+)$	Minimum (maximum) textual similarity between E and q^+.
$\mathcal{R}_k(q^+)$	The current k·th best minimum combined spatial-textual similarity of any object for any query $q \in q^+$.
$\mathcal{R}_k(q)$	The k·th best combined spatial-textual similarity of any object for q.
LO	A min-priority queue to keep the k objects with the best lower bounds found so far.
RO	A max-priority queue of objects whose upper bound is better than $\mathcal{R}_k(q^+)$.

Table 3.6: 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 3.7: 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	$(R_1, 1, 0), (R_2, 2, 0)$	$(R_4, 3, 0)$	$(R_5, 2, 0), (R_6, 3, 0)$

processing and I/O costs among the queries. Unlike the approach described in Section 3.3.2, we utilize a shared priority queue of objects and nodes during tree 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 5.2 presents the notation used in this Section.

3.3.3.1 Index: The 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 3.6 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 3.2a. The tree structure of the MIR-tree is the same as the IR-tree (Figure 3.3). 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.

Index Construction Cost. In contrast to the original IR-tree [28], the space requirement of the MIR-tree includes an additional weight stored for the minimum text similarity for each term in each node. Specifically, for a node N, if the number of terms is M, the additional space is required to store $\sum_{i=1}^{M} d_i$ number of weights, where d_i is the number of objects in the posting list of term t_i in node N. The construction time of the MIR-tree is very similar 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. As the split and merge of the nodes are executed in the same manner as the IR-tree, the update costs of the MIR-tree are also same as that of the IR-tree.

In this work, the proposed MIR-tree is an extension of the original IR-tree presented by Cong et al. [28], 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 of 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.

3.3.3.2 Query Grouping

Our goal is to access the necessary objects from disk, and avoid duplicate retrieval of objects for different queries. A group of queries is formed for this purpose, denoted as a "super-query" (q^+), and the objects are accessed using this group instead of individual queries.

Super-query Construction. The "super-query" (q^+) is constructed such that $q^+.l$ is the MBR enclosing the locations of all queries, $q^+.d^\cup$ is the union, and $q^+.d^\cap$ is the intersection of the keywords of all queries, respectively.

As an example, Figure 3.2b shows the locations of the queries $Q = q_1, q_2, \ldots, q_7$. The corresponding text descriptions are presented in Table 3.2. The location of the "super-query", $q^+.l$ is the MBR enclosing the locations for all the queries, shown with a dotted rectangle. Here, the intersection of the keywords of all the queries, $q^+.d^\cap$ is '`1000`' and the union, $q^+.d^\cup$ is '`1111`'.

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

3.3.3.3 Upper and Lower Bound Estimation

The maximum spatial-textual similarity between any node E of the MIR-tree and the super-query q^+ is computed as:

$$\mathrm{CS}^\uparrow(E, q^+) = \alpha \cdot \mathrm{SS}^\uparrow(E.l, q^+.l) + (1 - \alpha) \cdot \mathrm{TS}^\uparrow(E.d, q^+.d^\cup)\,,$$

where $\mathrm{SS}^\uparrow$ is the maximum spatial similarity computed from the minimum Euclidean distance between the two MBRs using Equation 3.2, and $\mathrm{TS}^\uparrow$ is the maximum textual similarity between $E.d$ and the union of the keywords of the queries, $q^+.d^\cup$, computed as:

$$\mathrm{TS}^\uparrow(E.d, q^+.d^\cup) = \sum_{t \in (q^+.d^\cup \cap E.d)} \boldsymbol{w}^\uparrow(E.d, t)\,,$$

where $\boldsymbol{w}^\uparrow(E.d, t)$ is the maximum weight of the term t in the associated document of node E.

The following lemma enables us to estimate an upper bound on the similarity between any query $q \in Q$, and any object node E using the super-query q^+, where E is a node of the MIR-tree.

ALGORITHM 3: TREE TRAVERSAL OF THE SF-GRP APPROACH

Input: A set of queries Q, a positive integer k, and an MIR-tree over the set of objects O.
Output: The top-k results for each query in Q.
$q^+.l \leftarrow \underset{q \in Q}{\text{MBR}}\ (q.l)$
$q^+.d^{\cup} \leftarrow \bigcup_{q \in Q} (q.d)$
$q^+.d^{\cap} \leftarrow \bigcap_{q \in Q} (q.d)$
Initialize max-priority queue PQ, RO, min-priority queue LO
$E \leftarrow$ MIR-tree(root)
ENQUEUE(PQ, E, CS$^{\downarrow}(E, q^+)$)
while PQ $\neq \varnothing$ **do**
  $E \leftarrow$ DEQUEUE(PQ)
  if E *is leaf* **then**
    if |LO| $< k$ **then**
      ENQUEUE(LO, E, CS$^{\downarrow}(E, q^+)$)
      if |LO| $= k$ **then**
        $\mathcal{R}_k(q^+) \leftarrow$ CS$^{\downarrow}$(TOP(LO), q^+)
    else if $CS^{\uparrow}(E, q^+) \geq \mathcal{R}_k(q^+)$ **then**
      ENQUEUE(LO, E, CS$^{\downarrow}(E, q^+)$)
      $o \leftarrow$ DEQUEUE(LO)
      $\mathcal{R}_k(q^+) \leftarrow$ CS$^{\downarrow}(o, q^+)$
      if $CS^{\uparrow}(o, q^+) \geq \mathcal{R}_k(q^+)$ **then**
        ENQUEUE(RO, o, CS$^{\uparrow}(o, q^+)$)
  else
    if |LO| $< k \parallel CS^{\uparrow}(E, q^+) \geq \mathcal{R}_k(q^+)$ **then**
      READ(E)
      for *each* $e \in E$ **do**
        ENQUEUE(PQ, e, CS$^{\downarrow}(e, q^+)$)
return INDIVIDUAL_TOPK(Q, k, q^+, LO, RO)

Lemma 1. *$\forall q \in Q$, $CS^{\uparrow}(E, q^+)$ is an upper bound estimation of $CS(E, q)$, such that, for any object node E, $CS(E, q) \leq CS^{\uparrow}(E, q^+)$.*

Proof. Recall that the $q^+.l$ 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$^{\uparrow}(E, q^+)$ is the maximum spatial similarity computed from the minimum Euclidean distance between the two MBRs of E and q^+ using Equation 3.2. As the location $q.l$ of any query $q \in Q$ is inside the rectangle $q^+.l$, the value SS(E, q) must be less than or equal to SS$^{\uparrow}(E, q^+)$. For textual similarity, as $q^+.d^{\cup} = \cup_{q \in Q} q.d$,

the maximum textual similarity score between any query $q \in Q$, and any object node E that can be achieved is $\text{TS}^{\uparrow}(E, q^+)$ from Equation 3.3. Since the spatial-textual score $\text{CS}(E, q)$ is the weighted sum of the corresponding spatial and textual scores, $\forall q \in Q$, the score $\text{CS}(E, q)$ must also be less than or equal to $\text{CS}^{\uparrow}(E, q^+)$. □

Lemma 1 guarantees that $\text{CS}^{\uparrow}(E, q^+)$ is a correct upper bound estimation of the similarity between a node E of the MIR-tree and any $q \in Q$, as the similarity score $\text{CS}(E, q)$ is always less than $\text{CS}^{\uparrow}(E, q^+)$. Similarly, a lower bound similarity can be computed as:

$$\text{CS}^{\downarrow}(E, q^+) = \alpha \cdot \text{SS}^{\downarrow}(E.l, q^+.l) + (1 - \alpha) \cdot \text{TS}^{\downarrow}(E.d, q^+.d^{\cap})$$

where $\text{SS}^{\downarrow}(E.l, q^+.l)$ is the minimum spatial similarity computed from the maximum Euclidean distance between the two MBRs and $\text{TS}^{\downarrow}(E.d, q^+.d^{\cap})$ is the minimum textual similarity between E and $q^+.d^{\cap}$ computed using the minimum weights of the terms in E. Similar to the upper bound estimation, the property $\forall q \in Q, \text{CS}(E, q) \geq \text{CS}^{\downarrow}(E, q^+)$ always holds for the lower bound estimation.

3.3.3.4 Algorithm

In this approach, 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: (i) a careful tree traversal; and (ii) an efficient top-k object computation of the individual queries. We utilize q^+ to access the MIR-tree and share I/O costs, and the similarity 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 3. Finally, the top-k results of the individual queries are verified by applying several pruning strategies as presented in Algorithm 4.

Filtering Step: Tree Traversal. The pseudocode is presented in Algorithm 3. Here, the MIR-tree is traversed for the super-query q^+ instead of the individual queries. Lines 3.3 - 3.5 show the construction of q^+ from Q. Initially, a max-priority queue PQ is created to keep track of the nodes that are yet to be visited, where the key is the lower bound similarity $\text{CS}^{\downarrow}$ w.r.t. q^+. A min-priority queue LO is also maintained to keep the k objects with the best lower bounds found so far. Lines 3.12 - 3.15 show how LO is initially filled up with k objects according to their lower bounds. The actual objects are used instead of object nodes in LO for

a better estimation of similarity. The k·th best lower bound similarity score, $\mathcal{R}_k(q^+)$ found so far is also maintained.

Since the score $\mathcal{R}_k(q^+)$ is the k·th best lower bound score for any query $q \in Q$, the similarity score of any unseen object o must be greater than or equal to $\mathcal{R}_k(q^+)$ to be one of the top-k objects of q. Therefore, we need to consider only those nodes E, where $\mathrm{CS}^{\uparrow}(E, q^+) \geq \mathcal{R}_k(q^+)$. 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 $\mathcal{R}_k(q^+)$ is also updated accordingly. If the object o dequeued from LO in this adjustment process has a better upper bound than the updated $\mathcal{R}_k(q^+)$, o is stored in a priority queue RO (shown is Lines 3.18 -3.21). Here, RO is a max-priority queue where the key is the upper bound similarity score w.r.t. q^+. 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 3.24 - 3.26. Finally, the objects in LO and RO are used to compute the top-k objects for individual queries in a later step (Line 3.27).

The MIR-tree is traversed 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 2. *The objects $O = o_1, o_2, \ldots, o_7$ in Figure 3.2a are indexed with an MIR-tree as shown in Figure 3.3. The queries $Q = q_1, q_2, \ldots, q_7$ are shown in Figure 3.2b, where the dotted rectangle is the MBR of the queries ($q^+.l$). Table 3.6 and Table 3.2 present the text descriptions. Let $k = 1$ and $\alpha = 0.5$. 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 $|LO| < k$, $LO : o_7$, $\mathcal{R}_k(q^+) = 0.7$*
5. *Dequeue R_3, as $CS^{\uparrow}(R_3, q^+) = 0.8$, enqueue o_5 in PQ*
   $PQ : (o_5, 0.7), (o_6, 0.5), (R_5, 0.3)$
6. *Dequeue o_5, as $CS^{\uparrow}(o_5, q^+) = 0.8$, enqueue o_5 in RO*
   $LO : o_7$, $RO : o_5$, $\mathcal{R}_k(q^+) = 0.7$; $PQ : (o_6, 0.5), (R_5, 0.3)$

7. *Dequeue* o_6*, as* $CS^{\uparrow}(o_6, q^+) = 0.9$*, enqueue* o_5 *in RO*
   $LO : o_7$*,* $RO : o_6, o_5$*,* $\mathcal{R}_k(q^+) = 0.7$*;* $PQ : (R_5, 0.3)$

8. *Dequeue* R_5*, as* $CS^{\uparrow}(R_5, q^+) = 0.6 < \mathcal{R}_k(q^+)$*, discard.*

ALGORITHM 4: Verification step of SF-Grp approach

```
4.1  Input: A set of queries Q, a positive integer k, a max-priority queue RO, and a
     min-priority queue LO.
4.2  Output: The top-k results for each query in Q.
4.3  Initialize an array H of |Q| min-priority queues for each q ∈ Q.
4.4  for each q ∈ Q do
4.5      for each o ∈ LO do
4.6          Enqueue(H_q, o, CS(o, q))
4.7      R_k(q) ← CS(Top(H_q), q)
4.8      for each o ∈ RO do
4.9          if CS↑(o, q+) < R_k(q+) then
4.10             Break
4.11         else if CS(o, q) ≥ R_k(q) then
4.12             Enqueue(H_q, o, CS(o, q))
4.13             Dequeue(H_q)
4.14             R_k(q) ← CS(Top(H_q), q)
4.15 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 4 summarizes this process. For each $q \in Q$, a min-priority queue H_q of objects is initialized where the key is the total similarity score of the object w.r.t. q. For each q, the similarity 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, $\mathcal{R}_k(q)$ computed so far is also stored (Lines 4.5 - 4.7).

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. q^+ is less than $\mathcal{R}_k(q)$, then o cannot be a top-k object of q. The subsequent objects of o in RO can also be pruned from consideration (Line 4.9) as RO is maintained in ascending order of the upper bound scores w.r.t. q^+. Otherwise, if $\mathrm{CS}(o, q) \geq \mathcal{R}_k(q)$, o is inserted into H_q. H_q is adjusted such that it contains k objects with the highest similarity scores, and $\mathcal{R}_k(q)$ is updated accordingly as shown in Lines 4.11 - 4.14. Finally, the priority

Table 3.8: Notation used in Section 3.4

Symbol	**D**escription
B	Block size of a posting list in a SIF index.
$\mathrm{ID}^{\Uparrow}$	The largest object ID in O.
$\boldsymbol{w}^{\uparrow}(O,t)$, $\boldsymbol{w}^{\uparrow}(b,t)$	Maximum text weight of a term t in the set of objects O (objects in block b).
MBR_t, $\mathrm{MBR}_{b,t}$	Minimum bounding rectangle of the objects in the posting list of t (in a block of the posting list of t).
CP_t	A pointer to the current posting of t.
νt	Pivot term, for which the accumulated maximum weight of the terms exceeds the threshold $\mathcal{R}_k(q)$.
ν	Pivot, the object ID of the pointer of the pivot term.
TL	The set of terms preceding νt, where the terms are sorted by the object IDs pointed by the corresponding pointers.
fst	First term of the sorted posting lists.
TU	All unique terms of $q \in Q$.
$\mathrm{CS}_b^{\uparrow}(\mathrm{TL}, q)$	The maximum combined spatial-textual similarity using the block level upper bounds of the terms in TL.
$\mathrm{SS}_b^{\uparrow}(\mathrm{TL}, q.l)$, $\mathrm{TS}_b^{\uparrow}(\mathrm{TL}, q.d)$	The maximum spatial (textual) similarity using the block level upper bounds of the terms in TL
$\mathrm{CS}_{\ell}^{\uparrow}(o, q)$, $\mathrm{TS}_{\ell}^{\uparrow}(o.d, q.d)$	The maximum combined spatial-textual similarity (textual similarity) using the location lookup table and the block level upper bounds where object o is stored.
$\nu^{\Downarrow}$	The minimum pivot ID from the current pivots of $q \in Q$.
$q^{\Downarrow}$	The query for which $\nu^{\Downarrow}$ is selected.

queue H_q for each query $q \in Q$ contains its top-k objects in reverse order, and $\mathcal{R}_k(q)$ is the spatial-textual similarity score of the k·th ranked object of the corresponding query.

Example 3. *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* $(o_i,$ $CS^{\downarrow}(o, q^+)$, $CS^{\uparrow}(o, q^+))$. *First, object* o_7 *from* LO *is considered. As* $CS(o_7, q_6) = 0.75$, *so,* $\mathcal{R}_k(q_6)$ *becomes* 0.75 *and* $H_{q_6} : o_7$. *Then the objects in* RO *are considered. Here,* $CS(o_6, q_6) = 0.85$, *so* $\mathcal{R}_k(q_6)$ *becomes* 0.85 *and* $H_{q_6} : o_6$ *as object* o_6 *has a better similarity score than* o_7 *w.r.t. query* q_6. *As the upper bound of the next object of* RO, $CS^{\uparrow}(o_5, q^+)$ *is* 0.8, *which is less than the current value of* $\mathcal{R}_k(q_6)$, *the processing for query* q_6 *terminates. The top-*1 *object of* q_6 *is* o_6, *where* $\mathcal{R}_k(q_6) = 0.85$. *The process is repeated for all* $q \in Q$.

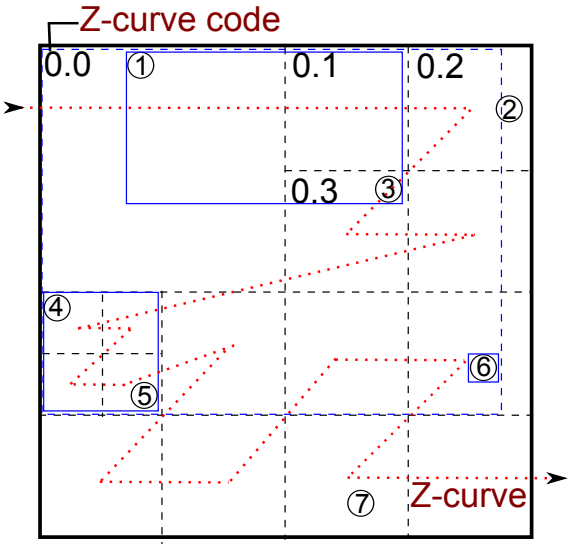


Figure 3.4: Z-curve ordering of objects

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 3.9: Z-order IDs and term frequencies

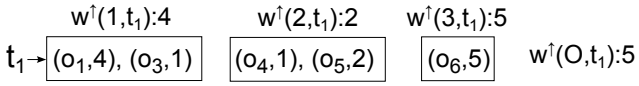


Figure 3.5: Posting list for the term 't_1' in SIF index (block size = 2)

3.4 Text Prioritization Approach

As discussed in Section 3.6, 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 (Section 2.2.3) is a state-of-the-art document-at-a-time, top-k algorithm commonly used in inverted indexes, we use this algorithm as a basis for our approach. In particular, we illustrate the mapping from textual-inverted-file to our proposed *Spatial Inverted File* (SIF) (Section 3.4.1), 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 (Section 3.4.2). Table 3.8 shows the notations used to present our approach in this Section.

3.4.1 Proposed Index: Spatial Inverted File

Recall the inverted file described in Section 2.2.3. 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

similarity 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 3.4. Figure 3.4 shows the locations of the same 7 objects $O = \{o_1, o_2, \ldots, o_7\}$ from Figure 3.2a, 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 [90], highlighted as the red dotted lines in Figure 3.4. The object IDs are assigned based on their position on this curve. Table 3.9 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 *location lookup table* 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, referred to as '*block lookup table*'. The size of the lookup tables are 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:

- 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 $\boldsymbol{w}^{\uparrow}(O, t)$. In this chapter, the textual similarity score for each term is computed using Equation 3.3.
- 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.

Similarly, for each block b of the posting list of term t, the following two values are maintained:

- The maximum textual similarity of t of the objects stored in b, denoted as $\boldsymbol{w}^{\uparrow}(b, t)$; and
- The block-level MBR – $\text{MBR}_{b,t}$ which is the minimum bounding rectangle that encloses all of the locations for the objects stored in block b.

Example 4. *Figure 3.5 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 3.3). In this example, the entries of the block lookup table 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 3.4, and the block-level MBR for each of the three blocks is highlighted by the three blue rectangles of solid borders respectively.

In this work, 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.

Comparison with SFC-Skip. Christoforaki et al. proposed an index called SFC-Skip [26] 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. In contrast, 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 query.

To summarize, inverted files are the most widely used index in information retrieval systems, and the SIF index along with the our traversal techniques 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.

3.4.2 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 3.4.3.

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 is computed as:

$$\mathrm{CS}^{\uparrow}(\mathrm{TL}, q) = \alpha \cdot \mathrm{SS}^{\uparrow}(\mathrm{TL}, q.l) + (1 - \alpha) \cdot \mathrm{TS}^{\uparrow}(\mathrm{TL}, q.d)\,, \tag{3.4}$$

where TL is a set of terms such that TL $\subseteq q.d$. In Section 3.4.3.1 and Algorithm 6, we will describe how TL is obtained. The upper bound of spatial similarity, $\mathrm{SS}^{\uparrow}(\mathrm{TL}, q.l)$ is computed as:

$$\mathrm{SS}^{\uparrow}(\mathrm{TL}, q.l) = \max_{t \in \mathrm{TL}} \mathrm{SS}^{\uparrow}(\mathrm{MBR}_t, q.l)$$

For each term $t \in \mathrm{TL}$, $\mathrm{SS}^{\uparrow}(\mathrm{MBR}_t, q.l)$ is computed in the same way as Equation 3.2:

$$\mathrm{SS}^{\uparrow}(\mathrm{MBR}_t, q.l) = 1 - \frac{\boldsymbol{d}^{\downarrow}(\mathrm{MBR}_t, q.l)}{\boldsymbol{d}^{\Uparrow}}, \tag{3.5}$$

where the notations $\boldsymbol{d}^{\downarrow}$ and $\boldsymbol{d}^{\Uparrow}$ carry the same meaning as Equation 3.2. As $\mathrm{SS}^{\uparrow}(\mathrm{MBR}_t, q.l)$ is computed from the minimum Euclidean distance between the rectangle MBR_t and the query location, $\mathrm{SS}^{\uparrow}(\mathrm{MBR}_t, q.l)$ 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 $\mathrm{TS}^{\uparrow}(\mathrm{TL}, q.d)$ is computed as the summation of maximum weights $\boldsymbol{w}^{\uparrow}(O, t)$ contributed by the posting list of each terms $t \in \mathrm{TL}$, i.e.,

$$\mathrm{TS}^{\uparrow}(\mathrm{TL}, q.d) = \sum_{t \in \mathrm{TL}} \boldsymbol{w}^{\uparrow}(O, t)$$

Block-level Upper Bound. As 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.

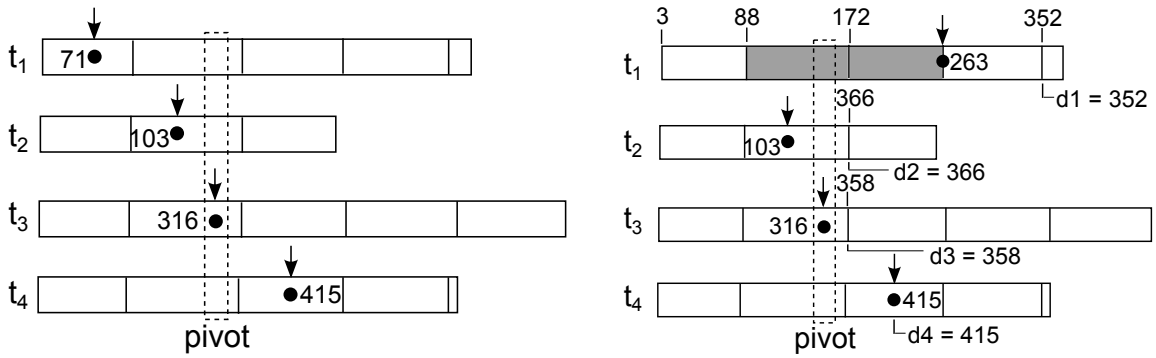
$$\mathrm{CS}_b^{\uparrow}(\mathrm{TL}, q) = \alpha \cdot \mathrm{SS}_b^{\uparrow}(\mathrm{TL}, q.l) + (1 - \alpha) \cdot \mathrm{TS}_b^{\uparrow}(\mathrm{TL}, q.d) \tag{3.6}$$

Here, b is a block in the inverted list of a term $t \in \mathrm{TL}$ that is currently being traversed, and TL $\subseteq q.d$. Let BL be the set of such blocks, one for each term of TL. For a block b, $\mathrm{SS}^{\uparrow}(\mathrm{MBR}_{b,t}, q.l)$ is computed from the minimum Euclidean distance between the $\mathrm{MBR}_{b,t}$ and $q.l$ using Equation 3.5. $\mathrm{SS}_b^{\uparrow}(\mathrm{TL}, q.l)$ is the maximum spatial similarity contributed by all $b \in \mathrm{BL}$. So,

$$\mathrm{SS}_b^{\uparrow}(\mathrm{TL}, q.l) = \max_{b \in BL} \{\mathrm{SS}^{\uparrow}(\mathrm{MBR}_{b,t}, q.l)\}$$

The upper bound of text similarity $\mathrm{TS}_b^{\uparrow}(\mathrm{TL}, q.d)$ is computed as the sum of the maximum weights $\boldsymbol{w}^{\uparrow}(b, t)$ contributed by each block $b \in \mathrm{BL}$ of the corresponding term $t \in \mathrm{TL}$, i.e.,

$$\mathrm{TS}_b^{\uparrow}(\mathrm{TL}, q.d) = \sum_{b \in BL} \boldsymbol{w}^{\uparrow}(b, t)$$



(a) Selecting Pivot (b) Skipping

Figure 3.6: List traversal for a single query.

Location Table-based Block-level Upper Bound. As the locations of the objects are stored separately in a location lookup table, 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 $t \in q.d$, where object o is stored. Let BO be the set of such blocks, one for each $t \in q.d$. This upper bound $\mathrm{CS}_\ell^\uparrow(o, q)$ is computed as:

$$\mathrm{CS}_\ell^\uparrow(o, q) = \alpha \cdot \mathrm{SS}(o.l, q.l) + (1 - \alpha) \cdot \mathrm{TS}_\ell^\uparrow(o.d, q.d)$$

where $\mathrm{TS}_\ell^\uparrow(o.d, q.d) = \sum_{b \in BL} \boldsymbol{w}^\uparrow(b, t)$. Here, the text component of both $\mathrm{CS}_\ell^\uparrow(o, q)$ and $\mathrm{CS}_b^\uparrow(\mathrm{TL}, q)$ are computed in similar manner, but $\mathrm{CS}_\ell^\uparrow(o, q)$ uses the exact location of o instead of the MBRs. Therefore, for any object o, the bound $\mathrm{CS}_\ell^\uparrow(o, q)$ is tighter than the corresponding bound $\mathrm{CS}_b^\uparrow(\mathrm{TL}, q)$.

3.4.3 Index Traversal Methods

In this section, we first describe a basic index traversal method using SIF for a single query, and show how unnecessary blocks can be skipped using the upper bounds in Section 3.4.2. Then we present an advanced index traversal method designed to answer multiple queries in a batch.

3.4.3.1 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 [11, 32], which improves the performance of Wand by enabling skipping blocks of

ALGORITHM 5: *GEOBW*

Input: A query q, a positive integer k, and an SIF over the set of objects O.
Output: The top-k result for q.
Initialize a min-priority queue H of size k for the current top-k objects.
$\mathrm{ID}^{\Uparrow} \leftarrow$ The largest object ID in O
$\mathcal{R}_k(q) \leftarrow 0$
for *each* $t \in q.d$ **do**
    $\mathrm{CP}_t \leftarrow$ First object ID of the posting list of t.
end $\leftarrow$ False
while *end is* False **do**
    Sort posting lists by CP_t.
    $\nu\mathrm{t} \leftarrow \mathrm{FindPivotTerm}(\mathcal{R}_k(q), q)$
    if $\nu\mathrm{t} = \varnothing$ **then** end $\leftarrow$ True
    $\nu \leftarrow \mathrm{CP}_{\nu\mathrm{t}}$
    if $\nu \geq \mathrm{ID}^{\Uparrow}$ **then** end $\leftarrow$ True
    for *each* $t \in q.d$, *where* $\mathrm{CP}_t < \nu$ **do**
        Forward CP_t, skip blocks containing only object ID $< \nu$
    if $CS_{\ell}^{\uparrow}(\nu, q) \geq \mathcal{R}_k(q)$ **then**
        fst $\leftarrow$ First term of the sorted posting lists.
        if $\mathrm{CP}_{\mathrm{fst}} = \nu$ **then**
            /* All the preceding lists contain the pivot object */
            Read blocks pointed by each $\mathrm{CP}_t \leq \nu$.
            Enqueue(H, ν, CS(ν, q))
            if $|\mathrm{H}| > k$ **then**
                Dequeue(H)
            $\mathcal{R}_k(q) \leftarrow$ CS(Top(H), q)
            Move all CP_t to the next object ID $> \nu$.
        else
            Choose term t with the largest IDF, and $\mathrm{CP}_t < \nu$. Move CP_t to the next object ID $\geq \nu$.
    else
        TL $\leftarrow$ All terms $t \in q.d$ where $\mathrm{CP}_t \leq \nu$.
        if $CS_{b}^{\uparrow}(\mathrm{TL}, q) < \mathcal{R}_k(q)$ **then**
            $d' \leftarrow$ First object ID of the block next to ν.
            Choose term t with the largest IDF, where $\mathrm{CP}_t \leq \nu$. Move CP_t to the next object ID $\geq d'$.
        else
            Choose term t with the largest IDF, where $\mathrm{CP}_t \leq \nu$. Move CP_t to the next object ID $> \nu$.
Return H

ALGORITHM 6: *FindPivotTerm*$(\mathcal{R}_k(q), q)$

```
6.1 TL ← ∅
6.2 /* Posting lists are sorted by CP_t. */
6.3 for each t ∈ q.d do
6.4 |   TL ← TL ∪ t
6.5 |   if CS↑(TL, q) ≥ R_k(q) then
6.6 |   |   Return t
6.7 Return ∅
```

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 5.

Given a query q (in form of $(q.l, q.d, k)$), where $q.l$ is a location, $q.d$ is the text description, and k is the number of objects to be returned, we maintain a min-priority queue H of objects of size at most k (Line 5.3) to order the top-k objects found so far. The key is the spatial-textual similarity score with respect to q, $\text{CS}(o, q)$, computed using Equation 3.1. Let $\mathcal{R}_k(q)$ be the spatial-textual similarity score of the current k·th ranked object in H.

Similar to both Wand and Block-max Wand, for each query term $t \in q.d$, a pointer to the current posting in the list, CP_t is maintained. Each pointer CP_t is initialized by pointing to the first element of its corresponding posting list (Lines 5.6 - 5.7). After initialization, the candidate object with the smallest ID that can be a top-k object, called the *pivot* (ν), 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 3.6, where the query terms $q.d = \{t_1, t_2, t_3, t_4\}$. Note that, 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 CP_t. An example using Figure 3.6a is presented below.

Example 5. *Figure 3.6a shows the pointer for each of the four (blocked) posting lists, which are arranged in ascending order of their CP_t. To determine the pivot object, we start computing the $CS^{\uparrow}(TL, q)$ for the lists from top to bottom, until we reach a score no less than $\mathcal{R}_k(q)$. This computation is shown in Algorithm 6. If no such condition occurs, then the algorithm terminates as no object can be better than the current top-k objects. In Figure 3.6a, suppose*

that $CS^{\uparrow}(TL, q) \geq \mathcal{R}_k(q)$ *happens for the third list from the top, the posting list of term* t_3*. The object ID* 316 *pointed by the pointer* CP_t *is the pivot object, i.e.,* $\nu = 316$*.*

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 block lookup table, which stores the first object ID for each block in the same sequence as they are stored in the posting lists. So the block lookup table can be used to skip unused blocks. For example in Figure 3.6b, the shaded blocks are skipped for the list of t_1 where only the objects with ID less than $\nu = 316$ are stored. All of the pointers are forwarded accordingly, and now point to blocks where the pivot ID may be found (Lines 5.15 - 5.16). 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 to be a top-k object. In this work, we adopt a tighter bound than BLOCK-MAX WAND, by using the object locations stored in the location lookup table. Recall the location lookup table is used to extend the block-level upper bound introduced at the end of Section 3.4.2, where o is the pivot object. This can be used to compute the bound $\mathrm{CS}_{\ell}^{\uparrow}(\nu, q)$ based on the exact location of the pivot from the location lookup table, and the $\boldsymbol{w}^{\uparrow}(b, t)$ of the blocks for which the current pointers CP_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 $\mathrm{CS}_{\ell}^{\uparrow}(\nu, q) \geq \mathcal{R}_k(q)$, 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 5.17 - 5.19). As shown in Lines 5.20 - 5.22, the corresponding blocks are retrieved from disk, and the score of the pivot object, $\mathrm{CS}(\nu, q)$, is computed using Equation 3.1. The priority queue H and the value $\mathcal{R}_k(q)$ is updated accordingly (Lines 5.23 - 5.25). All the pointers with $\mathrm{CP}_t \leq \nu$ are moved to the next object with ID greater than this pivot (Line 5.26).

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 5.28 - 5.28). Selecting the pointer of the term with the highest IDF is shown by Broder et al. [8] to have a

better gain in skipping. If $\mathrm{CS}_{\ell}^{\uparrow}(\nu, q) < \mathcal{R}_k(q)$, 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 compute the bound $\mathrm{CS}_{b}^{\uparrow}(\mathrm{TL}, q)$ using the MBRs and the $\boldsymbol{w}^{\uparrow}(b, t)$ for the set of corresponding blocks. If the condition $\mathrm{CS}_{b}^{\uparrow}(\mathrm{TL}, q) < \mathcal{R}_k(q)$ 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 5.31 - 5.33) in the following example.

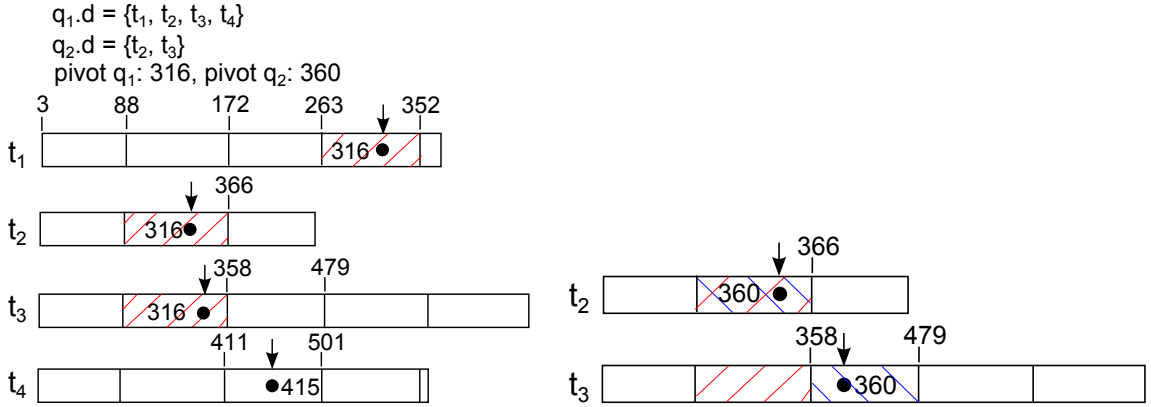
Example 6. *In Figure 3.6b, suppose that after computing* $CS_{b}^{\uparrow}(TL, q)$ *using the maximum values of the blocks, we find that pivot* 316 *cannot be a top-k object. Let* $d_1 = 352$ *be the first object ID of the successor block of* t_1*. Similarly, the other block boundaries for the posting lists appearing before the pivot is determined. As the pointers in postings lists beyond the pivot are already greater than or equal to the current pivot ID, the block boundaries of those lists is taken as their current object ID. Here,* $d_4 = 415$ *is the current object ID in the fourth list. 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 3.4.2 that $\mathrm{CS}_{\ell}^{\uparrow}(\nu, q)$ is a tighter upper bound than $\mathrm{CS}_{b}^{\uparrow}(\mathrm{TL}, q)$ for the pivot object, so $\mathrm{CS}_{b}^{\uparrow}(\mathrm{TL}, q) \geq \mathrm{CS}_{\ell}^{\uparrow}(\nu, q) \geq \mathrm{CS}(\nu, q)$. So, if $\mathrm{CS}_{\ell}^{\uparrow}(\nu, q) \geq \mathcal{R}_k(q)$ holds, then $\mathrm{CS}_{b}^{\uparrow}(\mathrm{TL}, q) \geq \mathcal{R}_k(q)$ 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 5.12); or (ii) all of the objects up to the maximum object ID in the dataset have been either processed or skipped (Line 5.14).

3.4.3.2 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 7 shows the pseudocode of our proposed approach, TF-MBW, to answer multiple queries as a batch. For each query $q \in Q$, a separate priority queue H_q is maintained that stores the current top-k objects of that query (Line 7.4).

Let TU be the union of the terms of all the queries in Q. We maintain a buffer of size $|\mathrm{TU}|$, denoted as Buf_t to keep the last accessed block for each term $t \in \mathrm{TU}$ by any query, or



(a) Retrieving blocks for q_1 (b) Retrieving and sharing blocks for q_2

Figure 3.7: List traversal for multiple queries.

more specifically, only one block for each $t \in$ TU. For each query $q \in Q$, the spatial-textual similarity score $\mathcal{R}_k(q)$ of the currently k·th ranked object, and the pointers $\text{CP}_{t,q}$ are also maintained (Line 7.7). We initialize the pivot object for each individual query ν_q once, in the same way as described in Section 3.4.3.1. If we reach the termination condition for a query q, there is no object left that can be a top-k object of that query, so we can exclude it from Q (Lines 7.8 - 7.14).

Let $\nu^{\Downarrow}$ be the smallest pivot ID among the current pivots for any of the queries $q \in Q$, and $q^{\Downarrow}$ be the corresponding query for which $\nu^{\Downarrow}$ is selected. In each iteration, we take $\nu^{\Downarrow}$ and process the query $q^{\Downarrow}$ for that pivot (Lines 7.16 - 7.18). While retrieving any block that is required to compute the total score of $\nu^{\Downarrow}$, one of the following two conditions can occur: (i) the block that contains the current $\nu^{\Downarrow}$ 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, and then the block is stored in *Buf* for the corresponding term after the computation. If condition (i) holds, then the block must be found 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 any previous step. For multiple queries, as we process the minimum of all the pivot IDs in each iteration, the $\nu^{\Downarrow}$ of a step is also greater than or equal to the minimum pivot ID, $\nu^{\Downarrow}$ of any previous step. Thus, the objects less than the minimum pivot ID, $\nu^{\Downarrow}$ 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

ALGORITHM 7: *TF-MBW*

7.1 **Input:** A set of queries Q, a positive integer k, and an SIF over the set of objects O.
7.2 **Output:** The top-k results for each query in Q.
7.3 Initialize an array H of $|Q|$ min-priority queues
7.4 TU $\leftarrow \bigcup_{q \in Q} q.d$
7.5 Initialize an array Buf of |TU| blocks
7.6 **for** *each* $q \in Q$ **do**
7.7 Execute Lines 5.5 - 5.7 of Algorithm 5
7.8 Sort posting lists by $\text{CP}_{t,q}$
7.9 $\nu\text{t} \leftarrow \text{FindPivotTerm}(\mathcal{R}_k(q), q)$
7.10 **if** $\nu\text{t} = \varnothing$ **then**
7.11 $Q \leftarrow Q - q$
7.12 $\nu_q \leftarrow \text{CP}_{\nu\text{t},q}$
7.13 **if** $\nu_q < \text{ID}^{\Uparrow}$ **then**
7.14 $Q \leftarrow Q - q$
7.15 **while** $Q \neq \varnothing$ **do**
7.16 $\nu^{\Downarrow} \leftarrow \min_{q \in Q}(\nu_q)$
7.17 $q^{\Downarrow} \leftarrow$ the query for which $\nu^{\Downarrow}$ is selected.
7.18 Execute Lines 5.14 - 5.17 of Algorithm 5 for $q^{\Downarrow}$
7.19 **for** *each* $t \in TU$, *where* $\text{CP}_{t,q^{\Downarrow}} \leq \nu_{q^{\Downarrow}}$ **do**
7.20 **if** Block pointed by $\text{CP}_{t,q^{\Downarrow}}$ NOT retrieved before **then**
7.21 $b \leftarrow$ Retrieve block pointed by $\text{CP}_{t,q^{\Downarrow}}$
7.22 Mark b as retrieved
7.23 $\text{Buf}_t \leftarrow b$
7.24 Execute Lines 5.20 - 5.26 of Algorithm 5 for $q^{\Downarrow}$
7.25 Execute Lines 7.8 - 7.14 of Algorithm 7 for $q^{\Downarrow}$
7.26 Return H

containing the current $\nu^{\Downarrow}$ is retrieved for any prior $\nu^{\Downarrow}$, 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 $q^{\Downarrow}$ as described for GEOBW (Line 7.24). The pivot object of the $q^{\Downarrow}$ is computed for the next iteration as shown in Line 7.25. We now illustrate I/O sharing among queries with the following example.

Example 7. *Figure 3.7 shows two queries* q_1 *and* q_2*, where* $q_1.d = \{t_1, t_2, t_3, t_4\}$ *and* $q_2.d = \{t_2, t_3\}$*. 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 minimum pivot ID,* $\nu^{\Downarrow}$ *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 3.7a) are retrieved from disk to compute this score, and these blocks are stored in the block buffer for the corresponding terms.*

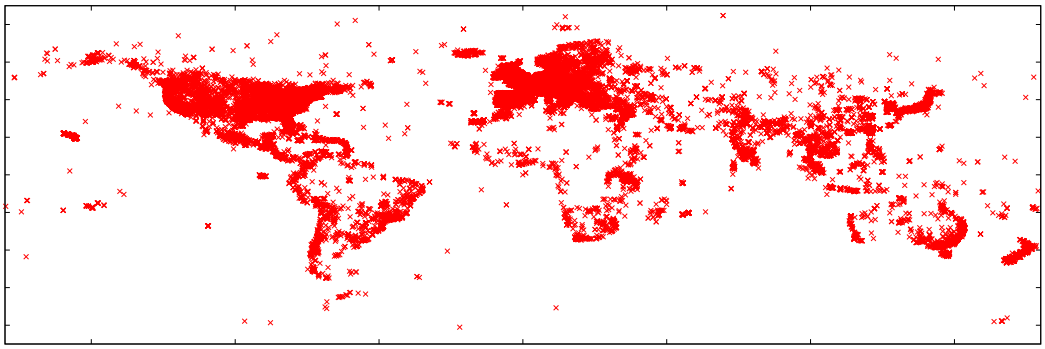
The pivot of q_1 *is computed again for the next iteration. Let, the* $\nu^{\Downarrow}$ *selected in the next iteration be* 360 *for* q_2*. After checking the conditions, suppose the score of* 360 *needs to be computed for* q_2*. The blocks that are required to be accessed for* q_2 *are shown with blue stripes in Figure 3.7b. As the block for the term* t_2 *was retrieved for* q_1 *previously, that block can be found in the block buffer, Buf for* t_2*. Thus the block shown with two color stripes (red and blue) are shared among the queries. The block stored in the block buffer for* t_3 *is not the one that is required by* q_2*. Therefore, we need to retrieve this block that contains the pivot object* 360*, and update Buf 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 $\mathcal{R}_k(q)$ 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 its top-k object, q is excluded from Q. We continue until Q is empty, which indicates that the result for all the queries have been found.

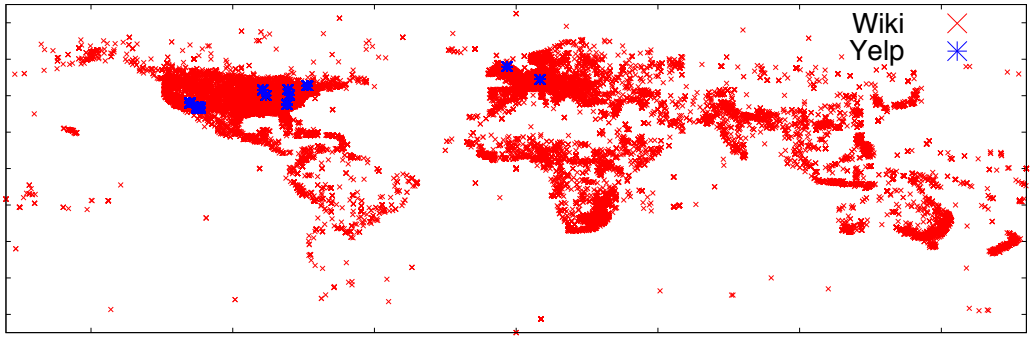
In the TF-MBW approach using the SIF index, a separate priority queue, each of size k is maintained for each query to track the current best objects, i.e., total $k \times |Q|$ objects for the batch. We also maintain a block buffer store the last recent retrieved block for each unique query term, so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

3.5 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: (i) using an IR-tree according to the approach described in Section 3.3.1, denoted as the *space-first baseline* (SF-BL) and (ii) using the GeoBW approach (3.4.3) over a SIF index presented in 3.4, denoted as the *text-first baseline* (TF-BL).



(a) Flickr dataset



(b) Wiki and Yelp dataset

Figure 3.8: Location of the Data objects

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

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 Mackenzie et al. [85], who generated a dataset from a subset of the TREC[3] ClueWeb09B collection. The documents were 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

[1] http://webscope.sandbox.yahoo.com/catalog.php?datatype=i&did=67
[2] http://www.yelp.com.au/dataset_challenge
[3] https://www.trec.nist.gov
[4] https://lemurproject.org/clueweb09/FACC1

Table 3.10: 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

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 4.4 lists the properties of the datasets. Figure 3.8 shows the location distribution of the objects, where the red and the blue points in Figure 3.8b 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 is 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 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 in the inverted lists was set to 128, and the page size was fixed at 1 kB for both 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

Table 3.11: 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	$5, 10, \mathbf{2}0, 30, 40$
MBR of batch as % of dataspace, $Area$	$1, 2, \mathbf{4}, 8, 16$
No. of Queries in a batch, $\|Q\|$	$\mathbf{1}00, 200, 400, 800, 1600$

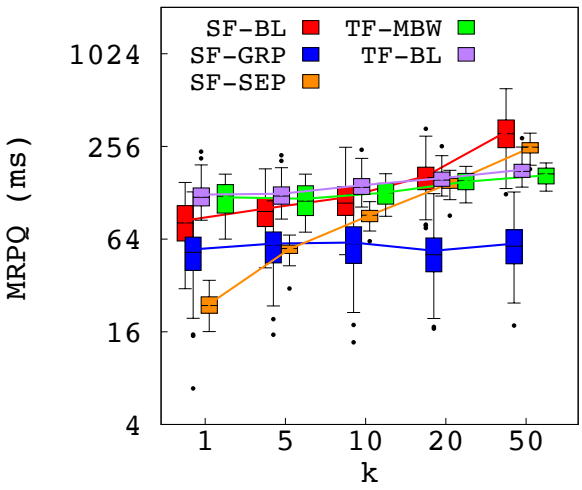
Table 3.12: % of blocks pruned for varying k in Flickr dataset

Approach \ k	1	5	10	20	50
SF-Grp	80.6	80.6	80.6	80.6	80.6
SF-Sep	86.8	82.8	77.9	71.3	59.3
TF-MBW	54.3	54.3	54.3	54.3	54.3

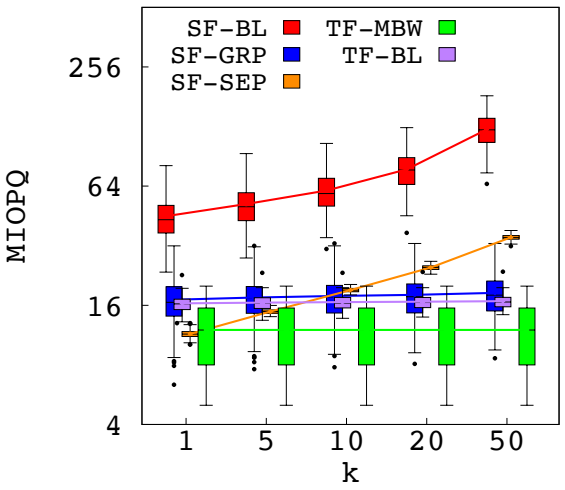
blocks contained in the list. In the experiments, the performance was evaluated using cold-start queries.

3.5.1 Performance evaluation

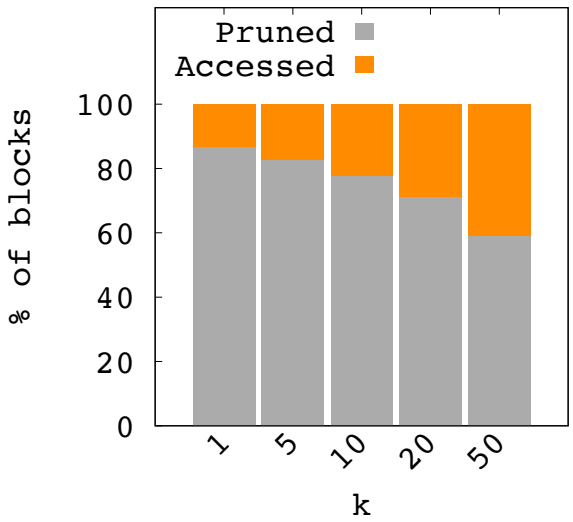
In this section, we evaluate and compare the performance of the approaches by varying several parameters. The parameter ranges are listed in Table 3.11 where the values in bold represent the default values. In all experiments, we vary a single parameter (while keeping the rest as the default settings) to study the impact on: (i) the Mean Runtime per Query (MRPQ), and (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, 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.



(a) Runtime

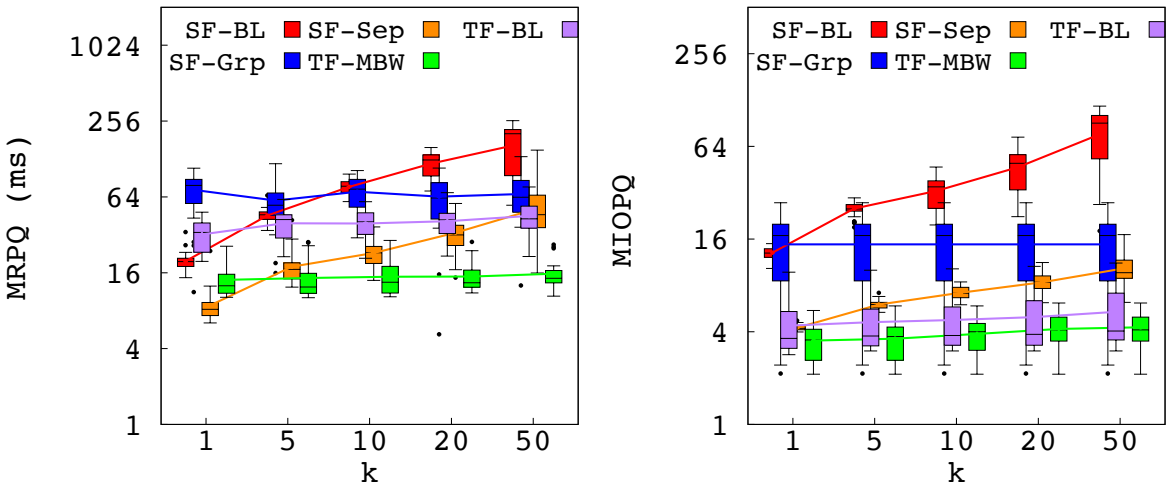


(b) I/O cost



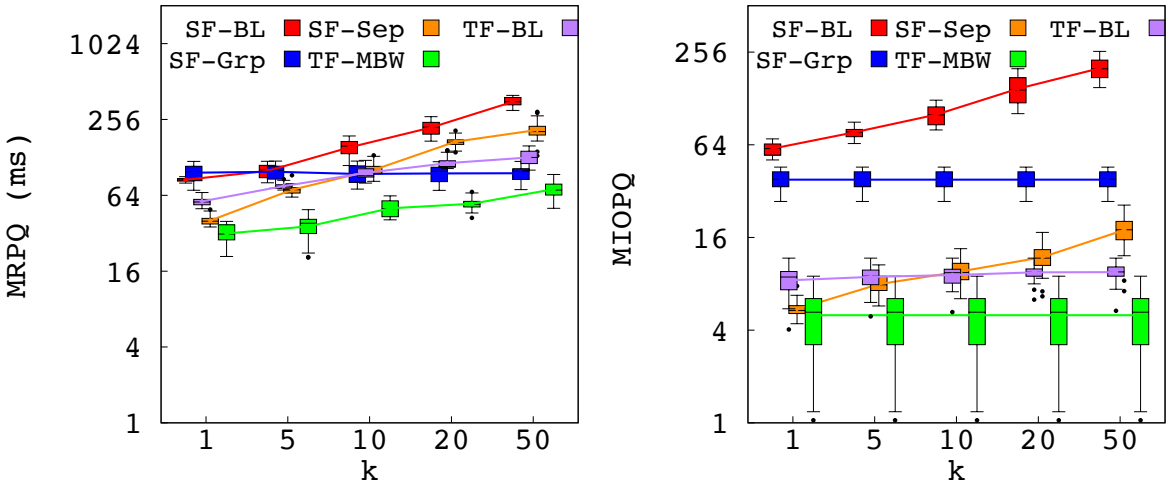
(c) % of blocks for SF-Sep approach

Figure 3.9: Effect of varying k for Flickr dataset



(a) Runtime (b) I/O cost

Figure 3.10: Effect of varying k for Yelp dataset



(a) Runtime (b) I/O cost

Figure 3.11: Effect of varying k for Wiki dataset

***Varying* k.** The experimental results when varying the top-k are shown in Figure 3.9, Figure 3.10, and Figure 3.11 for the Flickr, Yelp and Wiki dataset, respectively. The number of blocks pruned is shown as a percentage of the total number of blocks in Table 3.12 and Figure 3.9c shows the number of SF-SEP approach graphically for Flickr dataset. 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 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. For the same reason, although the percentage of blocks pruned is lower for TF-MBW than the other two approaches, the mean number of blocks retrieved is much less for TF-MBW approach than SF-SEP and SF-GRP. (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 improve the pruning effect. If many objects have a similarity score close to the threshold $\mathcal{R}_k(q)$, 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 the mean I/O costs per query for the three datasets in Figure 3.12. The impact on runtime is shown in Figure 4.14, Figure 3.14a, and Figure 3.14b 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 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

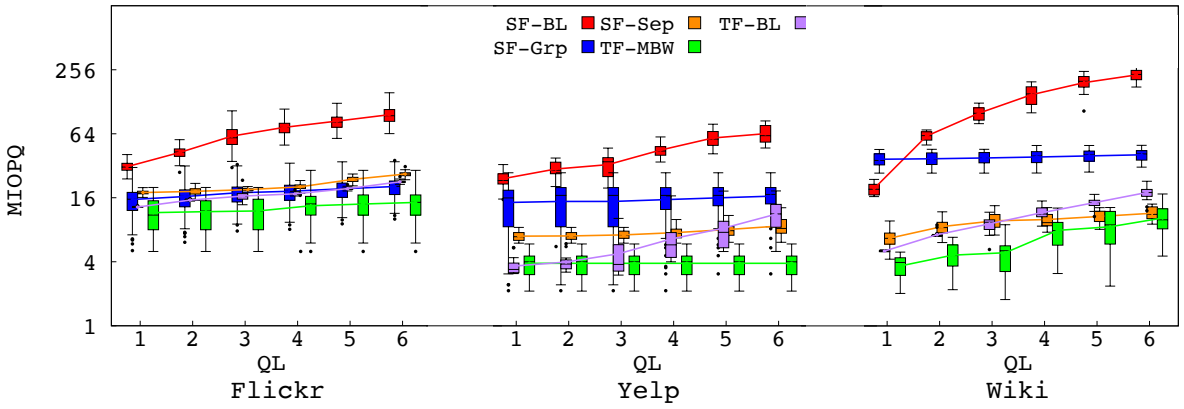
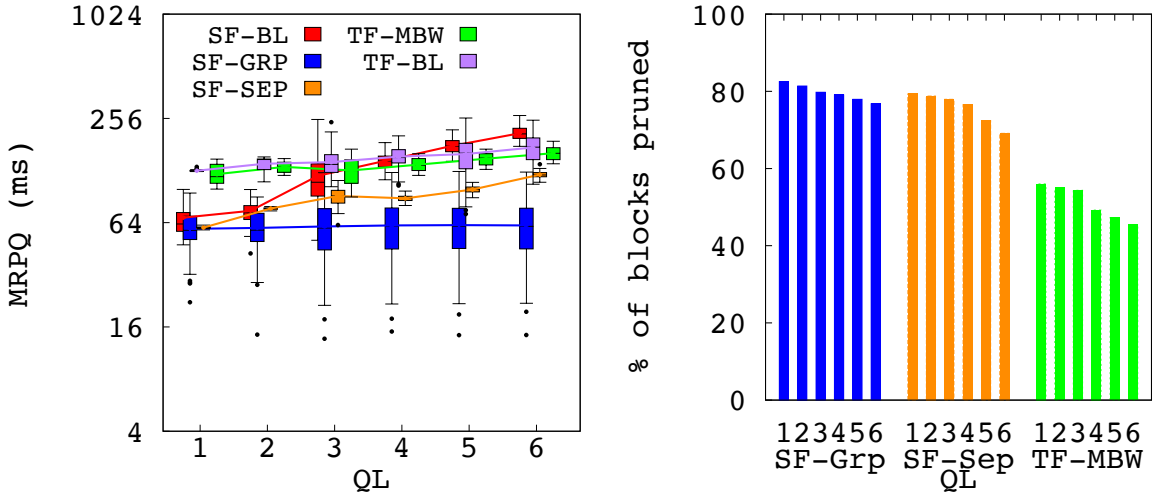


Figure 3.12: MIOPQ when varying QL



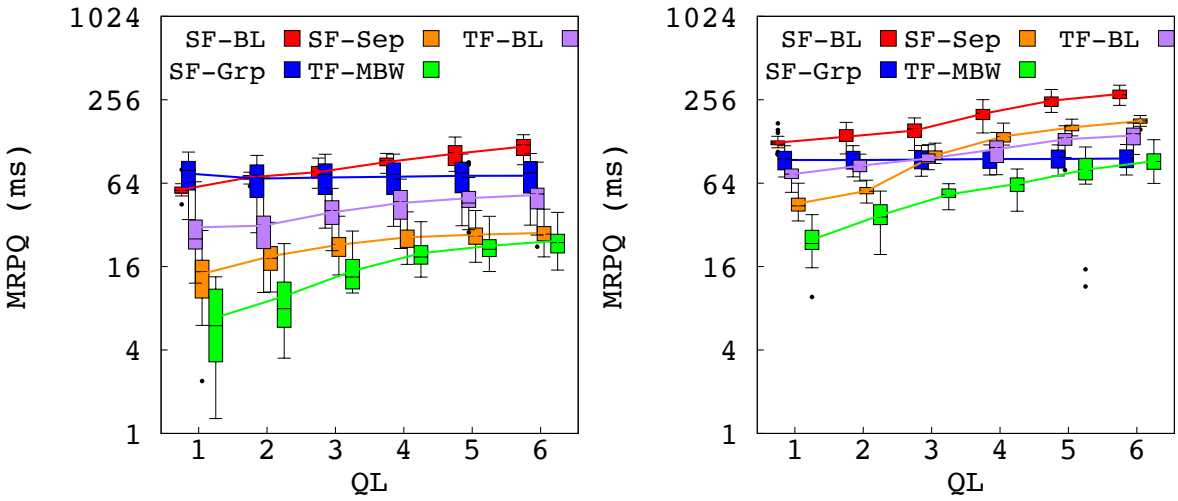
(a) Runtime

(b) % of blocks pruned

Figure 3.13: Effect of varying QL for Flickr dataset

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.

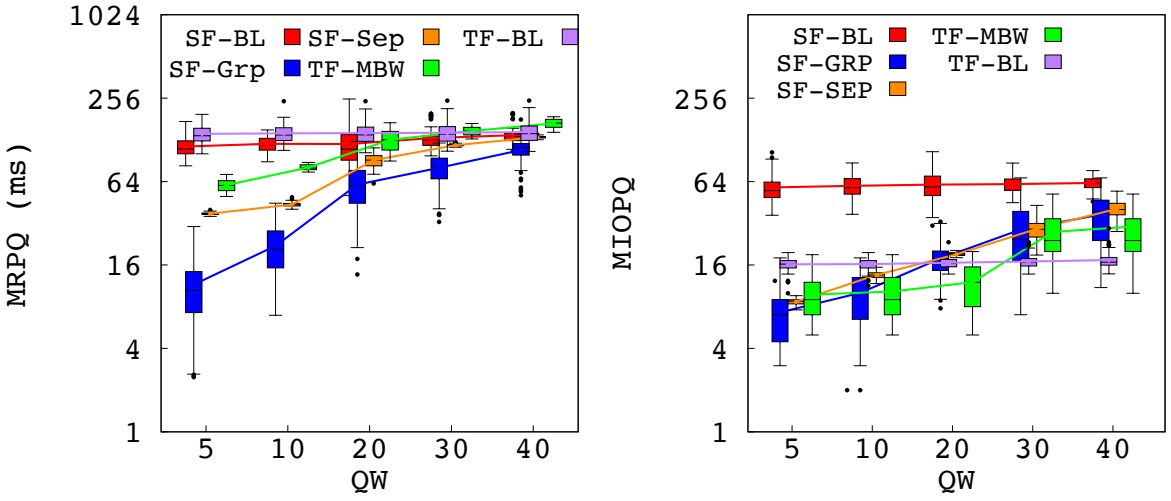
In the text intensive datasets (Yelp and Wiki), more objects contain the query keywords, and must be retrieved than in the Flickr dataset. Moreover, SF-GRP uses the MBR and the union of the query keywords to access the index, which may retrieve unnecessary objects that



(a) Runtime for Yelp dataset

(b) Runtime for Wiki dataset

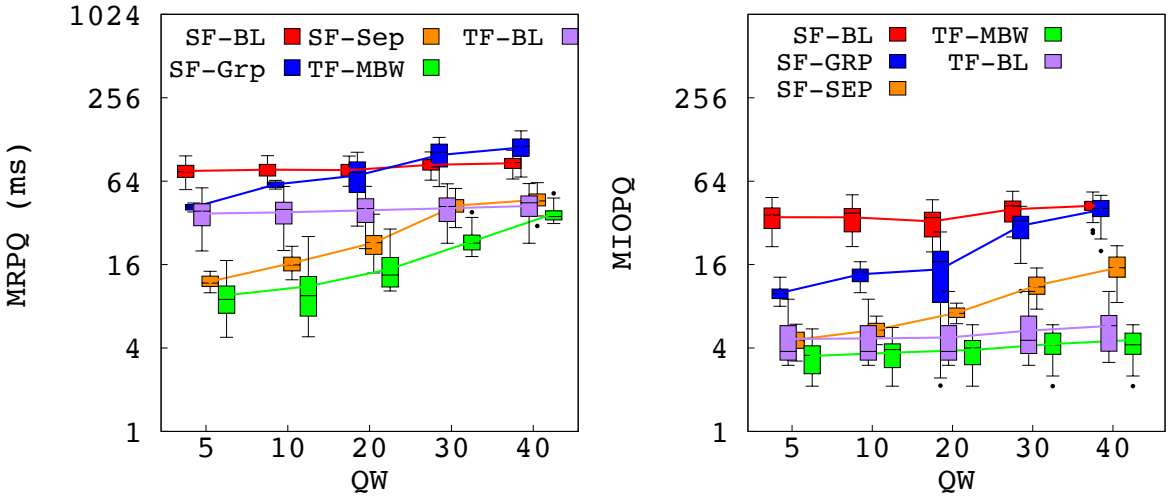
Figure 3.14: Effect of varying QL



(a) Runtime

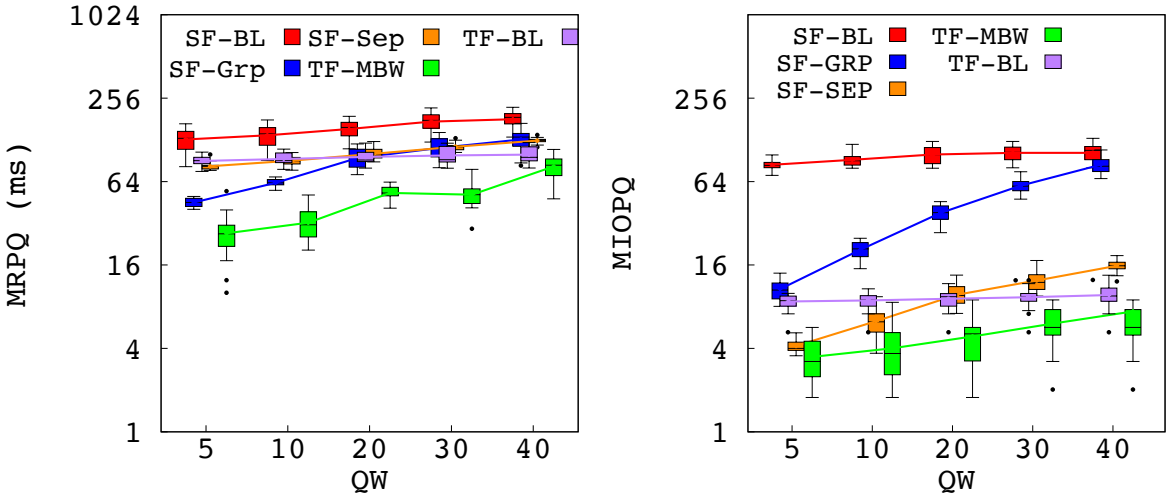
(b) I/O cost

Figure 3.15: Effect of varying QW for Flickr dataset



(a) Runtime (b) I/O cost

Figure 3.16: Effect of varying QW for Yelp dataset



(a) Runtime (b) I/O cost

Figure 3.17: Effect of varying QW for Wiki dataset

Table 3.13: MRPQ for varying α in Flickr dataset

Approach \ α	0.1	0.3	0.5	0.7	0.9
SF-SEP	87.6	88.9	91.4	93.3	94.4
SF-GRP	63.2	60.9	60.8	60.7	60.7
TF-MBW	132.9	130.1	128.2	126.1	124.2

do not actually score high with respect to 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 any of the individual queries are retrieved. Therefore, SF-SEP has better performance than SF-GRP in the text intensive datasets.

***Varying* QW.** Figure 3.15, Figure 3.16, and Figure 3.17 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.

***Varying* α.** Figure 3.18 shows the effect of varying α on each of the three datasets as a bubble graph, where the radius of a bubble is proportional to the MIOPQ of the corresponding approach. Table 3.13 shows the MRPQ when varying α for the Flickr dataset. Here, a higher value of α indicates more preference towards spatial similarity. As shown Figure 3.18, the radius of the bubbles do not vary much with respect to α. A possible explanation of the behaviour is that, as the queries share keywords and are located close-by, the total unique objects and the I/Os that need to be checked for the batch do not vary much for changing preference weight of the similarity measures. Hence, the mean costs of the queries in a batch are not affected much by α. The cost of the text-first approach TF-MBW do not vary with α as well, as in each

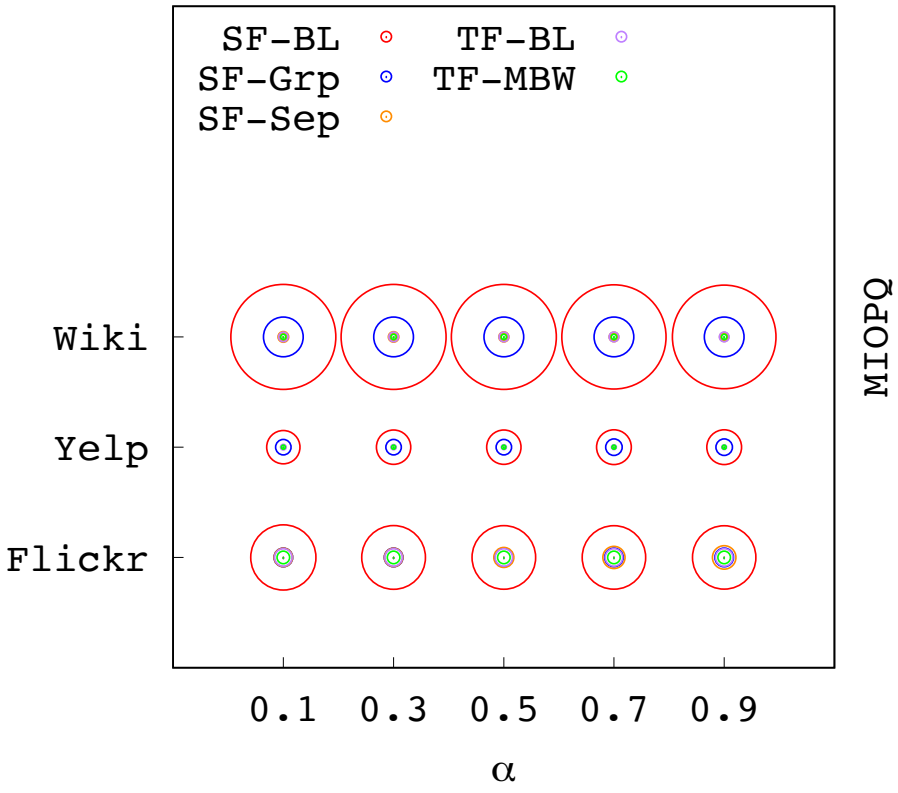
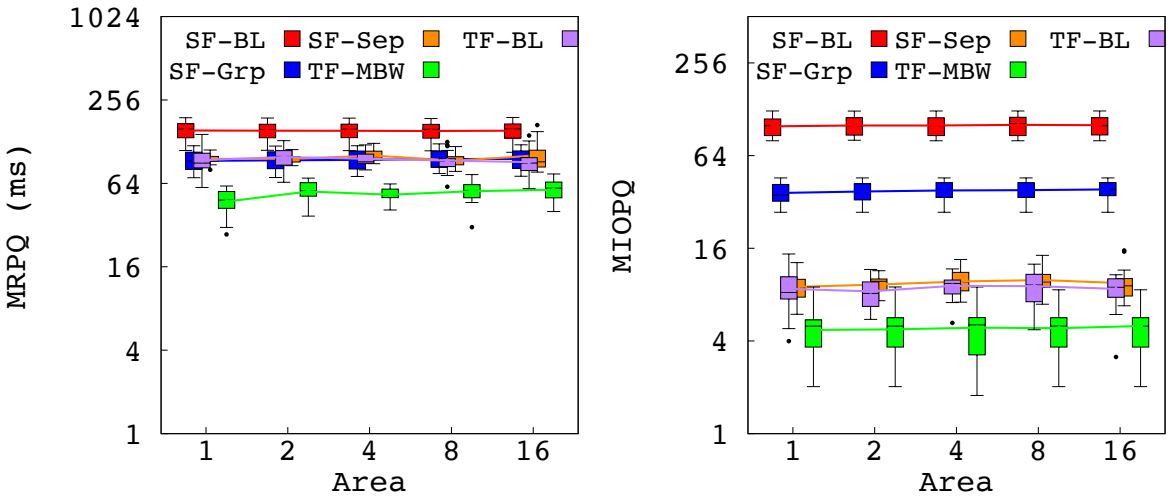


Figure 3.18: MIOPQ for varying α

posting list objects are organized to preserve spatial locality. The MRPQ of the approaches in the other two datasets have the similar trend.

***Varying* Area.** Figure 3.19 shows the impact when varying the area covered by the MBR of the query locations, presented as a percentage of the total dataspace for Wiki dataset. 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. The effect of varying *Area* is similar for the other two datasets.

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 $|Q|$ and the number of unique keywords in a batch, QW in order to evaluate scalability. Figure 3.20, Figure 3.21, and Figure 3.22 show the effect on the total runtime and



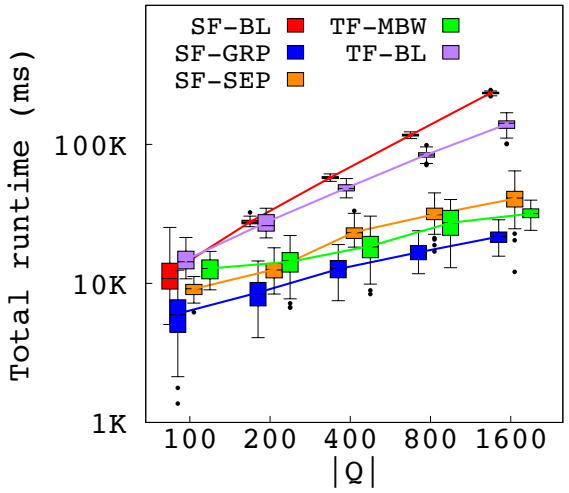
(a) Runtime

(b) I/O cost

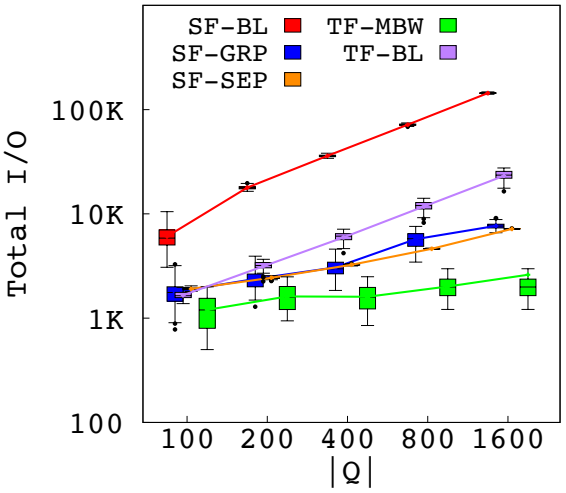
Figure 3.19: Effect of varying *Area* for Wiki dataset

the I/O cost of processing a batch of queries when varying $|Q|$ 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. Figure 3.20c shows the performance of the approaches as a graph of the total runtime vs. the total I/O cost for Flickr dataset. All of our proposed approaches are situated at the lower left corner (low runtime and low I/O cost), where the costs increase rapidly for the baseline approaches.

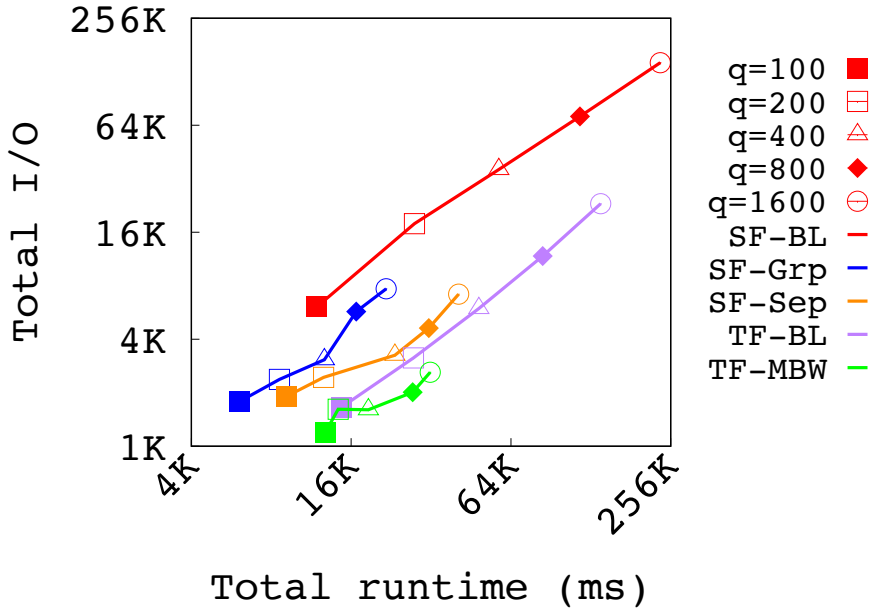
Figure 3.23, Figure 3.24, and Figure 3.25 shows the performance for varying both $|Q|$ and QW of the batch of queries. Here, we increase the number of unique keywords in a batch (QW) by 10% at each level of increment of $|Q|$. The performance of the two baselines are similar to the prior case, as increasing 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 are incurred for the other approaches, compared to the costs where only $|Q|$ is varied. The difference of the I/O costs of the approaches with respect to the baseline are annotated in Figure 3.20b and Figure 3.23b. As shown in the figures, the difference is higher when only $|Q|$ is changed, compared to the difference when both $|Q|$ and QW are changed.



(a) Runtime

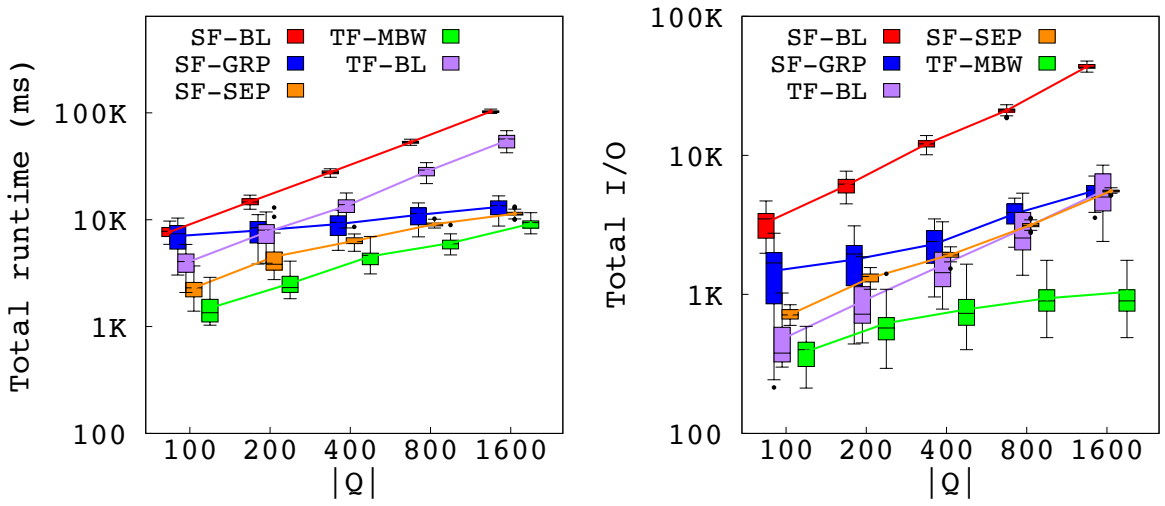


(b) I/O cost



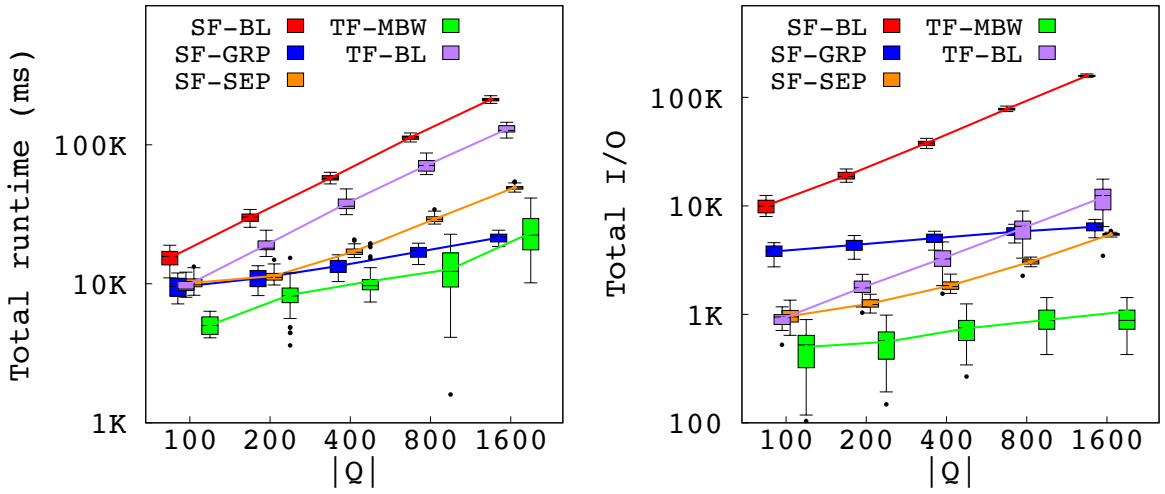
(c) Runtime vs. I/O cost

Figure 3.20: Effect of varying $|Q|$ for Flickr dataset



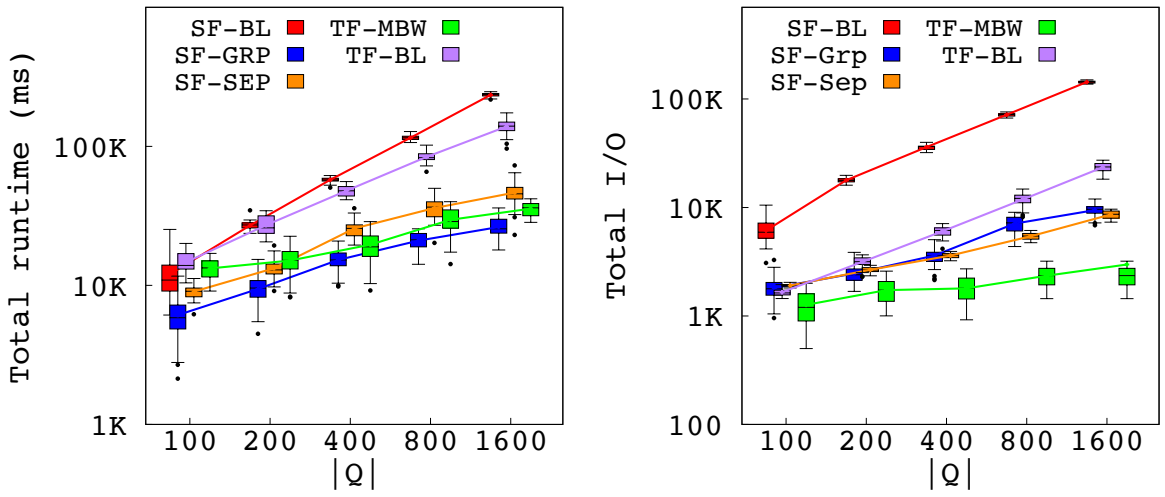
(a) Runtime (b) I/O cost

Figure 3.21: Effect of varying $|Q|$ for Yelp dataset



(a) Runtime (b) I/O cost

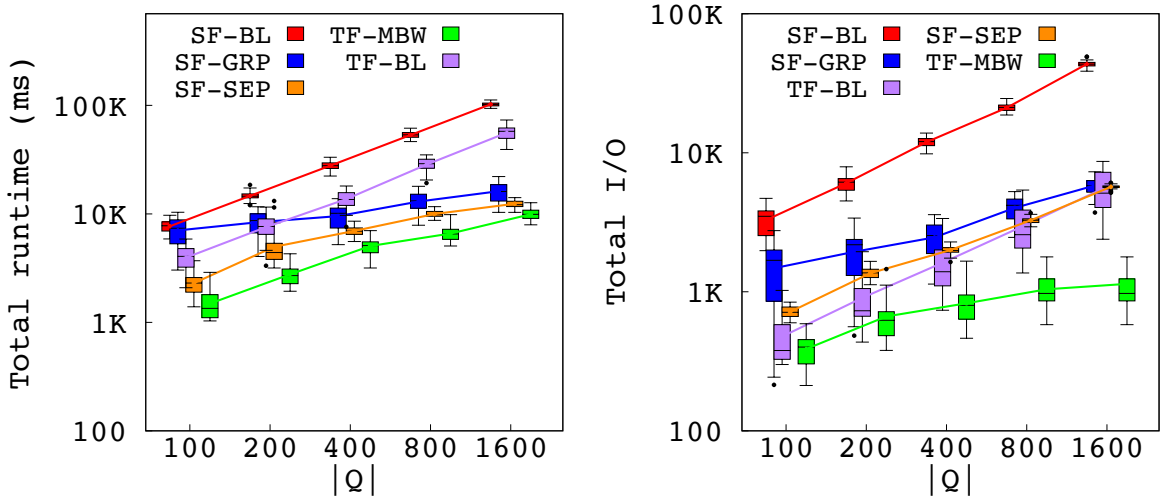
Figure 3.22: Effect of varying $|Q|$ for Wiki dataset



(a) Runtime

(b) I/O cost

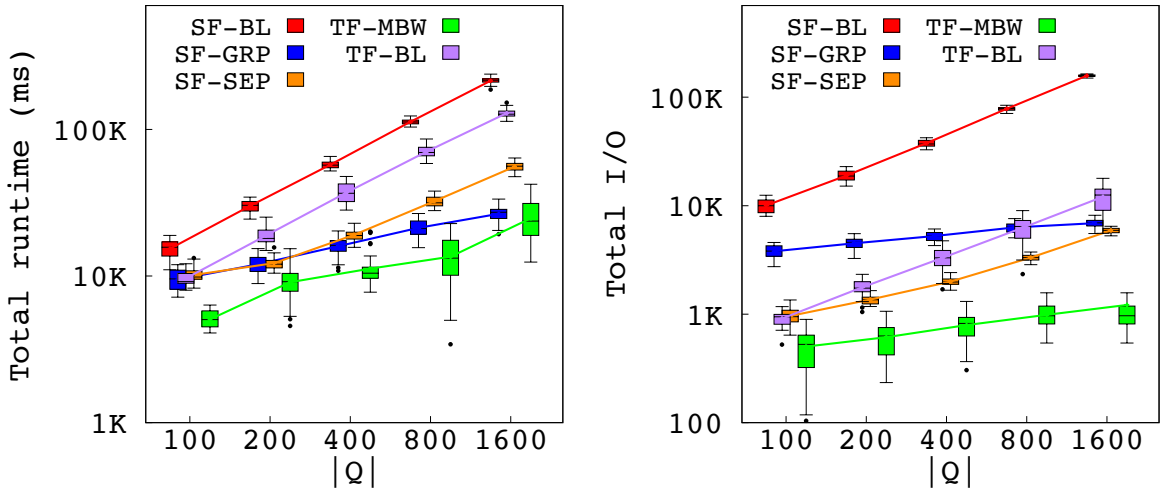
Figure 3.23: Effect of varying both $|Q|$ and QW for Flickr dataset



(a) Runtime

(b) I/O cost

Figure 3.24: Effect of varying both $|Q|$ and QW for Yelp dataset



(a) Runtime

(b) I/O cost

Figure 3.25: Effect of varying both $|Q|$ and QW for Wiki dataset

Summary. We conclude the experimental evaluation section by showing the space vs. the time efficiency trade-offs of our proposed approaches. Figure 3.26 shows the tradeoff between index size of the datasets and the mean runtime per query in the default settings. 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 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.

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.

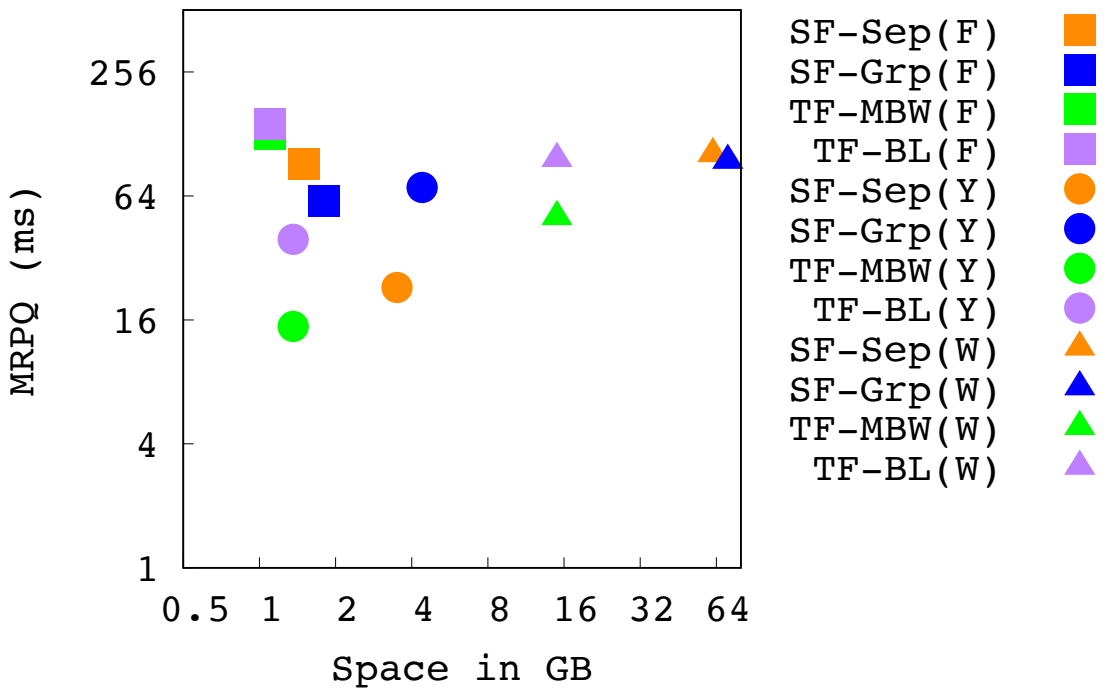


Figure 3.26: Space-time efficiency trade-offs for all indexing methods

3.6 Additional Related Work

The problem of processing multiple queries as a batch has been examined in the past in several different contexts. Sellis [109] and Hong et al. [59] 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 [98] and multiple nearest neighbor queries [22, 144]. 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. [144] 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 [98] 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. [33]. 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 present the related work on top-k queries over spatial-textual data in Chapter 2, we now review the work on batch processing of spatial-textual queries in this section.

Batch Processing of Top-k Spatial-Textual Queries. Although there are many approaches that explore the processing of individual top-k spatial-textual queries using spatial-first and/or text-first indexes (Table 2.4), there is only one work ([130]) which focuses on batch processing of queries.

Wu et al. [130] 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 [28, 129]. 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.

3.7 Conclusion

In this chapter, we have studied how to efficiently answer multiple top-k spatial-textual queries as a batch. In particular, the existing work on top-k spatial-textual queries are studied, and the following approaches are 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; and (iii) a text-first index, the SIF, which augments a standard inverted file with spatial information and a novel block-wise traversal technique in SIF to process multiple top-k spatial-textual queries.

The SF-Sep and SF-Grp approaches have adopted a space-first strategy to construct the index, which may have two drawbacks: (i) the index size can be impractically large; (ii) the index may not be easily integrated into an existing web search engine architecture. Therefore, a text-first index, SIF is proposed to better support text intensive data. 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 datasets, the performance and the space requirements of our approaches are compared for different settings.

Chapter 4

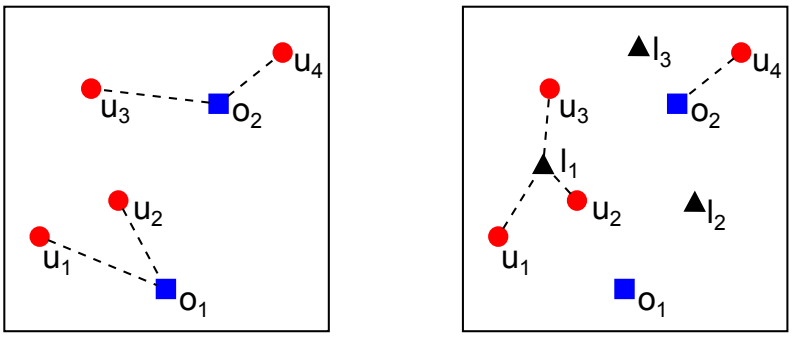
Finding the Optimal Location and Keywords in Obstructed and Unobstructed Space

The problem of optimal location selection based on maximized bichromatic reverse k nearest neighbor (RkNN) queries has been extensively studied in spatial databases. The maximized bichromatic RkNN problem is essentially an aggregation over the results of multiple top-k queries. That is, given a set of objects and a set of users, each user can be treated as a top-k query, where the top-k objects of a user are ranked based on a similarity function. The goal of the maximized bichromatic RkNN problem is to find the attributes (e.g., location) of an object so that the object will be in the top-k of the maximum number of users based on the same ranking function.

In this chapter, we present a related query, denoted as a Maximized Bichromatic Reverse Spatial-Textual k Nearest Neighbor (MaxST) query that finds two attributes: (i) location, and (ii) a set of keywords of an object, such that the object will be a top-k spatial-textual object of the maximum number of users.

4.1 Introduction

The optimal location selection problem is an important task in making business decisions. As a result, a number of studies have addressed different instances of this problem, and queries such as the *Maximized Bichromatic Reverse k Nearest Neighbor* (MaxBRkNN) queries [81, 126, 127,



(a) (b)

ID	Keywords
u_1	sushi, seafood
u_2	sushi
u_3	sushi, noodles
u_4	noodles
o_1	sushi
o_2	noodles

Figure 4.1: Example of a MaxST query. Here, $u_1, \ldots, u_4$ are the users, o_1, o_2 are the existing restaurants, and the locations where a new restaurant can be placed are ℓ_1, ℓ_2, ℓ_3. The table shows the corresponding text descriptions. The top-1 relevant restaurant of each user is shown with a connecting dotted line.

146] have received considerable attention in the spatial database community in recent years. Given two sets of data: (i) a set of users U, and (ii) a set of objects O over a shared spatial dataspace, a bichromatic reverse k nearest neighbor (BRkNN) query for an object $o \in O$ returns all the users $u \in U$ for which o is a kNN in O. A MaxBRkNN query finds the optimal region in that dataspace to place a new object, $p \notin O$ such that the number of users in the result of the BRkNN query issued by p is maximized.

A practical application of these queries is to find the location of a new business or a new facility that can serve the maximum number of customers. In the literature, spatial distance is usually the only relevance criteria considered. However, customers are generally interested in the products and services as well as the location. Therefore, a spatial-textual query is a natural extension in this setting. In the following, we present example application scenarios of the spatial-textual context.

Example 8. *Consider an application that finds the optimal location to open a new restaurant, and the items to include in the menu such that the restaurant will be one of the top-k relevant restaurants for the maximum number of customers. As an example in Figure 4.1,* $u_1, \ldots, u_4$ *are the four users where the locations are shown with circles;* o_1 *and* o_2 *are the existing restaurants shown with squares. The table shows the corresponding text descriptions. Here,* o_1 *is the most relevant restaurant for the users* u_1, u_2 *and* o_2 *is the most relevant restaurant for* u_3, u_4 *based on both spatial and textual similarity. Suppose a service provider wants to open a new restaurant,* p *in one of the available locations* ℓ_1, ℓ_2, ℓ_3 *shown with triangles in Figure 4.1b, and for cost*

Figure 4.2: An example to demonstrate the importance of visibility in placing advertisement billboard. The billboard in the front is visible from the road, while the billboard at the back is obstructed by the surrounding buildings.

reasons, the number of menu items that can be served is '1'. If a new restaurant p is placed in location ℓ_1, and the menu of p is 'sushi', then p becomes the most relevant restaurant for u_1, u_2, and u_3 as shown in Figure 4.1b. Note that p can be the most relevant restaurant for at most three users in this example.

Example 9. *In social media advertising, a user is shown a limited number of advertisements that are highly relevant to her location and preferences (top-k relevant advertisements). In this case, given a set of candidate locations and keyword choices, the application is to find the location and a limited number of keywords to include in an advertisement such that it is displayed to the maximum number of users.*

Despite significant progress on this problem, there is a research gap in finding both an optimal location and a keyword set for an object, which is an interesting extension of the problem. Moreover, as targeted applications become more practical, physical obstacles must also be factored into the solution. For example, visibility can have an important role in improving advertisement reach, frequency, and impact. While previous research has explored various visibility related queries, no prior work has considered the effect of visual obstacles in the context of BRkNN problems. Consider the following example application that highlights the importance of this factor.

Example 10. *Consider a company who wants to place a new billboard for an advertisement p. An important attribute in this scenario is the visibility of the billboard for potential customers in the presence of obstacles, such as buildings in a city. So given a set of potential locations where a billboard can be placed and a set of keywords appropriate for the advertisement, the problem is to find an optimal location and a limited number of keywords for p such that p is relevant and visually unobstructed for as many customers as possible. The scenario is demonstrated in Figure 4.2 for visibility, where the billboard in the front is visible from the road, while another billboard is mostly obstructed by its surrounding buildings. The location in the front is intuitively a better choice to place the billboard.*

In these examples, the underlying problem is to find the location and the text description for a specific product or service such that the product is one of the k most relevant objects for the maximum number of users, which is analogous to maximizing the size of the bichromatic reverse kNN of the product. Here, the ranking is based on both spatial and textual properties. In the first two examples, the spatial relevance of an object (advertisement) is the spatial proximity (Euclidean distance) from a user. In Example 10, the spatial relevance is the visibility of an object from a user in the presence of visual obstacles (buildings). An object o is considered visible from a user u if there is at least one point over o, such that the straight line connecting the point and u does not pass through any other objects (buildings). In contrast to the spatial proximity, the visibility of an object for a user depends on other objects in the dataset that are located in between the user and the object.

In this chapter, a new query variation is presented – the *Maximized Bichromatic Reverse Spatial Textual k Nearest Neighbor* (MaxST) query. Two different instances of spatial relevance are explored for ranking objects: (i) the Euclidean distance, and (ii) the visibility. A series of indexes and traversal algorithms are proposed to process MaxST queries for both of the instances. First, the GRP-TOPK approach is proposed to efficiently find the best location and keyword set combination that requires the computation of the top-k objects for all of the users. This approach can benefit from our solutions presented in Chapter 3 where the top-k spatial-textual objects of multiple users are processed jointly as a batch. Second, the INDIV-U approach is proposed, which avoids computing the top-k objects for the users that cannot affect the final result set. Third, the Modified IUR-tree (MIUR-tree) is proposed to store the users, where the motivation is to improve efficiency by applying the pruning techniques over a hierarchical structure of the users. To select the best candidates, an exact and an approximate algorithm are presented, and several pruning strategies are applied to answer MaxST queries efficiently.

The visibility of an object introduces additional challenges when computing spatial relevance in this problem, as the visibility of an object o w.r.t. a user u depends not only on their own locations, but also the on the other objects (visual obstacles) between o and u. We present a new index structure, the OIR-tree to support the visibility computation and extend the approaches for visibility as the spatial relevance in MaxST problem.

In summary, the key contributions of this chapter are:

- We introduce and address the MaxST problem for both unobstructed and obstructed space, where the spatial relevance of an object is based on Euclidean distance, and visibility from a user, respectively.
- Three different approaches are proposed to answer the MaxST query, where the approaches differ in the pruning techniques that are applied on the set of users.
- An exact and an approximate solution are proposed to select the best candidates efficiently.
- A comprehensive experimental study is presented to demonstrate the efficiency of our proposed techniques and compare relative performance.

For the rest of the chapter, the phrase "reverse k nearest neighbor (RkNN)" is used instead of "bichromatic spatial-textual reverse k nearest neighbor" when the context is clear. We also use the terms "top-k" and "kNN" interchangeably.

The rest of the chapter is organized as follows: Section 4.3 presents the preliminaries and defines the MaxST problem. Section 4.4 presents an overview of the proposed approaches. We propose three different approaches to answer a MaxST query: (i) the GRP-TOPK approach (Section 4.5), (ii) the INDIV-U approach (Section 4.6), and (iii) the INDEX-U approach (Section 4.7). The extension of the solutions are presented in Section 4.8 to support answering a MaxST query for visibility as the spatial relevance. We present our extensive experimental evaluation in Section 4.9, an overview of related work in Section 4.2, and conclude in Section 4.10.

4.2 Related Work

4.2.1 Spatial Databases

Relevant work from the spatial database domain can be categorized mainly as *Maximizing Bichromatic Reverse k Nearest Neighbor* (MaxBRkNN) and location selection queries.

MaxBRkNN queries. Wong et al. [126] introduced the MaxOverlap algorithm to solve the MaxBRkNN problem. The algorithm iteratively finds the intersection point of the Nearest Location Circles (NLCs) that are covered by the largest number of NLCs. The optimal region is the overlap of these NLCs. This work also supports ℓ-MaxBRkNN queries to find the ℓ best regions. In a later work, Wong et al. [127] extended the MaxOverlap algorithm to support the L_p-norm and three-dimensional space. However, the scalability of MaxOverlap is an issue, as the computation of the intersection points for the NLCs is expensive.

Other work exists that overcome the limitations of MaxOverlap. Zhou et al. [146] introduced the MaxFirst algorithm which iteratively partitions the space into quadrants and use the NLCs to prune the quadrants that cannot be a part of the result. Liu et al. [81] present the MaxSegment algorithm that transforms the optimal region search problem to the optimal interval search problem. They use a variant of plane sweep to find the optimal interval.

Approximate solutions have also been proposed to improve the efficiency. Lin et al. [77] proposed OptRegion where each NLC is approximated by the minimum bounding rectangle (MBR) and a sweep-line technique is used to find the overlapping NLCs. An estimation of the number of overlapping NLCs is computed using the MBRs to prune the intersection points. Alternately, Yan et al. [134] propose a grid-based approximation algorithm called FILM. Since the algorithm is approximate, the solution requires an order of magnitude less computation time than MaxOverlap. The authors extended FILM to answer the related problem of locating k new services that collectively maximize the total number of users.

These previous studies focus solely on spatial properties such as the intersection of geometric shapes [126, 127], space partitioning [146], or sweep-line techniques [77, 81] in the query processing methods. Therefore, it is not straightforward to extend these solutions to support the textual component of the MaxST query.

Location selection queries. Others have explored optimal location queries, which find a location for a new facility that minimizes the average distance from each customer to the closest facility [64, 141]. Zhang et al. [141] propose the *MDOL prog* algorithm which partitions space to find an optimal location. Jianzhong et al. [64] maintain the *influence set* of a potential location p that includes the customers for whom the nearest facility distance is reduced if a new facility is established at p. A similar problem was presented in other work [63, 102, 135, 136] which finds a location for a new server such that the maximum distance between the server and any client is minimized. Dimitris et al. [31] find a location that minimizes the sum of the distances from the users.

A *maximal influence query* ([19, 60, 78, 113, 132]) finds the optimal location to place a new facility such that the influence of that facility is maximized. Here, the influence of a location c represents the cardinality of customers whose corresponding nearest facility will be c if a new facility is established in c. These queries focus on an aggregation over distances from the query location, such as the average or the minimum distance. These works do not directly address the problem of this chapter, maximizing the reverse kNN users.

4.2.2 Spatial-Textual Databases

In the literature of spatial-textual queries, the reverse spatial-textual kNN (RSTkNN) and the maximizing reverse spatial-textual kNN (MaxST) are the most relevant to our problem.

RSTkNN. Given a dataset D of spatial-textual objects, a target query object q, an RSTkNN query finds all the objects in D that have q in their list of top-k relevant objects. The ranking of the objects use an objective function which combines both spatial proximity and text relevancy. Lu et al. [83] proposed the Intersection-Union R-tree (IUR-tree) index, and later presented a cost analysis for RSTkNN queries [84]. Each node of an IUR-tree consists of an MBR and two textual vectors: an intersection vector and a union vector. The weight of each term in the intersection (union) textual vector is the minimum (maximum) weight of the terms in the documents that are contained in the corresponding subtree. Each non-leaf node is also associated with the number of objects in the subtree rooted at that node.

In their proposed solution, an upper and a lower bound estimation of similarity is computed between each node of the IUR-tree and the k·th most similar object. A branch-and-bound algorithm is then used to answer the RSTkNN query. In this work, the computation of the bounds and the algorithm are designed for the monochromatic case only since both the data objects and the query objects belong to the same type, and the nodes of the tree store only one type of object.

MaxRSTkNN. Given a set of users and a set of facilities, Gkorgkas et al. [47] address the problem of selecting at most ω keywords as the text description of a specific facility, such that the facility will appear in the top-k results of the maximum number of users.

A recent work by Xie et al. [133] independently propose a solution of a subproblem of this article, where, given a set of keywords, the query is to find a region in space to establish a facility such that the facility will be a top-k spatial-textual object of the maximum number of

users. They present both an exact and an approximate solution based on Voronoi diagrams to find such regions.

4.2.3 Visibility Queries

Visibility problems studied in spatial databases usually involve finding the kNN [95, 125] or RkNN [41, 42, 43] objects for a given query point, where the shortest path between two points without crossing any obstacle is taken as the distance measure, denoted as the *obstructed distance*.

Masud et al. [89] propose the k *maximum visibility query* that finds top-k locations from a set of query locations with the maximum visibility of a target object T in the presence of obstacles. Choudhury et al. [25] present an efficient approach to construct a Visibility Color Map (VCM), where each point in the space is assigned a color value denoting the visibility measure of a query target. [103] study the problem of constructing a VCM for a moving target. In general, all of these approaches use the concept of an obstructed region to find the parts of the space that are visible from the query location(s), and then calculate the visibility value of the visible part of the target object.

Visual-textual queries. Zhang et al. [139] study the problem of finding the top-k objects based on the visibility and the textual similarity with respect to a query location and a set of query keywords. They propose a two pass method on the IR-tree where (i) the first pass iteratively explores the region around the query location to determine the obstructed and the visible region w.r.t. the query, and (ii) the second pass is used to calculate the visibility and textual similarity of the visible objects in a best-first manner.

Our approach to answer the MaxST query for visibility metric also follows a similar principle, where the obstructed regions are pre-computed and indexed, and then an OIR-tree is used to compute the visibility of the necessary objects and the candidates.

4.3 Problem Formulation

Let D be a spatial-textual bichromatic dataspace, where U is a set of users and O is a set of objects. Each object $o \in O$ is a pair $(o.l, o.d)$, where $o.l$ is a geo-spatial position (e.g., point, rectangle, or polygon) and $o.d$ is a set of keywords. Each user $u \in U$ is also defined as a similar pair $(u.l, u.d)$. For an object o, let $\mathcal{B}_o$ denote the set of users that have o as a top-k object based on a combined spatial and textual relevance score.

Table 4.1: Basic notation

Symbol	**Description**
U	The set of users.
L	The set of candidate spatial positions (as point, line, etc.).
W	The set of candidate keywords.
p	A specific object $p \notin O$ for which the optimal location and keyword set needs to be selected as result.
ℓ	The optimal location from L to select as result.
ω	The maximum number of keywords to select in the result.
W'	The best set of ω keywords from W to select as result.
Section 4.3:	
$\overline{\text{SS}}(o.l,\text{u.l})$	The visibility of object o w.r.t. the user u.
$\Delta o.l$	The segment of $o.l$ for which the distances and the orientations of all points can be considered as visually similar.
$\text{VL}(o.l, u.l)$, $\text{VL}_\Delta(\Delta o.l, u.l)$	The perceived length of o (the segment $\Delta o.l$) from $u.l$.
$\angle(o.l, u.l)$, $\angle(\Delta o.l, u.l)$	The angle between $o.l$ (the segment $\Delta o.l$) and the straight line connecting the midpoint of $o.l$ and $u.l$.
$\text{len}(o.l)$, $\text{len}(\Delta o.l)$	The length of $o.l$ (the segment $\Delta o.l$).

Definition 2. *A MaxST query $q(p, L, W, \omega, k)$ over D, where $p \notin O$ is a specific spatial-textual object, L is a set of spatial candidate locations (e.g., point, line, rectangle, etc.), W is a set of candidate keywords, ω is a positive integer where $\omega \leq |W|$, and k is the number of relevant objects to be considered, finds a location $\ell \in L$ and a set of keywords $W' \subseteq W$, $|W'| \leq \omega$ such that if $p.l = \ell$ and $p.d = W'$, the cardinality $|\mathcal{B}_p|$ is maximized. If p has any existing text description, then $p.d = (W' \cup p.d)$ and $p.l = \ell$ combinedly maximize $|\mathcal{B}_p|$.*

An object o is ranked based on a combined score of spatial and textual relevance with respect to a user u. Without loss of generality, we use Equation 3.1, the linear weighted combination score presented in Section 3.2 as the ranking function. In this chapter, the problem of MaxST is addressed for two different types of spatial similarity (which are measured using the location of the objects and the users), (i) the spatial proximity (Euclidean distance) of an object w.r.t. a user and (ii) the visibility of an object from a user in the presence of visual obstacles. Equation 3.2 is used to measure the spatial similarity when Euclidean distance is considered. The text similarity is measured by the TF·IDF metric. Now we present the visibility metric that we use as spatial similarity in this chapter.

Visibility measure. We now define visibility when a line is the geometric shape of the spatial data. For each $o \in O$, $o.l$ is shown as a line for the rest of the chapter for visibility, but the calculation is representative of any other geometric shape. An object o is considered visually

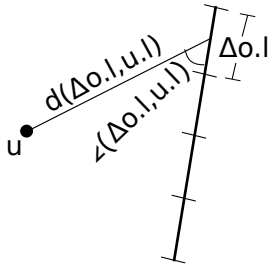


Figure 4.3: Measuring visibility of an object w.r.t. a user u based on distance and angle between them. The object is divided into infinitesimal segments such that for each segment $\Delta o.l$, the distances and the orientations of all points on $\Delta o.l$ w.r.t. u are assumed to be visually similar.

relevant to a user u if and only if at least one point of $o.l$ is visible from the user $u.l$, i.e., there exists a point 'a' over the line segment $o.l$ such that the straight line connecting a and $u.l$ does not pass through any other objects in O.

Previous work [25, 89, 139] has defined and used different metrics to quantify visibility. The metric, called "visual angle" used by Choudhury et al. [25] is the angle formed at the eye of a user by the extremities of an object viewed, which determines the perceived length of that object. We adopt this metric as the measure of visibility in this chapter. Specifically, the visibility measure in our work is computed as:

$$\overline{\text{SS}}(o.l, u.l) = \frac{2arctan(\text{VL}(o.l, u.l))}{180} \tag{4.1}$$

where the maximum possible visual angle is 180^o, which is used to normalize the value of $\overline{\text{SS}}(o.l, u.l)$ between [0,1], and $\text{VL}(o.l, u.l)$ is the perceived length of o from $u.l$.

The perceived length of an object mainly depends on the distance and the viewing angle between the user and the object. If an object is viewed from an oblique angle, the perceived length of that object becomes smaller than the original length. The perceived length of o also decreases with the increase of the distance between o and the user u. As different parts of an object always have different distances and orientations w.r.t. a user, we use the *cumulative approach* presented by Zhang et al. [139] to calculate visibility. Specifically, we divide the line $o.l$ into infinitesimal segments of size at most ϵ such that, for each segment, $\Delta o.l$, the distances and the orientations of all points on $\Delta o.l$ w.r.t. to a user u is perceived as visually similar. For example, in Figure 4.3, the segments are shown for an object o, and in Figure 4.5, the segments are shown for a particular object o_2.

Let the straight line connecting the midpoint of a line segment $\Delta o.l$ and the point location of the user $u.l$ create an angle $\angle(\Delta o.l, u.l)$ with $o.l$. Let the minimum distance of $\Delta o.l$ and $u.l$

be $\boldsymbol{d}^{\downarrow}(\Delta o.l, u.l)$. Then the perceived length of a segment $\Delta o.l$ w.r.t. $u.l$ is measured as:

$$\mathrm{VL}_{\Delta}(\Delta o.l, u.l) = \frac{\angle(\Delta o.l, u.l)}{90^o} \times \frac{len(\Delta o.l)}{\boldsymbol{d}^{\downarrow}(\Delta o.l, u.l)} \tag{4.2}$$

Here, if $\angle(\Delta o.l, u.l) = 90^o$, the perceived length of the line segment $\Delta o.l$ from a nominal distance is the same as its original length, $len(\Delta o.l)$. The perceived length of the entire $o.l$ w.r.t. $u.l$, $\mathrm{VL}(o.l, u.l)$, is obtained by summing up the perceived lengths of the segments $\Delta o.l$ that are visible from $u.l$:

$$\mathrm{VL}(o.l, u.l) = \sum_{\Delta o.l \text{ visible from } u.l} \mathrm{VL}_{\Delta}(\Delta o.l, u.l)$$

For example, in Figure 4.5, to obtain the visibility of o_2, $\overline{\mathrm{SS}}(o_2.l, u.l)$, the perceived length of the segments of o_2 that are visible from u (represented with black) must be individually computed, and summed as $\mathrm{VL}(o.l, u.l)$ in Equation 4.1 to get the visibility value of object o_2 w.r.t. the user u. The new basic notation (that are not presented in the previous chapters) used in the remainder of this chapter is shown in Table 4.1.

4.4 Solution Overview

One can think of a straightforward solution for the MaxST query consisting of the following steps: (i) Find the top-k spatial-textual objects for all users in U individually using any of the existing techniques. Let the relevance score of the k·th ranked object of a user $u \in U$ be $\mathcal{R}_k(u)$. (ii) Generate all possible combinations M of ω keywords from W. For each candidate location $\ell \in L$, and each keyword combination $m \in M$, the total relevance score of ℓ and $p.d \cup m$ are computed for the users $u \in U$, where $m \cap u.d \neq \emptyset$. If this score is greater than $\mathcal{R}_k$, u is a RkNN of the object p, where $p.l = \ell$ and $p.d = m$. The location and the keyword combination $\langle \ell, m \rangle$ for which the maximum number of users is a RkNN, is returned as the result.

Challenges. The straightforward method is computationally expensive for several reasons:

- Computing the top-k results for *all* users;
- Iterating over *all* of the candidate locations;
- Generating all combinations M of ω candidate keywords; and

- Computing the relevance scores for all of the candidate location tuples, and $m \in M$ with respect to each user $u \in U$.

We now show that the candidate selection part of the MaxST problem is NP-hard by reduction from the Maximum Coverage problem.

Lemma 2. *The MaxST problem is NP-hard.*

Proof. Given a collection of sets $S = \{S_1, S_2, \ldots, S_m\}$, and a positive integer n, the Maximum Coverage (MC) problem is to find a subset $S' \subseteq S$ such that $|S'| \leq n$ and the number of covered elements by S', $|\cup_{S_i \in S'} S_i|$ is maximized. The MC problem is NP-hard [58].

Consider a special case of the MaxST problem where $\alpha = 1$. Here, the similarity score of the objects that contain at least one of the user keywords are measured by the spatial proximity using Equation 3.1. Also assume that the number of candidate locations, $|L| = 1$ in this special case. So $p.l = \ell$, where ℓ is the only candidate location in L.

For each candidate keyword $w \in W$, let $\mathcal{B}_w$ be the set of users that have p as a top-k object when w is included in $p.d$. So, a collection of the set of users exists, one for each $w \in W$. The goal of a MaxST query is to select at most ω keywords from W, such that the number of users for which p is a top-k object is maximum. That is, solving the maximum coverage problem is equal to finding a subset of the candidate keywords, $W' \subseteq W$, where $|W'| \leq \omega$ maximizes $|\cup_{w \in W'} \mathcal{B}_w|$.

Again, $\mathcal{B}_w$ for each $w \in W$ corresponds to each set S_i of the MC problem, where each user $u_j \in \mathcal{B}_w$ corresponds to the element of S_i. Therefore, finding a subset W' of the candidate keywords where $|W'| \leq \omega$ maximizing $|\cup_{w \in W'} \mathcal{B}_w|$ is equivalent to solving the maximum coverage problem. □

Therefore, scanning all combinations is not practical for a large number of candidates. To overcome these limitations, we seek techniques that:

- Efficiently compute the top-k objects for the users;
- Avoid computing the top-k objects for the users that cannot affect the result; and
- Prune the candidate locations and keywords that cannot be part of the final result.

A series of approaches to answer the MaxST query are presented. In each of these approaches, the idea is to avoid processing the candidates that cannot be a result, avoid computing the top-k objects of the users that do not have any affect on the final optimal result, share

object retrieval costs among the users, and access the necessary objects only once. The methods differ in the pruning techniques that are applied to achieve these goals. In the following, an overview of three different solutions for MaxST queries are presented from a high level and their differences are explained.

- **GRP-TOPK approach**: This approach consists of two separate modules to answer the MaxST problem. First, the top-k objects for all users is computed jointly to address the first challenge mentioned above. This step can benefit from the approaches presented in Chapter 3. Next, several pruning techniques are applied to discard the candidates that cannot be a result, where the score of the k·th ranked object of the users are used to facilitate the pruning.

- **INDIV-U approach**: A limitation of the GRP-TOPK approach is that the top-k objects for all of the users must be computed. Therefore, the INDIV-U approach is proposed to avoid computing the top-k objects for users that do not affect the final result set. The idea is to estimate the number of objects that can have a higher similarity than a candidate for each user $u \in U$ with a single traversal of the objects. This number is used to prune the unpromising candidates and also the unnecessary users that cannot be an RkNN of any promising candidate.

- **INDEX-U approach**: In the previous approach, the users are pruned by checking each one individually against the candidates. So, a new index is proposed, the Modified IUR-tree (MIUR-tree) to store the users, where the motivation is that a hierarchical index structure over the set of users may exhibit a higher pruning capacity than checking them individually.

In Section 4.5 - 4.7 our proposed approaches to answer a MaxST query are presented, and the required steps are described using the Euclidean distance as spatial relevance. Then in Section 4.8, the extension of the solutions are presented to support answering a MaxST query in obstructed space, where the spatial relevance of an object is measured as its visibility w.r.t. a user. Table 4.2 presents the notation used in Section 4.5 - 4.7.

4.5 GRP-TOPK Approach

In this approach, we assume that the top-k objects for all users are computed as a first step to answering a MaxST query. In Chapter 3, three different techniques to jointly compute the

Table 4.2: Notation used in Section 4.5 - 4.7

Symbol	**Description**
Section 4.5:	
$\mathcal{B}_p$	The set of users that are a reverse kNN of an object p.
$\mathcal{B}^{\downarrow}{}_{\ell}$	The set of users that are definitely a reverse kNN of an object with location ℓ based on a lower bound.
$\mathcal{B}^{\uparrow}{}_{\ell}$	The set of users that can be a reverse kNN of an object with location ℓ based on an upper bound.
u^+	Super-user, constructed in the same way as q^+ by grouping the users in U.
$u^+.l$	Location of super-user.
$u^+.d^{\cap}$, $u^+.d^{\cup}$	Intersection (union) of the text of the users in super-user.
$\mathcal{R}_k(u^+)$	The current k·th best minimum combined spatial-textual similarity of any object for any user $u \in u^+$.
$\mathcal{R}_k(u)$	The k·th best combined spatial-textual similarity of any object for u.
$\mathrm{CS}^{\downarrow}(\ell, u^+)$, $\mathrm{CS}^{\uparrow}(\ell, u^+)$	Minimum (maximum) combined spatial-textual similarity of the object p w.r.t. u^+ when $p.l = \ell$.
$\mathrm{SS}^{\downarrow}(\ell, u^+)$, $\mathrm{SS}^{\uparrow}(\ell, u^+)$	Minimum (maximum) spatial similarity of the object p w.r.t. u^+ when $p.l = \ell$.
$\mathrm{TS}^{\downarrow}(\ell, u^+)$, $\mathrm{TS}^{\uparrow}(\ell, u^+)$	Minimum (maximum) textual similarity of the object p w.r.t. u^+ when $p.l = \ell$.
$\mathrm{CS}^{\downarrow}(\ell, u)$, $\mathrm{CS}^{\uparrow}(\ell, u)$	Minimum (maximum) combined spatial-textual similarity of the object p w.r.t. u when $p.l = \ell$.
$\mathrm{SS}^{\downarrow}(\ell, u)$, $\mathrm{SS}^{\uparrow}(\ell, u)$	Minimum (maximum) spatial similarity of the object p w.r.t. u when $p.l = \ell$.
$\mathrm{TS}^{\downarrow}(\ell, u)$, $\mathrm{TS}^{\uparrow}(\ell, u)$	Minimum (maximum) textual similarity of the object p w.r.t. u when $p.l = \ell$.
$W^{\uparrow}$	The set of ω number of keywords of the highest weights from $u^+.d^{\cup} \cap W$.
$W_u{}^{\uparrow}$	The set of ω number of keywords of the highest weights from $u.d \cap W$.
$W^{\uparrow}{}_{w,u}$	The set of ω number of keywords of the highest weights from $W \cap u.d$ such that $W^{\uparrow}{}_{w,u} \cap w \neq \varnothing$.
M	The set of all possible combinations of ω number of keywords from W.
Section 4.6:	
LO_u	A min-priority queue to keep the k objects with the best lower bounds w.r.t. u found so far.
Section 4.7:	
EU	A node of an MIUR-tree of the users.

top-k objects for multiple queries (users) are presented, any of the approaches can be applied. Any other approach to efficiently batch process multiple top-k spatial-textual queries (users) could also be used. As the details of efficient solutions to jointly compute the top-k objects for the users are described in Chapter 3, we proceed to present our method to efficiently find the best candidate combinations.

ALGORITHM 8: CANDIDATE SELECTION

Input: A set of users U, a set of candidate locations L, a set of candidate keywords W, the number of keywords to select ω, the number of top relevant objects k, and a particular object p.
Output: The optimal location from L and the set of ω number of keywords from W for p.
Initialize a max-priority queue QL.
$\mathcal{B}_p \leftarrow \varnothing$.
for *each* $\ell \in L$ **do**
  if $CS^{\uparrow}(\ell, u^+) \geq \mathcal{R}_k(u^+)$ **then**
    for *each* $u \in U$ **do**
      if $CS^{\uparrow}(\ell, u) \geq \mathcal{R}_k(u)$ **then**
        $\mathcal{B}^{\uparrow}{}_{\ell} \leftarrow u$
      if $CS^{\downarrow}(\ell, u) \geq \mathcal{R}_k(u)$ **then**
        $\mathcal{B}^{\downarrow}{}_{\ell} \leftarrow u$
  ENQUEUE(QL, ℓ, $|\mathcal{B}^{\uparrow}{}_{\ell}|$)
while QL $\neq \varnothing$ **do**
  $\ell \leftarrow$ DEQUEUE(QL)
  if $|\mathcal{B}^{\uparrow}{}_{\ell}| < |\mathcal{B}_p|$ **then** BREAK
  else if $|\mathcal{B}^{\downarrow}{}_{\ell}| \geq |\mathcal{B}_p|$ **then**
    $p.l \leftarrow \ell$
  else
    $W' \leftarrow$ Find best candidate keyword set for ℓ using either APPROXIMATE or EXACT method.
    $p'.l = \ell$; $p'.d = W'$
    if $|\mathcal{B}_{p'}| > |\mathcal{B}_p|$ **then**
      $p.l \leftarrow \ell$; $p.d \leftarrow W'$
return $\langle p.l, p.d \rangle$

4.5.1 Candidate Selection

We use the query grouping technique presented in Section 3.3.3.2 to group the users and construct the super-user, $\mathcal{R}_k(u^+)$, which is essentially the same as the 'super-query', q^+ in Chapter 3. Let the similarity score of the k·th ranked object with respect to each user $u \in U$ be $\mathcal{R}_k(u)$, and the k·th best lower bound similarity score with respect to the super-user be $\mathcal{R}_k(u^+)$ such that for any user $u \in U$, $\mathcal{R}_k(u^+) \leq \mathcal{R}_k(u)$. The details of finding the scores are presented in Section 3.3.3.4. These scores are used to facilitate the pruning of the un-promising candidates.

As shown in Lemma 2, even when there is only a single candidate location, the candidate keyword selection process alone is NP-hard. Therefore, we propose a *spatial-first* pruning

technique to select the best candidate combination of a location and a set of keywords. For each candidate location ℓ, the idea is to estimate the number of users that can be in $\mathcal{B}_p$ if $p.l = \ell$ for the specific object p. Then the candidates are considered in a best-first manner so that the most promising candidates are processed first. Several pruning strategies are used during this process, which are described now.

Algorithm 8 shows the pseudocode of the steps to select the candidate location and keywords for the MaxST problem. The pruning techniques used in this process use an upper and a lower bound estimation of relevance of the candidate combinations with respect to the users.

Upper bound estimation. For each $\ell \in L$, the combined spatial-textual upper bound similarity is computed in two steps: (i) for the super-user u^+, denoted as $\text{CS}^\uparrow(\ell, u^+)$ such that for any user $u \in U$, the similarity between p and u is at most the $\text{CS}^\uparrow(\ell, u^+)$, when $p.l = \ell$; and (ii) for each individual $u \in U$ such that the similarity $\text{CS}(p, u)$ is at most $\text{CS}^\uparrow(\ell, u)$, when $p.l = \ell$.

When using Euclidean distance for spatial similarity, the spatial upper bound $\text{SS}^\uparrow(\ell, u^+)$ is computed from Equation 3.2 using the minimum Euclidean distance between ℓ and u^+, as $u^+.l$ is the MBR for all of the users. For text relevance, a straightforward way is to consider the relevance as 1 (maximum), when the score is normalized within $[0, 1]$. But we can achieve a tighter bound using the following lemma:

Lemma 3. *Let $W^\uparrow$ be the set of ω keywords of the highest weights from $(u^+.d^\cup \cap W)$. The text relevance between $p.d$ and a user $u \in U$ after adding at most ω candidate keywords is always less than or equal to the score $TS((p.d \cup W^\uparrow), u^+.d^\cup)$,*

$$TS^\uparrow(p, u^+) = TS((p.d \cup W^\uparrow), u^+.d^\cup)\,.$$

Proof. The text relevance between a user u and p can change by adding only the keywords that are present in $u.d$. As $u^+.d^\cup$ is the union of the text of all the users in U, the text relevance w.r.t. any user u can be increased only by adding the candidate keywords that are present in $u^+.d^\cup$. Let w_1 and w_2 be two keywords in $W^\uparrow$ where the weight of w_1 is greater than the weight of w_2 and $\omega = 1$. If a user u has both w_1 and w_2 in the text description, then from Equation 3.3, the text relevance of p w.r.t. u obtained by adding w_1 must be greater than or equal to the text relevance obtained by adding w_2. Even if a user u does not have all of the keywords from $W^\uparrow$, the lemma still provides an upper bound estimation of text relevance that can be achieved by adding ω candidate keywords. □

So, the upper bound estimation of relevance of a candidate location w.r.t. the super-user u^+ is:

$$\mathrm{CS}^{\uparrow}(\ell, u^+) = \alpha \cdot \mathrm{SS}^{\uparrow}(\ell, u^+.l) + (1-\alpha) \cdot \mathrm{TS}^{\uparrow}(p, u^+)\,.$$

Similarly, an upper bound estimation of a candidate location ℓ w.r.t. any particular user u can be computed as:

$$\mathrm{CS}^{\uparrow}(\ell, u) = \alpha \cdot \mathrm{SS}^{\uparrow}(\ell, u.l) + (1-\alpha) \cdot \mathrm{TS}^{\uparrow}(p, u)\,.$$

Here, $\mathrm{TS}^{\uparrow}(p.d, u.d) = \mathrm{TS}(p.d \cup {W_u}^{\uparrow}, u.d)$, where ${W_u}^{\uparrow}$ is the set of ω keywords of the highest weights from $(u.d \cap W)$.

Lower bound estimation. For text relevance, the minimum score is computed from the original text description of p. The spatial lower bound is computed using the maximum Euclidean distance. So, the lower bound estimation of $\ell \in L$ w.r.t. u^+ is:

$$\mathrm{CS}^{\downarrow}(\ell, u^+) = \alpha \cdot \mathrm{SS}^{\downarrow}(\ell, u^+.l) + (1-\alpha) \cdot \mathrm{TS}^{\downarrow}(p.d, u^+.d^{\cap})\,.$$

Pruning techniques. We denote the set of users that can be in $\mathcal{B}_p$ for $p.l = \ell$ as ${\mathcal{B}}^{\uparrow}{}_{\ell}$, and the set of users that are definitely in $\mathcal{B}_p$ when $p.l = \ell$ as ${\mathcal{B}}^{\downarrow}{}_{\ell}$. The number of users that find p as a top-k object is initialized as an empty set. The steps and the pruning strategies employed in Algorithm 8 can be summarized as follows:

- **Initialize necessary user lists:** As the similarity score of the $k\cdot$th ranked object for any user u, $\mathcal{R}_k(u)$ satisfies the condition $\mathcal{R}_k(u^+) \leq \mathcal{R}_k(u)$. Therefore if $\mathrm{CS}^{\uparrow}(\ell, u^+) < \mathcal{R}_k(u^+)$, then no user can have p as a top-k object for the candidate location ℓ. Otherwise, $\mathrm{CS}^{\uparrow}(\ell, u)$ is computed for each user. If $\mathrm{CS}^{\uparrow}(\ell, u) \geq \mathcal{R}_k(u)$, then u is included in ${\mathcal{B}}^{\uparrow}{}_{\ell}$. A list of such users, ${\mathcal{B}}^{\uparrow}{}_{\ell}$ is obtained for each candidate location ℓ, (Lines 8.8-8.9). For each ℓ, if the lower bound similarity $\mathrm{CS}^{\downarrow}(\ell, u) \geq \mathcal{R}_k(u)$, then u is added to the corresponding list ${\mathcal{B}}^{\downarrow}{}_{\ell}$ (Lines 8.10-8.11).

- Here, a *best-first traversal* technique is exploited. A max-priority queue QL of candidate locations is maintained according to the cardinality $|{\mathcal{B}}^{\uparrow}{}_{\ell}|$, so that the most promising candidates are processed first. In each iteration the location, the current top location ℓ with the maximum $|{\mathcal{B}}^{\uparrow}{}_{\ell}|$ is selected (Line 8.14).

- If the cardinality of $\mathcal{B}^{\uparrow}{}_{\ell}$ of the current top location ℓ is less than the best $|\mathcal{B}_p|$ found so far, a better tuple from the subsequent entries of QL is not possible. Thus, the computation can be *early terminated* (Line 8.15).

- Since all users in $\mathcal{B}^{\downarrow}{}_{\ell}$ have p as a top-k object for $p.l = \ell$, irrespective of the keyword selection, a check to see if $|\mathcal{B}^{\downarrow}{}_{\ell}|$ is greater than the current best $|\mathcal{B}_p|$ can be used to *avoid computing the candidate keywords* for this condition (Lines 8.16-8.17).

- Otherwise, the best candidate keyword set, W' is determined for ℓ. An approximate or an exact method presented in the following section is used to select W' (Line 8.19). The location and the text description of p is updated with ℓ and W' accordingly (Lines 8.21-8.22).

4.5.2 Candidate Keyword Selection

Recall that the best candidate keyword set W' that provides the maximum cardinality of $\mathcal{B}_p$ has to be determined for $p.l = \ell$ (Line 8.19) in Algorithm 8, where ℓ is the location at the top of the priority queue at that iteration. As this is an NP-hard problem, an approximation algorithm is first developed. An exact method that uses several pruning strategies is also presented, which can serve as a naive baseline.

4.5.2.1 Approximate Algorithm

The candidate keyword selection problem was shown to be NP-hard in Lemma 2 using a reduction from the Maximum Coverage (MC) problem. For the MC problem, a greedy algorithm exists, which is a $(1 - 1/e) \simeq 0.632$ approximation algorithm. In the MC problem, the input is a collection of sets $S = \{S_1, S_2, \ldots, S_m\}$ and a number n. The greedy algorithm chooses a set in each step that contains the largest number of uncovered elements until exactly n sets are selected. This greedy algorithm was shown to be the best-possible polynomial time approximation algorithm for the MC problem by Feige [38]. Inspired by this algorithm, we propose an approximate algorithm to select the candidate keywords in our algorithm when $p.l = \ell$ (Line 8.19 of Algorithm 8). However, some preprocessing must be done before applying the greedy algorithm, and is discussed next.

Preprocessing. For each $w \in W$, we generate a list LW_w of the users such that these users can be in $\mathcal{B}_p$ based on an upper bound estimation, where $p.d = W'$ and $W' \cap w \neq \varnothing$. As the set

$\mathcal{B}^{\uparrow}{}_{\ell}$ is already generated based on this upper bound, only the users in $\mathcal{B}^{\uparrow}{}_{\ell}$ need to be considered for this step. Let $W^{\uparrow}{}_{w,u}$ be a set of the ω highest weighed keywords from $W \cap u.d$ such that $W^{\uparrow}{}_{w,u} \cap w \neq \varnothing$. When $p.d = W^{\uparrow}{}_{w,u}$ and $p.l = \ell$, a user u can be in $\mathcal{B}_p$ if $\mathrm{CS}(p,u) \geq \mathcal{R}_k(u)$. Such users are included in the corresponding list, LW_w for each $w \in W$.

Approximating the best candidate keyword set. Recall that in the MC problem, the objective is to find a subset $S' \subseteq S$ such that $|S'| \leq n$ and the number of covered elements by S', $|\cup_{S_i \in S'} S_i|$ is maximized, given a collection of sets $S = S_1, S_2, \ldots, S_m$ and a number n. In our case, the collection of the sets are the collection of LW_w for each w and the number n is ω. The greedy approach of MC is applied in our problem to find the best set of candidate keywords W' of size ω such that $|\cup_{w \in W'} \mathrm{LW}_w|$ is maximized. This set W' is returned as the best candidate keyword set for the location ℓ.

4.5.2.2 Exact Algorithm

The number of candidates can be small in some applications. Moreover, the search space can be pruned using several strategies when selecting the candidate keyword set. This motivates us to develop an exact algorithm for selecting the best keyword set W' in a MaxST query. The pseudocode is presented in Algorithm 9 and the pruning techniques are now explained.

- **Pruning users:** According to the definition of $\mathrm{CS}^{\uparrow}(\ell, u)$, only the users in $\mathcal{B}^{\uparrow}{}_{\ell}$ can have p as a top-k object when $p.l = \ell$. So only the users in $\mathcal{B}^{\uparrow}{}_{\ell}$ must be considered.

- **Pruning candidate keywords:** Let the union of the text description of the users in $\mathcal{B}^{\uparrow}{}_{\ell}$ be WU (Line 9.3). Only the candidate keywords that are contained in at least one of those users, $W \cap \mathrm{WU}$, are necessary.

- Let M be the set of the combinations of ω number of keywords from $W \cap \mathrm{WU}$. For a keyword combination $m \in M$, only those users where $m \cap u.d \neq \varnothing$ are processed.

- **Early termination:** If $|W \cap \mathrm{WU}| \leq \omega$, this is the only possible candidate keyword set. So the process terminates and $W \cap \mathrm{WU}$ is returned as the best candidate keyword set for ℓ as shown in Lines 9.6-9.7.

- If the lower bound relevance, $\mathrm{CS}^{\downarrow}(\ell, u) \geq \mathcal{R}_k(u)$, then u is included in $\mathcal{B}_{p'}$, where $p'.l = \ell$ and $p'.d$ is the candidate keyword combination m currently under consideration (Lines 9.14

ALGORITHM 9: EXACT METHOD TO SELECT CANDIDATE KEYWORD

9.1 **Input:** A set of users U, a candidate location $\ell \in L$ with the maximum $\mathcal{B}^{\uparrow}{}_{\ell}$ in the current iteration, a set of candidate keywords W, the number of keywords to select ω, and the number of top relevant objects k.
9.2 **Output:** The optimal set of ω keywords from W for the location ℓ.
9.3 $\mathrm{WU} \leftarrow \bigcup_{u \in \mathcal{B}^{\uparrow}{}_{\ell}} (u.d)$
9.4 $W' \leftarrow \varnothing$;
9.5 $best \leftarrow 0$
9.6 **if** $|W \cap WU| \leq \omega$ **then**
9.7 $\quad W' \leftarrow (W \cap \mathrm{WU})$
9.8 **else**
9.9 $\quad M \leftarrow$ combinations of ω number of keywords from $W \cap \mathrm{WU}$.
9.10 $\quad p'.l = \ell$
9.11 $\quad$ **for** *each* $m \in M$ **do**
9.12 $\quad\quad p'.d = m$
9.13 $\quad\quad$ **for** *each* $u \in \mathcal{B}^{\uparrow}{}_{\ell}$ **do**
9.14 $\quad\quad\quad$ **if** $CS^{\downarrow}(\ell, u) \geq \mathcal{R}_k(u)$ **then**
9.15 $\quad\quad\quad\quad \mathcal{B}_{p'} \leftarrow u$
9.16 $\quad\quad\quad$ **else if** $m \cap u.d \neq \emptyset$ **then**
9.17 $\quad\quad\quad\quad$ **if** $CS(p', u) \geq \mathcal{R}_k(u)$ **then**
9.18 $\quad\quad\quad\quad\quad \mathcal{B}_{p'} \leftarrow u$
9.19 $\quad\quad$ **if** $|\mathcal{B}_{p'}| > best$ **then**
9.20 $\quad\quad\quad W' \leftarrow m$
9.21 $\quad\quad\quad best \leftarrow |\mathcal{B}_{p'}|$
9.22 return W'

- 9.15). If the cardinality of $\mathcal{B}_{p'}$ is greater than that of the current best keyword combination, the current best is updated (Lines 9.19 - 9.21).

4.6 INDIV-U Approach

In this approach, instead of generating the list of the top-k objects for each of the users, the idea is to estimate the number of objects that have a higher similarity than each candidate w.r.t. each $u \in U$ with a single traversal of the objects. This number is used to prune the users and the candidates that cannot affect the result. The steps of the algorithm relies on computing the similarity estimations of the objects presented in Section 3.3.3.3 w.r.t. the users and the super-user. In the following we present the steps of our algorithm using these bounds.

ALGORITHM 10: INDIV-U APPROACH

10.1 **Input:** A set of users U, a set of candidate locations L, a set of candidate keywords W, the number of keywords to select ω, the number of top relevant objects k, a particular object p, and the MIR-tree of the set of objects O.
10.2 **Output:** The optimal location from L and the set of ω number of keywords from W for p.
10.3 Initialize max-priority queue QL, QE; a min-priority queue LS.
10.4 Initialize an array LO_u of $|U|$ min-priority queues for each $u \in U$.
10.5 $\text{LU} \leftarrow U$
10.6 $\langle \ell, W' \rangle_{\text{best}} \leftarrow \varnothing$
10.7 **for** *each* $\ell \in L$ **do**
10.8 $\quad \mathcal{B}^{\downarrow}{}_{\ell} \leftarrow \varnothing$; $\mathcal{B}^{\uparrow}{}_{\ell} \leftarrow U$
10.9 ENQUEUE(QE, MIR-tree(root), $\text{CS}^{\downarrow}$(MIR-tree(root), u^+))
10.10 **while** QE $\neq \varnothing$ **do**
10.11 $\quad E \leftarrow$ DEQUEUE(QE)
10.12 $\quad$ **if** *E is an object* **then**
10.13 $\quad\quad$ **if** $|\text{LS}| < k \parallel CS^{\uparrow}(E, u^+) \geq \mathcal{R}_k(u^+)$ **then**
10.14 $\quad\quad\quad$ ENQUEUE(LS, E, $\text{CS}^{\downarrow}(E, u^+)$)
10.15 $\quad\quad\quad$ Update $\mathcal{R}_k(u^+)$
10.16 $\quad\quad\quad$ **for** *each* $u \in \text{LU}$ **do**
10.17 $\quad\quad\quad\quad$ next $\leftarrow$ **true**
10.18 $\quad\quad\quad\quad$ **if** $|\text{LO}_u| < k \parallel CS^{\uparrow}(E, u) \geq \mathcal{R}_k(u)$ **then**
10.19 $\quad\quad\quad\quad\quad$ ENQUEUE(LO_u, E, $\text{CS}^{\downarrow}(E, u)$)
10.20 $\quad\quad\quad\quad\quad$ Update $\mathcal{R}_k(u)$
10.21 $\quad\quad\quad\quad\quad$ **for** *each* $\ell \in L$ **do**
10.22 $\quad\quad\quad\quad\quad\quad$ **if** *u is in* $\mathcal{B}^{\uparrow}{}_{\ell}$ *&* $|\text{LO}_u| \geq k$ *&* $CS^{\uparrow}(\ell, u) < \mathcal{R}_k(u)$ **then**
10.23 $\quad\quad\quad\quad\quad\quad\quad$ Remove u from $\mathcal{B}^{\uparrow}{}_{\ell}$
10.24 $\quad\quad\quad\quad\quad\quad$ **else** next $\leftarrow$ **false**
10.25 $\quad\quad\quad\quad\quad$ **if** next is ***true*** **then**
10.26 $\quad\quad\quad\quad\quad\quad$ Remove u from LU
10.27 $\quad$ **else if** $|\text{LS}| < k \parallel CS^{\uparrow}(E, u^+) \geq \mathcal{R}_k(u^+)$ **then**
10.28 $\quad\quad$ READ(E)
10.29 $\quad\quad$ **for** *each element e of E* **do**
10.30 $\quad\quad\quad$ **for** *each* $u \in \text{LU}$ **do**
10.31 $\quad\quad\quad\quad$ **if** $|\text{LO}_u| < k \parallel CS^{\uparrow}(E, u) \geq \mathcal{R}_k(u)$ **then**
10.32 $\quad\quad\quad\quad\quad$ ENQUEUE(QE, e, $\text{CS}^{\downarrow}(e, u^+)$)
10.33 $\quad\quad\quad\quad\quad$ BREAK
10.34 **for** *each* $\ell \in L$ **do**
10.35 $\quad$ **for** *each* $u \in \mathcal{B}^{\uparrow}{}_{\ell}$ **do**
10.36 $\quad\quad$ **if** $CS^{\downarrow}(\ell, u) \geq \mathcal{R}_k(u)$ **then** $\mathcal{B}^{\downarrow}{}_{\ell} \leftarrow u$
10.37 $\quad$ ENQUEUE(QL, ℓ, $|\mathcal{B}^{\uparrow}{}_{\ell}|$)
10.38 Execute Lines 8.13 - 8.23 of Algorithm 8.

4.6.1 Algorithm

We present an approach that avoids computing the top-k object for users that do not affect the final result. Here, we apply a spatial-first strategy as well. Algorithm 10 presents the pseudocode of the approach. Recall that the set of users that can be in $\mathcal{B}_p$ for $p.l = \ell$ is denoted as $\mathcal{B}^{\uparrow}{}_{\ell}$, and the set of users that are definitely in $\mathcal{B}_p$ when $p.l = \ell$ as $\mathcal{B}^{\downarrow}{}_{\ell}$. The algorithm works as follows:

- **Initialization:** For a candidate location $\ell \in L$, $\mathcal{B}^{\downarrow}{}_{\ell}$ is initialized with an empty set and $\mathcal{B}^{\uparrow}{}_{\ell}$ is initialized with the set U (Line 10.8). Let LU be the set of the users that are in $\mathcal{B}^{\uparrow}{}_{\ell}$ of at least one candidate $\ell \in L$, which is also initialized with all the users in U. For each user u, a priority queue LO_u is maintained to track the current top-k objects of u found so far. A separate priority queue LS is also maintained for the super-user to track k objects with the best lower bound similarity found so far.

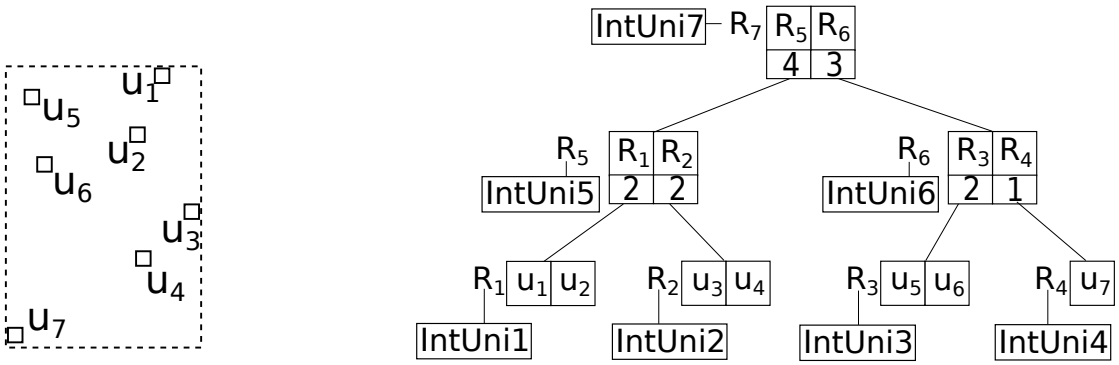
- Lines 10.12 - 10.20 show how LS and LO_u for each user u are filled with k objects with the highest lower bound similarity values found so far. The actual objects are used instead of object nodes in LS and each LO_u for better relevance estimations. The corresponding values of $\mathcal{R}_k(u)$ and $\mathcal{R}_k(u^+)$ are also updated.

- **Pruning object nodes using the super-user:** If the upper bound similarity $\text{CS}^{\uparrow}(E, u^+)$ is less than the current $\mathcal{R}_k(u^+)$, then E cannot contain any object that can be a top-k object for any users in U (Line 10.27). Otherwise, the upper bound similarity $\text{CS}^{\uparrow}(E, u)$ is computed w.r.t. each user $u \in \text{LU}$.

- **Pruning object nodes for individual users:** Similarly, if $\text{CS}^{\uparrow}(E, u)$ is less than the current $\mathcal{R}_k(u)$, then E for u does not need to be considered. If E is not needed for any of the users currently in LU, the subtree rooted at E can be pruned from further consideration (Lines 10.29 - 10.33).

- **Pruning users for candidate locations:** If the upper bound similarity $\text{CS}^{\uparrow}(\ell, u)$ of a candidate ℓ w.r.t. a user u is less than the current $\mathcal{R}_k(u)$, then u cannot be in $\mathcal{B}^{\uparrow}{}_{\ell}$ for $p.l = \ell$. If u is discarded from the $\mathcal{B}^{\uparrow}{}_{\ell}$ for all of the candidates, then u is removed from LU as well. Computing the top-k objects for such users can be avoided (Lines 10.21 - 10.26).

- The set of users that are definitely an RkNN for $p.l = \ell$, $\mathcal{B}^{\downarrow}{}_{\ell}$, are found using their corresponding $\mathcal{R}_k(u)$ and $\text{CS}^{\downarrow}(\ell, u)$ values (Lines 10.34 - 10.36).



(a) The set of example users

(b) The example Modified IUR-tree (MIUR tree)

ID	**Intersection** ($t_1t_2t_3t_4$)	**Union** ($t_1t_2t_3t_4$)
IntUni1	1 0 1 1	1 0 1 1
IntUni2	1 0 0 0	1 1 1 0
IntUni3	1 0 0 0	1 1 0 1
IntUni4	1 0 0 1	1 0 0 1
IntUni5	1 0 0 0	1 1 1 1
IntUni6	1 0 0 0	1 1 0 1
IntUni7	1 0 0 0	1 1 1 1

Figure 4.4: Example of constructing Modified IUR-tree (MIUR tree)

If a node E cannot be pruned, the entries of E are retrieved and placed in the queue. As traversal down the tree continues, the similarity bounds approach the actual values. When the leaf nodes are reached, the values for the actual objects are computed instead of nodes that cannot be pruned, and the objects that can be a top-k object of the necessary users are found. After the traversal of the MIR-tree, the list of users is obtained, $\mathcal{B}^{\uparrow}{}_{\ell}$ and $\mathcal{B}^{\downarrow}{}_{\ell}$ for each candidate location, and then the rest of the candidate selection process is the same as the GRP-TOPK approach (Lines 8.13 - 8.23 of Algorithm 8).

4.7 INDEX-U Approach

In the previous method, the users are pruned by checking them individually against the candidates and the retrieved objects. In this section we propose a new index, the Modified IUR-tree (MIUR-tree) to store the users, where the motivation is to improve efficiency by applying the pruning techniques over a hierarchical structure of the users instead of the individual users.

ALGORITHM 11: INDEX-U APPROACH

11.1 **Input:** A set of candidate locations L, a set of candidate keywords W, the number of keywords to select ω, the number of top relevant objects k, a particular object p, the MIR-tree on the set of objects O, and the MIUR-tree on the set of users U.
11.2 **Output:** The optimal location from L and the set of ω number of keywords from W for p.
11.3 Initialize a max-priority queue QL.
11.4 EU $\leftarrow$ MIUR-tree(root)
11.5 Compute $\mathcal{R}_k(\text{EU})$.
11.6 $\text{RO}_{\text{EU}} \leftarrow$ Set of objects o with $\text{CS}^{\uparrow}(o, \text{EU}) \geq \mathcal{R}_k(\text{EU})$ (by executing Algorithm 3 with EU in place of u^+)
11.7 **for** *each* $\ell \in L$ **do**
11.8 **if** $CS^{\uparrow}(\ell, \text{EU}) \geq \mathcal{R}_k(\text{EU})$ **then**
11.9 $\mathcal{B}^{\uparrow}{}_{\ell} \leftarrow$ EU
11.10 ENQUEUE(QL, ℓ, $|\mathcal{B}^{\uparrow}{}_{\ell}|$)
11.11 **while** QL $\neq \varnothing$ **do**
11.12 $\ell \leftarrow$ DEQUEUE(QL)
11.13 **if** $\mathcal{B}^{\uparrow}{}_{\ell} \neq \varnothing$ **then**
11.14 EU $\leftarrow$ The node in $\mathcal{B}^{\uparrow}{}_{\ell}$ with maximum users
11.15 **for** *each* *eu* $\in$ EU **do**
11.16 Update $\mathcal{R}_k(\text{eu})$ using the set of objects RO_{EU} by executing Lines 4.3 - 4.11 of Algorithm 4.
11.17 $\text{RO}_{\text{eu}} \leftarrow$ Set of objects $o \in \text{RO}_{\text{EU}}$ where $\text{CS}^{\uparrow}(o, \text{eu}) \geq \mathcal{R}_k(\text{eu})$
11.18 **for** *each* $\ell \in L$ **do**
11.19 **if** $\mathcal{B}^{\uparrow}{}_{\ell}$ contains EU $\parallel$ $CS^{\uparrow}(\ell, \text{eu}) \geq \mathcal{R}_k(\text{eu})$ **then**
11.20 $\mathcal{B}^{\uparrow}{}_{\ell} \leftarrow$ eu
11.21 Update QL.
11.22 **else**
11.23 Execute Lines 10.36- 10.38 of Algorithm 10.

Modified IUR-tree (MIUR-tree). An MIUR-tree is essentially an R-tree where each node is augmented with a reference to the union and the intersection vector of the keywords appearing in the subtree. Each node R contains a number of entries, each consists of a reference to a child node, the MBR of all entries of the child node, and an identifier of a vector of keywords. If R is a leaf node, this is the identifier of the vector of the text description of an object o. If R is a non-leaf node then it has a reference to the union and intersection of all text descriptions in the entries of the child node. It also contains the number of actual objects stored in the subtree rooted at R.

Figure 4.4 illustrates the MIUR-tree for $U = \{u_1, u_2, \ldots, u_7\}$ of Figure 4.4a, where the MBRs are constructed according to the IR-tree (Figure 4.4b). The table shows the text vectors of the nodes for the users presented in Table 3.2.

Algorithm. The pseudocode of the approach is presented in Algorithm 11. The root of the MIUR-tree is essentially the same as the super-user u^+ in the previous methods, where the MBR of the root encloses the location of all of the users. The MIR-tree of the objects is traversed starting at the root node, EU of the MIUR-tree, to obtain the k·th best lower bound $\mathcal{R}_k(\text{EU})$ and the list RO of the object, such that each $o \in \text{RO}$, $\text{CS}^{\uparrow}(o, \text{EU}) \geq \mathcal{R}_k(\text{EU})$ (Lines 11.5 - 11.6). The details of the traversal are explained in Algorithm 3 of the previous chapter.

For each $\ell \in L$, a list ${\mathcal{B}^{\uparrow}}_{\ell}$ is maintained, but unlike Algorithm 8, ${\mathcal{B}^{\uparrow}}_{\ell}$ may now contain user nodes. In each iteration, the location ℓ is selected with the maximum $|{\mathcal{B}^{\uparrow}}_{\ell}|$. If there is a user node in a ${\mathcal{B}^{\uparrow}}_{\ell}$, the number of actual users stored in that subtree is used to compute the number of users in ${\mathcal{B}^{\uparrow}}_{\ell}$. The following steps are executed to access the MIUR-tree:

1. If there is any non-leaf node in ${\mathcal{B}^{\uparrow}}_{\ell}$, the non-leaf node $EU \in {\mathcal{B}^{\uparrow}}_{\ell}$ is dequeued with the maximum number of users stored in the subtree.

    a) For each element eu $\in$ EU, Algorithm 4 is executed to update $\mathcal{R}_k$(eu) using the set of objects RO(EU) of its parent node, RO(eu) is also updated (Lines 11.16 - 11.17).

    b) For each $\ell \in L$ including ℓ, if EU $\in {\mathcal{B}^{\uparrow}}_{\ell}$, then ${\mathcal{B}^{\uparrow}}_{\ell}$ is updated with the users eu $\in$ EU based on the corresponding upper bound scores. The priority queue QL is also updated. In this way, a node of the MIUR-tree is only accessed at most once.

2. Otherwise, the rest of the algorithm to find the best candidate location and keyword set combination of the MaxST query is the same as Lines 10.36- 10.38 of Algorithm 10.

In this best-first method, the users that are in the list ${\mathcal{B}^{\uparrow}}_{\ell}$ of the most promising candidates are accessed first. In addition, the top-k objects are not computed for the users that are not necessary to determine the result candidate combinations.

4.8 Adding Visibility Requirements

In this section we extend our solutions to support answering the MaxST query in obstructed space, where the spatial relevance of an object is its visibility w.r.t. a user. The notation used in this section is presented in Table 4.3.

Table 4.3: Notation used in Section 4.8

Symbol	**Description**
Section 4.8.2:	
c	A cell of the auxiliary Quadtree.
$OR(u)$	The obstructed region of u.
$OL(c)$	A list of the users for which c is completely inside $OR(u)$.
$len^{\downarrow}(E)$, $len^{\uparrow}(E)$	The minimum (maximum) length of an object stored in the subtree rooted at node E.
$o.\theta$	The angle of o w.r.t. the Cartesian X-axis.
Section 4.8.3:	
$\chi_{E,u}$, $\hat{\chi}_{E,u}$	The set of leaf level cells of the Quadtree that are visible (not visible) from u and intersects with at least one object in node E.
χ_{E,u^+}, $\hat{\chi}_{E,u^+}$	The set of leaf level cells of the Quadtree that are visible (not visible) from at least one user u from u^+ and intersects with at least one object in node E.
$\boldsymbol{d}^{\downarrow}{}_{\perp}(o.l, u.l)$	The perpendicular distance of the line $o.l$ from the location $u.l$.
$\mathrm{VL}^{\downarrow}_{\Delta}(\Delta o.l, u.l)$, $\mathrm{VL}^{\uparrow}_{\Delta}(\Delta o.l, u.l)$	Minimum (maximum) perceived length of segment $\Delta o.l$ from $u.l$.
$\angle^{\downarrow}(\Delta o.l, u.l)$, $\angle^{\uparrow}(\Delta o.l, u.l)$	Minimum (maximum) angle between the segment $\Delta o.l$ and $u.l$.
$\angle^{\downarrow}(\Delta o.l, u^+)$, $\angle^{\uparrow}(\Delta o.l, u^+)$	Minimum (maximum) angle between the segment $\Delta o.l$ and any user u from u^+.

Challenges with visibility measures. The additional challenge of the visibility measure is that the visibility of an object o w.r.t. a user u depends on their locations, and the locations of the other objects (obstacles) in between them, where the spatial proximity (e.g., Euclidean distance) of o w.r.t. u depends only on their own locations. Therefore, to incorporate the visibility measure in the solutions presented in Sections 4.5 - 4.7, we seek techniques that:

- Efficiently estimate and calculate the visibility of an object or a candidate without requiring the retrieval of other objects whenever possible.
- Pruning of unnecessary objects, candidates, and users based on visibility.

Visibility extension overview. At the highest level, the following modifications are made in the solutions of Sections 4.5 - 4.7 to incorporate visibility and achieve the above goals:

1. The MIR-tree is extended, denoted as an *OIR-tree*, to store the set of objects O, along with some additional information to support visibility computation. Specifically, an auxiliary Quadtree structure is maintained to identify the portions of space that are visible to a user, and associate that information with the OIR-tree. A space partitioning technique

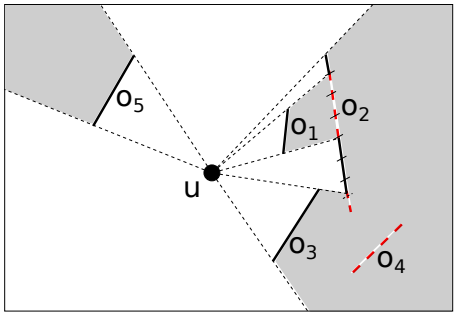


Figure 4.5: An example obstructed region. The visible portions of the objects from the user u are shown with solid black lines, and the obstructed portions are shown with dotted red lines.

is used to construct the auxiliary Quadtree to facilitate finding the visible segments of an object or a candidate location.

2. As our proposed solutions rely on estimating the similarity bounds, the visibility estimation bounds of a candidate and an object o w.r.t. a user (and super-user) using the nodes of the OIR-tree are presented. These bounds do not require the retrieval of the other objects in between o and the user to estimate visibility, which is a major contribution of this chapter.

3. Several additional pruning techniques are also presented which use visibility to prune unnecessary objects, users, and candidates.

In Section 4.8.2, the extension of our proposed indexes to incorporate the visibility measure is presented. The modifications of the algorithms to answer the MaxST query for visibility are explored in Section 4.8.3.

4.8.1 Preliminaries

In this section, the notion of an *obstructed region* of a user is presented, which is crucial to measure the visibility.

Definition 3. ***Obstructed region (OR(u)).*** *Given a set of obstacles O in space, an obstructed region w.r.t. a user u,* OR(u) *consists of all the points in the space such that a straight line connecting any of these points and $u.l$ passes through at least one object $o \in O$.*

Based on the definition, an important observation is: if an object is completely inside the obstructed region of a user u, no point of that object is visible from u. As shown in Figure 4.5,

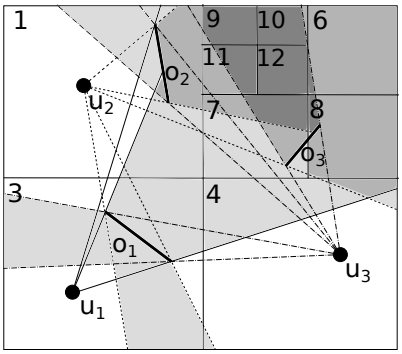
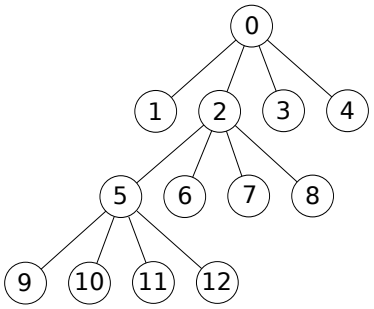


Figure 4.6: An example partitioning for the construction of the auxiliary Quadtree. A region that is not obstructed from any of the users is shown with white. A darker shade represents that the region is obstructed from more number of users than a region with a lighter shade.



Cell ID	OL(c)
c_2	u_1
c_5	u_2
c_6	u_2
c_{10}	u_3
c_{12}	u_3

Figure 4.7: The auxiliary Quadtree constructed for the example in Figure 4.6. The table shows the list of the users for the cells with at least one entry. The list of the users for a cell contains the user IDs for which the cell is completely obstructed.

the shaded region is the obstructed region of the user u due to the presence of the objects $o_1, \ldots, o_5$. The visible portions of the objects are represented with black, and the obstructed portions (the portions that are inside the obstructed region) are represented with red. As the object o_4 is completely inside the obstructed region of u, o_4 is not visible at all from u.

4.8.2 Index Structure

First we present the construction of the auxiliary Quadtree, and then the details of the modifications to the MIR-tree to construct a OIR-tree.

4.8.2.1 Auxiliary Quadtree

The purpose of this auxiliary structure is to quickly find which part of an object or a candidate is obstructed from a user. To achieve this goal, the obstructed region $OR(u)$ for each user $u \in U$ is first constructed. The space is then partitioned using a Quadtree. For each cell c of the Quadtree, a list of users, $OL(c)$, is maintained based on the visibility of c from those users. The idea is that the visibility of an object o w.r.t. any user can be estimated from the cells with which $o.l$ intersects. Moreover, if an object is completely inside cells that are obstructed from a user u, that object can be safely discarded from consideration for u.

A top-down approach is used to populate the list of users $OL(c)$ associated with each cell c of the Quadtree. Specifically, $OL(c)$ consists of the ID of the users, such that for each user $u \in OL(c)$, c is completely inside the obstructed region $OR(u)$. Starting from the root, such users are found and added in the list $OL(c)$ of the corresponding cell c. If a user u is included in $OL(c)$ of a cell c, it implies that all the descendent cells of c are also contained inside $OR(u)$. Therefore repeatedly storing u for the descendants of that cell c is not required. If an additional user u' is found for which a cell c is completely inside $OR(u')$, but u' is not included in any OL of the ancestor cells of c, only then is u' included in $OL(c)$. This is illustrated with the example shown in Figure 4.6 and Figure 4.7.

Example 11. *Let the space be partitioned using a Quadtree into* 10 *cells as in Figure 4.6. The list of users* OL(c) *for the cells with at least one entry are shown in the table of Figure 4.7. Here, the cell c_{10} is completely obstructed from the users u_1, u_2, u_3, where u_3 is stored in the user list of c_9, user u_2 is stored in the user list of its parent cell c_5, and so on.*

Separate from the Quadtree, the information of the obstructed region (the collection of polygons) for each user is also stored. Each leaf level cell of the Quadtree is associated with a pointer to the corresponding obstructed regions that intersect with the cell. These obstructed regions are later used to compute the exact visibility of the necessary objects and candidates.

Auxiliary Quadtree partitioning. The purpose of maintaining this structure is to efficiently find the visible segments of an object w.r.t. a user. As the visibility of an object o is calculated by taking the summation of the visibility of its small segments $\Delta o.l$ of length at most ϵ, this value is used to direct the partitioning process. Specifically, if a cell c intersects with or contains any object $o \in O$, then c is further partitioned until the diagonal length of the cell c is less than or equal to the threshold ϵ.

4.8.2.2 OIR-tree

The set of objects O needs to be indexed in a way that the visibility of an object w.r.t. any user can be estimated in an efficient way. We extend the MIR-tree (Section 3.3.3.1) to support the visibility and textual similarity computation of our problem, as close-by objects (objects in an MBR) are likely to have a similar visibility value from a user, and it is efficient to estimate the distance, and angle (discussed in details later) using the MBRs. We refer to this index as an OIR-tree. The OIR-tree is constructed in a similar manner to the original MIR-tree, where the MBRs of the line segments of the objects $o.l$ in O are used to construct the underlying R-tree. In addition, the following information is maintained:

- We maintain a reference to a cell of the auxiliary Quadtree with each node E of the OIR-tree, specifically, the cell c at the lowest level such that E is completely inside c. The idea is that if a node E is completely inside a cell c of the Quadtree, that means all of the objects stored in E are completely obstructed from the users in $OL(c)$ (and the users stored in the OL of the ancestors of c), so E cannot contain any top-k object of those users.

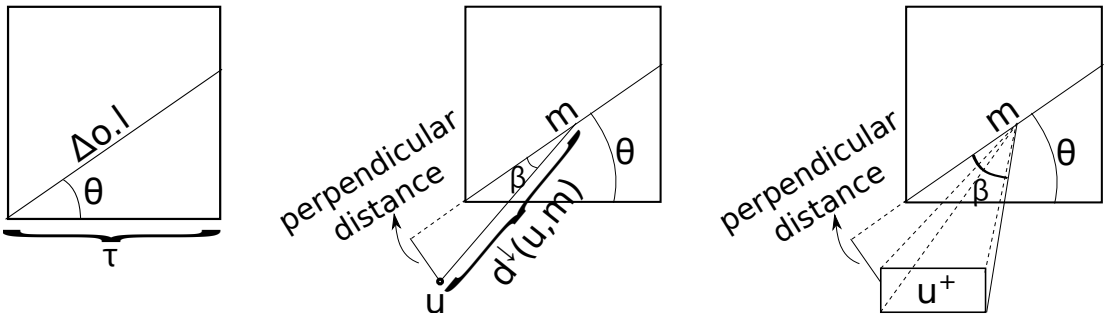
  Example 12. *In Figure 4.6, let the minimum bounding rectangle of the object o_3 be a node of the OIR-tree. This node intersects with the cells c_7 and c_8, but the cell c_2 is the lowest level cell for which this node is completely inside. Therefore, the reference to cell c_2 is associated with this node.*

- The maximum and the minimum of the lengths of the objects stored in the subtree rooted at node E, denoted as $len^{\uparrow}(E)$ and $len^{\downarrow}(E)$, respectively, are stored.

Angle lookup table. A lookup table is maintained with the angle of each object $o \in O$ w.r.t. the Cartesian X-axis, denoted as $o.\theta$. This angle is later used to derive the angle of an object w.r.t. any user in query time.

4.8.3 Visibility Bounds

Here we present an upper and a lower bound of visibility estimation of an object w.r.t. a user and the super-user using the node of the OIR-tree. Similar to the previous instance of MaxST with Euclidean distance as spatial similarity, these bounds are used to limit the number of top-k object computations of the necessary users, and facilitate the pruning of the candidates. We



(a) Calculating an upper bound length of $\Delta o.l$ inside a cell with the length of the larger side as τ.

(b) Calculating an upper bound of angle β between $o.l$ and u.

(c) Calculating an upper bound of angle β w.r.t. u^+.

Figure 4.8: Calculating upper bound using the angle θ between $o.l$ and the Cartesian x-axis.

also present bounds for the candidates to apply in the algorithms. The maximum (minimum) textual similarity $\mathrm{TS}^{\uparrow}(E.d, u.d)$ ($\mathrm{TS}^{\downarrow}(E.d, u.d)$) is computed from the maximum (minimum) TF-IDF value of the terms in $E.d \cap u.d$ in the same way as described in the previous sections.

4.8.3.1 Visibility Bounds of an OIR-tree Node of Objects

Here, we present how to compute the maximum (minimum) visibility value of any object in a node E of the OIR-tree w.r.t. a user u such that $\forall o \in O, \overline{\mathrm{SS}}^{\uparrow}(E.l, u.l) \geq \overline{\mathrm{SS}}(o, u)$ ($\overline{\mathrm{SS}}^{\downarrow}(E.l, u.l) \leq \overline{\mathrm{SS}}(o, u)$).

Upper bound visibility for a user. Note that, if a node E is completely inside the obstructed region of a user u, all the objects in E are completely obstructed from u as well. Such a node E cannot contain any top-k object of u. As the visibility of an object depends on its visible length, distance, and angle w.r.t. a user, a straightforward approach to compute the upper bound visibility of a node E for u is to use the maximum length of any object stored in E, $\mathrm{VL}^{\uparrow}(E.l, u.l) = len^{\uparrow}(E)$ as the visible length. The minimum Euclidean distance between u and E is used, and the angle is 90^o (the angle for which the perceived length of any object is maximum). Although this bound is easy to compute, it is a loose bound, as the actual maximum *visible length* of any object in E can be very different from $len^{\uparrow}(E)$ as a result of object obstruction. Therefore, techniques to compute a upper bound that estimates a tighter visible length is now presented.

From the reference of the Quadtree cell associated with E, first the leaf level cells of the Quadtree that intersect with E are found. Recall that a cell c is completely obstructed from the users in $OL(c)$ and the users in the lists OL of the ancestors of c. Let $\chi_{E,u}$ be the set of the leaf level cells of the Quadtree that are both visible from a user u and intersects with at least one object in E. Now an angle lookup table that keeps the angle $o.\theta$ of each object o with the Cartesian x-axis is used to compute tighter upper bound estimations as follows:

- First $o.\theta$ is used to better estimate the maximum length of a segment $\Delta o.l$ that is contained in a cell $x_i \in \chi_{E,u}$. The process is demonstrated using Figure 4.8a. Let the length of the larger side of cell x_i be τ. As the length of a line segment with a slope inside a rectangle is maximum when it goes through a corner of a larger side of x_i, the maximum length of $\Delta o.l$ can be calculated as:
$$\Delta o.l = \frac{\tau}{\cos o.\theta}\,.$$
- Now, a tighter upper bound of the angle between $o.l$ and u is estimated instead of using 90^o. Recall Section 4.8.1, the small segments $\Delta o.l$ of size ϵ are partitioned such that the orientations of all points on $\Delta o.l$ w.r.t. to a user u can be considered as visually similar. Therefore, a point over $\Delta o.l$ is chosen, m. The mid-point of the line segment might be chosen assuming that it goes through a corner of a larger side of x_i, but choosing any other point will have an insignificant visual difference. Then the angle $\beta = \angle(\Delta o.l, u.l)$ can be calculated as:
$$\sin\beta = \frac{\boldsymbol{d}^{\downarrow}{}_{\perp}(o.l, u.l)}{\boldsymbol{d}^{\downarrow}(o.l, u.l)}\,.$$
The calculation is shown with Figure 4.8b, where the perpendicular distance of the line $o.l$ from the user location is $\boldsymbol{d}^{\downarrow}{}_{\perp}(o.l, u.l)$.
- Finally, the minimum Euclidean distance between cell x_i and $u.l$ is used instead of the distance between E and u, the value of β as the upper bound of $\angle(\Delta o.l, u.l)$, and the value of $\Delta o.l$ is computed to get $\mathrm{VL}^{\uparrow}_{\Delta}(\Delta o.l, u.l)$ for an individual segment. The $\mathrm{VL}^{\uparrow}(E, u)$ is obtained by taking the summation of these values from the cells $\chi_{E,u}$. If this summation value is greater than the maximum length of an object in E, $len^{\uparrow}(E)$, then $len^{\uparrow}(E)$ is taken as the upper bound visible length.

Upper bound visibility for the super-user. As the algorithms to answer MaxST rely on the bounds of a node of objects for u^+ to filter the objects and candidates efficiently, the steps to compute the upper bound visibility of a node E w.r.t. u^+, $\overline{\text{SS}}^{\uparrow}(E.l, u^+.l)$ such that $\forall u \in u^+$, $\forall o \in E$, $\overline{\text{SS}}^{\uparrow}(E.l, u^+.l) \geq \overline{\text{SS}}(o.l, u.l)$ is now presented.

Let χ_{E,u^+} be the set of the leaf level cells of the Quadtree that intersect with at least one object $o \in E$ and visible from at least one user $u \in u^+$. The upper bound w.r.t. u^+ is computed in a similar manner using χ_{E,u^+}. To calculate the upper bound of the angle w.r.t. the super-user, the angle is computed using the same technique mentioned above for each of the four corner points of the MBR of u^+. Let the maximum of these angles be $\beta = \angle^{\uparrow}(o.l, u^+)$, where β is between 0 and 90 degrees. The angle of a line segment $\Delta o.l$ from any user location u inside the MBR of u^+ will be less than $\angle^{\uparrow}(o.l, u^+)$. This upper bound calculation of angle is illustrated in Figure 4.8c. The maximum visible length, the angle upper bound, and the minimum Euclidean distance are used to compute the final upper bound w.r.t. the super-user.

Lower bound visibility. Let $\hat{\chi}_{E,u}$ be the set of the leaf level cells that are *obstructed* from u and intersects with at least one object $o \in E$. The maximum length of a segment $\Delta o.l$ that is contained inside a cell $\hat{x_i} \in \hat{\chi}_{E,u}$ is computed in the same manner as the upper bound calculation. As these segments are obstructed, the sum of the maximum lengths of $\Delta o.l$ from $\hat{\chi}_{E,u}$ represent the maximum length of $o.l$ obstructed from u. Therefore, if this maximum obstructed length of $o.l$ is subtracted from $len^{\downarrow}(E)$ (the minimum length of any object in E), a lower bound of the visible length of $o.l$ is obtained. If this value becomes negative for any $o \in E$, the lower bound visibility of E, $\overline{\text{SS}}^{\downarrow}(E.l, u.l)$ is taken as 0. Otherwise, to get a lower bound on the angle, first the angle $\beta = \angle(\Delta o.l, u.l)$ is computed for the object o that intersects with a cell $x_i \in \chi_{E,u}$ as above. If multiple objects intersect with x_i then the minimum of their angles, $\angle^{\downarrow}(\Delta o.l, u.l)$, is used. The lower bound of the visible length, the angle $\angle^{\downarrow}(\Delta o.l, u.l)$, and the maximum Euclidean distances between E and u are used to compute lower bound visibility with the Equation. 4.2 and Equation 4.1.

Lower bound visibility for the super-user. This bound is calculated in the same way as for an individual user, except that the calculation is done using the set $\hat{\chi}_{E,u^+}$ of the leaf level cells that are *obstructed* from at least one user $u \in U$, and intersecting with at least one object $o \in E$.

4.8.3.2 Visibility Bounds of the Candidates

Similar to the unobstructed instance of the MaxST problem, the upper (lower) bound of a candidate is computed by assuming the maximum (minimum) text similarity that can be achieved from the candidate keyword set and the keyword constraints. Now we present the techniques to calculate the upper and the lower bound visibility of a candidate location $\ell \in L$ w.r.t. a user u and the super-user.

Upper bound visibility. Let the set of the leaf level cells that intersect with ℓ and visible from u be $\chi_{\ell,u}$. Note that the diagonal length of these cells can be larger than ϵ. Unlike the upper bound calculation for an object, the length of ℓ inside each cell $x_i \in \chi_{\ell,u}$ can be easily calculated using the techniques to find the intersection points of a line segment and a rectangle. For each x_i, the length of ℓ inside x_i, the minimum distance between x_i and u, and the actual angle between the portion of ℓ inside x_i and u is used to calculate an upper bound of the visible length of that portion of ℓ. Finally, the sum of the upper bounds of visible lengths is taken to get the upper bound of the visibility of ℓ w.r.t. u using Equation 4.1.

To calculate the upper bound visibility w.r.t. u^+, similar to the calculation for an object, the set χ_{ℓ,u^+} of the leaf level cells that intersect with ℓ and visible from at least one user $u \in U$ are obtained. Similarly, the upper bound of the angle is the maximum of the angles calculated from the four corner points of u^+. Then the exact length of ℓ inside each cell $x_i \in \chi_{\ell,u^+}$ is computed. These values and the minimum distance between ℓ and u^+ are used to get the upper bound visibility of a candidate w.r.t. the super-user.

Lower bound visibility. Let $\hat{\chi}_{\ell,u}$ be the set of the leaf level cells that are *obstructed* from u and intersect with the candidate ℓ. The length of ℓ that are contained inside each cell $\hat{x}_i \in \hat{\chi}_{\ell,u}$ are computed. These obstructed lengths are subtracted from the actual length of ℓ to find the visible length of ℓ w.r.t. u. Then the lower bound is calculated using the maximum Euclidean distance and the actual angle between ℓ and u. For the lower bound w.r.t. the super-user, the calculation is done similarly, but using the set of the leaf level cells that are *obstructed* from at least one user $u \in U$.

Computing the final visibility of a candidate. If a candidate line cannot be pruned using the bounds, the actual visibility w.r.t. some users may need to be computed. Recall from the construction of the auxiliary Quadtree that each leaf level cell is associated with a pointer to the corresponding obstructed regions (the collection of polygons) that intersect with the cell. If

Table 4.4: Description of datasets

Property	**Flickr**	**LA**	**Yelp**
Total objects	1,000,000	542,310	61,185
Total unique terms	166,317	52,731	266,869
Avg unique terms per object	6.9	8.5⋆	398.7
Total terms in dataset	6,936,385	274,577	77,838,026

⋆The avg unique terms of the objects (buildings) that are associated with a text description.

the diagonal length of a cell $x_i \in \chi_{\ell,u}$ is greater than ϵ, the obstructed regions of u are retrieved that intersect with x_i by following the pointer to find the segments of ℓ that are actually visible from u.

4.9 Experimental Evaluation

In this section, the experimental evaluation for our three proposed algorithms are presented, (i) Grp-topk, (ii) Indiv-U, and (iii) Index-U are used to process the MaxST query for the two instances of spatial relevance in the ranking of an object, (i) the Euclidean distance and (ii) the visibility. We also compare our approaches with a straightforward baseline, where the top-k objects of each user is computed individually, and after some basic filtering, all the possible combination of the candidates are checked against the users to find the best candidate combination (Section 4.4).

Datasets and setup. All experiments were conducted on three different datasets, (i) the Flickr dataset [1], (ii) the LA dataset [2], and (iii) the Yelp dataset [3].

The LA dataset contains the 2D footprint of 542, 310 buildings in Los Angeles. The text description of 31, 526 POIs are collected from Foursqure for this area, and each text description is assigned to the corresponding building of that POI.

The properties of the Flickr and the Yelp dataset are the same as described in Section 3.5 of the previous Chapter. For the experiments on visibility, the point locations of Flickr and Yelp dataset are converted to rectangles (representing building footprints), where the distributions of the size of the rectangles follow the same distribution of the LA dataset. Table 4.4 lists the

[1] http://webscope.sandbox.yahoo.com/catalog.php?datatype=i&did=67
[2] http://egis3.lacounty.gov/dataportal/2011/04/28/countywide-building-outlines
[3] http://www.yelp.com.au/dataset_challenge

Table 4.5: Parameters

Parameter	**Description**	**Range**
k	No. of objects to consider	$5, \mathbf{10}, 20, 50, 100$
α	Preference parameter	$0.1, 0.3, \mathbf{0.5}, 0.7, 0.9$
UL	No. of keywords per user	$1, 2, \mathbf{3}, 4, 5, 6$
UW	No. of total unique keywords of users	$5, 10, \mathbf{20}, 30, 40$
$Area$	MBR of users' as % of dataspace	$1, 2, \mathbf{4}, 8, 16$
$\|L\|$	No. of candidate locations	$1, 20, 50, \mathbf{100}, 300$
ω	No. of keywords to select	$1, 2, 3, 4, \mathbf{5}, 6, 7, 8$
$\|U\|$	No. of users	$\mathbf{100}, 500, 1K, 2K, 4K$

Table 4.6: The performance of the exact and approximate approach for varying k, corresponding to Figure 4.11b (Euclidean distance as spatial relevance).

Dataset \ k		1	5	10	20	50
Flickr	Runtime(E)	9K	15K	26K	33K	36K
	Runtime(A)	7.3	28.8	38.6	49.4	52.5
	Ratio(A)	0.66	0.78	0.87	0.92	0.98
	TR(E)	0.87	0.78	0.64	0.53	0.50
	TR(A)	0.83	0.89	0.93	0.96	0.99
LA	Runtime(E)	25K	26K	26K	26K	27K
	Runtime(A)	56.4	54.44	56.13	53.21	53.85
	Ratio(A)	0.93	0.96	0.96	0.97	0.97
	TR(E)	0.54	0.53	0.53	0.53	0.50
	TR(A)	0.96	0.98	0.98	0.99	0.99
Yelp	Runtime(E)	14K	19K	21K	25K	28K
	Runtime(A)	44.3	44.5	45	44.8	45.6
	Ratio(A)	0.99	0.99	0.99	0.99	0.99
	TR(E)	0.75	0.66	0.61	0.55	0.50
	TR(A)	1.00	1.00	1.00	1.00	1.00

properties of the datasets. The set of users U are generated in the same technique described to generate the set of queries in Section 3.5. In this chapter, we generated 50 such sets of users and reported the average performance. The experimental setup and the machine configuration are also the same.

4.9.1 Performance Evaluation

In this section, we evaluate and compare the performance of the approaches by varying several parameters. The performance evaluation of our proposed approaches consist of the following two components:

- Computing the top-k objects for the users, where (i) the baseline (BL) approach computes the top-k objects for all of the users individually; (ii) the GRP-TOPK approach groups the users and computes the top-k objects of all the users jointly (Grp); and (iii) the INDIV-U approach that uses the same joint processing techniques of the GRP-TOPK approach, but uses the candidates to avoid computing the top-k object for some users (Indiv).
- Finding the best location and keywords combination from the given set of candidates, where we compare the performances of the exact (E) and the approximate (A) methods.

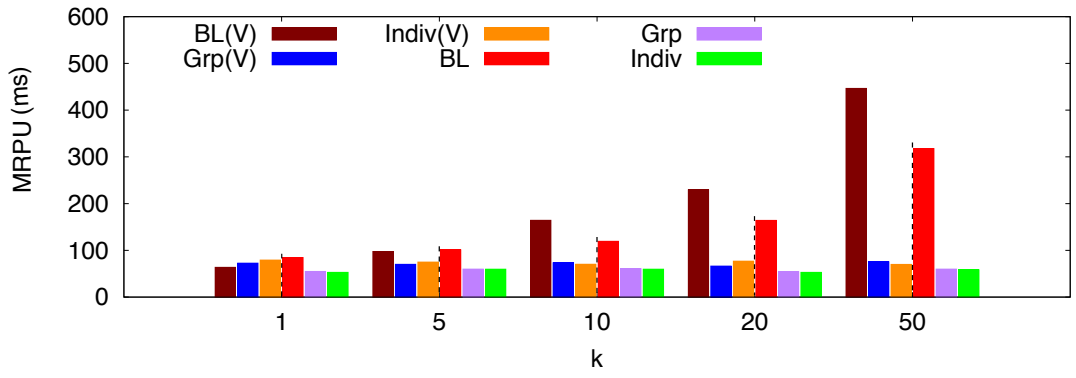
We choose the SF-GRP approach presented in Section 3.3.3 to compute the top-k objects of the users jointly for the GRP-TOPK approach and report the performance. As the top-k object computation and the candidate selection steps are interleaved in the INDEX-U approach, the performance of the INDEX-U is reported as the total cost when varying the number of users at the end of this section.

The parameter ranges are listed in Table 4.5 where the values in bold represent the default values. In all experiments, a single parameter is varied while keeping the rest as the default settings to study the impact on: (i) the mean runtime per user (MRPU) to compute the top-k objects; (ii) the mean I/O cost per user (MIOPU) to compute the top-k objects; (iii) the total runtime of selecting the best candidate; and (iv) the approximation ratio of the approximate and the exact methods. This value is the ratio between the number of RkNN users of the best candidate returned by the approximate method, over the number of RkNN users of the best candidate returned by the exact method. Scalability is evaluated by varying the total number of users, and by reporting (i) the total runtime, and (ii) the total I/O cost for computing the top-k objects, instead of the mean values. We also evaluate the approaches for both the Euclidean distance and the visibility (V) as the spatial relevance in the MaxST problem. The runtime of all experiments is reported in milliseconds (ms).

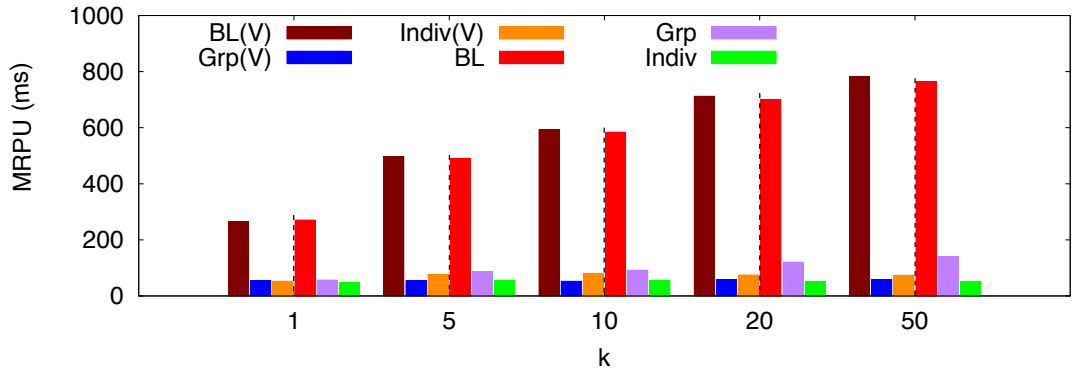
To better comprehend the trade-off between efficiency and the approximation quality, we present a trade-off score TR, where the runtime and the approximation ratio are combined by a linear weighted function. Specifically,

$$\mathrm{TR} = \gamma \times \mathrm{T} + (1 - \gamma) \times \mathrm{Ratio}\,,$$

where T is the runtime normalized between $[0, 1]$ and γ is the preference weight between T and Ratio. We use $\gamma = 0.5$ to present the trade-off scores in the rest of the chapter.



(a)

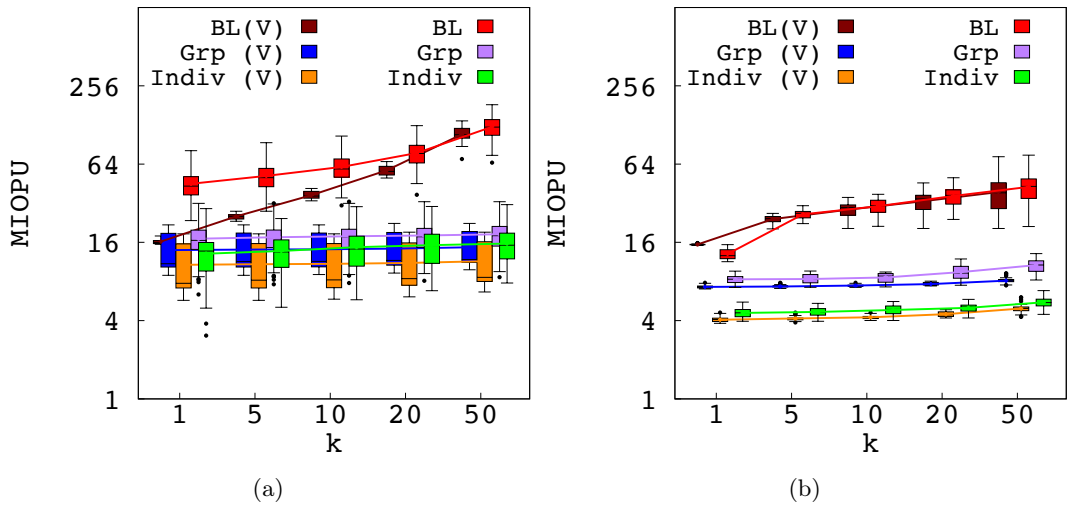


(b)

Figure 4.9: The effect on mean runtime per user to find the top-k objects for varying k in (a) Flickr and (b) LA dataset.

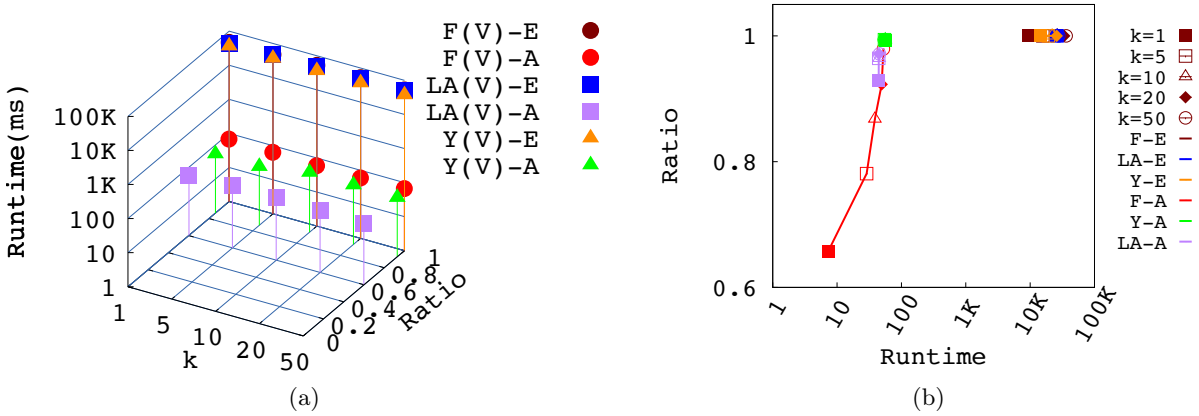
Varying k. Figure 4.9 and Figure 4.10 show the impact of varying k, and the mean costs of computing the top-k objects of the users for Flickr and LA datasets, respectively. Since the Grp and the Indiv approach use several pruning strategies, and avoid visiting any page multiple times, the costs are significantly lower than the baseline (BL). Both the mean I/O costs and the mean runtime per user are slightly less for the Indiv approach than the Grp approach, as the Indiv approach prunes some users as well, where the Grp approach computes the top-k objects for all users jointly.

The costs in the LA dataset are much higher than the costs in the Flickr dataset for the same settings as the half a million data points of the LA dataset are densely located in the dataspace, whereas the Flickr dataset contains 1 million data points scattered all over the



(a) (b)

Figure 4.10: The effect on mean I/O cost per user to find the top-k objects for varying k in (a) Flickr and (b) LA dataset.



(a) (b)

Figure 4.11: Candidate selection runtime and approximation ratio tradeoff for varying k where the spatial relevance is (a) visibility and (b) Euclidean distance.

world. Therefore, many objects in the LA dataset have very close similarity values for a user compared to the other dataset, and must be retrieved in the process. A lower cost incurs for spatial similarity based on visibility than that of Euclidean distance in LA dataset, as more objects are pruned based on their visibility than Euclidean distance based bounds.

Although the trend of the changes in the costs are similar for the Euclidean distance and the visibility, the MRPU for visibility is much higher than the Euclidean distance, as there are additional calculations required (finding the visible segments of a node/object and calculating

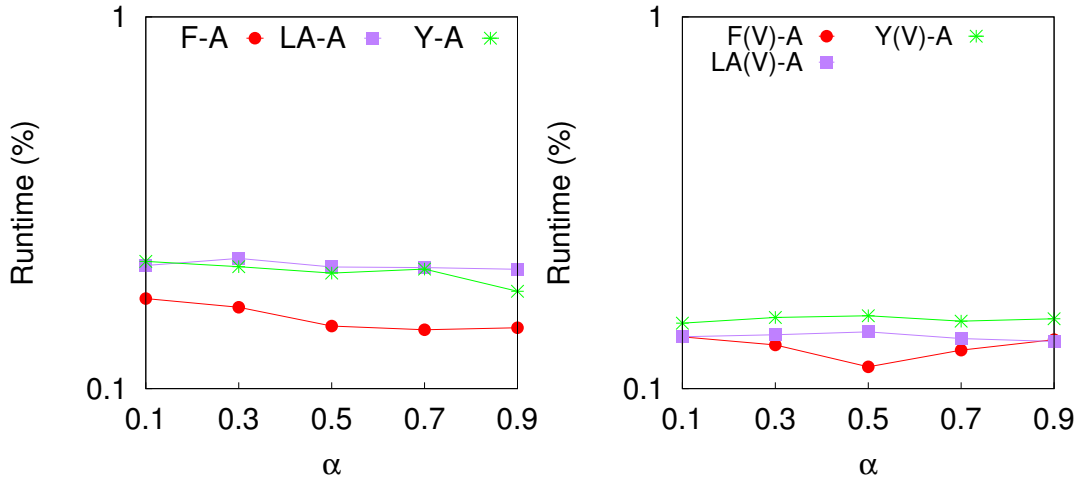


Figure 4.12: Percentage of runtime w.r.t. the exact method for varying α, where the spatial relevance is (a) Euclidean distance and (b) visibility.

angle bounds) for visibility.

Figure 4.11a and Figure 4.11b demonstrate the tradeoff between the runtime and the quality of the approximation to select the best candidate combinations for Euclidean distance and visibility, respectively, in all three datasets. For both cases, the runtime of the exact (E) method does not vary much for k as it uses only basic pruning techniques, and exhaustively computes all candidate combinations. The approximation ratio of the exact method is shown as 1 (the best ratio) to help compare with the approximate method. The runtime and the accuracy of the approximation increase with k, as more candidates are eligible to be included in the answer of the MaxST query. For all three datasets, the approximate method (A) selects the candidate combination of keywords greedily, and thus requires around 3 orders of magnitude less computational time than the exact method. We show the trade-off scores for both exact and approximate methods in Table 4.6 for $\gamma = 0.5$, i.e., equal weight on both values.

Varying α. A higher value of α indicates more preference towards spatial similarity. Similar to the result found in Chapter 3, as the MBR of the users' locations, and the union of the users' keywords remain the same, the cost of the top-k computation of our proposed approaches remains almost constant when α is varied. Figure 4.12 and Figure 4.13 show the runtime and the approximation quality when varying α. The runtimes are shown as a percentage of the exact method. The dotted line (at 100%) represent the runtime of the exact method.

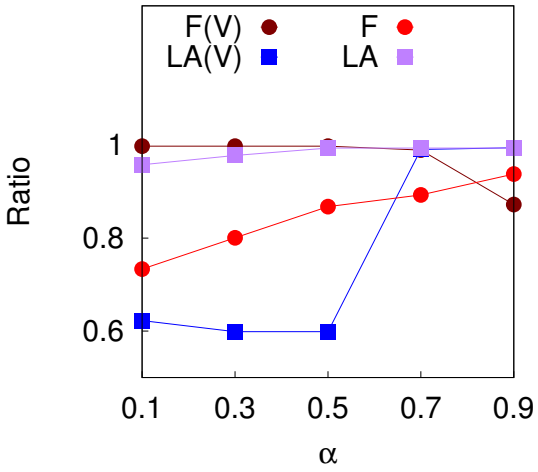
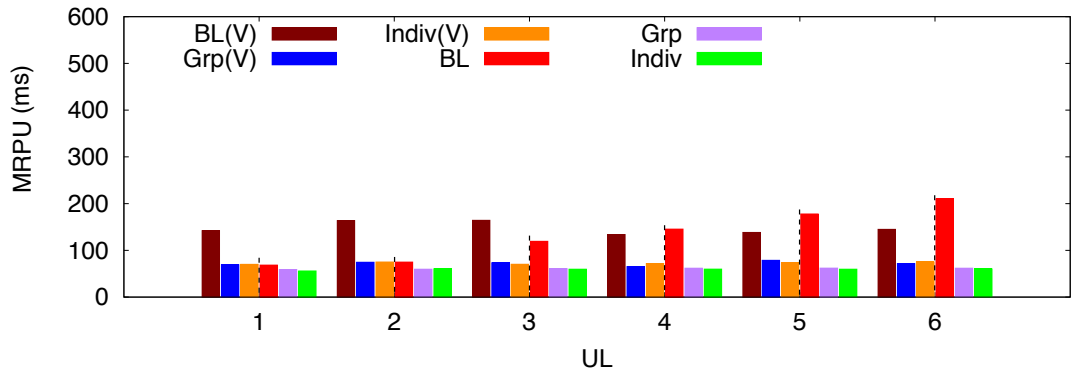


Figure 4.13: Approximation ratio for varying α.

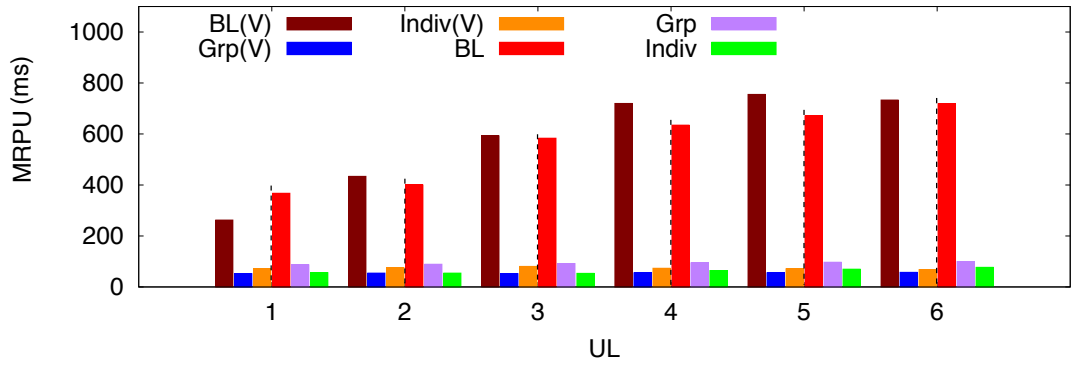
The approximation ratio in the LA dataset is comparatively low for lower values of α, but rapidly increases as α increases. The reason is that as the location density of the objects in LA dataset is very high, the visibility scores (spatial similarity) for most of the relevant objects are usually very low. Therefore, the total similarity score of an object is more sensitive to the textual score (thereby, the choice of the candidate text) in the LA dataset when compared to the other datasets. Therefore, the accuracy of the approximate method increases rapidly as α increases for the LA dataset when a higher weight is given to the spatial similarity. The approximation ratio for Yelp (not shown in Figure) does not vary much with α.

***Varying* UL.** We now vary the number of keywords per user, and present the effect on performance in Figure 4.14 and Figure 4.15 for the Flickr and LA datasets. The cost of BL increases proportionally with the increase of UL for the Euclidean distance metric, as more objects become relevant to each user. The increase in the costs of the baseline is not that prominent for the visibility, as those additional objects may not be visible to that user. The mean costs of our proposed Grp and Indiv approaches, where the users are grouped together as a super-user, do not vary much. The reason is that although UL increases, the total number of unique keywords in the group (UW) remains constant, so the number of objects retrieved remains unchanged as well.

Figure 4.16 shows the runtime and the approximation ratio to find the result candidate



(a)



(b)

Figure 4.14: The effect on mean runtime per user to find the top-k objects for varying UL in (a) Flickr and (b) LA dataset.

combination. Here, the number of users that are a reverse kNN of the result, increase with the increase of UL for both exact and approximate methods, and the approximation quality increases as well.

***Varying* UW.** Figure 4.17 and Figure 4.18 show the effect on performance when varying the total number of unique keywords for the group of the users in the Flickr dataset. Here, a lower value indicates that the queries share more keywords. As the Grp and the Indiv approaches exploit shared I/Os among users, it outperforms the baseline, and the benefit is greater as overlap increases.

Figure 4.19a and Figure 4.19b show the runtime of the exact and the approximate ap-

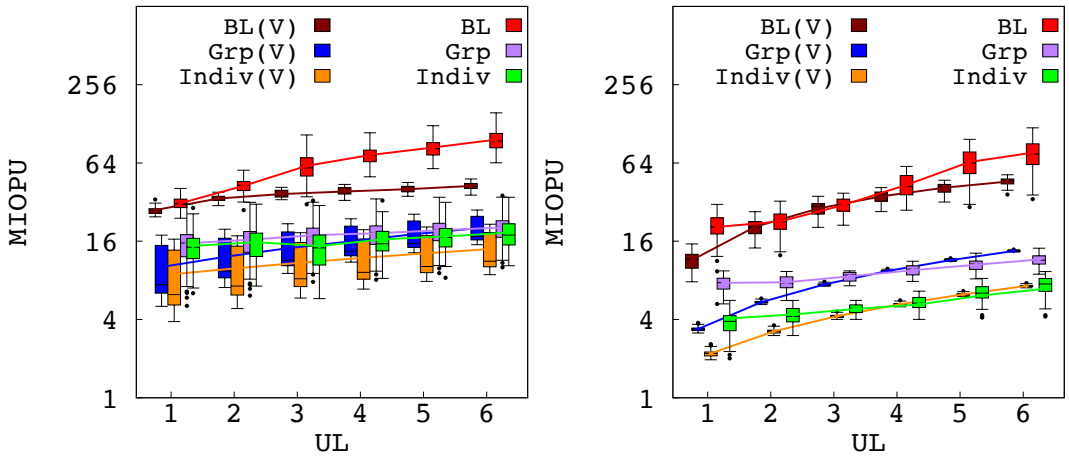


Figure 4.15: The effect on mean I/O cost per user to find the top-k objects for varying UL in (a) Flickr and (b) LA dataset.

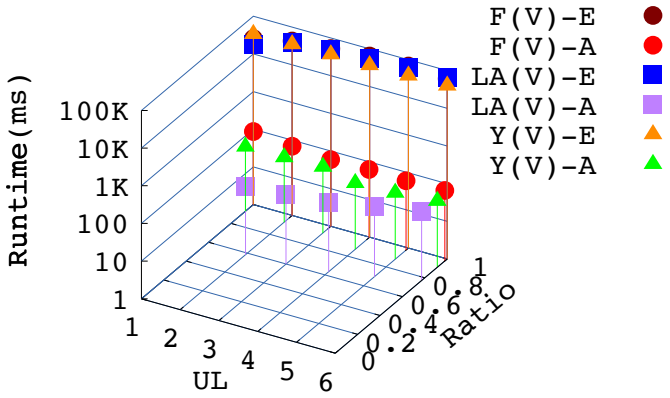


Figure 4.16: Candidate selection runtime and approximation ratio tradeoff for varying UL.

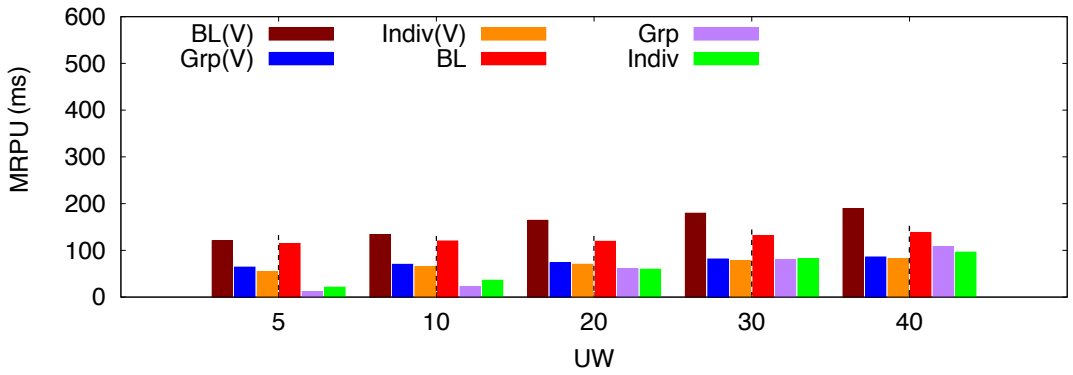


Figure 4.17: Effect on mean runtime per user for varying UW to find the top-k objects on Flickr dataset.

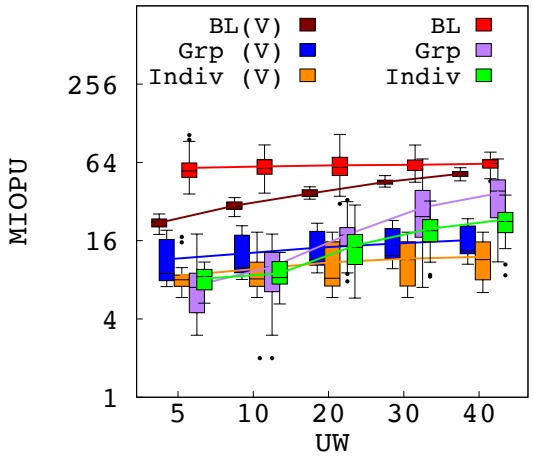
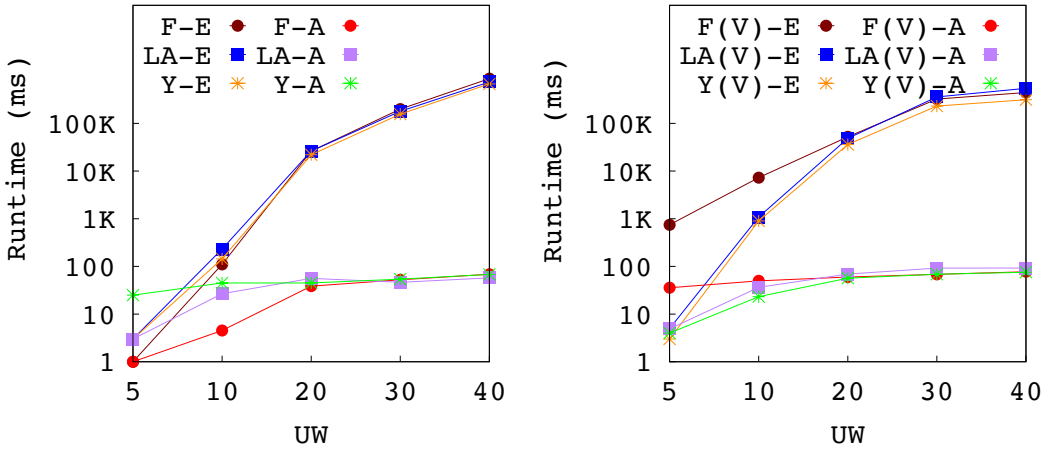


Figure 4.18: Effect on mean I/O cost per user for varying UW to find the top-k objects on Flickr dataset.

proaches for selecting the candidate in all three datasets for the Euclidean distance and the visibility as spatial relevance, respectively. As the set of keywords UW is also the set of candidate keywords, the runtime of candidate selection increases for both methods. As the exact method checks all possible combinations of keywords after applying some basic pruning, the runtime for the exact method increases rapidly when compared to the approximate method. The runtime for visibility metric is more than that of the Euclidean distance for the same settings, as each visibility calculation is more computationally costly than a single Euclidean distance calculation.



(a) (b)

Figure 4.19: Runtime to select the candidates for varying UW, where the spatial relevance is (a) Euclidean distance and (b) visibility.

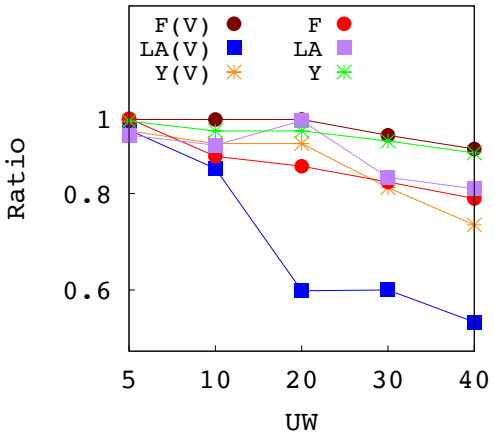


Figure 4.20: Approximation ratio for varying UW.

Figure 4.20 shows the effect on the approximation ratio when varying UW for all three datasets, and for both spatial relevance metrics. As UW increases, the possible combinations of the candidate keywords also increases. Therefore, the accuracy of the approximate method is very high for the lower values of UW (almost 1), and decreases gradually as UW increases.

Varying ω. The performance when varying the number of candidate keywords to select ω is shown in Figure 4.21. As the top-k object computation for the users does not depend on ω,

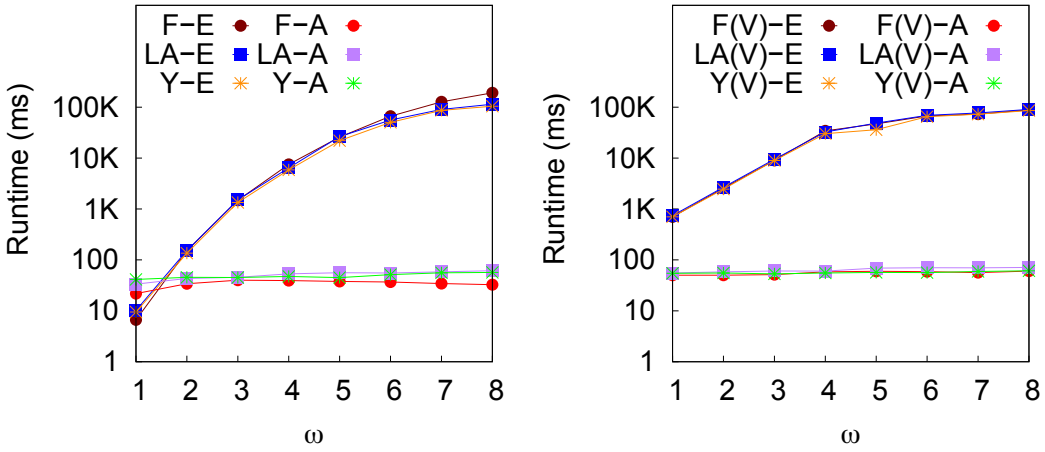


Figure 4.21: Runtime to select the candidates for varying ω, where the spatial relevance is (a) Euclidean distance and (b) visibility.

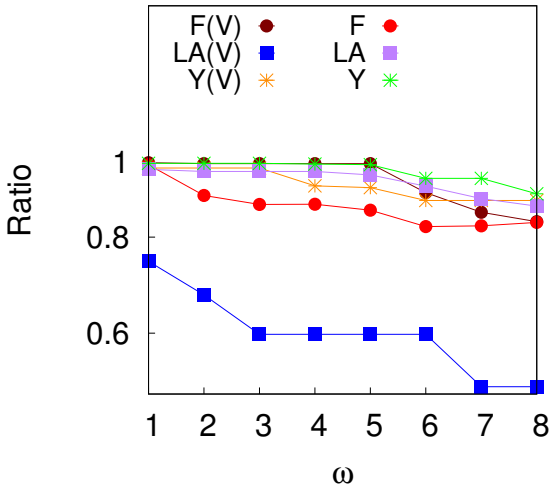


Figure 4.22: Approximation ratio for varying ω.

we only show the performance of the exact and the approximate methods. As the number of keyword combinations increases with ω, the runtime of both the exact and the approximate approach increase for both spatial similarity metrics (Figure 4.21a and Figure 4.21b). As the number of combinations to check in the exact method increases exponentially with the increase of ω, the benefit of the approximate approach is higher as ω increases. The number of users that are a reverse kNN of the result candidate combination increases as ω increases, and the

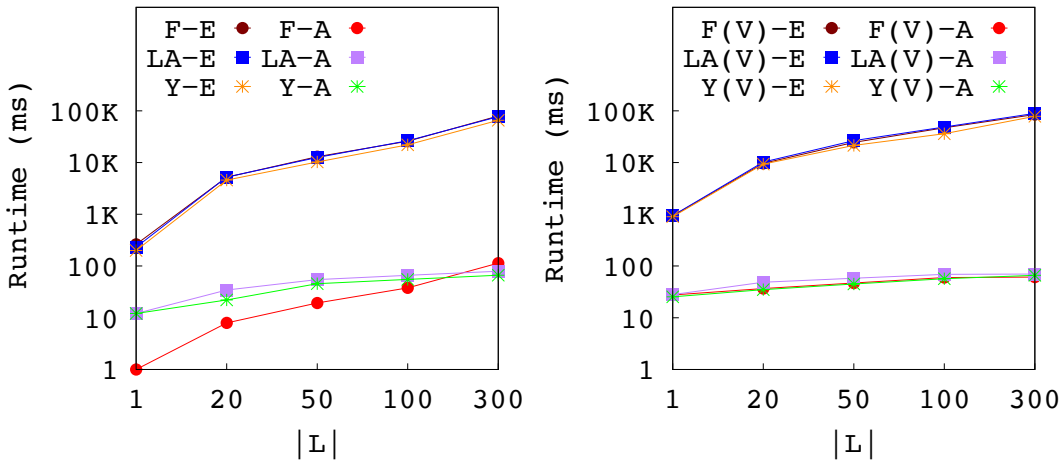


Figure 4.23: Runtime to select the candidates for varying $|L|$, where the spatial relevance is (a) Euclidean distance and (b) visibility.

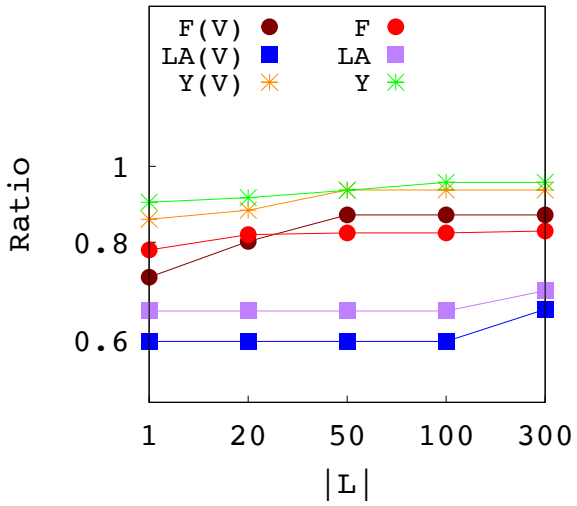
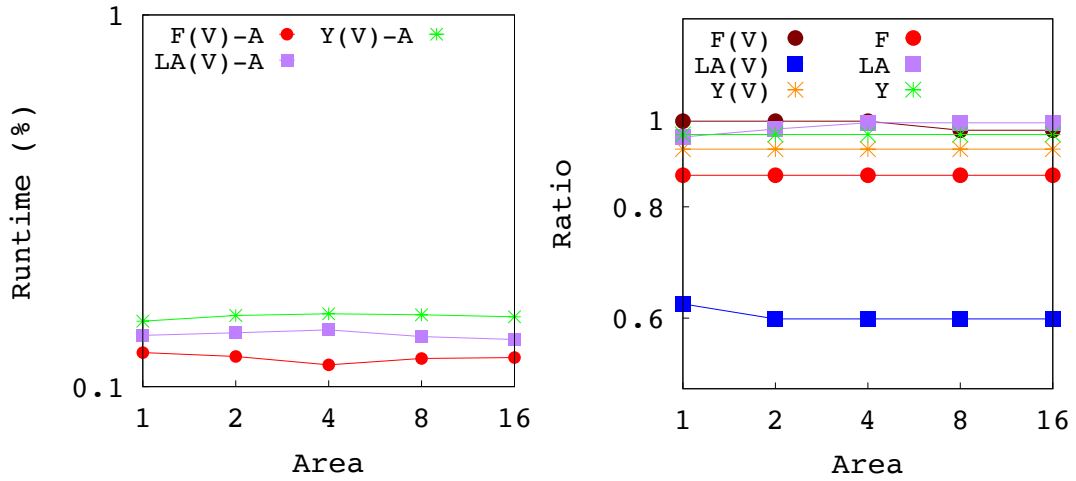


Figure 4.24: Approximation ratio for varying $|L|$.

accuracy of the approximate method gradually drops. Figure 4.22 demonstrates this scenario for all of the datasets.

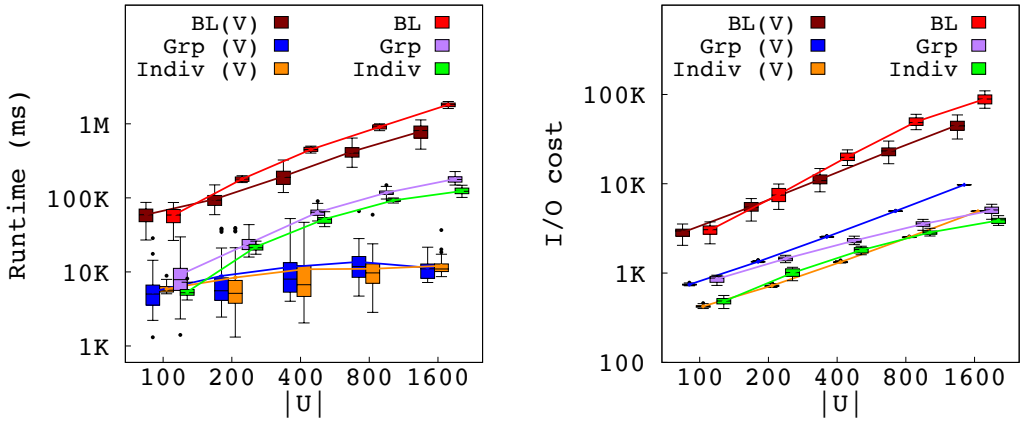
***Varying* $|L|$.** Figure 4.23a and Figure 4.23b show the runtime when varying the number of candidate locations $|L|$ for the two different spatial similarity metrics respectively. As the total number of possible candidate combinations increase with the increase of $|L|$, the runtime



(a) Percentage of runtime of the approximate method w.r.t. the exact method (visibility as spatial relevance)

(b) Approximation ratio

Figure 4.25: Effect of varying Area.

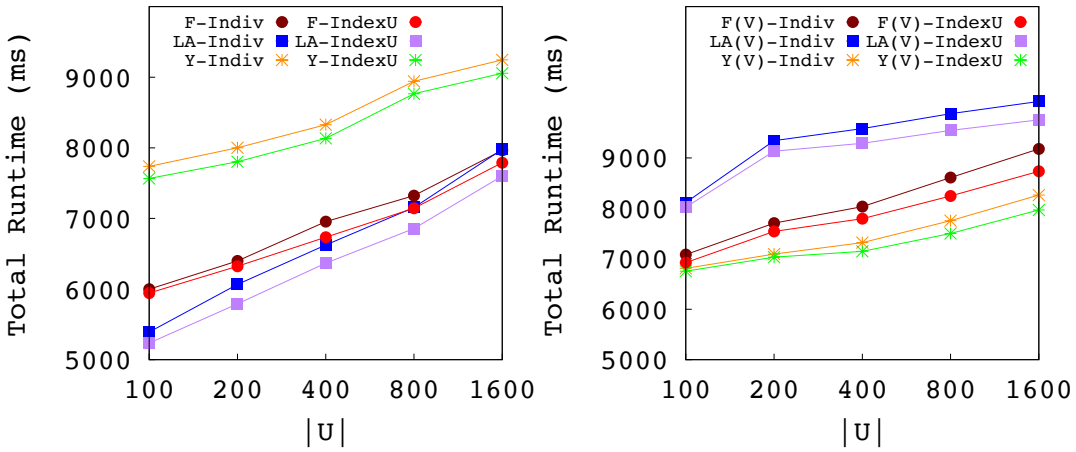


(a) Runtime for finding top-k

(b) I/O cost for finding top-k

Figure 4.26: Effect of varying $|U|$ for LA dataset.

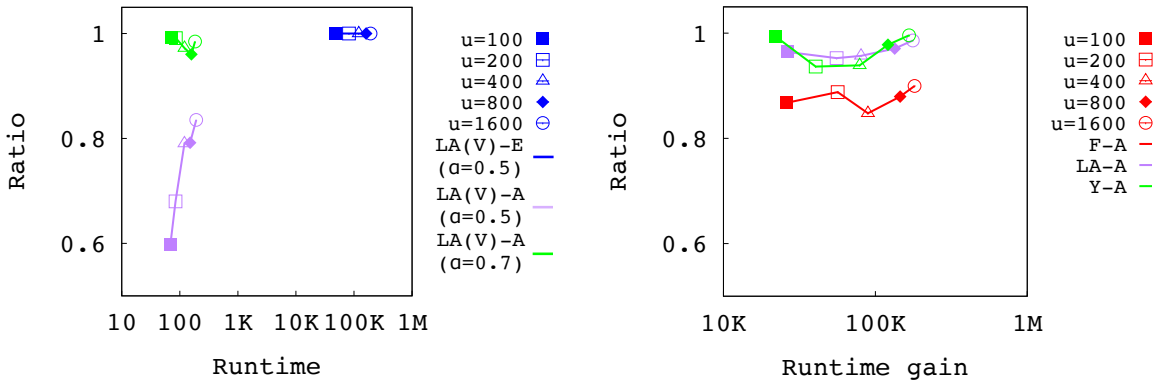
increases proportionally with $|L|$ for both methods. As the visibility needs to be calculated for each qualifying candidate locations, the runtime for visibility is higher than that of the Euclidean distance. The accuracy of the approximation increases slightly for a higher value of $|L|$, as more candidate locations become potential results.



(a) Runtime (Euclidean distance as spatial relevance)

(b) Runtime (visibility as spatial relevance)

Figure 4.27: Effect of user pruning by varying $|U|$.



(a) Approximate approach runtime vs ratio (visibility as spatial relevance)

(b) Approximate approach runtime vs ratio (Euclidean distance as spatial relevance)

Figure 4.28: Candidate selection runtime and approximation ratio tradeoff for varying $|U|$.

Varying Area. Similar to the findings in Chapter 3, the performance of the approaches do not vary much with the change in the area of the users' locations when computing the top-k objects. Figure 4.25a shows the runtime to find the candidates as a percentage of the exact method. Figure 4.25b shows the approximation ratio of the approaches. The exact and the approximate method differ in selecting the candidate keywords, therefore the performance do not vary much with the area of the users' locations.

Table 4.7: The performance of the exact and approximate approach for varying $|U|$ (visibility as spatial relevance)

Dataset	$\|U\|$	100	200	400	800	1600
Flickr	Runtime(E)	52K	82K	121K	180K	190K
	Runtime(A)	59.9	73.4	99.9	121.3	145.3
	Ratio(A)	0.99	0.98	0.98	0.95	0.97
	TR(E)	0.86	0.78	0.68	0.53	0.50
	TR(A)	1.00	0.99	0.99	0.97	0.99
LA	Runtime(E)	48K	81K	120K	161K	192K
	Runtime(A)	69.0	84.4	120.2	150.4	191.9
	Ratio(A)	0.60	0.68	0.79	0.79	0.84
	TR(E)	0.87	0.79	0.69	0.58	0.50
	TR(A)	0.80	0.84	0.90	0.90	0.92
Yelp	Runtime(E)	36K	67K	100K	164K	189K
	Runtime(A)	56.7	69.5	85.5	91.3	128.4
	Ratio(A)	0.93	0.94	0.97	0.97	0.96
	TR(E)	0.90	0.82	0.73	0.57	0.50
	TR(A)	0.96	0.97	0.98	0.98	0.98

Varying $|U|$. We evaluate the scalability of our proposed approaches by varying the number of users $|U|$. We show the runtime and the I/O costs of computing the top-k objects instead of the mean values for the LA dataset in Figure 4.26. Similar to our previous finding in Chapter 3, as the number of users increases, the cost of the BL increases proportionally as BL processes the users one by one. As the I/Os are shared among users, the advantage of our proposed approaches becomes more prominent when the number of users are higher.

Figure 4.27 shows the performance of the Indiv-U and the Index-U approaches as the total runtime of answering a MaxST query, including both the top-k object computation and the candidate selection time of the approximate approach. Both approaches apply pruning techniques to prune unnecessary users, where the Indiv-U approach applies the techniques over individual users, and the Index-U approach finds them using a MIUR-tree index of the users. In all cases, the Index-U approach performs better than Indiv-U, as a hierarchy of the users helps prune branches of the index (multiple users together), thus reducing overall time. The benefit of the Index-U approach slightly improves w.r.t. an increasing $|U|$.

Efficiency vs. approximation quality tradeoff. Figure 4.28a and Figure 4.28b show the efficiency versus the approximation quality tradeoff for varying the number of users for visibility and Euclidean distance, respectively. Figure 4.28a presents the tradeoff in the LA dataset, and Table 4.7 presents the performance of the approaches for visibility in all of the three datasets.

Clearly, the approximate approach is about $3-4$ orders of magnitude faster than the exact method, as the approximate approach selects the candidate combination of keywords greedily.

In Figure 4.28a, the performance of the approximation approach is shown for $\alpha = 0.5$ and $\alpha = 0.7$ for varying $|U|$ in LA dataset. The approximation ratio for a lower value of α is comparatively low for lower values of $|U|$, but the approximation ratio does not vary much for the other two datasets (Table 4.7). This can be explained similarly when the value of α is varied using Figure 4.13. As the LA dataset is highly dense, the visibility score for most of the relevant objects are usually very low, and thus the total similarity score of an object is more sensitive to the textual score for the default value of α (0.5) than the other datasets. As $|U|$ increases, the greedy selection of the candidate keywords can increase relevance with more users. The advantage of approximation in both efficiency and quality is more prominent for higher values $|U|$ and α. The trade-off scores for both methods are shown in Table 4.7 for $\gamma = 0.5$.

4.9.2 Summary

The experiments in this chapter has shown that all of our proposed methods significantly outperform the baseline in all three datasets. The Indiv-U approach always performs slightly better than the Grp-topk approach, as the Indiv-U approach employs some additional pruning over the set of users. The Index-U approach further improves the performance by applying the pruning techniques over a hierarchy of the users, thus reducing the runtime. As the approximate approach greedily selects the candidate keywords, the approximate approach is consistently 3 to 4 orders of magnitude faster than the exact method. The approximation ratio of the approximate approach is about $0.9-1$ in Flickr dataset for both visibility and Euclidean distance. As the objects in the LA dataset is densely located, the approximation ratio becomes better for a higher preference to the textual similarity score.

4.10 Conclusion

In this chapter, we presented solutions to efficiently answer a *Maximized Bichromatic Reverse Spatial Textual k Nearest Neighbor* (MaxST) query, which finds an optimal location and a set of keywords for an object so that the object has the maximum number of RkNNs. We proposed three different solutions to answer the MaxST query for the spatial-textual data, and we presented the necessary calculations for the approaches using two different spatial similarity

metrics, (i) the Euclidean distance and (ii) the visibility. In all of our proposed methods, we improved the overall efficiency by pruning the candidates, sharing the processing and I/O costs of the multiple users, and avoiding multiple retrievals of the same object, even for visibility, where the visibility calculation of an object requires the location of the other objects as well. We also proposed an approximate solution for the keyword selection component of the problem.

Through extensive experiments using three publicly available datasets, we compared the performance of our proposed approaches in different settings. In summary, we find that the approximate algorithm, which greedily selects the keywords, is around $2-3$ orders of magnitude faster than an exact method. The GRP-TOPK approach that groups the users and finds the top-k objects of the users jointly, performs about $2-3$ times better than the baseline, and the benefit increases when using a visibility metric. As the INDIV-U approach prunes users as well, the performance of the INDIV-U approach is close to or better than the GRP-TOPK approach in all settings tested. The INDEX-U performs slightly better than the INDIV-U approach, at the cost of maintaining an additional index for the users, but the benefits increase as the number of users increases.

Chapter 5

Top-m Rank Aggregation of Spatial Objects in Streaming Queries

Rank aggregation is the problem of combining multiple rank orderings to produce a single ordering of the objects. Thus, aggregating the ranks of spatial objects can provide key insights into the importance of the objects in many different scenarios, where the rank of an object with respect to an individual query is computed based on its distance from the query location. This translates into a natural extension of the problem that finds the top-m objects with the highest aggregate rank over multiple queries.

As new queries are continuously being issued, clearly the rank aggregations change over time, and recency can also play an important role when interpreting the final results. In this chapter, we consider the problem of top-m rank aggregation of spatial objects in streaming queries, where, given a set of objects and a stream of proximity based spatial queries, the problem is to report the updated top-m objects with the highest aggregate rank over a subset of the most recent queries from the stream.

5.1 Introduction

Rank aggregation is a classic problem in the database community which has seen several important advances over the years [2, 34, 35, 36, 86]. Informally, *rank aggregation* is the problem of combining multiple rank orderings to produce a single "best" ordering. Typically, this translates into finding the top-m objects with the highest aggregate rank, where the algorithms used for ranking and aggregation can take several different forms. Common ranking and aggrega-

tion metrics include majority ranking (sum, average, median, and quantile), consensus-based ranking (Borda count), and pairwise disagreement based ranking (Kemeny optimal aggregation) [34, 35]. Rank aggregation has a wide variety of practical applications such as determining election winners, sports analytics, collaborative filtering, meta-search, and aggregation in database middleware.

One such application area where rank aggregation can be applied is in spatial computing [111]. In spatial databases for example, a fundamental problem is to rank objects based on their proximity from a query location, where two pervasively used spatial query types are k-nearest neighbor (kNN) query and range query (Chapter 1). The results of a range query are often returned as a list ranked by their distance from the query location [115]).

A proximity based spatial query is an important tool that provides partially ranked lists over a set of objects. Each object o receives a different ranking (or is not ranked at all) which depends on the query location. Thus, aggregating the ranks of spatial objects can provide key insights into the importance of the objects in many different scenarios. A recent example that inspires this work is a real estate data visual analytics system called Houseseeker[1] [75]. Houseseeker facilitates finding houses based on spatial preference. The ranking of each house is based on the distance from a preferred location (close to a school, railway station, supermarket, etc.). A house that is ranked high by many users' queries has a higher aggregate rank, and is therefore more popular in the market. Clearly, popularity in this context changes continuously over time as new buyers search for houses. Continuously monitoring the housing properties with the highest aggregate rank, regardless of how rank is defined is of practical importance to both buyers and sellers: (i) the buyers can continuously be informed of the trending houses on the market, and make informed buying decisions; and (ii) the "hot" houses on the market can be used by a system like Houseseeker to find similar houses which are not currently for sale to provide insights and notifications to potential sellers on the best time to enter the market.

In this chapter, we consider the problem of top-m rank aggregation of spatial objects for streaming queries, where, given a set of objects O, a stream of spatial queries (kNN or range), the problem is to report the m objects from O with the highest aggregate rank. Here, an object that satisfies the query constraint is ranked based on its distance from the query location, and the aggregation is computed using all of the individual rank orderings. To maintain recency information and minimize memory costs, a sliding window model is imposed on the query stream, and a query is valid only while it remains in the window. Two of the most common models

[1] http://115.146.89.158

for sliding windows, the *count-based* window and the *time-based* window [93] are considered for this chapter.

Our work draws inspiration from three different domains – spatial databases, continuous queries, and rank aggregation. While several seminal papers have considered various combinations of these three domains, no previous work has considered approaches to combining all three. We summarize previous work, and present the differences between previous best solutions in these problem domains and our work in Section 5.2.

In the domain of rank aggregation, previous solutions have addressed the problem of incrementally computing individually ranked lists using *on-demand* algorithms [36, 86]. However, these approaches do not consider streaming queries, and there is no obvious way to easily extend these approaches to the sliding window problem, where queries are also removed from a sliding window. A related body of work is to find the most frequent objects over a stream, which is essentially a count aggregation [29, 99]. However, our goal is to efficiently compute the aggregation without explicitly requiring the ranked results to be fully computed for every query separately, so these approaches are not directly amenable to our problem.

In the domain of continuous spatial queries, objects are streaming, but the queries do not change [7, 68, 93]. Continuous result updates of top-k queries where the query location is moving have also been extensively studied in the literature [13, 72]. These approaches make the assumption that a query location can move only to an adjacent location, and construct a *safe region* around the query, such that the top-k results do not change as long as the query location remains in the safe region. These problems are subtly different from the streaming query problem explored in this chapter, where each new query location can be anywhere in space and the query does not move.

In the domain of spatial databases, the maximized reverse top-k [73, 132, 138] and the maximized bichromatic reverse k nearest neighbors (RkNN) explored in Chapter 4 is a related body of work. Although the count of RkNN is also an aggregation, these solutions do not consider the rank position of the objects for the aggregation. Rather, the approaches rely on properties of skyline or k-skyband queries, or region overlap to estimate the number of RkNN for an object 4.2, where determining the ranked position of an object is not straightforward. Moreover, to the best of our knowledge, there is no previous work on the continuous case of finding the object with the maximum number of RkNN for streaming queries.

To summarize, aggregating spatial object rankings can provide key insights into the importance of objects in many different problem domains. Our proposed solutions are generic and applicable to many different spatial rank aggregation problems, and a variety of different

query types such as range queries, k-nearest neighbor (kNN) queries, and reverse kNN (RkNN) queries can be adapted and used within our framework.

The contributions of this chapter are:

- We propose and formalize the problem of top-m rank aggregation on a sliding window of spatial queries, which draws inspiration from the three classical problem domains – rank aggregation, continuous queries, and spatial databases (Section 5.3).
- An exact solution is proposed to continuously monitor the top-m ranked objects (Section 5.3).
- We propose an approximation algorithm with bounded error guarantees to maximize the reuse of the computations from previous queries in the current window, and show how to incrementally update the top-m results only when necessary (Section 5.5). In particular, the following three technical contributions have been made: (i) the notion of *safe ranking* is introduced to determine whether the result set in a previous window is still valid or not in the current window; (ii) the computation of the *validation objects* is presented, which are able to limit the number of objects to be updated in the result set; and (iii) a new index, the *Inverted Rank File* (IRF), is presented and used to bound the error of the solution.

The rest of the chapter is organized as follows. Section 5.2 reviews previous related work. Section 5.3 presents the problem definition and an exact solution. Section 5.4 shows how to compute the lower and upper bounds for rank aggregation using an inverted rank file, which provides a foundation for an approximate solution to the rank aggregation problem introduced in Section 5.5. In Section 5.6, we validate our approach experimentally. Finally, we conclude and discuss future work in Section 5.7.

5.2 Related Work

Since this chapter draws inspiration from three different problem domains in database area – rank aggregation, spatial queries and streaming queries, we review the related work for each domain and combinations of two domains (if any).

5.2.1 Rank Aggregation

Given a set of ranked lists, where objects are ranked in multiple lists, the problem is to find the top-m objects with the highest aggregate rank. This is a well studied and classic problem, mostly for its importance in determining winners based on the ranks from different voters [3, 34, 35, 36, 49, 86, 87].

The approaches of Fagin et al. [35] and Dwork et al. [34] assume that the ranked lists exist before aggregation, and explore exact and approximate solutions for *Kendall optimal aggregation*, and the related problem of Kemeny optimal aggregation, which is known to be NP-Hard for 4 or more lists. When the complete ranked lists are not available a priori, or random access in a ranked lists is expensive, Mamoulis et al. [86] and a variant presented by Fagin et al. [36] have shown that the ranked objects for an individually ranked list can be computed incrementally one-by-one using an on-demand aggregation. However, such incremental approaches [36, 86] are not straightforward to extend to sliding windows where queries are also removed from the result set as new queries arrive.

5.2.2 Top-k Aggregation over Streaming Data

A related body of work on aggregation is to find the most frequent items, or finding the majority item over a stream of data [29, 80]. This problem is essentially an aggregation of the count of the data, which is studied for sliding window models as well [71, 99]. The solutions can be categorized mainly as sampling-based, counting-based, and hashing-based approaches. The goal of the approaches is to identify the high frequency items and maintain their frequency count as accurately as possible in limited space. As the result of the queries are not readily available for the count aggregation in our problem, these approaches cannot be directly applied.

5.2.3 Database Queries

The relevant work can be categorized mainly as - (i) moving, (ii) streaming, and (iii) maximum top-k queries.

Moving queries. In spatial databases, given a moving query and a set of static objects, the problem is to report the query result continuously as the query location moves [13, 15, 54, 72, 96]. The most common assumption made to improve efficiency is that a query location can move only to a neighboring region [13, 15]. By maintaining a *safe region* around the query location, a result set is updated only when the query moves out of the safe region. Thus, both the

computation and communication cost to report updated results are reduced. Li et al. [72] substitute the safe region with a set of *safe guarding objects* around the query location such that as long as the current result objects are closer to query than any safe guarding objects, the current result remains valid.

The problem of continuously updating kNN results when both the query location and the object locations can move was initially explored by Mouratidis et al. [92]. Mouratidis et al. solve the problem by using a conceptual partitioning of the space around each query, where the partitions are processed iteratively to update results when the query or any of the objects moves. In contrast, streaming queries studied in this paper can originate anywhere in space and not move, thus safe region based approaches are not applicable.

Query processing over streaming objects. Many different streaming problems have been explored over the years, among which the problem of continuous maintenance of query results [7] is most closely related to our problem. Bohm et al. [7] explore an expiration time based recency approach where objects are only valid in a fixed time window. Other related work explored sliding window models where objects are valid only when they are contained in the sliding window [68, 91, 93, 101]. The two most common variants of sliding windows are - (i) count based windows which contain the $|\mathbb{W}|$ most recent data objects; and (ii) time-based windows which contain the objects whose time-stamps are within $|\mathbb{W}|$ most recent time units. Note that the number of objects that can appear within a time-based window can vary, when the number of objects in a count based window are fixed.

The general approach in all of these solutions is as follows – Queries are registered to an object stream, and as a new object arrives, the object is reported to the queries if it qualifies as a result for that query. The solutions rely on the idea of a skyline, where the set of objects that are not dominated by any other object in any dimension must be considered. In these models, the queries are static, and the skyline is computed for a query. Newly arriving objects can be pruned based on the properties of the skyline. The key difference between our problem and related streaming problems is that objects are static in our model while the queries are streaming.

Maximizing reverse top-k. Another related body of work is reverse top-k querying [23, 67, 73, 121]. Given a set of objects and a set of users, the problem is to find the object that is a top-k object of the maximum number of users. Li et al. [73] explore solutions for spatial databases using precomputed Voronoi diagrams. Other solutions use a skyline and a k-skyband

to estimate the number of users that have an object as a top-k result. A k-skyband contains the objects that are dominated by at most $k-1$ objects. Unlike top-k queries, the number of result objects of a range query is not fixed, therefore maintaining a skyband is not straightforward for range queries.

A related problem in spatial databases is to find a region in space such that if an object is placed in that region, the object will have the maximum number of reverse kNN users (Section 4.2.1). Solutions for this problem depend on static queries (users), and are therefore not directly applicable to our problem. Moreover, these solutions do not consider the rank position of the object in the top-k results in their solutions.

Gkorgkas et al. [46] study the problem of maximizing RkNN in temporal data, which returns the object with the highest continuity score (the maximum number of consequent intervals for which an object o is a top-m object based on its RkNN count). Although the problem is scoped temporally, both the queries and the objects in the database are static.

5.3 Preliminaries

In this section, we formulate the problem of top-m rank aggregation on a sliding window of spatial queries, present the ranking measure, and then present a straightforward baseline solution to answer the problem.

5.3.1 Problem Formulation

Table 5.1 presents the basic notation used in the remainder of the paper. Let O be a set of objects where $o \in O$ is a point in dataspace D. Now consider a stream of user queries that is an infinite sequence $\langle q_1, q_2, \dots \rangle$ in order of their arrival time. Each query q is a point in D, associated with a spatial constraint, $\mathrm{Con}(q)$, such as range or kNN. In this chapter, we focus primarily on range queries, but our solutions are easily generalized to other spatial query types.

We adopt the sliding window model where a query is only valid while it belongs to the current sliding window $\mathbb{W}$. We present our work for a count-based sliding window, and then discuss the extensions required for a time-based window in Section 5.5.6. A count-based window contains the $|\mathbb{W}|$ most recent items, ordered by arrival time. Before defining our problem, we first present a rank aggregation measure of an object for a window of $|\mathbb{W}|$ queries, denoted as *popularity*.

Table 5.1: Basic notation in Chapter 5

Symbol	**Description**
$\mathbb{W}$	Sliding window of $\lvert\mathbb{W}\rvert$ most recent queries.
$\mathrm{Con}(q)$	Spatial constraint (range or kNN) of query q.
$\boldsymbol{r}(o,q)$	Ranked position of o based on $\boldsymbol{d}^{\downarrow}(o,q)$.
O_q^+	The set of objects in O that satisfy $\mathrm{Con}(q)$.
$\rho(o,\mathbb{W})$	Popularity (aggregated rank) of o for queries in $\mathbb{W}$.
qo	The least recent query, which is excluded from $\mathbb{W}$.
qn	The most recent query, which is added to $\mathbb{W}$.
c	A leaf level cell of a quadtree.
$\boldsymbol{d}^{\downarrow}(o,c_q)$ ($\boldsymbol{d}^{\uparrow}(o,c_q)$)	The minimum (maximum) Euclidean distance between o and any query in cell c_q.
$\boldsymbol{r}^{\downarrow}(o,c_q)$ ($\boldsymbol{r}^{\uparrow}(o,c_q)$)	Lower (upper) bound rank of o for any query q in cell c_q.
B	Block size of the rank lists.
$\boldsymbol{d}^{\downarrow}(b,c_q)$ ($\boldsymbol{d}^{\uparrow}(b,c_q)$)	The minimum (maximum) distance between any object in a block b and any query in cell c_q.

Popularity Measure. Each query q partitions O into two disjoint sets: $O_q^+ = \{o \in O \mid o \text{ satisfies } \mathrm{Con}(q)\}$ and $O \backslash O_q^+ = \{o \in O \mid o \text{ does not satisfy } \mathrm{Con}(q)\}$. Each object $o^+ \in O_q^+$ is ranked based on the minimum Euclidean distance from q, $\boldsymbol{d}^{\downarrow}(o,q)$. Other distance measures can also be used to rank the objects. The rank of o w.r.t. q, $\boldsymbol{r}(o,q) = i$, is defined as the i·th position of $o \in O_q^+$ in an ordered list indexed from $i = 1$ to $|O_q^+|$ where $\boldsymbol{d}^{\downarrow}(o_i^+, q) \leq \boldsymbol{d}^{\downarrow}(o_{i+1}^+, q)$.

The popularity of an object $o \in O$ in a sliding window $\mathbb{W}$ of queries is an aggregation of the ranks of o with respect to the queries in $\mathbb{W}$. We now formally define *Popularity* ($\boldsymbol{\rho}$) as a rank aggregation function for a sliding window of $|\mathbb{W}|$ queries. Other similar aggregation functions are applicable to our problem as well.

$$\boldsymbol{\rho}(o,\mathbb{W}) = \frac{1}{|\mathbb{W}|}\sum_{i=1}^{|\mathbb{W}|}\begin{cases} |O| - \boldsymbol{r}(o,q_i) + 1 & \text{where } o \in O_{q_i}^+, \\ 0 & \text{otherwise.} \end{cases}$$

A higher value of $\boldsymbol{\rho}(o,\mathbb{W})$ indicates higher popularity. If an object does not satisfy the constraint of a query, the contribution in the aggregation for that query is zero. We now formally define our problem as follows:

Definition 4. ***Top-m popularity in a sliding window of spatial queries (T$m\rho$Q) problem.*** *Given a set of objects O, the number of objects to monitor m, and a stream of spatial queries $\langle q_1, q_2, \ldots \rangle$, a top-$m$ popularity in a sliding window of spatial queries (T$m\rho$Q) problem*

is to maintain an aggregate result set R, such that $R \subseteq O$, $|R| = m$, $\forall o \in R$, $o' \in O \backslash R$, $\boldsymbol{\rho}(o, \mathbb{W}) \geq \boldsymbol{\rho}(o', \mathbb{W})$, where $\mathbb{W}$ contains the $|\mathbb{W}|$ most recent queries.

5.3.2 Baseline

A straightforward approach to continuously monitor the top-m popular objects in $\mathbb{W}$ consists of the following steps:

- Each time a new query, qn arrives, compute the rank of the objects in O^+_{qn} that satisfy $\text{Con}(qn)$.
- Update the popularity ρ of each object $o \in O^+_{qn}$ for qn, and each object $o' \in O^+_{qo}$ for the query qo. Here, qo is the least recent query that is removed from $\mathbb{W}$ as qn arrives.
- Sort all of the objects that are contained in O^+_q for at least one query q in the current window, and return the top-m objects with the highest $\boldsymbol{\rho}$ as R. As there is no prior work on aggregating spatial query results in a sliding window (See Section 5.2), we consider this straightforward solution as a baseline approach.

Unfortunately, the baseline approach is computationally expensive for several reasons:

- For each query q in the window, the ranks of all objects that satisfy the $\text{Con}(q)$ must be computed. As the number of objects in each range can be very large, and the queries can arrive at a high rate, this step incurs a high computational overhead.
- Each time the sliding window shifts, $\boldsymbol{\rho}$ for a large number of objects may need to be updated.
- The union of all of the objects that satisfy $\text{Con}(q)$ for each query in the current window must be sorted by the updated $\boldsymbol{\rho}$.

To overcome these limitations, we seek techniques which avoid processing objects for the query stream that cannot affect the top-m objects in R. This minimizes the number of popularity computations that must occur. Two possible approaches to accomplish this, are: accurately estimate the rank of the objects for newly arriving queries, or reuse the computations from prior windows efficiently. We consider both of these approaches in the following sections.

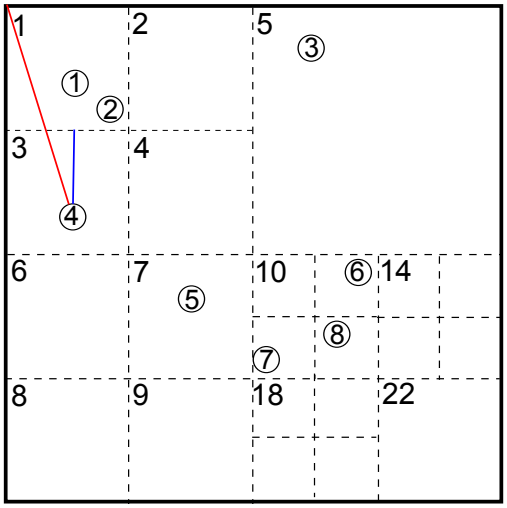


Figure 5.1: An example of rank bound computation of object o_4 for any query in cell c_1. The maximum and the minimum distance of o_4 and any point in c_1 is shown with a red and a blue line, respectively.

	$\text{minDist}_{1,1}$:0	$\text{minDist}_{1,2}$:6	$\text{minDist}_{1,3}$:16	$\text{minDist}_{1,4}$:22
c_1 →	(o_1,1), (o_2,1)	(o_4,2), (o_5,4)	(o_3,4), (o_7,5)	(o_6,6), (o_8,6)
	$\text{minDist}_{2,1}$:1	$\text{minDist}_{2,2}$:6	$\text{minDist}_{2,3}$:10	$\text{minDist}_{2,4}$:18
c_2 →	(o_1,1), (o_2,1)	(o_3,3), (o_4,3)	(o_5,3), (o_7,5)	(o_6,6), (o_8,6)
⋮				
	$\text{minDist}_{22,1}$:2	$\text{minDist}_{22,2}$:8	$\text{minDist}_{22,3}$:20	$\text{minDist}_{22,4}$:24
c_{22} →	(o_7,1), (o_8,1)	(o_6,2), (o_5,4)	(o_3,5), (o_4,5)	(o_2,7), (o_1,8)

Figure 5.2: An example inverted rank file for the objects $o_1, \ldots, o_8$ in Figure 5.1.

5.4 Rank Bounds and Indexing

In this section, we first present how to compute an upper bound and a lower bound for the rank of an object w.r.t. an unseen query, and then propose an indexing approach referred to as an *Inverted Rank File* (IRF) that can be used to estimate the rank of objects for arriving queries.

5.4.1 Computing Rank Bounds

Here, we assume that the space has been partitioned into cells (the space partitioning step is explained in Section 5.5.1). The rank bound for an object o w.r.t. a cell c_q is computed as follows. For any query q arriving with a location in cell c_q, the rank of o satisfies the condition,

$\boldsymbol{r}^{\downarrow}(o, c_q) \leq \boldsymbol{r}(o, q) \leq \boldsymbol{r}^{\uparrow}(o, c_q)$, where $\boldsymbol{r}^{\downarrow}(o, c_q)$ and $\boldsymbol{r}^{\uparrow}(o, c_q)$ are the lower and the upper bound rank of o for any query q in cell c_q, respectively.

Lower bound rank. The lower bound rank is computed from the number of objects $o' \in O \backslash o$ that are definitely closer to q in c_q than o. Note that a smaller rank indicates a smaller Euclidean distance from the query. Specifically, let ℓ_n be the number of objects $o' \in O \backslash o$ such that $\boldsymbol{d}^{\uparrow}(o', c_q) \leq \boldsymbol{d}^{\downarrow}(o, c_q)$, where $\boldsymbol{d}^{\uparrow}(o', c_q)$ ($\boldsymbol{d}^{\downarrow}(o, c_q)$) are the maximum (minimum) Euclidean distance between o' (o) and cell c_q. Therefore, even if the query location q is the closest point of c_q to o, there are still at least ℓ_n objects closer to q than o. So the rank of o must be greater than ℓ_n for any query in c_q, i.e., $\boldsymbol{r}^{\downarrow}(o, c_q) = \ell_n + 1$.

Upper bound rank. Let, u_n be the number of objects $o' \in O \backslash o$ such that, $\boldsymbol{d}^{\downarrow}(o', c_q) < \boldsymbol{d}^{\uparrow}(o, c_q)$ for any query q in c_q. Therefore, even if a query q arrives at the farthest location in c_q from o, there are at most u_n objects that can possibly be closer to q than o. So the rank of o cannot be greater than $u_n + 1$ for any query in c_q, resulting in $\boldsymbol{r}^{\uparrow}(o, c_q) = u_n + 1$.

Example 13. *In Figure 5.1, let $O = \{o_1, o_2, \ldots, o_8\}$ be the set of objects and c_1 be a cell in D. The minimum and maximum distances between c_1 and an object o_4 are shown with a blue and a red line, respectively. Here, only the maximum distance between c_1 and o_1 is less than $\boldsymbol{d}^{\downarrow}(o_4, c_1)$. Therefore, $\boldsymbol{r}^{\downarrow}(o_4, c_1) = 1 + 1 = 2$. The minimum distance between c_1 and each of the objects o_1, o_2 and o_3 is less than $\boldsymbol{d}^{\uparrow}(o_4, c_1)$, so, $\boldsymbol{r}^{\uparrow}(o_4, c_1) = 3 + 1 = 4$.*

5.4.2 Indexing Rank

We present an indexing technique called an *Inverted Rank File* (IRF) where the dataspace D is partitioned into different cells, and the bounds of each object's rank for queries appearing in the cell is precomputed. The rank information is indexed such that, if a query q arrives anywhere inside a cell c_q, the rank of any object for q can be estimated. A quadtree structure is employed to partition D into cells. We present the general structure of an IRF, then in Section 5.5.1 we present a space partitioning approach to approximately answer T$m\rho$Q with bounded error.

Inverted rank file. An IRF consists of a collection of all leaf level cells of the quadtree, and a set of *rank lists*, one for each leaf level cell c of the quadtree. Each rank list is a sorted sequence of tuples of the form $\langle o, \boldsymbol{r}^{\downarrow}(o, c_q) \rangle$, one for each object $o \in O$, sorted in ascending order of $\boldsymbol{r}^{\downarrow}(o, c_q)$. If multiple objects have the same $\boldsymbol{r}^{\downarrow}(o, c_q)$ for a cell c_q, those tuples are

Table 5.2: Notation used in Section 5.5

Symbol	**Description**
ϵ	Approximation parameter.
$\mathbb{W}_{i-1}$, $\mathbb{W}_i$	Two consecutive windows, where $\mathbb{W}_i$ is derived from $\mathbb{W}_{i-1}$ by excluding qo and adding qn, $\lvert\mathbb{W}_i\rvert=\lvert\mathbb{W}_{i-1}\rvert$.
$\hat{\boldsymbol{r}}(o,q)$ ($\hat{\boldsymbol{\rho}}(o,\mathbb{W})$)	Approximation of $\boldsymbol{r}(o,q)$ ($\rho(o,\mathbb{W})$).
R_i	The set of result objects for a window $\mathbb{W}_i$.
o_m	The m·th object from the set R_{i-1}.
$\hat{\boldsymbol{r}}^{\downarrow}(b,q)$ ($\hat{\boldsymbol{r}}^{\uparrow}(b,q)$)	Lower (upper) bound rank of any object in block b for q.
$\hat{\boldsymbol{\rho}}(o_{m+1},\mathbb{W}_{i-1})$	The approximate popularity of top $(m+1)$·th object from R_{i-1} in previous window $\mathbb{W}_{i-1}$.
$\hat{\boldsymbol{\rho}}(o_m,\mathbb{W}_{i-1}\backslash qo)$	The approximate popularity of top m·th object from R_{i-1}, updated w.r.t. excluding q_o from Window $\mathbb{W}_{i-1}$.
$\hat{\boldsymbol{\rho}}(o_m,\mathbb{W}_i)$	The approximate popularity of top m·th object from R_i.
$\zeta(\hat{\boldsymbol{r}}(o,q))$	Contribution of q to the approximate popularity of o.
Δ_o	The popularity gain of object o for the current window.
$\Delta_o^{\uparrow}$	The maximum popularity gain among all objects for the current window.
Δ_b	An upper bound of the gain of the objects $o \in b$.
BSR	Block-level safe rank.
OSR	Object-level safe rank.
VO	The set of validation objects.

sorted by $\boldsymbol{d}^{\downarrow}(o,c_q)$. Here, $\boldsymbol{r}^{\downarrow}(o,c_q)$ is the lower bound rank of o for a query q coming in cell c_q. Each rank list is stored as a sequence of blocks of a fixed length, B. Each block b of the rank list for cell c_q is associated with the minimum distance between c_q and any object in b, where $\boldsymbol{d}^{\downarrow}(b,c_q) = \min_{o\in b}\boldsymbol{d}^{\downarrow}(o,c_q)$.

Example 14. *Figure 5.1 shows an example partitioning of the space for the objects* $O = \{o_1,o_2,\ldots,o_8\}$. *After the partitioning step, the quadtree has* 22 *disjoint leaf cells,* $c_1,\ldots,c_{22}$. *Figure 5.2 shows the resulting IRF index for block size* $B = 2$, *where each leaf level cell of the quadtree is associated with a sorted list of tuples of the form* $\langle o, \boldsymbol{r}^{\downarrow}(o,c_q)\rangle$. *As a specific example, the lower bound rank of the object* o_4 *for the quadtree cell* c_1 *is* 2, *and the tuple* $\langle o_4, 2\rangle$ *is stored in the second block of the rank list of* c_1. *The minimum distance between each cell and any object in each block of its rank list is also shown in Figure 5.2.*

5.5 Approximate Solution

As queries can arrive at a very high rate, there may be instances where the cost of computing the exact solution is too expensive. In this section, we show how to exploit the rank bounds to find an approximate solution with a guaranteed bound for the T$m\rho$Q problem. Table 5.2 summarizes the notation used to present the approximate solution. At the highest level, the approximate solution consists of the following steps:

1. A space partitioning technique is presented (in Section 5.5.1) to construct an IRF index that supports an incremental computation of the approximate solution of T$m\rho$Q (in Section 5.5.2). The approximation error bound is discussed in Section 5.5.5.

2. A *safe rank* is computed which represents a threshold, if this threshold is exceeded by a result object currently in R, that object must remain in R as a valid result. Specifically, the current safe rank can be computed by combining: (i) a *block-level* and (ii) an *object-level safe rank* (Section 5.5.3)

3. If the ranks for all of the result objects are *safe*, R does not need to be updated. Otherwise, more verification must be done to determine if any object can affect R. This can be achieved using a second technique called *validation objects*, which incrementally identifies the objects that can affect R (Section 5.5.4). As long as the current result objects have a higher popularity than the validation objects, R does not need to be updated.

4. If R must be updated, the approximate popularity of the affected objects are computed. We show that the computations of the prior windows can be used to efficiently approximate the popularity scores of the objects that must change in the current window (Section 5.5.4).

5.5.1 Space Partitioning

Ideally, rank bound estimations should be as close as possible to the actual rank of each object. If the quadtree leaf cell where a query q arrives is as small as a single point location (the same as the location of q), then both the upper and the lower bound ranks of any object will be exactly the same as the actual rank of that object for q. However, if the space is partitioned in this way, then the number of cells will be infinite. Therefore, we propose a partitioning technique which guarantees that the difference between the bound of any object's rank and its true rank is bounded by a threshold, ϵ. Specifically, for any $o \in O$, and any leaf level quadtree cell c,

the difference between the upper and the lower bound rank must be within a *percentage of the lower bound rank*:

$$\boldsymbol{r}^{\uparrow}(o,c) - \boldsymbol{r}^{\downarrow}(o,c) \leq \epsilon \times \boldsymbol{r}^{\downarrow}(o,c) \tag{5.1}$$

Otherwise, c is further partitioned until the condition holds. As an example, let $\epsilon = 0.5$. For an object o, and a cell c_i, let $\boldsymbol{r}^{\downarrow}(o,c_i) = 10$ and $\boldsymbol{r}^{\uparrow}(o,c_i) = 20$. So, c_i needs to be further partitioned for o until the condition is met. Let for another cell c_j, $\boldsymbol{r}^{\downarrow}(o,c_j) = 100$ and $\boldsymbol{r}^{\uparrow}(o,c_j) = 120$. Now cell c_j does not need to be partitioned for o since $120 - 100 \leq 0.5 \times 100$.

The intuition behind this partitioning scheme becomes quite clear when the notion of "top" ranked objects is taken into consideration. Getting the exact position of the highest ranked object matters much more than getting the exact position of the object at the thousandth position. So, the granularity of exactness in our inequality degrades gracefully with the true rank of the object.

Partitioning process. The partitioning can be achieved iteratively, where the quadtree root is initialized with the entire space D. The process starts from the root and recursively partitions D. If the partitioning condition is not satisfied for an object o and a cell c, partitioning of c continues until Condition (1) is met. The process terminates when $\forall o$, the condition holds for all of the current leaf level quadtree cells c.

Why use a quadtree?. We use a quadtree to partition the space and then organize the spatial information for each quadtree cell. The rationale is as follows:

- A quadtree partitions the space into mutually-exclusive cells. In contrast, MBRs in an R-tree may have overlaps, so a query location can overlap with multiple partitions, making it difficult to estimate the object ranks in new queries.
- A quadtree is update-friendly, and the partition granularity can be dynamically changed using ϵ to improve the accuracy of the approximation. This allows performance to be quickly and easily tuned for different collections.
- In a quadtree, a cell c is partitioned only when any rank bounds for c do not satisfy Condition (1). In contrast, if a regular grid structure of equal cell size is used, enforcing partitioning using Condition (1) will result in unnecessary cells being created.

ALGORITHM 12: T$m\rho$Q

Input: Window $\mathbb{W}_i$, number of result objects m, the result objects R_{i-1} of the previous window $\mathbb{W}_{i-1}$, and the $(m+1)$·th best popularity $\hat{\boldsymbol{\rho}}(o_{m+1}, \mathbb{W}_{i-1})$ of the previous window $\mathbb{W}_{i-1}$.
Output: Result objects R_i of the current window $\mathbb{W}_i$.
Initialize a max-priority queue PQ.
$R_i \leftarrow \varnothing$;
$qn \leftarrow \mathbb{W}_i \backslash \mathbb{W}_{i-1}$;
$qo \leftarrow \mathbb{W}_{i-1} \backslash \mathbb{W}_i$
for $o \in R_{i-1}$ **do**
  $\hat{\boldsymbol{\rho}}(o, \mathbb{W}_{i-1} \backslash qo) \leftarrow \hat{\boldsymbol{\rho}}(o, \mathbb{W}_{i-1}) - \zeta(\hat{\boldsymbol{r}}(o, qo))/|\mathbb{W}_i|$
$\hat{\boldsymbol{\rho}}(o_m, \mathbb{W}_{i-1} \backslash qo) \leftarrow$ The approximate popularity of top m·th object from R_{i-1} after updating for qo.
BSR $\leftarrow$ Block_safe_rank($\hat{\boldsymbol{\rho}}(o_{m+1}, \mathbb{W}_{i-1})$,$\hat{\boldsymbol{\rho}}(o_m, \mathbb{W}_{i-1} \backslash qo)$,PQ)
for $o \in R_{i-1}$ **do**
  $\hat{\boldsymbol{\rho}}(o, \mathbb{W}_i) \leftarrow \hat{\boldsymbol{\rho}}(o, \mathbb{W}_{i-1} \backslash qo) + \frac{1}{|\mathbb{W}|} \times \zeta(\hat{\boldsymbol{r}}(o, qn))$
  if $\hat{\boldsymbol{r}}(o, qn) \leq$ BSR & $o \in O^+_{qn}$ **then**
    $R_i \leftarrow o$
if $|R_i| < m$ **then**
  $\hat{\boldsymbol{\rho}}(o_m, \mathbb{W}_i) \leftarrow$ Current m·th best popularity of R_{i-1} in $\mathbb{W}_i$.
  OSR $\leftarrow$ Object_safe_rank($\hat{\boldsymbol{\rho}}(o_{m+1}, \mathbb{W}_{i-1}), \hat{\boldsymbol{\rho}}(o_m, \mathbb{W}_i)$, PQ)
  for $o \in R_{i-1} \backslash R_i$ **do**
    if $\hat{\boldsymbol{r}}(o, qn) \leq$ OSR & $o \in O^+_{qn}$ **then**
      $R_i \leftarrow o$
if $|R_i| < m$ **then**
  VO $\leftarrow$ Validation_objects($\hat{\boldsymbol{\rho}}(o_m, \mathbb{W}_i)$, PQ)
  if VO $\neq \varnothing$ **then**
    $R_i \leftarrow$ Update_results(VO, $R_{i-1} \backslash R_i$)
return R_i

5.5.2 Framework of Approximate Solution

In this section, we first introduce how to compute the approximate popularity of an object for a given sliding window, then we show how to aggregate the top-m approximate results. Since this section is all about how to compute the approximate popularity of objects, we use the terms popularity and approximate popularity interchangeable, unless specified otherwise.

First, a lemma is presented to show that the rank of any object o for a query q arriving in a cell c can be estimated using only the lower bound rank, $\boldsymbol{r}^{\downarrow}(o, c)$ within an error bound.

Lemma 4. *For any object $o \in O$, and any query q arriving in cell c, $\boldsymbol{r}^{\downarrow}(o,c) \leq \boldsymbol{r}(o,q) \leq (1+\epsilon) \times \boldsymbol{r}^{\downarrow}(o,c)$ always holds.*

Proof. The rank bounds are computed such that $\boldsymbol{r}^{\downarrow}(o,c) \leq \boldsymbol{r}(o,q) \leq \boldsymbol{r}^{\uparrow}(o,c)$ always holds. For any object $o \in O$, and for any leaf level cell c of the quadtree, the space is partitioned in a way that guarantees $\boldsymbol{r}^{\uparrow}(o,c) - \boldsymbol{r}^{\downarrow}(o,c) \leq \epsilon \times \boldsymbol{r}^{\downarrow}(o,c)$, so, clearly $\boldsymbol{r}^{\downarrow}(o,c) \leq \boldsymbol{r}(o,q) \leq (1+\epsilon) \times \boldsymbol{r}^{\downarrow}(o,c)$ also holds. □

Based on Lemma 4, we approximate the rank of an object with an error bound as:

$$\hat{\boldsymbol{r}}(o,q) = (1 + \frac{\epsilon}{2}) \times \boldsymbol{r}^{\downarrow}(o,c) \tag{5.2}$$

Corollary 1. *For any object $o \in O$, and any query q arriving in cell c, $|\boldsymbol{r}(o,q) - \hat{\boldsymbol{r}}(o,q)| \leq \epsilon/2 \times \boldsymbol{r}^{\downarrow}(o,c)$ always holds.*

Proof. Here, the maximum difference between $\boldsymbol{r}^{\downarrow}$(o,c) and $\hat{\boldsymbol{r}}$(o,q), and the maximum difference between $\boldsymbol{r}^{\uparrow}$(o,c) and $\hat{\boldsymbol{r}}$(o,q), are both $\epsilon/2 \times \boldsymbol{r}^{\downarrow}(o,c)$. Therefore, the proof follows from Lemma 4. □

The approximate popularity $\hat{\boldsymbol{\rho}}(o, \mathbb{W})$ of an object o for the queries q in a $\mathbb{W}$ can be computed using rank approximation:

$$\hat{\boldsymbol{\rho}}(o, \mathbb{W}) = \frac{1}{|\mathbb{W}|} \sum_{i=1}^{|\mathbb{W}|} \begin{cases} |O| - \hat{\boldsymbol{r}}(o, q_i) + 1 & \text{where } o \in O_{q_i}^{+}, \\ 0 & \text{otherwise.} \end{cases} \tag{5.3}$$

Updating a count based sliding window $\mathbb{W}_i$ of queries from the previous window $\mathbb{W}_{i-1}$ can be formulated as replacing the least recent query qo by the most recent query qn. As a result, only the leaf level cells (in the quadtree) that contain qn and qo need to be found, namely c_{qn} and c_{qo}. The rank lists corresponding to these cells can be quickly retrieved from the IRF index. For each window $\mathbb{W}_i$, assume that $m+1$ objects with the highest $\hat{\boldsymbol{\rho}}$ are computed, where the top-m objects are returned as the result R_i of T$m\rho$Q for $\mathbb{W}_i$, and the popularity of the $(m+1)$·th object is used in the next window to identify the safe rank and the validation objects efficiently.

The steps for updating the approximate solution of T$m\rho$Q for a window $\mathbb{W}_i$ are shown in Algorithm 12. Note that the necessary notation is defined in Table 5.2. Here, $\zeta(o,q)$ is the contribution of q to the popularity of o, and is computed as:

$$\zeta(\hat{\boldsymbol{r}}(o,q)) = \begin{cases} |O| - \hat{\boldsymbol{r}}(o,q) + 1 & \text{where } o \in O_{q}^{+}, \\ 0 & \text{otherwise.} \end{cases}$$

First, the approximate popularity of the result objects $o \in R_{i-1}$ for the excluded query qo, $\hat{\boldsymbol{r}}(o, qo)$ is updated. Let the updated m·th highest popularity from R_{i-1} be $\hat{\boldsymbol{\rho}}(o_m, \mathbb{W}_{i-1} \backslash qo)$ (Lines 12.7 - 12.9 in Algorithm 12). The rest of the algorithm consists of three main components - (i) computing the safe rank in two steps (block-level and object-level safe rank), (ii) finding the set of validation objects, and (iii) updating R_i.

Locating an object in IRF. Since some of the steps in Algorithm 12 require finding the entry of a particular object in a rank list of IRF, we first present an efficient technique for locating objects, and then describe the remaining steps of the algorithm. We start with a lemma that find a relation between the minimum Euclidean distance of the objects from a cell c and the lower bound ranks of the objects for any query in c.

Lemma 5. *For any two objects $o_i, o_j \in O$ and a cell c, if $\boldsymbol{r}^{\downarrow}(o_i, c) \leq \boldsymbol{r}^{\downarrow}(o_j, c)$, then $\boldsymbol{d}^{\downarrow}(o_i, c) \leq \boldsymbol{d}^{\downarrow}(o_j, c)$ always holds.*

Proof. We prove the lemma using proof by contradiction. Assume that $\boldsymbol{d}^{\downarrow}(o_i, c) > \boldsymbol{d}^{\downarrow}(o_j, c)$ is true. In the rank list of IRF, the entries $\langle o, \boldsymbol{r}^{\downarrow}(o, c)\rangle$ are sorted in ascending order of the lower bound rank, $\boldsymbol{r}^{\downarrow}(o, c)$. Here, $\boldsymbol{r}^{\downarrow}(o, c)$ is the number of objects o' (plus 1) that are guaranteed to be closer to c than o. So, $\boldsymbol{d}^{\uparrow}(o', c) \leq \boldsymbol{d}^{\downarrow}(o, c)$. Let $\mid O'_i \mid = \ell_i$ be the set of all the objects from O such that $\forall o'_i \in O'_i$, $\boldsymbol{d}^{\uparrow}(o'_i, c) \leq \boldsymbol{d}^{\downarrow}(o_i, c)$, and $\mid O'_j \mid = \ell_j$ be the set of objects where $\forall o'_j \in O'_j$, $\boldsymbol{d}^{\uparrow}(o'_j, c) \leq \boldsymbol{d}^{\downarrow}(o_j, c)$. As $\boldsymbol{r}^{\downarrow}(o_i, c) \leq \boldsymbol{r}^{\downarrow}(o_j, c)$, then $\ell_i \leq \ell_j$ is also true.

Since $\boldsymbol{d}^{\downarrow}(o_i, c) > \boldsymbol{d}^{\downarrow}(o_j, c)$ was assumed to be true, $\boldsymbol{d}^{\uparrow}(o'_j, c) \leq \boldsymbol{d}^{\downarrow}(o_j, c) < \boldsymbol{d}^{\downarrow}(o_i, c)$. Therefore, o'_i and o'_j both are in the set of objects from O that satisfy $\boldsymbol{d}^{\uparrow}(o'_i, c) \leq \boldsymbol{d}^{\downarrow}(o_i, c)$, and $\boldsymbol{d}^{\uparrow}(o'_j, c) < \boldsymbol{d}^{\downarrow}(o_i, c)$, respectively. Hence, $O'_j \subseteq O'_i$, so $\ell_j \leq \ell_i$ must be true. But this is a contradiction. Therefore, $\boldsymbol{d}^{\downarrow}(o_i, c) > \boldsymbol{d}^{\downarrow}(o_j, c)$ cannot be true. If $\boldsymbol{r}^{\downarrow}(o_i, c) \leq \boldsymbol{r}^{\downarrow}(o_j, c)$ is true, $\boldsymbol{d}^{\downarrow}(o_i, c) \leq \boldsymbol{d}^{\downarrow}(o_j, c)$ must hold. □

Lemma 5 shows that sorting the objects by $\boldsymbol{r}^{\downarrow}(o, c)$ is equivalent to sorting the objects by their minimum Euclidean distance to c, $\boldsymbol{d}^{\downarrow}(o, c)$. If multiple objects have the same $\boldsymbol{r}^{\downarrow}(o, c)$, they are already stored as sorted by $\boldsymbol{d}^{\downarrow}(o, c)$ as described in Section 5.4.2. Therefore, we can locate the position of o in the rank list of cell c by performing the following steps:

- Compute the minimum Euclidean distance $\boldsymbol{d}^{\downarrow}(o, c)$ of c from o.
- As described in Section 5.4.2, each block b of the rank list for cell c is associated with the minimum distance between c_q and any object in b, $\boldsymbol{d}^{\downarrow}(b, c)$, so a binary search on $\boldsymbol{d}^{\downarrow}(b, c)$ can be performed to find the position of the block b where o is stored.

- Perform a linear scan in that block to find the entry for o.

The entire process has a $\mathcal{O}(log_2(|O|/B) + B)$ time complexity, where B is the number of objects in a block.

5.5.3 Safe rank

Recall that in Algorithm 12 the purpose of finding a safe rank is to minimize the number of updates in R_{i-1} (result objects in the previous window $\mathbb{W}_{i-1}$) to get the result set R_i (in the current window $\mathbb{W}_i$) whenever the sliding window shifts. In particular, the idea is to compute the safe rank OSR for the objects $o \in R_{i-1}$ such that, if $\hat{\boldsymbol{r}}(o, qn) < \text{OSR}$, then no other object from $o' \in O \backslash R_{i-1}$ can have a higher $\hat{\boldsymbol{\rho}}$ than o, thereby o is a valid result in R_i as well. Note that a smaller value of rank implies a higher contribution to the popularity measure. The safe rank is defined w.r.t. the current window $\mathbb{W}_i$ by default.

Before presenting the computation of an object's safe rank, the concept of **popularity gain** is introduced. The popularity gain of an object o, denoted by Δ_o, results from replacing the least recent query qo by the most recent query qn, i.e., the popularity difference of o in the current window $\mathbb{W}_i$ from the previous window $\mathbb{W}_{i-1}$ as:

$$\Delta_o = \hat{\boldsymbol{\rho}}(o, \mathbb{W}_i) - \hat{\boldsymbol{\rho}}(o, \mathbb{W}_{i-1}) = \frac{\zeta(\hat{\boldsymbol{r}}(o, qn)) - \zeta(\hat{\boldsymbol{r}}(o, qo))}{|\mathbb{W}_i|} \tag{5.4}$$

Here, if o does not satisfy the query constraint $\text{Con}(q)$, the contribution of q to the popularity of o, $\zeta(\hat{\boldsymbol{r}}(o, qn)) = 0$. Let $\Delta_o^{\uparrow}$ denote the maximum popularity gain among all objects (in the current window $\mathbb{W}_i$). Then the popularity of any object $o' \in O \backslash R_{i-1}$ can be at most $\hat{\boldsymbol{\rho}}(o_{m+1}, \mathbb{W}_{i-1}) + \Delta_o^{\uparrow}$, where $\hat{\boldsymbol{\rho}}(o_{m+1}, \mathbb{W}_{i-1})$ is the $(m+1)$·th highest approximate popularity in the previous window $\mathbb{W}_{i-1}$. In other words, if the updated popularity of an object $o \in R_{i-1}$ is higher (better) than $\hat{\boldsymbol{\rho}}(o_{m+1}, \mathbb{W}_{i-1}) + \Delta_o^{\uparrow}$, then such an o is guaranteed to remain in R_i, which inspires the design of object-level safe rank OSR as:

$$\hat{\boldsymbol{\rho}}(o_m, \mathbb{W}_{i-1} \backslash qo) + \frac{|O| - \text{OSR} + 1}{|\mathbb{W}_i|} \geq \hat{\boldsymbol{\rho}}(o_{m+1}, \mathbb{W}_{i-1}) + \Delta_o^{\uparrow} \tag{5.5}$$

Since a lower rank indicates a higher contribution to the popularity, the gain will be maximized when the difference between $\zeta(\hat{\boldsymbol{r}}(o, qn))$ and $\zeta(\hat{\boldsymbol{r}}(o, qo))$ is maximized. Therefore, the goal of minimizing the updates of R_{i-1} can be reduced to the challenge of how to compute a tight estimation of $\Delta_o^{\uparrow}$.

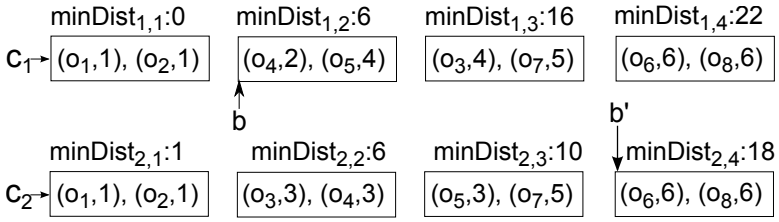


Figure 5.3: Example of computing block-level gain, where block b is currently under consideration. The most recent query qn and the least query qo are in cell c_1 and c_2 respectively.

5.5.3.1 Block-level Popularity Gain

Since the objects are arranged blockwise in an IRF index, and each object o is sorted by the lower bound rank $\boldsymbol{r}^{\downarrow}(o, c_q)$ in ascending order, a *block-level gain* can also be used as the first step in finding a tighter estimation of the maximum gain. A block-level maximum gain $\Delta_b^{\uparrow}$ is computed such that $\Delta_b^{\uparrow} \geq \Delta_o^{\uparrow}$, which can be used to find the block-level safe rank, BSR. If the rank of any result object is not better than BSR for qn, then an object-level maximum gain, $\Delta_o^{\uparrow}$ is computed. The object-level safe rank OSR can be computed using this value, where OSR $\geq$ BSR, as a lower value of rank implies a higher gain. If the rank of any result object is still not safe, then the validation objects (proposed in Section 5.5.4) must be checked to decide if R_{i-1} needs to be updated. Here, a part of the safe rank calculations can be reused to find the validation objects, which will be explained in Section 5.5.4.

Block-level gain computation. Given a block b from the rank list of qn, the block-level gain Δ_b is an overestimation of the gain of the objects $o \in b$, such that $\Delta_b \geq \Delta_o$. As the gain is maximized when the difference between $\zeta(\hat{\boldsymbol{r}}(o, qn))$ and $\zeta(\hat{\boldsymbol{r}}(o, qo))$ is maximized, a technique to compute Δ_b can be actualized by finding: (i) a lower bound estimation of the rank of any object $o \in b$ for qn, namely $\hat{\boldsymbol{r}}^{\downarrow}(b, qn)$, where, $\hat{\boldsymbol{r}}^{\downarrow}(b, qn) \leq \hat{\boldsymbol{r}}(o, qn)$; and (ii) an upper bound estimation of the rank that any object $o \in b$ can have for qo, denoted as $\hat{\boldsymbol{r}}^{\uparrow}(b, qo)$, such that $\hat{\boldsymbol{r}}(o, qo) \leq \hat{\boldsymbol{r}}^{\uparrow}(b, qo)$.

Since the objects are sorted in ascending order of lower bound ranks in the IRF index, the lower bound rank of the first entry of b is implicitly $\hat{\boldsymbol{r}}^{\downarrow}(b, qn)$. Here, $\forall o \in b, \hat{\boldsymbol{r}}^{\downarrow}(b, qn) \leq \boldsymbol{r}^{\downarrow}(o, qn)$ holds by definition.

Next, for the same block b of the rank list of qn, finding the maximum rank $\hat{\boldsymbol{r}}^{\uparrow}(b, qo)$ that any object $o \in b$ can have for qo is needed. To achieve this, a block b' is found such that all of the objects $o \in b$ are guaranteed to be in the rank list of qo before b'. As the objects are sorted by $\boldsymbol{r}^{\downarrow}(o, c_{qo})$ in the rank list of c_{qo}, $\boldsymbol{r}^{\downarrow}(o', c_{qo})$ is guaranteed to be greater than that of

any object in b', where o' is the first entry of b'. Therefore, $\boldsymbol{r}^{\downarrow}(o', qo)$ is taken as the upper bound estimation, $\hat{\boldsymbol{r}}^{\uparrow}(b, qo)$.

For a tight estimation of $\hat{\boldsymbol{r}}^{\uparrow}(b, qo)$, the block b' with the smallest $\boldsymbol{r}^{\downarrow}(o', qo)$ must be found. As the objects and blocks of a rank list are sorted by the minimum Euclidean distance from the corresponding cell (Section 5.4.2), and $\forall o \in b$, $\boldsymbol{d}^{\downarrow}(o, c_{qo}) \leq \boldsymbol{d}^{\uparrow}(b, c_{qo})$, a binary search over the blocks of the rank list of qo is performed to find the first position of the block b' where $\boldsymbol{d}^{\uparrow}(b, c_{qo}) \leq \boldsymbol{d}^{\downarrow}(b', c_{qo})$. Here, $\boldsymbol{d}^{\uparrow}(b, c_{qo})$ is computed as the maximum Euclidean distance between the minimum bounding rectangle of the objects $o \in b$ and cell c_{qo}.

Example 15. *In Figure 5.3, let c_1 and c_2 be the cell where qn and qo arrive respectively. The constraint of both queries are satisfied by all of the objects. Let $b = \langle (o_4, 2), (o_5, 4) \rangle$ be the block of the rank list of c_1 currently under consideration. Here, $\hat{\boldsymbol{r}}^{\downarrow}(b, qn) = 2$, which is the lower bound of the first entry of b. Let $\boldsymbol{d}^{\uparrow}(b, c_2) = 14$, which is computed from the MBR of block b and cell c_2. Now, a binary search is performed with the value 14 over the minimum distance of the blocks in c_2. Here, we get the block $b' = \langle (o_6, 6), (o_8, 6) \rangle$ in the rank list of c_2, as $\boldsymbol{d}^{\downarrow}(b', c_2) = 18$, which is the smallest value of $\boldsymbol{d}^{\downarrow}$ greater than 14, shown with an arrow. So, $\hat{\boldsymbol{r}}^{\uparrow}(b, qo) = 6$ is the lower bound rank of the first entry of b'.*

5.5.3.2 Block-level Safe Rank

By making use of the values $\hat{\boldsymbol{r}}^{\downarrow}(b, qn)$ and $\hat{\boldsymbol{r}}^{\uparrow}(b, qo)$ of block b, a block-level estimation of the maximum gain for $\mathbb{W}_i$ can found, and a block-level safe rank BSR can be computed, as shown in Algorithm 13. Algorithm 13 shows the steps to compute the block-level safe rank by finding the maximum gain of a block using the rank lists of qn and qo. A max-priority queue PQ is used to keep track of blocks that must be visited, where the key is Δ_b. Here, Δ_b is an overestimation of the gain of the objects in b. For any object $o \in b$, $\Delta_o \leq \Delta_b$, is computed in Line 13.10 as -

$$\Delta_b \leftarrow \frac{1}{|\mathbb{W}_i|}\zeta(\hat{\boldsymbol{r}}^{\downarrow}(b, qn)) - \zeta(\hat{\boldsymbol{r}}^{\uparrow}(b, qo))$$

Recall that in the IRF index, each object o in the rank list is sorted in ascending order of the lower bound rank w.r.t. the cell c_{qn}, and the traversal starts from the beginning of the rank list of c_{qn} so that the objects with a higher gain are most likely to be explored first. The traversal continues until the subsequent blocks of the rank lists of qn cannot have a better gain than the current maximum gain $\Delta_b^{\uparrow}$ found so far. Here, the **terminating condition** of Line 13.14 is:

$$\frac{\zeta(\hat{\boldsymbol{r}}^{\downarrow}(b, qn))}{|\mathbb{W}_i|} < \Delta_b^{\uparrow}.$$

ALGORITHM 13: BLOCK_SAFE_RANK

13.1 **Input:** $\hat{\boldsymbol{\rho}}(o_{m+1}, \mathbb{W}_{i-1})$ - $(m+1)$·th highest popularity of $\mathbb{W}_{i-1}$, $\hat{\boldsymbol{\rho}}(o_m, \mathbb{W}_{i-1} \backslash qo)$ - m·th highest popularity from R_{i-1} after updating for qo, and PQ - a max-priority queue.
13.2 **Output:** Block based safe rank - BSR.
13.3 $b \leftarrow$ First block in the rank list of c_{qn}.
13.4 $\Delta_b^{\uparrow} \leftarrow 0$
13.5 **do**
13.6 $\quad \hat{\boldsymbol{r}}^{\downarrow}(b, qn) \leftarrow \boldsymbol{r}^{\downarrow}(o, c_{qn})$ of the first entry o from b.
13.7 $\quad \boldsymbol{d}^{\uparrow}(b, c_{qo}) \leftarrow$ Maximum Euclidean distance between b, c_{qo}.
13.8 $\quad b' \leftarrow$ First block position of c_{qo}, where $\boldsymbol{d}^{\uparrow}(b, c_{qo}) \leq \boldsymbol{d}^{\downarrow}(b', c_{qo})$.
13.9 $\quad \hat{\boldsymbol{r}}^{\uparrow}(b, qo) \leftarrow \boldsymbol{r}^{\downarrow}(o', c_{qo})$ of the first entry o' of b'.
13.10 $\quad \Delta_b \leftarrow \dfrac{\zeta(\hat{\boldsymbol{r}}^{\downarrow}(b, qn)) - \zeta(\hat{\boldsymbol{r}}^{\uparrow}(b, qo))}{|\mathbb{W}|}$
13.11 $\quad$ ENQUEUE $(\mathrm{PQ}, b, \Delta_b)$
13.12 $\quad \Delta_b^{\uparrow} \leftarrow \Delta_{\mathrm{top(PQ)}}$
13.13 $\quad b \leftarrow$ NEXT (c_{qn})
13.14 **while** *b cannot have a better gain than* $\Delta_b^{\uparrow}$
13.15 BSR $\leftarrow$ Compute from $\hat{\boldsymbol{\rho}}(o_{m+1}, \mathbb{W}_{i-1})$,$\hat{\boldsymbol{\rho}}(o_m, \mathbb{W}_{i-1} \backslash qo)$, $\Delta_b^{\uparrow}$ as Equation 5.6.
13.16 return BSR

Lastly, in Line 13.15 the maximum gain value $\Delta_b^{\uparrow}$ is used to compute the block-level safe rank as follows:

$$\hat{\boldsymbol{\rho}}(o_m, \mathbb{W}_{i-1} \backslash qo) + \frac{|O| - \mathrm{BSR} + 1}{|\mathbb{W}_i|} \geq \hat{\boldsymbol{\rho}}(o_{m+1}, \mathbb{W}_{i-1}) + \Delta_b^{\uparrow}. \tag{5.6}$$

5.5.3.3 Object-level Safe Rank

If the rank of any object $o \in R_{i-1}$ for qn is not smaller (better) than the block-level safe rank BSR, then the object-level safe rank is computed, where OSR $\geq$ BSR is used to further determine whether the result needs to updated or not (Lines 12.15 - 12.20 in Algorithm 12).

Algorithm 14 shows a best-first approach to compute the maximum object level gain $\Delta_o^{\uparrow}$, using the same priority queue PQ maintained in the block-level computation. In each iteration, the top element E of PQ is dequeued. If E is a block, the approximate rank of each $o \in E$ for qn and qo is computed using the lower bound rank in the corresponding rank lists. The objects are then enqueued in PQ according to the gain computed using Equation 5.4. If E is an object, then the gain is returned as $\Delta_o^{\uparrow}$ (Lines 14.6 - 14.7). The object-level safe rank, OSR, is then computed in the same manner as Equation 5.6 with $\Delta_o^{\uparrow}$.

ALGORITHM 14: OBJECT_SAFE_RANK

14.1 **Input:** $\hat{\boldsymbol{\rho}}(o_{m+1}, \mathbb{W}_{i-1})$ - $(m+1)$·th highest popularity of $\mathbb{W}_{i-1}$, $\hat{\boldsymbol{\rho}}(o_m, \mathbb{W}_{i-1} \backslash qo)$ - updated m·th highest popularity from R_{i-1} after removing qo, PQ - a max-priority queue from BLOCK_SAFE_RANK.
14.2 **Output:** Object-level safe rank, OSR.
14.3 **while** PQ *not empty* **do**
14.4 $E \leftarrow$ DEQUEUE (PQ)
14.5 **if** *E is object* **then**
14.6 $\Delta_o^{\uparrow} \leftarrow \Delta_E$
14.7 BREAK
14.8 **else**
14.9 **for** *o in E* **do**
14.10 $\Delta_o \leftarrow \frac{\zeta(\hat{\boldsymbol{r}}(o, qn)) - \zeta(\hat{\boldsymbol{r}}(o, qo))}{|\mathbb{W}|}$
14.11 ENQUEUE (PQ, o, Δ_o)
14.12 OSR $\leftarrow$ Compute from $\hat{\boldsymbol{\rho}}(o_m, \mathbb{W}_i)$,$\hat{\boldsymbol{\rho}}(o_m, \mathbb{W}_{i-1} \backslash qo)$,$\Delta_o^{\uparrow}$ (by Equation 5.5).
14.13 return OSR

5.5.4 Validation Objects

If the rank of any object $o \in R_{i-1}$ is not safe, a set of validation objects VO is found such that, as long as $\forall vo \in$ VO, $\hat{\boldsymbol{\rho}}(o, \mathbb{W}_i) \geq \hat{\boldsymbol{\rho}}(vo, \mathbb{W}_i)$, o is a valid result object of R_i. We present an efficient approach to incrementally identify VO. Furthermore, we show that if the result needs to be updated, the new result objects also must come from VO.

First, after a new query qn arrives, the approximate rank for each object $o \in R_{i-1}$ is computed, and the appropriate popularity scores are updated. Let the updated m·th highest approximate popularity from R_{i-1} be $\hat{\boldsymbol{\rho}}(o_m, \mathbb{W}_i)$ (Line 12.16 of Algorithm 12). The priority queue PQ maintained for safe rank computation is used to find the set VO of validation objects, where $\hat{\boldsymbol{\rho}}(o_m, \mathbb{W}_i)$ is used as a threshold to terminate the search.

A best-first search is performed using PQ to find the objects that have gain high enough to be a result. Specifically, if the dequeued element E from PQ is a block, the $\hat{\boldsymbol{r}}$ of each object o in E is computed for qn and qo in the same manner as described for the object-level safe rank computation. As the popularity of an object $o \in O \backslash R_{i-1}$ can be at most $\hat{\boldsymbol{\rho}}(o_{m+1}, \mathbb{W}_{i-1}) + \Delta_o$, an object o is included in the validation set if o satisfies the following condition:

$$\hat{\boldsymbol{\rho}}(o_{m+1}, \mathbb{W}_{i-1}) + \Delta_o \geq \hat{\boldsymbol{\rho}}(o_m, \mathbb{W}_i) \tag{5.7}$$

As PQ is a max-priority queue which is maintained for the gain of the objects and the blocks, the process can be safely terminated when the gain of a dequeued element E does not satisfy the condition in Equation 5.7. If no validation object is found, that means there is no object that can have a higher popularity than the current results, so the result set R_{i-1} (of previous window $\mathbb{W}_{i-1}$) is the result of current window $\mathbb{W}_i$. Otherwise, the popularity of each object in $R_{i-1} \backslash R_i$ needs to be checked against the popularity of the validation objects $vo \in$ VO to update the result.

5.5.4.1 Updating Results

As described in Section 5.5.4, the set of validation objects VO is computed such that no object $O \backslash$VO can have a higher popularity than any of the objects in R_{i-1}. Therefore, only objects in VO are considered when updating the result set. To update the results using the objects $vo \in$ VO, the popularity of vo for the current window must be computed. Therefore, an efficient technique to compute the popularity of the validation objects is now presented.

Computing $\hat{\rho}$ of the validation objects. As the popularity gain of each $vo \in$ VO has already been computed as described in Section 5.5.4, it is sufficient to find the $\hat{\boldsymbol{\rho}}(vo, \mathbb{W}_{i-1} \backslash qo)$ and use it to compute $\hat{\boldsymbol{\rho}}(vo, \mathbb{W}_i)$. Since the popularity of every object for every window is not computed, a straightforward way to compute $\hat{\boldsymbol{\rho}}(vo, \mathbb{W}_{i-1} \backslash qo)$ is to find the rank of vo for each $q \in \mathbb{W}_{i-1} \backslash qo$ using the corresponding rank lists. However, this approach is computationally expensive, especially when the window size is large. Moreover, if vo was a validation object or a result object in a prior window $\mathbb{W}_{i-y}$, then the same computations are repeated unnecessarily for the queries shared by the windows (the queries contained in $\mathbb{W}_i \cap \mathbb{W}_{i-y}$).

Therefore, if $\hat{\boldsymbol{\rho}}$ of a result or a validation object is computed for a window $\mathbb{W}_{i-y}$, the aim is to reuse this computation for later windows in an efficient way. This can be accomplished by storing the popularity of a subset of "necessary" objects from prior windows for later reuse. We show that the choice of these limited number of windows is optimal, and storing the popularity for any additional windows cannot reduce the computational cost any further.

Choosing the limited number of prior windows. The popularity computations can be reused if the number of shared queries among the windows is greater than the number of queries that differ. Otherwise, the popularity must be computed for the window $\mathbb{W}_i$ from scratch rather than reusing the popularity computations from $\mathbb{W}_{i-y}$. Specifically, let Y be the number of shared queries among windows $\mathbb{W}_i$, $\mathbb{W}_{i-y}$ ($Y = |\mathbb{W}_i \cap \mathbb{W}_{i-y}|$), $Q_o = \mathbb{W}_{i-y} \backslash \mathbb{W}_i$, and

$Q_n = \mathbb{W}_i \backslash \mathbb{W}_{i-y}$. So in a count based window, $|Q_n| = |\mathbb{W}_i| - Y$ and $|Q_o| = |\mathbb{W}_i| - Y$, as each time a new query is inserted, the least recent query is removed from the window. If the number of computations required for the shared queries is greater than the number of computations for $|Q_n| + |Q_o|$, i.e., $Y \geq 2(|\mathbb{W}_i| - Y)$, then computations can be reused. So the number of shared queries, Y, should be greater than or equal to $2|\mathbb{W}_i|/3$ for efficient reuse.

Reusing popularity computations. If the condition $Y \geq 2|\mathbb{W}_i|/3$ holds, the popularity of an object o computed for $\mathbb{W}_{i-y}$ can be used to compute $\hat{\boldsymbol{\rho}}(o, \mathbb{W}_i)$ as follows:

$$\hat{\boldsymbol{\rho}}(o, \mathbb{W}_i) = \hat{\boldsymbol{\rho}}(o, \mathbb{W}_{i-y}) + \frac{\sum_{qn \in Q_n} \zeta(\hat{\boldsymbol{r}}(o, qn)) - \sum_{qo \in Q_o} \zeta(\hat{\boldsymbol{r}}(o, qo))}{|\mathbb{W}_i|} \tag{5.8}$$

Popularity lookup table. A popularity lookup table is maintained with the popularity of the result and validation objects for the most recent $2|\mathbb{W}_i|/3$ windows. If a validation object vo of the current window $\mathbb{W}_i$ is found in the lookup table, the popularity is computed using Equation 5.8. Otherwise, the popularity of vo is computed from the rank lists of the queries in $\mathbb{W}_i$. The popularity vo for $\mathbb{W}_i$ is then added to the popularity lookup table for later windows.

Obtaining results. The objects $vo \in \text{VO}$ are considered one by one to update the results. After computing the popularity of an object $vo \in \text{VO}$, if $\hat{\boldsymbol{\rho}}(vo, \mathbb{W}_i) > \hat{\boldsymbol{\rho}}(o_m, \mathbb{W}_i)$, then vo is added to R_i. The set R_i is adjusted such that it contains m objects with the highest $\hat{\boldsymbol{\rho}}$, and the value of $\hat{\boldsymbol{\rho}}(o_m, \mathbb{W}_i)$ is adjusted accordingly. During this process, if the overestimated popularity of an object vo computed with Equation 5.7 is less than the updated $\hat{\boldsymbol{\rho}}(o_m, \mathbb{W}_i)$, that object can be safely discarded from consideration without computing its popularity.

5.5.5 Approximation Error Bound

Here, we formalize the approximation error bound of our proposed approach. For any object $o \in O$, and any window $\mathbb{W}$ of queries, the ratio between $\hat{\boldsymbol{\rho}}(o, \mathbb{W})$ and $\rho(o, \mathbb{W})$ is bounded.

Lemma 6. *For any object $o \in O$, and a window $\mathbb{W}$ of queries, the approximation ratio,*
(i) $\hat{\boldsymbol{\rho}}(o, \mathbb{W})/\rho(o, \mathbb{W}) \leq 1 - \frac{\epsilon}{2 \times |O|}$ when $\rho(o, \mathbb{W}) \geq \hat{\boldsymbol{\rho}}(o, \mathbb{W})$; and
(ii) $\rho(o, \mathbb{W})/\hat{\boldsymbol{\rho}}(o, \mathbb{W}) \leq 1 - \frac{\epsilon}{2 \times |O|}$ when $\hat{\boldsymbol{\rho}}(o, \mathbb{W}) \geq \rho(o, \mathbb{W})$ always holds.

Proof. Here, $\hat{\boldsymbol{r}}(o,q)$ is the average value of the $\boldsymbol{r}^{\downarrow}(o,c)$ and $\boldsymbol{r}^{\uparrow}(o,c) = (1+\epsilon) \times \boldsymbol{r}^{\downarrow}(o,c)$, where c is the cell that contains q. Therefore, the difference between $\boldsymbol{r}(o,q)$ and $\hat{\boldsymbol{r}}(o,q)$ is maximum when $\boldsymbol{r}(o,q) = \boldsymbol{r}^{\downarrow}(o,c)$ or $\boldsymbol{r}(o,q) = \boldsymbol{r}^{\uparrow}(o,c)$.

From Equation 5.3, the difference between the exact and the approximate popularity computation of an object o is derived from substituting the $\boldsymbol{r}(o,q)$ by $\hat{\boldsymbol{r}}(o,q)$ for each query q in $\mathbb{W}$. If o does not satisfy $\mathrm{Con}(q)$, the contribution to the popularity for q is 0 for both cases. Therefore, the difference between $\rho(o,\mathbb{W})$ and $\hat{\boldsymbol{\rho}}(o,\mathbb{W})$ is maximum when either (i) $\boldsymbol{r}(o,q_i) = \boldsymbol{r}^{\downarrow}(o,c_i)$, or (ii) $\boldsymbol{r}(o,q_i) = \boldsymbol{r}^{\uparrow}(o,c_i)$ for each q_i in $\mathbb{W}$. We denote $\lambda_i = \sum_{i=1}^{|\mathbb{W}|} \boldsymbol{r}^{\downarrow}(o,c_i)$ for ease of presentation.
(i) If $\boldsymbol{r}(o,q_i) = \boldsymbol{r}^{\downarrow}(o,c_i)$ for each q_i in $\mathbb{W}$, then

$$(1)\ \ \rho(o,\mathbb{W}) = \frac{\sum_{i=1}^{|\mathbb{W}|} |O| - \boldsymbol{r}^{\downarrow}(o,c_i) + 1}{|\mathbb{W}|}, \text{ and}$$

$$(2)\ \ \hat{\boldsymbol{\rho}}(o,\mathbb{W}) = \frac{\sum_{i=1}^{|\mathbb{W}|} |O| - (1+\epsilon/2) \times \boldsymbol{r}^{\downarrow}(o,c_i) + 1}{|\mathbb{W}|} \text{ (from Equation 5.2).}$$

$$\begin{aligned}
\rho(o,\mathbb{W}) - \hat{\boldsymbol{\rho}}(o,\mathbb{W}) &= \frac{\sum_{i=1}^{|\mathbb{W}|} -\boldsymbol{r}^{\downarrow}(o,c_i) + (1+\epsilon/2)\boldsymbol{r}^{\downarrow}(o,c_i)}{|\mathbb{W}|} \\
&= \epsilon/2 \times \frac{\lambda_i}{|\mathbb{W}|} \\
\frac{\rho(o,\mathbb{W})}{\rho(o,\mathbb{W}) - \hat{\boldsymbol{\rho}}(o,\mathbb{W})} &= \frac{\sum_{i=1}^{|\mathbb{W}|} |O| - \boldsymbol{r}^{\downarrow}(o,c_i) + 1}{\epsilon/2 \times \lambda_i} \\
&= \frac{\mathbb{W} \times |O| + \mathbb{W} - \lambda_i}{\epsilon/2 \times \lambda_i}
\end{aligned}$$

Here, $\boldsymbol{r}^{\downarrow}(o,c)$ is the lower bound rank estimation, and the rank of an object is between $[1,|O|]$, hence, $\mathbb{W} \leq \lambda_i \leq \mathbb{W} \times |O|$. Therefore, the value of the nominator $\mathbb{W} \times |O| + \mathbb{W} - \lambda_i$ is also between $[\mathbb{W}, \mathbb{W} \times |O|]$. So by setting the lowest value of λ_i in the equation, we get the following inequality,

$$\begin{aligned}
\frac{\rho(o,\mathbb{W})}{\rho(o,\mathbb{W})-\hat{\boldsymbol{\rho}}(o,\mathbb{W})} &\leq \frac{\mathbb{W}\times|O|+\mathbb{W}-\mathbb{W}}{\epsilon/2\times\mathbb{W}} \\
&\leq \frac{|O|}{\epsilon/2} \\
&\leq \frac{2\times|O|}{\epsilon} \\
\Rightarrow \frac{\rho(o,\mathbb{W})-\hat{\boldsymbol{\rho}}(o,\mathbb{W})}{\rho(o,\mathbb{W})} &\geq \frac{\epsilon}{2\times|O|}, \text{ (by taking the inverse)} \\
\Rightarrow 1-\frac{\hat{\boldsymbol{\rho}}(o,\mathbb{W})}{\rho(o,\mathbb{W})} &\geq \frac{\epsilon}{2\times|O|} \\
\Rightarrow \frac{\hat{\boldsymbol{\rho}}(o,\mathbb{W})}{\rho(o,\mathbb{W})} &\leq 1-\frac{\epsilon}{2\times|O|}
\end{aligned}$$

(ii) If $\boldsymbol{r}(o,q_i)=\boldsymbol{r}^{\uparrow}(o,c_i)$ for each q_i in $\mathbb{W}$, then

$$(1)\ \ \rho(o,\mathbb{W})=\frac{\sum_{i=1}^{|\mathbb{W}|}|O|-(1+\epsilon)\times\boldsymbol{r}^{\downarrow}(o,c_i)+1}{|\mathbb{W}|}, \text{ and}$$

$$(2)\ \ \hat{\boldsymbol{\rho}}(o,\mathbb{W})=\frac{\sum_{i=1}^{|\mathbb{W}|}|O|-(1+\epsilon/2)\times\boldsymbol{r}^{\downarrow}(o,c_i)+1}{|\mathbb{W}|} \text{ (from Equation 5.2).}$$

$$\begin{aligned}
\hat{\boldsymbol{\rho}}(o,\mathbb{W})-\rho(o,\mathbb{W}) &= \frac{\sum_{i=1}^{|\mathbb{W}|}-(1+\epsilon/2)\times\boldsymbol{r}^{\downarrow}(o,c_i)+(1+\epsilon)\times\boldsymbol{r}^{\downarrow}(o,c_i)}{|\mathbb{W}|} \\
&= \epsilon/2\times\frac{\lambda_i}{|\mathbb{W}|} \\
\frac{\hat{\boldsymbol{\rho}}(o,\mathbb{W})}{\hat{\boldsymbol{\rho}}(o,\mathbb{W})-\rho(o,\mathbb{W})} &= \frac{\sum_{i=1}^{|\mathbb{W}|}|O|-(1+\epsilon/2)\times\boldsymbol{r}^{\downarrow}(o,c_i)+1}{\epsilon/2\times\lambda_i} \\
&= \frac{\mathbb{W}\times|O|+\mathbb{W}-(1+\epsilon/2)\times\lambda_i}{\epsilon/2\times\lambda_i}
\end{aligned}$$

Setting the lowest value of $\lambda_i = \mathbb{W}$ in the equation produces the following inequality,

$$\begin{aligned}\frac{\hat{\boldsymbol{\rho}}(o,\mathbb{W})}{\hat{\boldsymbol{\rho}}(o,\mathbb{W}) - \rho(o,\mathbb{W})} &\leq \frac{\mathbb{W} \times |O| + \mathbb{W} - (1+\epsilon/2) \times \mathbb{W}}{\epsilon/2 \times \mathbb{W}} \\ &\leq \frac{|O| - \epsilon/2}{\epsilon/2} \\ \Rightarrow \frac{\hat{\boldsymbol{\rho}}(o,\mathbb{W}) - \rho(o,\mathbb{W})}{\hat{\boldsymbol{\rho}}(o,\mathbb{W})} &\geq \frac{\epsilon/2}{|O| - \epsilon/2}, \text{ (by taking the inverse)}\end{aligned}$$

Since the value ϵ is between $[0, |O| - 1]$, the denominator $|O| - \epsilon/2$ can be a maximum of $|O|$.

$$\begin{aligned}1 - \frac{\rho(o,\mathbb{W})}{\hat{\boldsymbol{\rho}}(o,\mathbb{W})} &\geq \frac{\epsilon/2}{|O| - \epsilon/2} \geq \frac{\epsilon/2}{|O|} \\ \Rightarrow \frac{\rho(o,\mathbb{W})}{\hat{\boldsymbol{\rho}}(o,\mathbb{W})} &\leq 1 - \frac{\epsilon}{2 \times |O|}\end{aligned}$$

□

5.5.6 Extending the Solution for Time-based Window

Given a fixed time interval, a time-based sliding window $\mathbb{W}$ contains all of the queries that have arrived within the most recent interval. In contrast to a count based window, the number of new queries included, and the number of queries excluded from a time based window can vary at each interval. Let the set of new queries included be Q'_n and the set of queries excluded be Q'_o for a time based window. Then, the following modifications of the solution presented in Section 5.5 can be made to support this window type:

1. Similar to a count based window, if the aggregated rank estimation of a result object o is high enough after a shift of the sliding window such that no other object can have a higher popularity, o must remain as a result object. Therefore, the popularity gain and the safe rank needs to be computed for the sets Q'_n and Q'_o for the queries each time the sliding window shifts. For example, the object level gain is computed as:

$$\Delta_o = \frac{1}{|\mathbb{W}_i|}\zeta(\sum_{qn \in Q'_n} \hat{\boldsymbol{r}}(o, qn)) - \zeta(\sum_{qo \in Q'_o} \hat{\boldsymbol{r}}(o, qo)).$$

2. If the current result objects are not safe, we need to find the validation objects. Similar to a count based window, a best-first search can be performed to find the objects that can have a higher popularity than the current results, where the gain values are computed for the sets Q'_n and Q'_o.

Note that, although the solution can be extended for a time based window, the same approximation bound is not applicable as the window size varies at each interval. We leave this and other window-based variations of our solution to future work.

5.6 Experimental Evaluation

In this section, we present the experimental evaluation for our proposed approach to monitor the top-m popular objects in a sliding window of streaming queries. As there is no prior work that directly answers this problem (Section 5.2), we compare our approximate solution (proposed in Section 5.5), denoted by AP, with the baseline exact approach (proposed in Section 5.3.2), denoted by BS.

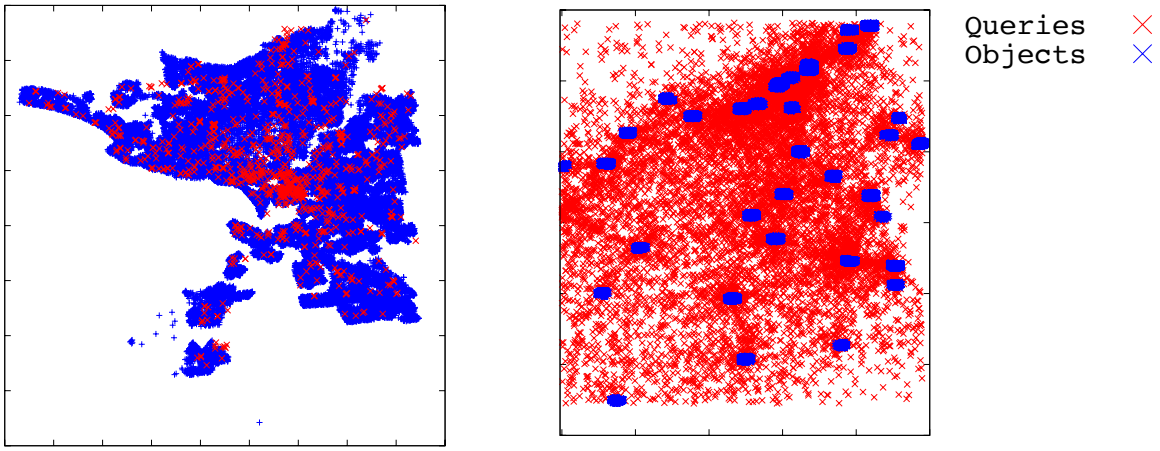
5.6.1 Experiment Settings

All algorithms were implemented in C++. Experiments were ran on a 24 core Intel Xeon $E5-2630$ 2.3 GHz using 256 GB RAM, and 1TB 6G SAS 7.2K rpm SFF (2.5-inch) SC Midline disk drives. As queries may arrive as a high rate, all index structures are memory resident.

Datasets and query generation. All experiments were conducted using two real datasets, (i) Melb dataset at a city scale and (ii) Foursq[2] dataset at a country scale.

The Melb dataset contains $52,913$ real estate properties sold in Melbourne in 2013-2015, collected from the real estate advertising site[3]. As a property can be sold multiple times over the period, only the first sale was retained in the dataset. The locations of the queries in the Melb dataset were created by using locations of 987 facilities (train stations, schools, hospitals, supermarkets, and shopping centers) in this city. We generated two sets of queries from these locations, each of size $20K$. Repeating queries were created using two different approaches: (i) uniform (U); and (ii) skewed (S) distribution, respectively. Each query is equally likely to be repeated for the uniform distribution, thus the distribution of the query set upsized in the first method is likely to follow the same distribution of the original query set. As some queries are more likely to be repeated for the skewed distribution, the second set of upsized queries represents the scenario where some locations (e.g., train stations) are highly preferred than the others. The radius of the queries are varied as an experimental parameter, and is discussed further in Section 5.6.2.

[2]https://sites.google.com/site/yangdingqi/home/foursquare-dataset
[3]http://www.realestate.com.au



(a) Melb (b) Foursq

Figure 5.4: Dataset and query locations

The Foursq dataset contains 304, 133 points of interest (POI) from Foursquare[4] in 34 cities spread across USA. The queries for the Foursq dataset were generated using the user check-ins. From the check-ins of each user, we generated a query, where the query location was the centroid of all the check-ins of that user, and the query radius was set as the minimum distance that covers these check-ins. If a user has only one check-in record, we set the query location as the check-in location, and the radius of the query is randomly assigned from another user. As a result, a total of 22, 442 queries were generated for the Foursq dataset.

Since Melb dataset represents a real city-level data and queries are real facilities in that city, it is the best candidate for an effectiveness study; as a result we conducted efficiency and effectiveness study on Melb. Since the queries generated for Foursq spread over a much wider area, it is more suitable for the efficiency and scalability study[5]. Nonetheless, we also used Foursq to validate our effectiveness study.

After initializing the sliding window, we evaluated the performance of both BS and AP approaches for $10K$ query arrivals in the stream, i.e., $10K$ shifts of the sliding window. We repeated the process 50 times for different sets of queries of the same parameter setting, and report the mean performance. For the Melb dataset, the arrival order of the queries was

[4]https://foursquare.com

[5]Note that, in reality one seldom issues a spatial range query while the candidates are objects spread over the whole big country

Table 5.3: Index construction time in minutes

ϵ	**Melb**	**Foursq**
1	184	1201
2	176	1096
3	130	971
4	67	511
5	65	487

Table 5.4: Parameters

Parameter	**Description**	**Range**
$\mathbb{W}$	Window size	$100, 200, \mathbf{400}, 800, 1600$
m	Number of results to return	$1, 5, \mathbf{10}, 20, 50, 100$
Query radius	Query radius as percentage of the dataspace	$1, 2, \mathbf{4}, 8, 16$
ϵ	Threshold of the cell size	$1, 2, \mathbf{3}, 4, 5$
B	Block size	$32, 64, \mathbf{128}, 256$

randomly generated. For the Foursq dataset, the arrival order of a query in the stream was obtained from the most recent check-in time of the corresponding user. Figure 3.8 shows the location distribution of the objects, and the queries for both datasets, where the blue and the red points represent object locations and query locations respectively. Note that, for the Foursq dataset, the POIs are clustered in large cities (i.e., blue clusters). As a user may check-in in different cities, the queries (which are the centroid of the check-in locations) are distributed in different locations across USA.

The index construction time for different values of ϵ for both datasets are shown in Table 5.3. As more partitioning of the space is required for a smaller ϵ, the index construction time is higher for lower values of ϵ.

Evaluation and Parameterization. We studied the efficiency, scalability and effectiveness for both the baseline approach (BS), and the approximate approach (AP) by varying several parameters. The parameter of interest and their ranges are listed in Table 5.4, where the values in bold represent the default values. For all experiments, a single parameter varied while keeping the rest as the default settings. For efficiency and scalability, we studied the impact of each parameter on: (i) the number of objects whose popularity are computed per query (OPQ), to update the answer of T$m\rho$Q; and (ii) the mean runtime per query (MRPQ), which is the mean value of 50 runs.

In order to measure the *effectiveness* of our approximate approach, the impact of each parameter on the following two metrics are studied:

1. **Approximation ratio**: For a window $\mathbb{W}$, for each $o_i \in R$, $o'_i \in \hat{R}$, where i is the corresponding position of the object in the top-m results, we compute the approximation ratio as -

$$ratio = max\left(\frac{\hat{\rho}(o'_i, \mathbb{W})}{\rho(o_i, \mathbb{W})}, \frac{\rho(o_i, \mathbb{W})}{\hat{\rho}(o'_i, \mathbb{W})}\right)$$

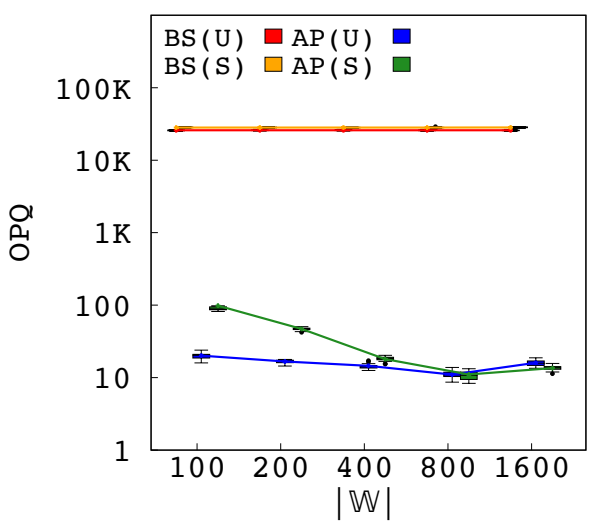
   We report the average approximation ratio of the sliding window by varying different parameters. As the approximate popularity of an object is an aggregation over the estimated ranks, the approximation ratio may not be "1" (the best approximation ratio) even if the approximate result object list $\hat{R}$ is exactly the same as that result list returned by the baseline (where the ratio is calculated as the ratio between the exact and the approximate popularity). Therefore, we present the following metric to demonstrate the similarity of the approximate result object lists with the baseline.

2. **Percentage of result overlap**: For a window $\mathbb{W}$, let $|R| = |O|$, where R is the sorted list of all of the objects according to their exact popularity. We report the similarity between the result list returned by the approximate approach, $\hat{R}$ with R at different depths. Specifically, for each result object $o'_i \in \hat{R}$, where $|\hat{R}| = m$, we record the percentage of objects in $\hat{R}$, overlapping with the top-k objects of R, where k is varied from 10 to 200. For instance, when $m = 50$, we compute how many objects in the top-50 approximate result that also appear in the top-50, top-75, ..., top-150 exact results. We report the percentage of the shared objects for different choices of k, averaged by $10,000$ shifts of the sliding window.

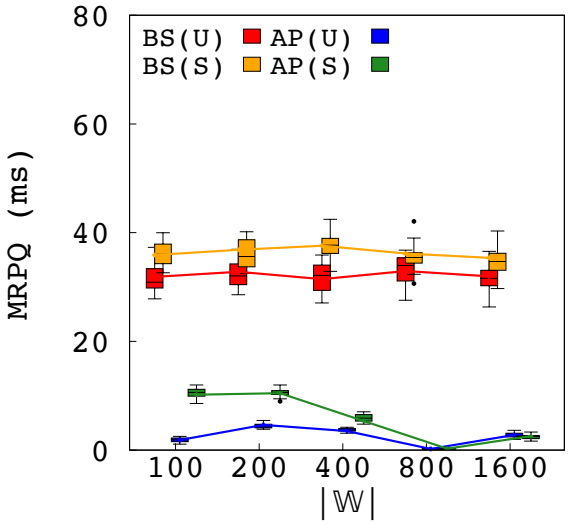
5.6.2 Efficiency and Scalability Evaluation

Varying $|\mathbb{W}|$**.** Figure 5.5 and Figure 5.6 show the impact of varying the number of queries in the sliding window, $|\mathbb{W}|$, for Melb and Foursq, respectively. For Melb, the experiments were conducted using uniform and skewed query sets, while the Foursq query set is derived directly from user check-ins.

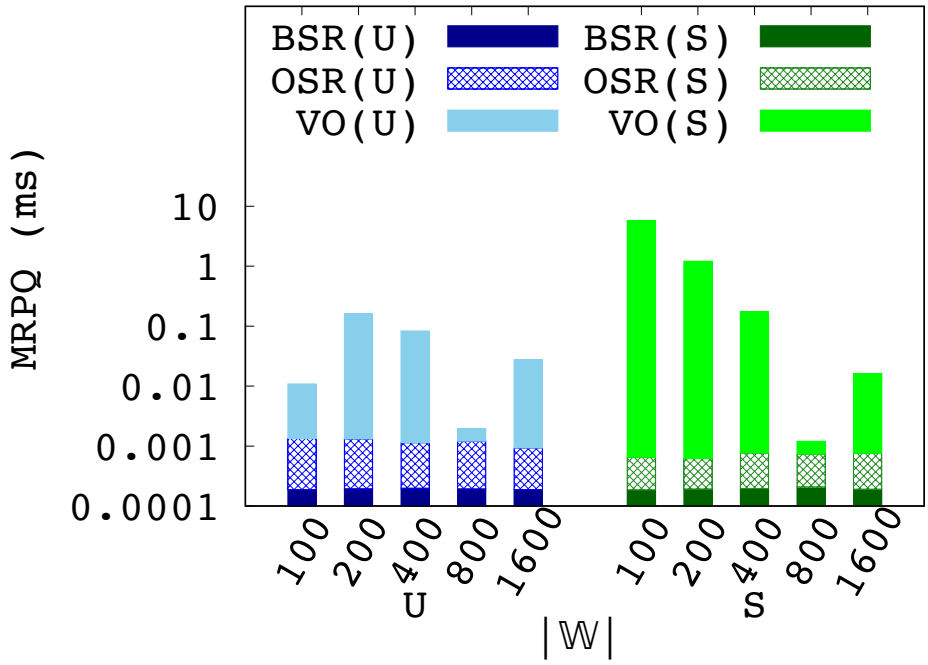
For both datasets, the number of popularity computations required by the approximate approach is about 3 orders of magnitude less than the baseline. The reason is two-fold: (i) In the approximate approach, we compute the popularity of only the objects necessary to update the result. If the result objects of the previous window are found as valid, we do not need to



(a) Objects computed

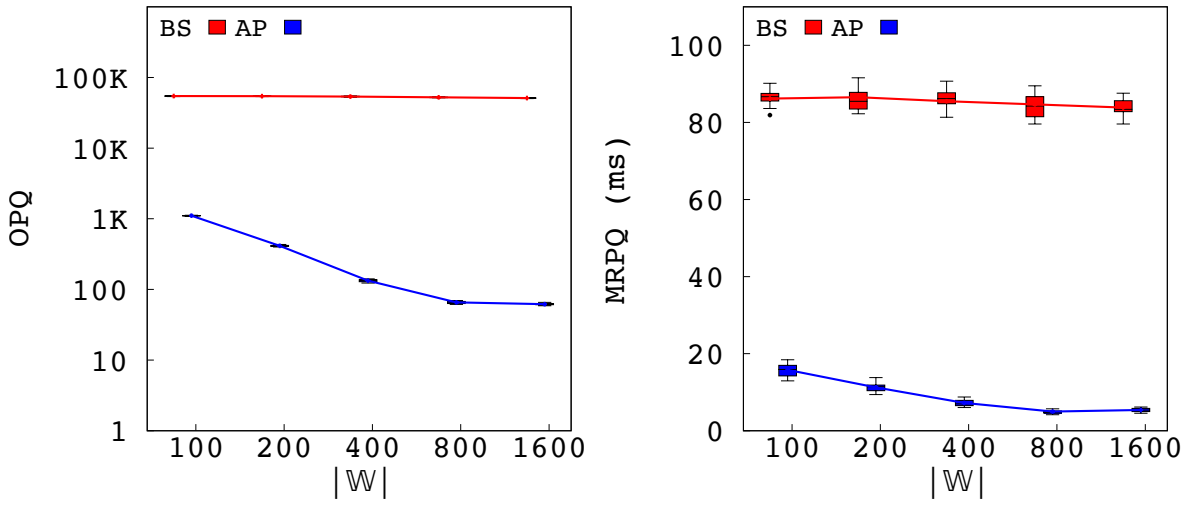


(b) Runtime



(c) Breakdown of runtime

Figure 5.5: Effect of varying |𝕎| on Melb dataset

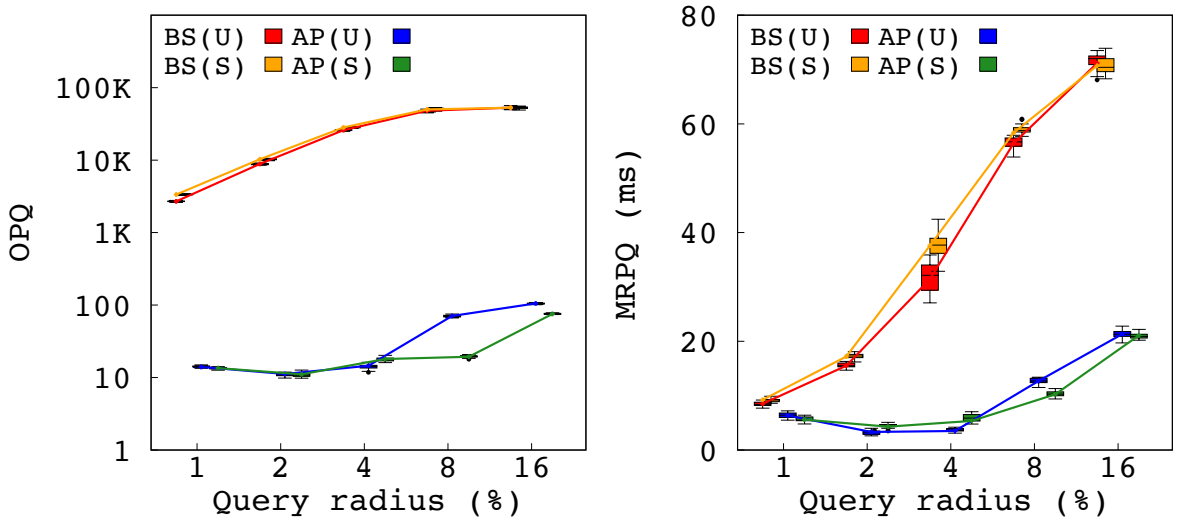


(a) Runtime (b) Objects computed

Figure 5.6: Effect of varying $|\mathbb{W}|$ on Foursq dataset

compute the popularity of any additional object. In contrast, the baseline solution must update the popularity for all of the objects that satisfy the query constraint. (ii) Since the popularity function is an average aggregation (see Section 5.3), the popularity of an object usually does not change drastically as $|\mathbb{W}|$ increases. Therefore, the result objects in a window are more likely to stay valid in subsequent windows for larger values of $|\mathbb{W}|$, thereby requiring even fewer objects being checked. As shown in Figure 5.5b and Figure 5.6b, fewer popularity computation directly translates to lower running time.

In Melb, the performance in both uniform and skewed query sets improves as $|\mathbb{W}|$ increases, but drops slightly from $|\mathbb{W}| = 800$ to $|\mathbb{W}| = 1600$ for the approximate approach. The reason is explained with Figure 5.5c, where the total runtime is shown as a breakdown of the computation time for (i) block-level safe rank, (ii) object-level safe rank, and (iii) validation object computation for both uniform and skewed query sets. If the results are *not valid* for a window, we need to check the validation objects, which is a subset of the objects that satisfy the constraint of at least one query in the current window. As the results are likely to update less often for larger $|\mathbb{W}|$, the time for validation object computation decreases from $\mathbb{W} = 100$ to 800. But when an update in the results is inevitable, more objects are required to be checked for a larger $|\mathbb{W}|$. Here, that scenario occurs for $|\mathbb{W}| = 1600$.



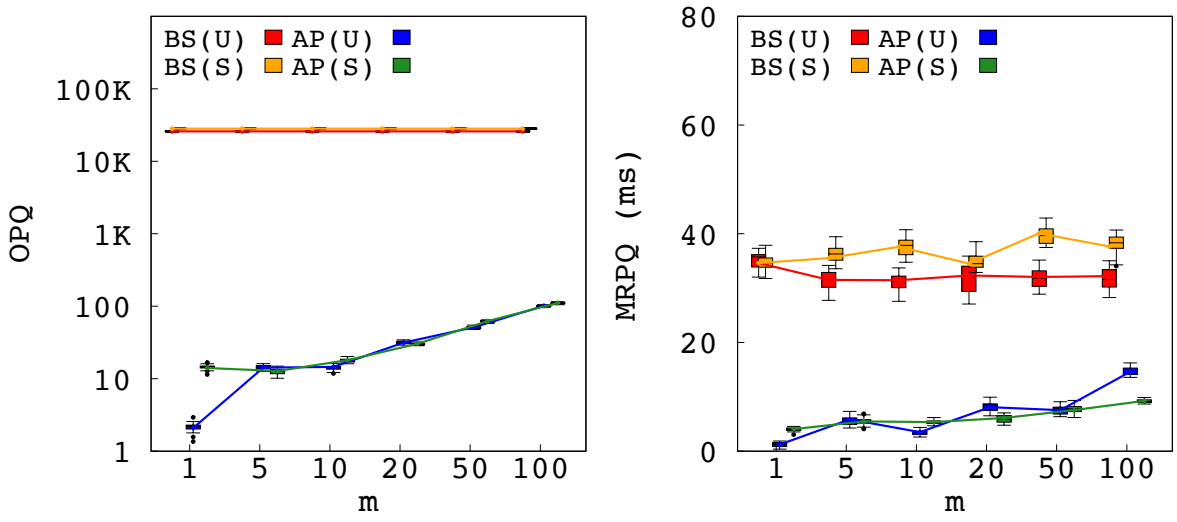
(a) Objects computed

(b) Runtime

Figure 5.7: Effect of varying query radius on Melb dataset

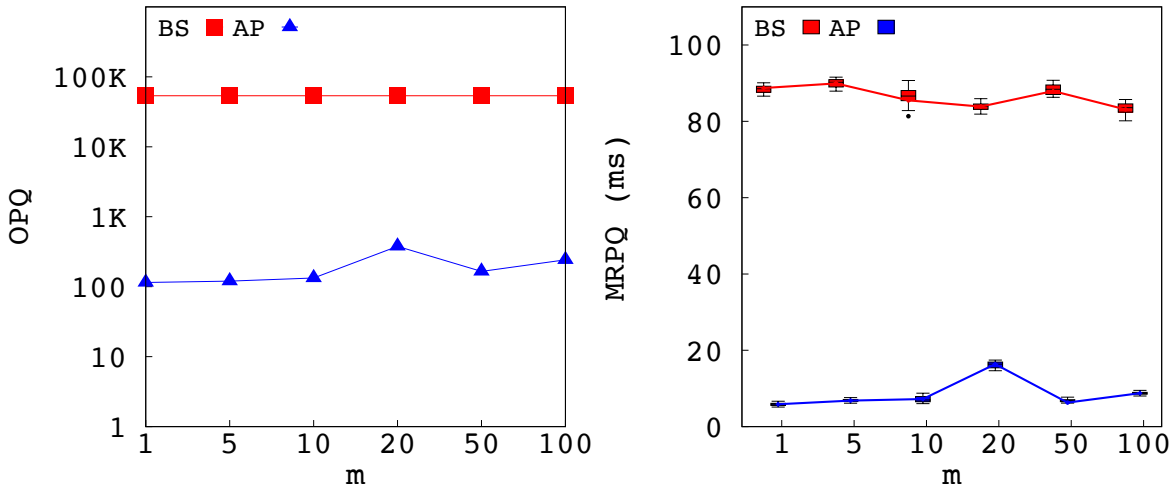
Varying query range. Figure 5.7 shows the performance when varying the radius of each query as a percentage of the dataspace. We vary the query radius only for Melb, as we use the radius that covers the check-in locations of a user as the query radius in the Foursq dataset. Here, the number of objects that fall into the query range grows as query radius increases. Therefore, the performance of the baseline declines rapidly when the query radius increases. In contrast, the approximate approach computes the popularity of only the objects that can be a result, which is a subset of the objects that fall within the query range. Thus, the approximate approach outperforms the baseline, and the benefit is more significant as the query radius increases.

Varying m***.*** The experimental results when varying the number of result objects, m, are shown in Figure 5.8 and Figure 5.9 for Melb and Foursq, respectively. Here, the performance of the baseline does not vary much, as the baseline computes the popularity for all of the objects that fall within the query range regardless of the value of m. The approximate approach outperforms the baseline, because the approximate approach considers only the objects that can potentially be in the top-m results. As more objects qualify to be a result, the performance of the approximate approach as m increases. Both the runtime and the objects checked per query show very little variance for the 50 repeated runs with the different set of queries of the same setting. Therefore, we show graphs with only the mean values for the rest of this section.



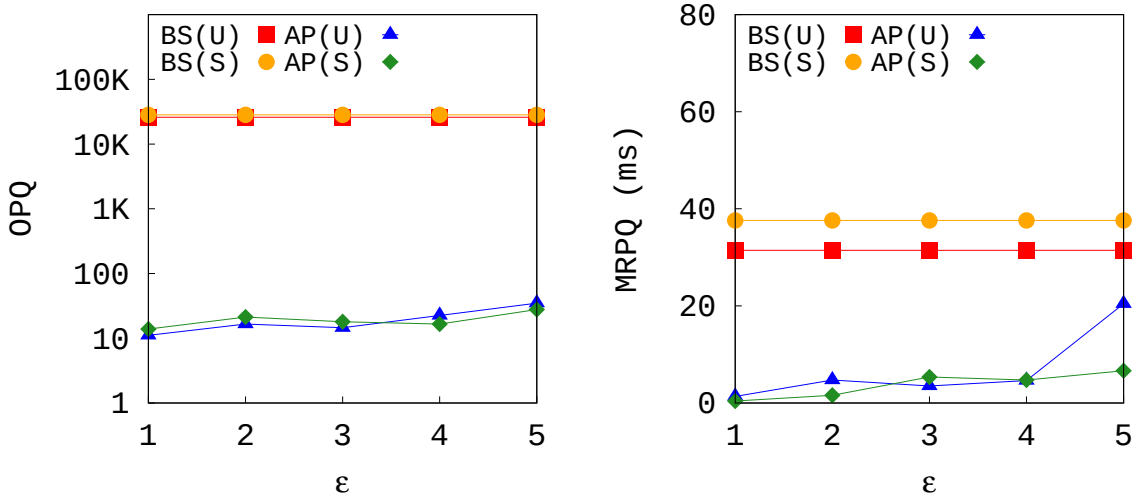
(a) Objects computed (b) Runtime

Figure 5.8: Effect of varying m on Melb dataset



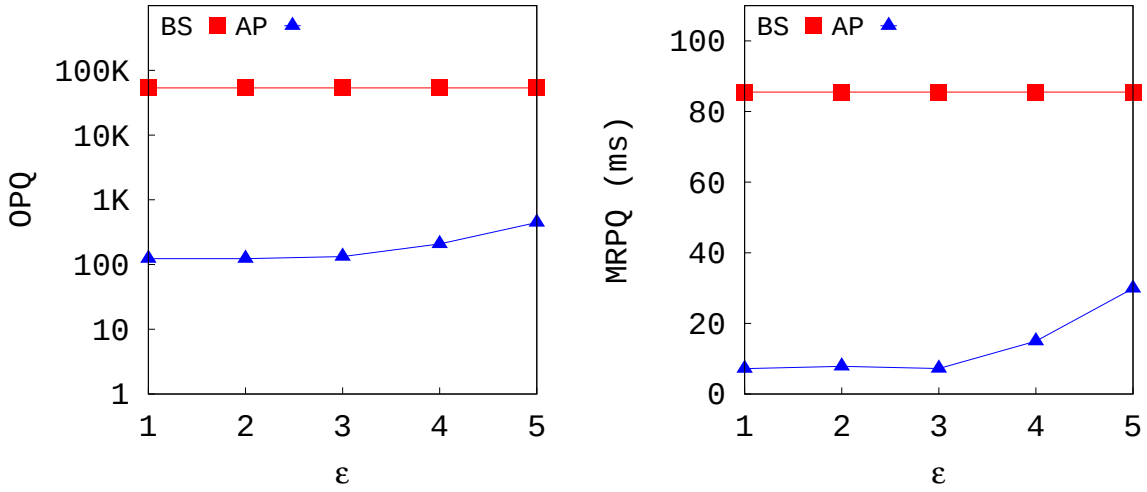
(a) Objects computed (b) Runtime

Figure 5.9: Effect of varying m on Foursq dataset



(a) Objects computed (b) Runtime

Figure 5.10: Effect of varying ϵ on Melb dataset



(a) Objects computed (b) Runtime

Figure 5.11: Effect of varying ϵ on Foursq dataset

Varying ϵ***.*** Figure 5.10 shows the performance of the approaches when varying the approximation parameter ϵ for Melb dataset. The approximate approach consistently outperforms the baseline for all choices of ϵ. As the rank of an object is more accurately approximated for a smaller value of ϵ, it leads to checking fewer number of objects and a lower runtime. As a result, the performance of the approximate approach gradually decreases with the increase of ϵ.

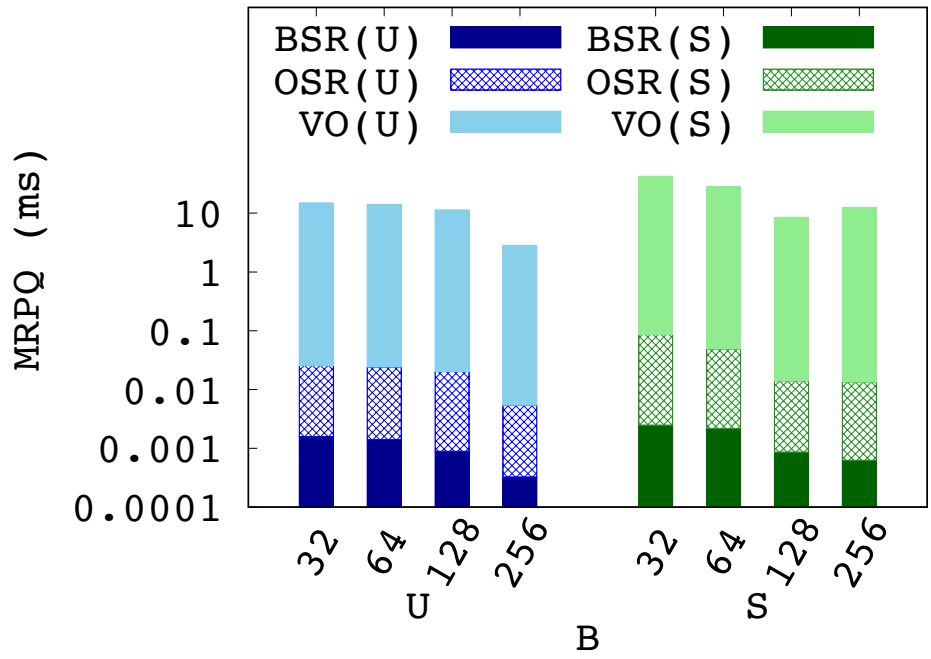


Figure 5.12: Effect of varying B on Melb dataset

Varying B***.*** By varying the block size B of the rank lists, we find that the number of objects to check does not vary with B. If the result of a window needs to be updated, the same set of validation objects are retrieved regardless of the rank list block size. So, we only show the runtime for varying B in Figure 5.12. For each B, the total runtime is shown as a breakdown of the computation time for (i) block-level safe rank, (ii) object-level safe rank, and (iii) validation object computation for both uniform and skewed query sets. From Figure 5.12 we can conclude that:

- As the total number of blocks decreases for higher B, the time required to compute the block-level safe rank also decreases; and
- The validation object lookups dominate the computational costs of the approximate solution.

5.6.3 Effectiveness Evaluation

Varying $|\mathbb{W}|$***.*** Table 5.5 shows the average approximation ratio for both datasets. As popularity is an average aggregation of $|\mathbb{W}|$ ranks, the difference between the approximate and the

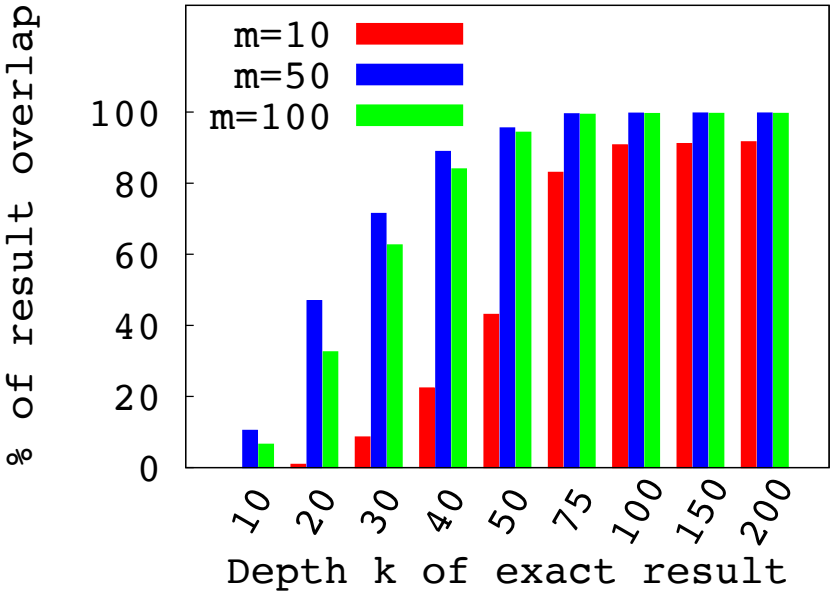


Figure 5.13: Percentage of result overlap for varying m in Melb dataset

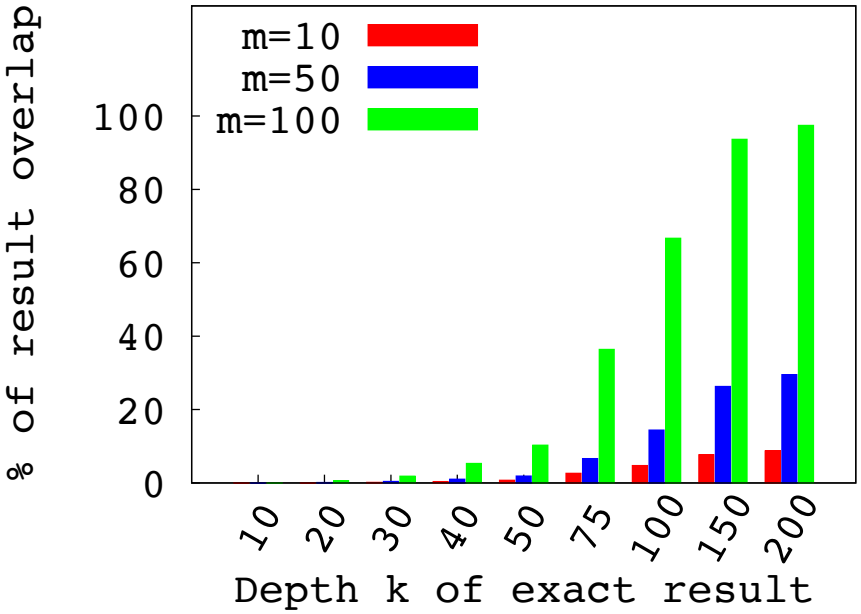


Figure 5.14: Percentage of result overlap for varying m in Foursq dataset

Table 5.5: Approximation ratio for varying $|\mathbb{W}|$

Dataset	$\|\mathbb{W}\|$	100	200	400	800	1600
Melb	U	2.12	1.60	1.57	1.67	1.55
	S	3.19	1.55	2.14	1.30	1.34
Foursq		2.76	6.87	3.49	3.15	2.33

126.3	119.1	125
126.5	120.5	126.3
127.1	121.8	127.1
129.2	122.7	128.9
129.2	123.6	129.6
131.8	124.4	130.2
132.3	125.9	131.3
202.5	194.8	201
203.3	195.1	207.1
204.9	201.7	210.5

(a)

50495.6	51576.2	51983.7
50495.3	51575.9	51983.5
50495.1	51575.6	51983.2
50494.8	51575.4	51982.9
50494.6	51575.1	51982.7
50494.3	51574.9	51982.4
50494	51574.6	51982.2
50493.8	51574.4	51981.9
50493.5	51574.1	51981.7
50493.3	51573.9	51981.4

(b)

Figure 5.15: Popularity values of top-10 objects in (a) Melb dataset, (b) Foursq dataset

exact rank of an object is more likely to decrease of a higher value of $\mathbb{W}$, thus the ratio gradually improves (becomes closer to '1') as $|\mathbb{W}|$ increases. However, the change does not follow any obvious pattern. The explanation for this random behaviour is that, the approximation ratio is the ratio between the exact and the approximate popularity, so if both popularity values do not change at the same rate with $|\mathbb{W}|$, their ratios do not change in the same way.

Varying m**.** As shown in Figure 5.4a, the query locations originally follow a skewed distribution, and most of the query locations are clustered in a small area (which is the central business district of that city), while the rest of the queries are scattered regionally for Melb dataset. In the uniform query set, the queries are repeated uniformly, thus the upsized query set also follows the same (skewed) distribution of the original query set. For this reason, we evaluated our effectiveness as a percentage of result overlap (between the top-m approximate results and the top-k exact results) when using the uniformly upsized query set to capture a more realistic scenario.

The percentage of result overlap between the top-m approximate results and the top-k exact results for Melb dataset are shown in Figure 5.13, where k ranges from 10 to 200 and we set three choices of m (10, 50, 100). As k increases, the overlap percentage also increases. For

Table 5.6: Approximation ratio for varying m

Dataset \ m		1	5	10	20	50	100
Melb	U	3.00	4.79	1.57	1.56	1.49	1.49
	S	5.61	2.03	2.14	1.60	1.17	1.16
Foursq		1.57	2.62	2.68	3.31	3.37	2.47

$m = 50$ and 100, the overlap percentage quickly reaches 90% when $k = 50$. Note that, if multiple objects have the same popularity value, we treat their rank position in the result as equivalent.

Figure 5.14 shows the overlap percentage for the Foursq dataset. Although the overlap approaches 100% for higher m, the overlap is not as high as in the Melb dataset for lower m. The reason is as follows. As shown in Figure 3.8 the objects in Foursq are clustered into cities, where the query locations are distributed all over the dataspace. Therefore, the popularity values of most of the objects in a city are very close to each other. Figure 5.15a shows a screenshot of the top-10 popularities computed in the baseline at three example instances for Foursq dataset. As we can see, *the final rank of two objects can be very far away for a slight difference in their popularity values*; for example, in the first instance, the difference between every adjacent objects' popularity is only 0.25 on average, while the absolute values are at the scale of 50K. In contrast, the screenshot of the top-10 popularity values returned by the baseline for Melb dataset in Figure 5.15a shows that the absolute values are at the scale of a few hundreds, and the difference between the adjacent objects' popularity is higher for Melb dataset than Foursq dataset.

Table 5.6 shows the approximation ratio when varying m in both datasets. We find that the approximation ratio keeps improving as m increases, since most of the objects in the top-m ranked list have very similar scores in both their exact and approximate popularity when $m < 100$. For Foursq dataset, the ratio slightly decreases at $m = 100$. This behaviour can be explained in the same way as mentioned for varying $\mathbb{W}$, that is, as the approximation ratio is the ratio between two popularities, so if both popularity values do not change at the same rate with m, their ratios do not change in the same way.

Varying query range. The approximation ratio of the results w.r.t. varying query ranges is shown in Table 5.7. The approximation ratio does not indicate any obvious patterns since the approximation computation does not depend on the query radius, or the number of objects falling in that range. The approximation ratio is higher for skewed distribution of queries. The

Table 5.7: Approximation ratio for varying query radius

Dataset	Query radius	1	2	4	8	16
Melb	U	2.55	1.55	1.57	1.61	1.63
	S	3.32	2.88	2.14	2.39	3.55

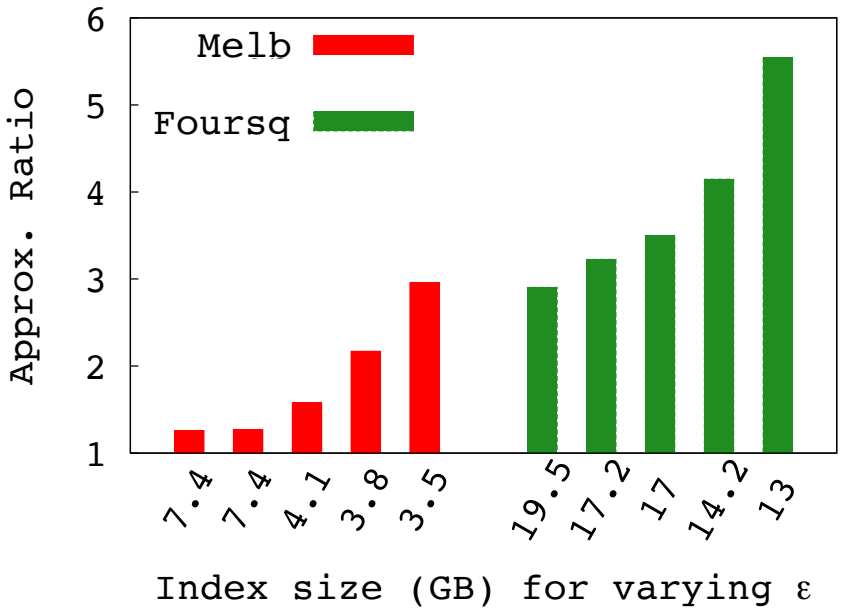


Figure 5.16: Index size vs. approximation ratio for varying ϵ

reason is as follows. For skewed distribution of queries, the ranks of an object for most of the queries in the window are likely to be close to each other. Therefore, the difference between the exact and the approximate popularity of a result object (which is likely to be ranked high for many queries in the window) can be large for a slight difference between its exact and approximate rank.

Varying ϵ, space vs. effectiveness tradeoff. Figure 5.16 shows the tradeoff between the space requirement and the effectiveness as approximation ratio for varying ϵ. Here, the x-axis represents the index size in GB for both datasets, where ϵ is varied from 1 to 5. Since the approximate popularity of an object becomes closer to the exact popularity as ϵ decreases, the approximation ratio also improves for smaller ϵ.

5.6.4 Summary

This chapter has shown that the approximate method significantly outperforms the exact method in terms of efficiency for all parameter settings in both datasets. The approximate method is about 8 times faster, and requires to check about 3 orders of magnitude less number of objects than the exact method. The approximation quality depends on different factors. For Melb dataset, where both the objects and the queries are clustered in a large city, the experiments have shown a high approximation quality (the approximate results have 90% overlap with the exact solution for m higher than 50). The approximate approach may not have shown a notable effectiveness for Foursq dataset, where the query locations are scattered in the dataspace, but the absolute popularity values of the results returned by the approximate method are found to be very close to the results returned by the exact method. The approximation quality for both datasets is higher for smaller values of ϵ, at the cost of a higher space requirement and higher construction time of the index.

5.7 Conclusion

In this chapter, we proposed the problem of top-m rank aggregation of spatial objects for streaming queries. This chapter combines three important problem domains (rank aggregation, continuous queries and spatial databases) into a single context. We have showed how to bound the rank of an object for any unseen query, and proposed an exact solution for the problem. We proposed an approximate solution with a guaranteed error bound, in which we presented *safe ranking* to determine whether the current result is still valid or not when new queries arrive, and *validation objects* to limit the number of objects to update in the top-m results. We conducted a series of experiments on two real datasets, and show that the approximate approach is about 3 orders of magnitude more efficient than the exact solution on the datasets, and the results returned by the approximate approach have more than a 90% overlap with the exact solution for m higher than 50.

Chapter 6

Conclusions and Future Work

This thesis presents novel approaches to process multiple queries on spatial and spatial-textual data. In contrast to the traditional setting, where the queries are processed independent of each other, we devise techniques to solve the problems that require processing multiple queries together. The focus of our proposed approaches is to share the common computations and the I/O operations among multiple queries to improve the overall efficiency. Specifically, we address three different research gaps outlined in Table 2.5. The key insights and contributions of the thesis are summarized below:

In our first study we investigate the performance of our proposed index structures that prioritize different data properties and show the results to batch process multiple spatial-textual top-k queries. As we have focused on spatial-textual data, the index can be constructed by either prioritizing the spatial properties, and/or the textual properties. Specifically, we present three different traversal techniques on different index structures to exploit the similarities between queries in a batch, (i) SF-Sep on IR-tree, (ii) SF-Grp on MIR-tree, and (iii) TF-MBW on SIF index. 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 for this problem.

We address the maximized reverse kNN problem for spatial-textual data as our second study. We show the problem is NP-hard, and present an approximate approach to determine the optimal result. The approximate approach is shown to be around $2-3$ orders of magnitude faster than a baseline in our experimental studies. As human visibility plays an important role in the target applications of this problem, we extend the proposed solutions for visibility as the spatial similarity metric. In this problem, given two sets of data, we present three

different techniques that solve the problem by indexing either or both of the sets of data. In particular, (i) the GRP-TOPK approach groups the set of users and index the set of objects with an MIR-tree, (ii) the INDIV-U approach only indexes the objects but does not apply any indexing on the users, and (iii) the INDEX-U maintains an additional index for the users.

Finally, the problem of rank aggregation is introduced and addressed for a sliding window of streaming spatial queries. This work draws inspiration from the three classical problem domains - rank aggregation, continuous query, and spatial databases. Here, we propose an approximation algorithm with bounded error guarantees to maximize the reuse of the computations in the stream, and incrementally update the results only when necessary. In particular, the following three technical contributions have been made: (i) we propose the notion of 'safe rank' to determine whether the current result set is valid or not for a change in the sliding window of queries; (ii) we propose the notion of 'validation objects' to limit the number of objects to be checked while updating the result set; and (iii) we present a new index structure, an Inverted Rank File (IRF) to answer the problem and show how the index can be used to bound the error of the solution.

Future directions. While the processing of spatial and spatial-textual queries has received considerable attention, there are many opportunities for further research. In the following, we highlight some of the promising directions.

1. **Query processing on dynamic data.** As shown in Table 2.5, there are still many research gaps for processing both individual and multiple queries on dynamic data. For example, (i) The spatial-textual moving reverse kNN query is not yet studied in literature. An example application of the query is in online games, where a moving player can monitor the reverse k nearest neighbors matching her preferences to form teams. The computations can be shared for multiple players to improve efficiency; (ii) As users check-in to new locations in social media, the users can be considered as streaming queries. The problem of continuously updating the optimal set of keywords based on maximized reverse kNNs can be explored, where an example application is to update a digital display board of advertisement for these streaming users.

2. **Processing multiple complex queries.** There have been many studies on different complex types of spatial and spatial-textual queries. For example, collective keyword search [10] returns a set of objects that collectively covers the query keywords and located in a close-by space, prestige queries [9] takes the presence of nearby objects into account

that are also relevant to a query in the ranking of an object, diversified search [140] considers both the relevance and the diversity of the results. However, the batch processing of queries and other problems regarding multiple queries have been studied only for the basic query types (range, kNN, RkNN, etc.) so far, and devising the techniques to apply to the batch of complex queries remains largely unexplored.

3. **Batch process of heterogenous queries.** The current research work on batch processing focus on homogeneous queries, i.e., the same type of queries are grouped together for processing. As the main goal of batch processing is to share the computations among queries, the grouping of different types of queries where the sharing can be improved is an interesting direction. For example, if the number of answers of a range query is equal or greater than k, the answer of a top-k query with the same location can be answered from the results of the range query.

4. **Batch process and rank aggregation of queries on complex data.** Most of the existing studies on spatial queries consider both the objects and the queries as point locations. However, in real-world applications objects and queries may have non-point geometries such as lines, multi-lines, polygons, etc. Another common assumption in the spatial-textual databases is that an object is associated with only one location, where this may not be true for a news article referring to multiple locations, a social media post, or a photo album. In addition, any real-life data is usually associated with several attributes other than location and text, for example, time, social connection, rating, etc., which provides rich semantic information. Although the temporal and social dimensions are studied in the literature [79, 112], the batch processing of the queries on these complex data types are yet to be explored. We have addressed rank aggregation on spatial objects only, where exploring the problem for spatial-textual and other types of data can have different challenges.

5. **Batch process of queries with different k values.** As the approaches proposed in Chapter 3 group the queries together to traverse the index, the solutions are not easily extendable to answer the queries with different k values. However, based on the application, the number of required results can be different for different users. Extending the solutions for different k values is an intuitive improvement that can be explored as a future work.

6. **Multiple query processing with in memory data structures.** The issue of should the data structures be Òon diskÓ or Òin memoryÓ is a hotly debated topic in academia and industry these days. There are advantages and disadvantages to both approaches. Assuming everything is entirely in memory could lead to entirely new data structures, and almost certainly different trade-offs in the ones explored in this thesis. Exploring multiple query processing on spatial and spatial-textual data is an interesting direction for future work.

Bibliography

[1] E. Achtert, H.-P. Kriegel, P. Kröger, M. Renz, and A. Züfle. Reverse k-nearest neighbor search in dynamic and general metric databases. In *EDBT*, pages 886–897, 2009. Cited on pages 28 and 29.

[2] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: ranking and clustering. *JACM*, 55(5):23, 2008. Cited on page 135.

[3] J. J. Bartholdi, C. A. Tovey, and M. A. Trick. The computational difficulty of manipulating an election. *Social Choice and Welfare*, 6(3):227–241, 1989. Cited on page 139.

[4] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *SIGMOD*, pages 322–331, 1990. Cited on page 17.

[5] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975. Cited on page 17.

[6] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970. Cited on page 20.

[7] C. Bohm, B. C. Ooi, C. Plant, and Y. Yan. Efficiently processing continuous k-NN queries on data streams. In *ICDE*, pages 156–165, 2007. Cited on pages 29, 137, and 140.

[8] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In *CIKM*, pages 426–434, 2003. Cited on pages 21, 27, and 59.

[9] X. Cao, G. Cong, and C. S. Jensen. Retrieving top-k prestige-based relevant spatial web objects. *PVLDB*, 3(1-2):373–384, 2010. Cited on page 178.

[10] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective spatial keyword querying. In *SIGMOD*, pages 373–384, 2011. Cited on page 178.

[11] K. Chakrabarti, S. Chaudhuri, and V. Ganti. Interval-based pruning for top-k processing over compressed lists. In *ICDE*, pages 709–720, 2011. Cited on page 56.

[12] W. W. Chang and H. J. Schek. A signature access method for the starburst database system. In *VLDB*, pages 145–153, 1989. Cited on page 25.

[13] M. Cheema, W. Zhang, X. Lin, Y. Zhang, and X. Li. Continuous reverse k nearest neighbors queries in Euclidean space and in spatial networks. *VLDB Journal*, 21(1): 69–95, 2012. Cited on pages 137 and 139.

[14] M. A. Cheema, X. Lin, Y. Zhang, W. Wang, and W. Zhang. Lazy updates: an efficient technique to continuously monitoring reverse kNN. *PVLDB*, 2(1):1138–1149, 2009. Cited on page 29.

[15] M. A. Cheema, L. Brankovic, X. Lin, W. Zhang, and W. Wang. Multi-guarded safe zone: An effective technique to monitor moving circular range queries. In *ICDE*, pages 189–200, 2010. Cited on pages 28, 29, and 139.

[16] M. A. Cheema, L. Xuemin, Z. Wenjie, and Z. Ying. Influence zone: Efficiently processing reverse k nearest neighbors queries. In *ICDE*, pages 577–588, 2011. Cited on pages 28 and 29.

[17] M. A. Cheema, W. Zhang, X. Lin, and Y. Zhang. Efficiently processing snapshot and continuous reverse k nearest neighbors queries. *VLDB Journal*, 21(5):703–728, 2012. Cited on page 29.

[18] M. A. Cheema, W. Zhang, X. Lin, Y. Zhang, and X. Li. Continuous reverse k nearest neighbors queries in Euclidean space and in spatial networks. *VLDB Journal*, 21(1): 69–95, 2012. Cited on page 29.

[19] J. Chen, J. Huang, Z. Wen, Z. He, K. Taylor, and R. Zhang. Analysis and evaluation of the top-k most influential location selection query. *Knowledge and Information Systems*, 43(1):181–217, 2015. Cited on page 89.

[20] L. Chen, G. Cong, and X. Cao. An efficient query indexing mechanism for filtering geo-textual data. In *SIGMOD*, pages 749–760, 2013. Cited on pages 29 and 30.

[21] L. Chen, G. Cong, C. S. Jensen, and D. Wu. Spatial keyword query processing: an experimental evaluation. In *PVLDB*, pages 217–228, 2013. Cited on pages 24, 26, 27, and 33.

[22] Y. Chen and J. Patel. Efficient evaluation of all-nearest-neighbor queries. In *ICDE*, pages 1056–1065, 2007. Cited on pages 29 and 80.

[23] L. Chen-Yi, K. Jia-Ling, and A. P. Chen. Determining k-most demanding products with maximum expected number of total customers. *TKDE*, 25(8):1732–1747, 2013. Cited on page 140.

[24] Z. Chengyuan, Z. Ying, Z. Wenjie, and L. Xuemin. Inverted linear quadtree: Efficient top k spatial keyword search. In *ICDE*, pages 901–912, 2013. Cited on pages 22, 27, 28, 29, and 32.

[25] F. M. Choudhury, M. E. Ali, S. Masud, S. Nath, and I. E. Rabban. Scalable visibility color map construction in spatial databases. *Information Systems*, 42:89 – 106, 2014. Cited on pages 90 and 92.

[26] M. Christoforaki, J. He, C. Dimopoulos, A. Markowetz, and T. Suel. Text vs. space: efficient geo-search query processing. In *CIKM*, pages 423–432, 2011. Cited on pages 22, 26, 28, 29, and 54.

[27] D. Comer. Ubiquitous B-tree. *ACM Computing Surveys*, 11(2):121–137, 1979. Cited on page 18.

[28] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB*, 2(1):337–348, 2009. Cited on pages 22, 23, 24, 27, 28, 29, 32, 33, 36, 37, 45, and 81.

[29] G. Cormode and M. Hadjieleftheriou. Finding frequent items in data streams. *PVLDB*, 1(2):1530–1541, 2008. Cited on pages 137 and 139.

[30] P. Deepak and P. M. Deshpande. Efficient RkNN retrieval with arbitrary non-metric similarity measures. *PVLDB*, 3(1-2):1243–1254, 2010. Cited on pages 28 and 29.

[31] P. Dimitris, S. Qiongmao, T. Yufei, and M. Kyriakos. Group nearest neighbor queries. In *ICDE*, pages 301–312, 2004. Cited on page 88.

[32] S. Ding and T. Suel. Faster top-k document retrieval using block-max indexes. In *SIGIR*, pages 993–1002, 2011. Cited on page 56.

[33] S. Ding, J. Attenberg, R. Baeza-Yates, and T. Suel. Batch query processing for web search engines. In *WSDM*, pages 137–146, 2011. Cited on pages 33 and 81.

[34] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW*, pages 613–622, 2001. Cited on pages 135, 136, and 139.

[35] R. Fagin, R. Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *SIGMOD*, pages 301–312, 2003. Cited on pages 135, 136, and 139.

[36] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *Journal of computer and system sciences*, 66(4):614–656, 2003. Cited on pages 135, 137, and 139.

[37] C. Faloutsos and S. Christodoulakis. Signature files: An access method for documents and its analytical performance evaluation. *ACM Transactions on Information Systems*, 2(4):267–288, 1984. Cited on pages 20 and 26.

[38] U. Feige. A threshold of Ln N for approximating set cover. *Journal of the ACM*, 45(4): 634–652, 1998. Cited on page 100.

[39] I. D. Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In *ICDE*, pages 656–665, 2008. Cited on pages 25, 28, and 29.

[40] R. A. Finkel and J. L. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, 1974. Cited on page 16.

[41] Y. Gao, B. Zheng, G. Chen, W. C. Lee, K. C. K. Lee, and Q. Li. Visible reverse k-nearest neighbor query processing in spatial databases. *TKDE*, 21(9):1314–1327, 2009. Cited on page 90.

[42] Y. Gao, J. Yang, G. Chen, B. Zheng, and C. Chen. On efficient obstructed reverse nearest neighbor query processing. In *GIS*, pages 191–200, 2011. Cited on page 90.

[43] Y. Gao, Q. Liu, X. Miao, and J. Yang. Reverse k-nearest neighbor search in the presence of obstacles. *Information Sciences*, 330:274–292, 2016. Cited on page 90.

[44] I. Gargantini. An effective way to represent Quadtrees. *Communications of the ACM*, 25 (12):905–910, 1982. Cited on page 27.

[45] P. Ghaemi, K. Shahabi, J. P. Wilson, and F. Banaei-Kashani. Continuous maximal reverse nearest neighbor query on spatial networks. In *SIGSPATIAL*, pages 61–70, 2012. Cited on pages 29 and 30.

[46] O. Gkorgkas, A. Vlachou, C. Doulkeridis, and K. Nørvåg. Discovering influential data objects over time. In *SSTD*, pages 110–127, 2013. Cited on page 141.

[47] O. Gkorgkas, A. Vlachou, C. Doulkeridis, and K. Nørvåg. Maximizing influence of spatio-textual objects based on keyword selection. In *SSTD*, pages 413–430, 2015. Cited on pages 29, 30, and 89.

[48] B. Goodwin, M. Hopcroft, D. Luu, A. Clemmer, M. Curmei, S. Elnikety, and Y. He. Bitfunnel: Revisiting signatures for search. In *SIGIR*, pages 605–614, 2017. Cited on pages 20 and 25.

[49] J. Guntzer, W. T. Balke, and W. Kiessling. Towards efficient multi-feature queries in heterogeneous environments. In *ITCC*, pages 622–628, 2001. Cited on page 139.

[50] L. Guo, J. Shao, H. Aung, and K.-L. Tan. Efficient continuous top-k spatial keyword queries on road networks. *GeoInformatica*, pages 1–32, 2014. Cited on page 29.

[51] L. Guo, D. Zhang, G. Li, K.-L. Tan, and Z. Bao. Location-aware pub/sub system: When continuous moving queries meet dynamic event streams. In *SIGMOD*, pages 843–857, 2015. Cited on pages 29 and 30.

[52] L. Guohui, L. Yanhong, L. Jianjun, L. Shu, and Y. Fumin. Continuous reverse k nearest neighbor monitoring on moving objects in road networks. *Information Systems*, 35(8): 860–883, 2010. Cited on page 29.

[53] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984. Cited on pages 17, 28, and 29.

[54] M. Hasan, M. A. Cheema, X. Lin, and Y. Zhang. Efficient construction of safe regions for moving kNN queries over dynamic datasets. In *SSTD*, pages 373–379, 2009. Cited on page 139.

[55] M. Hasan, M. A. Cheema, W. Qu, and X. Lin. Efficient algorithms to monitor continuous constrained k nearest neighbor queries. In *DASFAA*, pages 233–249, 2010. Cited on pages 28 and 29.

[56] D. Hilber. Über die stetige Abbildung einer Linie auf ein Flächenstück. *Mathematische Annalen*, 38(3):459–460, 1891. Cited on page 18.

[57] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Transactions on Database Systems*, 24(2):265–318, 1999. Cited on pages 19, 24, 28, and 29.

[58] D. Hochbaum. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, 1997. Cited on page 94.

[59] M. Hong, M. Riedewald, C. Koch, J. Gehrke, and A. Demers. Rule-based multi-query optimization. In *EDBT*, pages 120–131, 2009. Cited on page 80.

[60] J. Huang, Z. Wen, J. Qi, R. Zhang, J. Chen, and Z. He. Top-k most influential locations selection. In *CIKM*, pages 2377–2380, 2011. Cited on page 89.

[61] W. Huang, G. Li, K.-L. Tan, and J. Feng. Efficient safe-region construction for moving top-k spatial keyword queries. In *CIKM*, pages 932–941, 2012. Cited on page 29.

[62] H. Huiqi, L. Yiqun, L. Guoliang, F. Jianhua, and T. Kian-Lee. A location-aware publish/subscribe framework for parameterized spatio-textual subscriptions. In *ICDE*, pages 711–722, 2015. Cited on pages 29 and 30.

[63] J. J. Cardinal and S. Langerman. Min-max-min geometric facility location problems. In *EWCG*, pages 149–152, 2006. Cited on page 88.

[64] Q. Jianzhong, Z. Rui, L. Kulik, D. Lin, and X. Yuan. The min-dist location selection query. In *ICDE*, pages 366–377, 2012. Cited on page 88.

[65] J. M. Kang, M. F. Mokbel, S. Shekhar, X. Tian, and Z. Donghui. Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors. In *ICDE*, pages 806–815, 2007. Cited on page 29.

[66] H. Kido, Y. Yanagisawa, and T. Satoh. An anonymous communication technique using dummies for location-based services. In *ICPS*, pages 88–97, 2005. Cited on page 32.

[67] J.-L. Koh, C.-Y. Lin, and A. P. Chen. Finding k most favorite products based on reverse top-t queries. *PVLDB*, 23(4):541–564, 2014. Cited on page 140.

[68] F. Korn, S. Muthukrishnan, and D. Srivastava. Reverse nearest neighbor aggregates over data streams. In *PVLDB*, pages 814–825, 2002. Cited on pages 29, 137, and 140.

[69] R. K. V. Kothuri, S. Ravada, and D. Abugov. Quadtree and R-tree indexes in Oracle spatial: A comparison using GIS data. In *SIGMOD*, pages 546–557, 2002. Cited on page 17.

[70] N. Koudas, B. C. Ooi, K.-L. Tan, and R. Zhang. Approximate NN queries on streams with guaranteed error/performance bounds. In *VLDB*, pages 804–815, 2004. Cited on page 29.

[71] L. K. Lee and H. F. Ting. A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In *PODS*, pages 290–297, 2006. Cited on page 139.

[72] C. Li, Y. Gu, J. Qi, G. Yu, R. Zhang, and W. Yi. Processing moving kNN queries using influential neighbor sets. *PVLDB*, 8(2):113–124, 2014. Cited on pages 28, 29, 137, 139, and 140.

[73] C.-L. Li, E. T. Wang, G.-J. Huang, and A. L. P. Chen. Top-n query processing in spatial databases considering bi-chromatic reverse k-nearest neighbors. *Information Systems*, 42: 123–138, 2014. Cited on pages 137 and 140.

[74] G. Li, Y. Wang, T. Wang, and J. Feng. Location-aware publish-subscribe. In *SIGKDD*, pages 802–810, 2013. Cited on pages 29 and 30.

[75] M. Li, Z. Bao, T. Sellis, and S. Yan. Visualization-aided exploration of the real estate data. In *ADC*, pages 435–439, 2016. Cited on page 136.

[76] Z. Li, K. C. K. Lee, B. Zheng, W.-C. Lee, D. L. Lee, and X. Wang. IR-tree: An efficient index for geographic document search. *TKDE*, 23(4):585–599, 2011. Cited on pages 22, 24, 28, 29, 32, and 33.

[77] H. Lin, F. Chen, Y. Gao, and D. Lu. OptRegion: Finding optimal region for bichromatic reverse nearest neighbors. In *DASFAA*, pages 146–160, 2013. Cited on pages 29, 30, and 88.

[78] Q. Lin, C. Xiao, M. A. Cheema, and W. Wang. Finding the sites with best accessibilities to amenities. In *DASFAA*, pages 58–72, 2011. Cited on page 89.

[79] C. Lisi, C. Gao, C. Xin, and T. Kian-Lee. Temporal spatial-keyword top-k publish/subscribe. In *ICDE*, pages 255–266, 2015. Cited on pages 29, 30, and 179.

[80] H. Liu, Y. Lin, and J. Han. Methods for mining frequent items in data streams: an overview. *Knowledge and Information Systems*, 26(1):1–30, 2011. Cited on page 139.

[81] Y. Liu, R.-W. Wong, K. Wang, Z. Li, C. Chen, and Z. Chen. A new approach for maximizing bichromatic reverse nearest neighbor search. *Knowledge and Information Systems*, 36(1):23–58, 2013. Cited on pages 29, 30, 83, and 88.

[82] H. Lu, C. S. Jensen, and M. L. Yiu. Pad: Privacy-area aware, dummy-based location privacy in mobile services. In *MobiDE*, pages 16–23, 2008. Cited on page 32.

[83] J. Lu, Y. Lu, and G. Cong. Reverse spatial and textual k nearest neighbor search. In *SIGMOD*, pages 349–360, 2011. Cited on pages 25, 28, 29, and 89.

[84] Y. Lu, J. Lu, G. Cong, W. Wu, and C. Shahabi. Efficient algorithms and cost models for reverse spatial-keyword k-nearest neighbor search. *ACM Transactions on Database Systems*, 39(2):1–46, 2014. Cited on pages 28, 29, and 89.

[85] J. Mackenzie, F. M. Choudhury, and J. S. Culpepper. Efficient location-aware web search. In *ADCS*, pages 1–8, 2015. Cited on pages 27, 28, and 64.

[86] N. Mamoulis, K. H. Cheng, M. L. Yiu, and D. W. Cheung. Efficient aggregation of ranked inputs. In *ICDE*, pages 72–84, 2006. Cited on pages 135, 137, and 139.

[87] N. Mamoulis, M. L. Yiu, K. H. Cheng, and D. W. Cheung. Efficient top-k aggregation of ranked inputs. *ACM Transactions on Database Systems*, 32(3):19, 2007. Cited on page 139.

[88] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. Cited on page 11.

[89] S. Masud, F. M. Choudhury, M. E. Ali, and S. Nutanong. Maximum visibility queries in spatial databases. In *ICDE*, pages 637–648, 2013. Cited on pages 90 and 92.

[90] G. M. Morton. A computer oriented geodetic data base and a new technique in file sequencing. Technical report, IBM Ltd., 1966. Cited on pages 18 and 53.

[91] K. Mouratidis and D. Papadias. Continuous nearest neighbor queries over sliding windows. *TKDE*, 19(6):789–803, 2007. Cited on pages 29 and 140.

[92] K. Mouratidis, D. Papadias, and M. Hadjieleftheriou. Conceptual partitioning: an efficient method for continuous nearest neighbor monitoring. In *SIGMOD*, pages 634–645, 2005. Cited on pages 28, 29, and 140.

[93] K. Mouratidis, S. Bakiras, and D. Papadias. Continuous monitoring of top-k queries over sliding windows. In *SIGMOD*, pages 635–646, 2006. Cited on pages 29, 137, and 140.

[94] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems*, 9(1):38–71, 1984. Cited on page 16.

[95] S. Nutanong, E. Tanin, and R. Zhang. Incremental evaluation of visible nearest neighbor queries. *TKDE*, 22(5):665–681, 2010. Cited on page 90.

[96] S. Nutanong, R. Zhang, E. Tanin, and L. Kulik. Analysis and evaluation of V*-kNN: an efficient algorithm for moving kNN queries. *VLDB Journal*, 19(3):307–332, 2010. Cited on pages 28, 29, and 139.

[97] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. Efficient OLAP operations in spatial data warehouses. In *SSTD*, pages 443–459, 2001. Cited on page 25.

[98] A. N. Papadopoulos and Y. Manolopoulos. Multiple range query optimization in spatial databases. In *ADBIS*, pages 71–82, 1998. Cited on pages 29, 80, and 81.

[99] O. Papapetrou, M. Garofalakis, and A. Deligiannakis. Sketch-based querying of distributed sliding-window data streams. *PVLDB*, 5(10):992–1003, 2012. Cited on pages 137 and 139.

[100] J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *SIGIR*, pages 275–281, 1998. Cited on page 19.

[101] K. Pripužić, I. P. Žarko, and K. Aberer. Top-k/w publish/subscribe: A publish/subscribe model for continuous top-k processing over data streams. *Information Systems*, 39:256 – 276, 2014. Cited on page 140.

[102] J. Qi, Z. Xu, Y. Xue, and Z. Wen. A branch and bound method for min-dist location selection queries. In *ADC*, pages 51–60, 2012. Cited on page 88.

[103] I. E. Rabban, K. Abdullah, M. E. Ali, and M. A. Cheema. Visibility color map for a fixed or moving target in spatial databases. In *SSTD*, pages 197–215, 2015. Cited on page 90.

[104] S. E. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In *TREC*, pages 109–126, 1994. Cited on pages 19 and 20.

[105] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg. Efficient processing of top-k spatial keyword queries. In *SSTD*, pages 205–222, 2011. Cited on pages 22, 25, 27, 28, 29, and 32.

[106] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *SIGMOD*, pages 71–79, 1995. Cited on pages 19, 28, and 29.

[107] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information processing and management*, 24(5):513–523, 1988. Cited on page 35.

[108] SearchEngineLand. Microsoft: 53 percent of mobile searches have local intent. http://searchengineland.com/microsoft-53-percent-of-mobile-searches-havelocal-intent-55556. [Online; accessed 18-07-2017]. Cited on pages 9 and 33.

[109] T. K. Sellis. Multiple-query optimization. *ACM Transactions on Database Systems*, 13 (1):23–52, 1988. Cited on page 80.

[110] T. K. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-tree: A dynamic index for multi-dimensional objects. In *VLDB*, pages 507–518, 1987. Cited on page 17.

[111] S. Shekhar, S. K. Feiner, and W. G. Aref. Spatial computing. *Communications of the ACM*, 59(1):72–81, 2016. Cited on page 136.

[112] A. Shraer, M. Gurevich, M. Fontoura, and V. Josifovski. Top-k publish-subscribe for social annotation of news. *PVLDB*, 6(6):385–396, 2013. Cited on page 179.

[113] Y. Sun, J. Huang, Y. Chen, R. Zhang, and X. Du. Location selection for utility maximization with capacity constraints. In *CIKM*, pages 2154–2158, 2012. Cited on page 89.

[114] Y. Tao and C. Sheng. Fast nearest neighbor search with keywords. *TKDE*, 26(4):878–888, 2014. Cited on pages 22, 27, 28, and 29.

[115] Y. Tao, V. Hristidis, D. Papadias, and Y. Papakonstantinou. Branch-and-bound processing of ranked queries. *Information Systems*, 32(3):424–445, 2007. Cited on page 136.

[116] Y. Tao, D. Papadias, X. Lian, and X. Xiao. Multidimensional reverse kNN search. *VLDB Journal*, 16(3):293–316, 2007. Cited on pages 28 and 29.

[117] ThinkWithGoogle. Understanding consumers' local search behavior. https://www.thinkwithgoogle.com/research-studies/how-advertisers-can-extend-their-relevance-with-search.html, 2014. [Online; accessed 18-07-2017]. Cited on pages 9 and 33.

[118] X. Tian and Z. Donghui. Continuous reverse nearest neighbor monitoring. In *ICDE*, pages 77–77, 2006. Cited on page 29.

[119] H. Turtle and J. Flood. Query evaluation: Strategies and optimizations. *Information Processing and Management*, 31(6):831–850, 1995. Cited on page 20.

[120] S. Vaid, C. Jones, H. Joho, and M. Sanderson. Spatio-textual indexing for geographical search on the web. In *SSTD*, pages 218–235, 2005. Cited on pages 28 and 29.

[121] A. Vlachou, C. Doulkeridis, K. Nørvåg, and Y. Kotidis. Identifying the most influential data objects with reverse top-k queries. *PVLDB*, 3(1-2):364–372, 2010. Cited on page 140.

[122] X. Wang, Y. Zhang, W. Zhang, X. Lin, and W. Wang. Selectivity estimation on streaming spatio-textual data using local correlations. *PVLDB*, 8(2):101–112, 2014. Cited on page 29.

[123] X. Wang, Y. Zhang, W. Zhang, X. Lin, and W. Wang. AP-Tree: efficiently support location-aware publish/subscribe. *VLDB Journal*, 24(6):823–848, 2015. Cited on pages 29 and 30.

[124] X. Wang, Y. Zhang, W. Zhang, X. Lin, and W. Wang. AP-Tree: Efficiently support continuous spatial-keyword queries over stream. In *ICDE*, pages 1107–1118, 2015. Cited on pages 29 and 30.

[125] Y. Wang, Y. Gao, L. Chen, G. Chen, and Q. Li. All-visible-k-nearest-neighbor queries. In *DEXA*, pages 392–407, 2012. Cited on page 90.

[126] R. C.-W. Wong, M. T. Özsu, P. S. Yu, A. W.-C. Fu, and L. Liu. Efficient method for maximizing bichromatic reverse nearest neighbor. *PVLDB*, 2(1):1126–1137, 2009. Cited on pages 29, 30, 83, and 88.

[127] R. C.-W. Wong, M. T. Özsu, A. W.-C. Fu, P. S. Yu, L. Liu, and Y. Liu. Maximizing bichromatic reverse nearest neighbor for lp-norm in two and three-dimensional spaces. *PVLDB*, 20(6):893–919, 2011. Cited on pages 83 and 88.

[128] D. Wu, Y. Man Lung, C. S. Jensen, and G. Cong. Efficient continuously moving top-k spatial keyword query processing. In *ICDE*, pages 541–552, 2011. Cited on page 29.

[129] D. Wu, G. Cong, and C. S. Jensen. A framework for efficient spatial web object retrieval. *PVLDB*, 21(6):797–822, 2012. Cited on pages 22, 23, 24, 28, 29, 32, 33, and 81.

[130] D. Wu, M. L. Yiu, G. Cong, and C. S. Jensen. Joint top-k spatial keyword query processing. *TKDE*, 24(10):1889–1903, 2012. Cited on pages 29, 33, and 81.

[131] D. Wu, M. L. Yiu, and C. S. Jensen. Moving spatial keyword queries: Formulation, methods, and analysis. *ACM Transactions on Database Systems*, 38(1):1–47, 2013. Cited on page 29.

[132] T. Xia, D. Zhang, E. Kanoulas, and Y. Du. On computing top-t most influential spatial sites. In *PVLDB*, pages 946–957, 2005. Cited on pages 89 and 137.

[133] X. Xie, X. Lin, J. Xu, and C. Jensen. Reverse keyword-based location search. In *ICDE*, pages 375–386, 2017. Cited on page 89.

[134] D. Yan, R. C.-W. Wong, and W. Ng. Efficient methods for finding influential locations with adaptive grids. In *CIKM*, pages 1475–1484, 2011. Cited on page 88.

[135] D. Yan, Z. Zhao, and W. Ng. Efficient algorithms for finding optimal meeting point on road networks. *PVLDB*, 4(11):968–979, 2011. Cited on page 88.

[136] D. Yan, Z. Zhao, and W. Ng. Efficient processing of optimal meeting point queries in Euclidean space and road networks. *Knowledge and Information Systems*, 42(2):319–351, 2015. Cited on page 88.

[137] M. Yu, G. Li, and J. Feng. A cost-based method for location-aware publish/subscribe services. In *CIKM*, pages 693–702, 2015. Cited on pages 29 and 30.

[138] L. Zhan, Y. Zhang, W. Zhang, and X. Lin. Finding top k most influential spatial facilities over uncertain objects. In *CIKM*, pages 922–931, 2012. Cited on page 137.

[139] C. Zhang, L. Shou, K. Chen, and G. Chen. See-to-retrieve: efficient processing of spatio-visual keyword queries. In *SIGIR*, pages 681–690, 2012. Cited on pages 90 and 92.

[140] C. Zhang, Y. Zhang, W. Zhang, X. Lin, M. A. Cheema, and X. Wang. Diversified spatial keyword search on road networks. In *EDBT*, pages 367–378, 2014. Cited on page 179.

[141] D. Zhang, Y. Du, T. Xia, and Y. Tao. Progressive computation of the min-dist optimal-location query. In *VLDB*, pages 643–654, 2006. Cited on page 88.

[142] D. Zhang, K.-L. Tan, and A. K. H. Tung. Scalable top-k spatial keyword search. In *EDBT*, pages 359–370, 2013. Cited on pages 22, 26, 28, 29, 32, and 33.

[143] D. Zhang, C.-Y. Chan, and K.-L. Tan. Processing spatial keyword query as a top-k aggregation query. In *SIGIR*, pages 355–364, 2014. Cited on pages 26 and 28.

[144] J. Zhang, N. Mamoulis, D. Papadias, and Y. Tao. All-nearest-neighbors queries in spatial databases. In *SSDBM*, pages 297–306, 2004. Cited on pages 29 and 80.

[145] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.-Y. Ma. Hybrid index structures for location-based web search. In *CIKM*, pages 155–162, 2005. Cited on pages 22, 28, and 29.

[146] Z. Zhou, W. Wu, X. Li, M. L. Lee, and W. Hsu. MaxFirst for MaxBRkNN. In *ICDE*, pages 828–839, 2011. Cited on pages 29, 30, 84, and 88.

[147] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(2):6, 2006. Cited on pages 19 and 20.

[148] J. Zobel, A. Moffat, and K. Ramamohanarao. Inverted files versus signature files for text indexing. *ACM Transactions on Database Systems*, 23(4):453–490, 1998. Cited on pages 20 and 25.