University of Piraeus Department of Digital Systems

MSc Programme of Digital Systems and Services

A study on algorithms for maximizing the influence score of spatio-textual objects

Network-Oriented Information Systems

Stella Maropaki

Athens, March 2016

Πανεπιστημιο Πειραια Τμημα Ψηφιακων Συστηματων

Π.Μ.Σ. Ψηφιακα Συστηματα και Υπηρεσιες

Μελέτη αλγορίθμων για τη μεγιστοποίηση βαθμού επιρροής σε **Spatio-Textual** αντικείμενα

Δικτυοκεντρικά Πληροφοριακά Συστήματα

Στέλλα Μαροπάκη

Αθήνα, Μάρτιος 2016

Abstract

Nowadays, more and more applications are used that manage spatial objects annotated with textual descriptions. Advanced query operators and data indexes have become, not just useful, but indispensable, in order to help users handle the huge amount of available data by answering efficiently in their queries. With these data, users are offered the opportunity to pose spatio-textual queries with their preferences. The results of such a query consists of spatio-textual objects ranked according to their distance from a desired location and to their textual relevance to the query. A problem that arises from this context is how to select a set of at most b keywords to enhance the description of a spatial object, in order to make the object appear in the TOP_k results of as many users as possible. This problem is referred in later work as Best Term and it is proven that it is NP-hard.

In this thesis we study the design and development of an algorithm that approximately solves this problem. The presented algorithm focuses on using efficiently the data structure of an IR-tree index, that is build over the spatio-textual data, in order to compute the b keywords needed. An extended number of experiments will be demonstrated that will show the effectiveness of the proposed algorithm. A comparative performance analysis will be provided for this algorithm and the already introduced algorithms as baselines. As it will be shown by the experimental studies, this algorithm is an efficient solution for the Best Term problem.

Περίληψη

Στις μέρες μας, χρησιμοποιούνται όλο και περισσότερο εφαρμογές που διαχειρίζονται χωρικά αντικείμενα σε συνδυασμό με περιγραφές κειμένου. Ανεπτυγμένοι τύποι ερωτημάτων και ευρετήρια δεδομένων, έχουν γίνει όχι μόνο χρήσιμα, άλλα και σχεδόν απαραίτητα, ώστε να βοηθούν τους χρήστες να χειρίζονται τον μεγάλο όγκο των διαθέσιμων δεδομένων, απαντώντας τους αποτελεσματικά στα ερωτήματά τους. Με αυτά τα δεδομένα, δίνεται στους χρήστες η δυνατότητα να θέσουν spatio-textual ερωτήματα με τις προτιμήσεις τους. Τα αποτελέσματα ενός τέτοιου ερωτήματος, συνιστώνται από spatio-textual αντικείμενα, καταταγμένα ανάλογα την απόστασή τους από μία επιθυμητή τοποθεσία και την λεκτική ομοιότητά τους με το ερώτημα. Ένα πρόβλημα που προκύπτει, είναι το πώς να επιλέξεις το πολύ b λέξεις κλειδιά, ενισχύοντας την περιγραφή ενός χωρικού αντικειμένου, ώστε να εμφανίζεται αυτό στα TOP_k αποτελέσματα, σε όσο το δυνατόν περισσότερους χρήστες. Το πρόβλημα αυτό θα αναφέρεται στο εξής ως Best Term, και αποδεικνύεται ότι είναι NP-hard.

Σε αυτήν την διπλωματική εργασία, μελετάμε τον σχεδιασμό και την ανάπτυξη ενός αλγορίθμου που λύνει προσεγγιστικά αυτό το πρόβλημα. Ο αλγόριθμος που παρουσιάζεται, εστιάζει στο να χρησιμοποιεί αποτελεσματικά την δομή δεδομένων ενός IR-tree,που δημιουργήθηκε από τα spatio-textual δεδομένα, ώστε να υπολογίζει τις b λέξεις που απαιτούνται. Θα παρουσιαστεί ένας εκτενής αριθμός πειραμάτων, που αποδεικνύουν την αποτελεσματικότητα του αλγόριθμου. Επίσης, θα δοθεί μια συγκριτική ανάλυση απόδοσης μεταξύ αυτού του αλγορίθμου και των ήδη υπαρχόντων. Όπως θα φανεί από την μελέτη των πειραματικών αποτελεσμάτων, ο αλγόριθμος αυτός είναι μια αποδοτική λύση για το Best Term πρόβλημα.

Acknowledgements

First of all, I would like to thank my supervisor, Dr. Christos Doulkeridis, for his support, advising and guidance. Also Dr. Kjetil Nørvåg for his help and cooperation during the fulfillment of this thesis.

I would also like to thank Orestis Gkorgkas for his suggestions, help and patience during the implementation of the work.

Last but not least, I would like to thank my family and friends for their support and encouragement.

Stella Maropaki
Athens, March 2016

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Nowadays, numerous applications provide services to the users concerning location and textual relevance, and so spatio-textual search became more and more popular. The object of a spatio-textual query is to retrieve a ranked set of TOP_k spatio-textual objects that are close to the query prefered location and have high textual similarity with the preferred keywords. For example, consider a database that contains information about recreational establishments, such as hotels. Each tuple of the database is represented as a point in a data space annotated with their facilities as keywords. A user query could be a search for hotels in a specific city, or area of interest and a set of facilities included in the hotels, such as "pool" or "restaurant".

1.1 Thesis Contribution

An interesting problem that arises form the context of the spatio-textual TOP_k queries is how to improve the ranking of a spatio-textual object for as many users as possible. This problem is reduced to the problem of how to select a set of at most b keywords to enhance the description of a spatial object, in order to make the object appear in the TOP_k results of as many users as possible. This problem is referred as Best Term problem, and is proven to be NP-hard. This work presents an approximate algorithm to solve the Best Term problem, including a variation of an already existing algorithm. Finally, a comparative experimental analysis is demonstrated, that shows that the proposed algorithms are an efficient solution to the problem and need less processing time and are less resource demanding.

1.2 Thesis Outline

In Chapter 2 we describe the work that has already been made in topics related to this thesis and explicate why this work cannot apply to our problem. Chapter 3 gives a brief introduction about the preliminaries needed for this thesis, such as the TOP_k and the Reverse TOP_k query and their use. Furthermore basic background information about R-tree index and its variations is provided. The problem statement of this thesis is described thoroughly in Chapter 4, where the spatio-textual objects and the Best Term problem are defined. In Chapter 5 two baseline algorithms are described and their advantages and disadvantages are discussed. Chapter 6 explicates the idea of the approach in this thesis, describes the algorithm implemented and defines its complexity. The experimental results of this thesis are presented in Chapter 7 and a conclusion is made in Chapter 8 along with some future work proposals.

Chapter 2

Related Work

In this Chapter the work that has already been introduced and is related to this thesis will be described. A brief description on what algorithms have already been used will be given, and the reason they cannot be used in this work will be explained.

2.1 Keyword Recommendation

Zhang et al. [10] proposed a novel algorithm for advertising keywords recommendation for short-text web pages by leveraging the contents of Wikipedia. Their work is based on the fact, that Wikipedia contains numerous entities that have links to other entities on related topics. More specifically, considering a target web page, they proposed the use of a content-biased Page Rank on the Wikipedia graph, to rank the related entities. In addition to this, they added an advertisement-biased factor in the model, so high-quality advertising keywords could be recommended. The approach on solving the problem of the keywords recommendation consists of two stages. In the first stage, candidate keywords are extracted from the target web page. In the second stage they are using a random walk based algorithm applied to the Wikipedia graph, so related keywords to the target web page are recommended. Given these two facts, advertising keywords that are both relevant to a target web page and valuable for advertising are recommended. As shown by the experimental results, this proposal approach produces substantial improvement in the precision of the top 20 recommended keywords on short-text web pages over existing approaches.

The work of Fuxman et al. [3] is based on the idea, that search engines are paid by businesses that are interested in displaying ads for their site alongside the search results. Given that, their main problem to handle is the keyword generation. Given a business that is interested in launching a campaign, suggest keywords that are related to the specific campaign. For this reason they are taking a different approach than Zhang et al. by suggesting the use of the query logs of the search engine. In order to solve the problem, they identify queries related to a campaign by exploiting the associations between queries and URLs as they captured by the user's click. As a result, these queries are suitable for giving good keyword suggestions, since they capture the "wisdom of the crowd" as to what is related to a site. They produce the desired result by formulating the problem as a semi-supervised learning problem and by proposing algorithms within the Markov Random Field model. The experimental part of this work is based on real query logs and clearly shows the algorithms that being used, scale to large query logs and produce meaningful results with minimal effort. The main advantages of this method are two. First, this method strongly exploits the significance of "wisdom of the crowd" for keyword generation and second, suggested keywords take into account the click-through traffic that they generate in the search engine and as a result they are more directly monetizable.

Ravi et al. [6] aims towards the automatic construction of online ad campaigns. For this purpose they propose a variety of algorithmic methods to generate bid phrases. Their approach consists of two main phases. In the first phase candidate bid phrases are generated by a number of methods. In this phase among other methods it is introduced a (monolingual) translation model capable of generating phrases not contained within the text of the input. In the second phase candidate bid phrases are ranked in a probabilistic framework by using both the translation model as well as a bid phrase language model. In order to achieve the desired results experiments were made with a large collection of existing ads and their landing pages. Every bid phrase associated to a given ad becomes a "labeled" instance of the form (landing page, bid phrase). After that they evaluate their methods based on how well they can predict these bid phrases given the landing page.

The aim of the aforementioned approaches is to identify potentially relevant queries to the advertised products and form bid phrases based on the identified queries. This approach is inherently different, because the above techniques try to predict relevant queries and do not consider the relevance of the advertised product in relation to similar products. In addition, they do not consider top-k search criteria as the appearance of a product in a search result is decided mainly on the bidding strategy. On the contrary, the aim of this work is to enhance the description of a spatio-textual object and to increase the number of queries for which the target product appears in the top-k list of the search results. In this effort, it is taken into consideration not only the user preferences, but also the rest of the spatio-textual objects that are relevant to those queries.

Chapter 3

Preliminaries

In this Chapter an introduction is made in the TOP_k and in the Reverse TOP_k queries. Their definition is set and basic computing algorithm is described in the Section 3.1 and the Section 3.2. Moreover, basic background is given about the R-tree index and its variation in the Section 3.3, while in the Section 3.4 the IR-tree variation that has been used in this thesis is described.

3.1 TOP_k query

Nowadays that there is a huge amount of data available on the Web and in various databases, the users prefer to retrieve a limited set of k answers that best suit their preferences. The TOP_k query was introduced in order to help users retrieve the best answers that they prefer in a short time. Furthermore, a ranked query in a huge dataset will take a long time. With TOP_k query a user will get the ranked results during the execution of the query.

3.1.1 TOP_k query Definition

In order to get the TOP_k results a ranking should be made to the data. The ranking of tuples of the data is based on an aggregated score that occurs when a scoring function f is applied on certain attributes of the data. This scoring function declares the users' preferences and is also called "preference function". The most common scoring function is the linear or the weighted $f(tuple) = w_1 * attribute_1 + w_2 * attribute_2 + ... + w_n * attribute_n$, where $w_1 + w_2 + ... + w_n = 1$. In this kind of scoring functions great weight value w_i denotes high preference in feature i. The definition of the TOP_k query is the following:

Definition 1 *: (TOP_k **query**) Given O a set of objects o where o=[attr$_1$,attr$_2$,...,attr$_n$], a set of users U and a number k such as $0 \leq k$, the $TOP_k(u)$ set will hold the best k objects specified by the user's weights u, where $u = [w_1, w_2, ..., w_n]$. More specific, $TOP_k(u) \subseteq O$, $|TOP_k(u)| = k$ and $\forall o_1, o_2 : o_1 \in TOP_k(u), o_2 \in O - TOP_k(u)$ it holds that $f(o_1, u) \geq f(o_2, u)$.*

By the definition 1 of the TOP_k query it is obvious that only k tuples are returned as a result from the dataset, and these k tuples are determined by the scoring function. The scoring functions are different for each user, and each k tuple set is different for different functions.

3.1.2 TOP_k query Computation

The naive algorithm to compute the TOP_k result set computes the score of all the tuples in the dataset, sorts them according to their score and then chooses the first k and returns them as a result. But the number of tuples in a dataset could be very big and the number k could be very small. For these the naive algorithm is not efficient. The best case scenario would be if only the TOP_k tuples' scores would be computed.

The onion technique [1] uses the precomputed convex hulls of the dataset. The convex hull [9] of a set of points X in the Euclidean space is the smallest region of points that inside this region they can be paired with a straight line and that line is also in the region. The algorithm uses the convex hulls of the dataset in order to compute the TOP_k results. The top-1 results are in the outside layer of the convex hulls i.e. in the layer 1, the top-2 results are either one of the other points in the outside layer of the convex hulls or in the next layer, i.e. in the layer 2. This procedure is repeated until all the TOP_k results are found. In the figure 3.1 an example of the convex hulls and the layers of the TOP_k computation are shown.

The drawback of this algorithm is that for n points in d dimensions the complexity of computing the convex hulls is $O(n^{d/2})$. Also this algorithm does not support updates

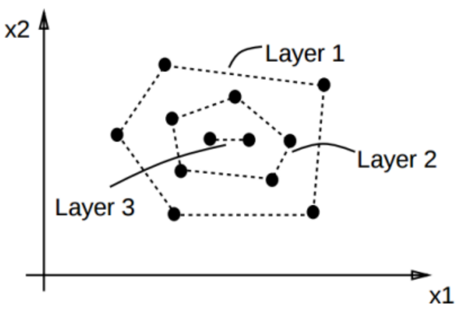


Figure 3.1: Onion method TOP_k computation

on the dataset since it has to compute again all the convex hulls each time a point is inserted, deleted or changed in the dataset.

Another method of computing the TOP_k query result set is the Branch-and-bound (BRS) [7] that pipelines the results in descending order by their score. This technique uses an R-tree to index the dataset and to compute the score of each leaf and non-leaf node, as shown in figure 3.2. For a leaf entry the score is it's score. The score of an intermediate entry is the largest score of any points that may lie in it's subtree. In order to compute the TOP_k result set, BRS uses a sorted heap to traverse the R-tree nodes according to their score in a descending order. At each step, the algorithm de-heaps the entry having the largest score and if it is a leaf, the corresponding data point is returned as a result, since it is guaranteed to have the largest score. If the de-heaped entry is a non-leaf node in the R-tree, its child nodes are heaped according to their score. The algorithm terminates when k data points have been returned.

This method has a small processing cost for all queries because it is not depended on the scoring function of each user. The Branch-and-bound algorithm also has the minimum possible space overhead since it uses only a sorted heap. It is also supports updates on the dataset, since it is easy to update an R-tree and in most databases it is a trivial process. Furthermore this algorithm can be used in many other ranked query variations and not just the TOP_k with only small changes.

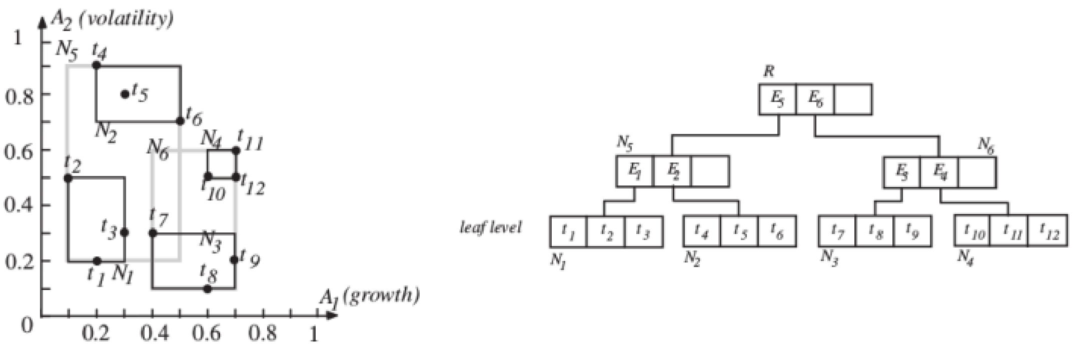


Figure 3.2: Branch-and-bound TOP_k computation

3.2 Reverse TOP_k query

The TOP_k query is a user-based query and so it is studied mainly from this angle for efficient query processing. The Reverse TOP_k query is, as it's name implies, the reverse process of a TOP_k query and is studied from the perspective of the product manufacturer. More specific, given a point in a dataset, and a set of users, the result of the Reverse TOP_k are the user preferences for which this query point is in the TOP_k query result set.

3.2.1 Reverse TOP_k query Definition

The Reverse TOP_k query has been studied in two variations [8], the Monochromatic and the Bichromatic. In the Monochromatic algorithm, the user preferences are not known, only their distribution, and an estimation is made on the impact the query point will have. This variation is useful in business analysis when no user preferences are known. In the Bichromatic algorithm, the user preferences are known, and the purpose of the algorithm is to identify the users that are interested in the given query point. This variation is important for example in commercials, in order to show the correct commercial to the user interested in it.

The definitions of both Reverse TOP_k query algorithms are the following:

Definition 2 *: **(Monochromatic Reverse** TOP_k **query)** Given a point q and a positive number k, as well as a dataset S, the result set of the Monochromatic Reverse*

TOP_k ($mRTOP_k(q)$) query of point q is a collection of d-dimensional vectors w_i, for which $\exists p \in TOP_k(w_i)$ such that $f_{w_i}(q) \leq f_{w_i}(p)$.

Definition 3 : ***(Bichromatic Reverse*** TOP_k ***query)*** *Given a point q and a positive number k, as well as a dataset S and W, where S represents data points and W is a data set containing different weighting vectors, a weighting vector $w_i \in W$ belongs to the Bichromatic Reverse TOP_k ($bRTOP_k(q)$) result set of q, if and only if $\exists p \in TOP_k(w_i)$ such that $f_{w_i}(q) \leq f_{w_i}(p)$.*

As indicated in the definitions, the result set of the Bichromatic algorithm contains a finite number of weighting vectors (user preferences), while the Monochromatic algorithm aims to describe the parts of the solution space that satisfy the query.

3.3 R-tree

Before the R-trees introduced, the B-trees where used in literature. Many variations exists, such as B^+-tree and B^*-tree, and so the keyword "B-tree" stands for all of them. They were designed to handle one-dimensional data, like integers and strings, where an ordering relation can be defined, and so they have been a standard access method in many application systems for typical transaction processing.

B-trees were only able to handle one-dimensional data and could not cover the requirements of many new application areas with different types of data, such as geographical, medical and scientific applications. Therefore a novel access method was proposed, the R-tree [5]. This structure aimed at handling geometrical data, in more than one dimension, and many improving variations have been proposed for various instances and environments.

The basic property of the R-tree is that it groups dynamically a set of d-dimensional objects by representing them with the minimum bounding d-dimensional rectangles (MBR) of nearby objects. R-tree as a tree has nodes and all nodes are implemented as disk pages. The leaf nodes of the tree contain pointers to the database objects and the non-leaf nodes hold the corresponding MBRs that aggregate the MBRs of its children nodes.

Furthermore R-trees are highly balanced trees and because of their dynamic data structure, global reorganization is not needed for insertions and deletions.

The entries of the leaf nodes of an R-tree is a tuple of the form $< mbr, oid >$, where mbr is the MBR that contains the object and oid is the object's identifier. The leafs can have maximum M entries and minimum m, where $m \leq {}^{M}/_{2}$. The non-leaf nodes can also have the same number of entries, i.e. between m and M. Each entry in the non-leaf nodes is also a tuple of the form $< mbr, p >$, where mbr is the MBR that contains the MBRs contained in this child and p is the pointer to a child node. The minimum number of entries allowed in the root of the R-tree is 2 and all leaves of the R-tree must be in the same level.

It is important to note that the MBRs may overlap and an MBR could be included in more than one node but it is assigned to only one of them. For this, a spatial search could search in many nodes in order to confirm whether a specific MBR exists or not. In order to improve the search and search as less nodes as possible, the minimization of the area of each MBR is necessary. For the same set of d-dimensional objects different R-trees can be constructed. This depends on the order of the insertions and deletions of the objects.

3.3.1 R^*-tree

A widely accepted variation on the R-tree is the R^*-tree that was introduced in order to improve the R-tree and its features [5]. As discussed in the Section 3.3, the R-tree tries to minimize the area of each MBR. R^*-tree has more criteria that intuitively improve the performance of query processing. These criteria are the following:

- *Minimization of the area covered by each MBR.* This is the same criterion that is also examined in the R-tree. It aims to minimize the area covered by each MBR and so reduce the dead space inside the MBR but not affect the enclosed rectangles. This helps to reduce the number of paths examined during query processing and searching.

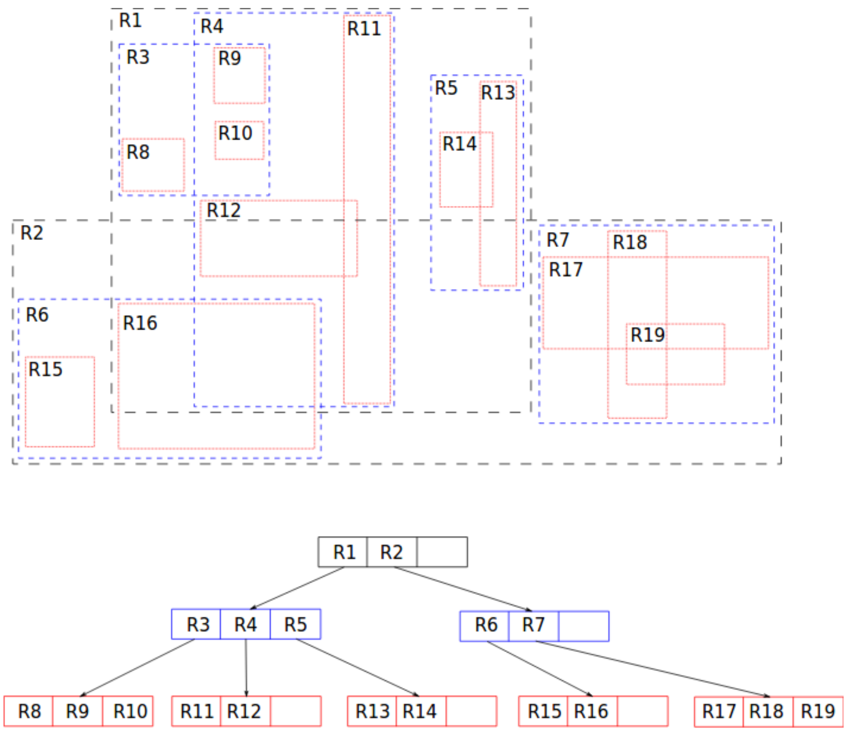


Figure 3.3: *R*-tree example

- *Minimization of overlap between MBRs.* This criterion also helps to reduce the number of paths examined during query processing and searching, since the larger the overlapping between MBRs, the larger is the expected number of paths followed for a query.

- *Minimization of MBR margins (perimeters).* This criterion is used to improve the performance of queries that have a large quadratic shape. It aims to create more quadratic rectangles, that are more easily packed. This helps the corresponding MBRs at upper levels to be smaller and thus the minimization is indirectly achieved.

- *Maximization of storage utilization.* When utilization is low, more nodes tend to be invoked during query processing. This holds especially for larger queries, where a significant portion of entries satisfies the query. Moreover, the tree height increases with decreasing node utilization.

When using the R^*-tree, an approach to find the best possible combinations of these criteria is necessary, especially when some of the criteria are contradictory.

3.4 IR-tree

The IR-tree was introduced in [2] to help the needs of a TOP_k query that takes into account both location proximity and text relevancy for points of interest with associated text. This type of query is called a *location-aware* TOP_k *text retrieval query* (LkT). An IR-tree is essentially an R-tree with inverted files. The idea is that each node of the IR-tree should record a summary of the location information and the textual content of all the objects in the sub-tree rooted at the node.

In a leaf node, the IR-tree, contains entries that have the form $< O, rectangle, O.di >$. O refers to an object in the database D, $rectangle$ is the MBR of the object O, and $O.di$ is the identifier of the document of the object O. In addition, a leaf node contains also a pointer to an inverted file for the text documents of the objects being indexed. This inverted file has a vocabulary for all distinct terms in the document of object O and a set of posting lists, each of which relates to a term t. Each posting list is a sequence of pairs $< d, w_{d,t} >$, where d refers to a document containing term t, and $w_{d,t}$ is the weight of term t in document d. Furthermore the inverted file can be distributed across several machines while this is not easily possible for the R-tree. Also it is more efficient to store each inverted file contiguously, rather than as a sequence of blocks or pages that are scattered across a disk. For these, the inverted file is stored separately from the leaf node.

In a non-leaf node, the IR-tree, contains entries that have the form $< cp, rectangle, cp.di >$. cp is the address of a child node of R, $rectangle$ is the MBR of all rectangles in entries of the child node, and $cp.di$ is the identifier of a pseudo document. A pseudo document represents all documents in the entries of the child nodes, enabling the estimation of a bound of the text relevancy to a query of all documents contained in the subtree rooted at cp. The weight of each term t in the pseudo document referenced by $cp.di$ is the maximum weight of the term in the documents contained in the subtree rooted at node cp.
In the figure 3.4 an example of an IR-tree is shown. In table 3.1 the inverted files of the leaf nodes are shown and in table 3.2 the inverted files of the non-leaf nodes are shown.

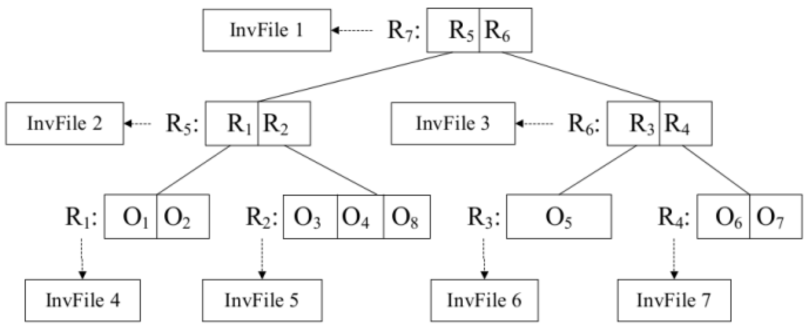


Figure 3.4: *IR*-tree example

Vocabulary	InvFile4	InvFile5	InvFile6	InvFile7
t_1	$< O_1$.doc,5>	$< O_3$.doc,2>	$< O_5$.doc,5>	$< O_7$.doc,7>
t_2	$< O_1$.doc,5>, $< O_2$.doc,4>	$< O_3$.doc,1>	$< O_5$.doc,5>	$< O_7$.doc,2>
t_3	$< O_1$.doc,6>	$< O_4$.doc,1>, $< O_8$.doc,3>		$< O_6$.doc,3>
t_4		$< O_3$.doc,5>	$< O_5$.doc,3>	$< O_6$.doc,2>, $< O_7$.doc,5>

Table 3.1: Leaf inverted files

Vocabulary	InvFile1	InvFile2	InvFile3
t_1	$< R_5$.doc,5>, $< R_6$.doc,7>	$< R_1$.doc,5>, $< R_2$.doc,2>	$< R_3$.doc,5>, $< R_4$.doc,7>
t_2	$< R_5$.doc,5>, $< R_6$.doc,5>	$< R_1$.doc,5>, $< R_2$.doc,1>	$< R_3$.doc,5>, $< R_4$.doc,2>
t_3	$< R_5$.doc,6>, $< R_6$.doc,3>	$< R_1$.doc,6>, $< R_2$.doc,3>	$< R_4$.doc,3>
t_4	$< R_5$.doc,5>, $< R_6$.doc,5>	$< R_2$.doc,5>	$< R_3$.doc,3>, $< R_4$.doc,5>

Table 3.2: Non-leaf inverted files

Chapter 4

Problem Definition

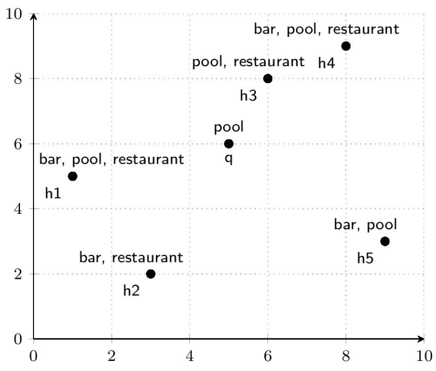
This Chapter describes the problem statement of this thesis. In order to define the problem first definitions should be provided for the Spatio-Textual Object in Section 4.1, the User Preference in Section 4.2 and the TOP_k query for Spatio-Textual Object in Section 4.3. Consequently the Best Term Problem Definition will be provided in Section 4.4.

4.1 Spatio-Textual Object

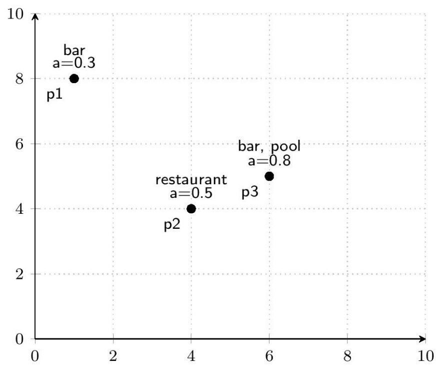
A Spatio-Textual Object o is an object that consists of keyword terms and location. More specific, it is a tuple with the form $< o.T, o.L >$, where $o.T$ are the keyword terms of the object and $o.L$ is the location of the object in the $\mathbb{R}^2$ area.

Database with this kind of objects could be a database that holds information about hotels, where the hotels have coordinates that denote their place and keywords that denote their facilities, as in the figure 4.1a. In this figure there are five hotels and for example the hotel h_1 is depicted as the tuple $< [1, 5], [bar, pool, restaurant] >$.

Let O be a set of spatio-textual objects. The vocabulary of the set O is the set A that consists of all the distinct keyword terms of all the objects o that belong to the set O. More specific, $A = \bigcup_{o \in O} o.T$. Continuing the example of figure 4.1a, the vocabulary is $A = \{\text{bar, pool, restaurant}\}$.



(a) Hotels



(b) Users

Figure 4.1: Spatio-Textual Objects and User Preferences Example

4.2 User Preferences

Users have preferences over the keywords and the location on the objects. Like objects, User Preferences are also a tuple with the form $< u.T, u.L, a >$, where $u.T$ are the desired keyword terms, $u.L$ is the location of the user in the $\mathbb{R}^2$ area and a is a number $a \in [0, 1]$ that indicates the importance of the location over matching the keyword terms.

Given a user preference, a score can be assigned to each object, according to the user's location and keywords, matching over the object's location and keywords, using the equation 4.1.

$$f(o, u) = \alpha * \delta(o.L, u.L) + (1 - \alpha) * \theta(o.T, u.T) \tag{4.1}$$

where $\delta(o.L, u.L)$ is the spatial distance computed with the euclidean distance, and $\theta(o.T, u.T)$ is the textual distance between the object's and the user's tuples computed as $1 - |o.T \cap u.T| * |u.T|^{-1}$. Other textual similarity measures could be as well used, such as cosine similarity or jaccard similarity coefficient. In this work they are not used since it is needed to do a feature ranking and the provider isn't worse if it has more keywords than a user requests.

In the figure 4.1b there is an example of three User Preferences, with their location, their desired keyword and their alpha factor. User Preference p_1 for instance is depicted as the tuple $< [1, 8], [restaurant], 0.3 >$ and the score of hotel h_1 is $f(h_1, p_1) = 0.9$.

4.3 Spatio-Textual Queries

The Spatio-Textual TOP_k query is simply a TOP_k query that takes into consideration both spatial and textual relevance with a scoring function such as the one in the equation 4.1. The Spatio-Textual objects are ranked with their score and the k best are returned to the user. In the context of this thesis we assume that the best preference is the one with the minimum score.

Definition 4 *: **(Spatio-Textual TOP_k query)** Given O a set of spatio-textual objects o, a user set of preferences U and a number k such as $0 \leq k$, the $TOP_k(u)$ set will hold the best k spatio-textual objects specified by the user preferences u. More specific, $TOP_k(u) \subseteq O$, $|TOP_k(u)| = k$ and $\forall o_1, o_2 : o_1 \in TOP_k(u), o_2 \in O - TOP_k(u)$ it holds that $f(o_1, u) \leq f(o_2, u)$.*

With this definition we can say that a user can *see* the objects that are included in his TOP_k result, or an object is *visible* from the users that have this object in their TOP_k sets. In order to find how many users can see a specific object, the reverse process of TOP_k should be defined, and so we define the Reverse TOP_k.

Definition 5 *: **(Spatio-Textual Reverse TOP_k query)** Given an object q, a set of user preferences U and a number k such as $0 \leq k$, the $RTOP_k(q)$ set will hold all the user preferences u_i from the set U for which the object q is visible, therefore q is included in the TOP_k of u_i. More specific, $RTOP_k(q) \subseteq U$ and $u \in RTOP_k(q)$ if and only if $\exists p \in TOP_k(u_i)$ such that $f(q, u_i) \leq f(p, u_i)$*

The number of users from which an object o is visible is called the Influence Score of the object, $I(o)$. Therefore $I(o) = |RTOP_k(o)|$. Continuing the example given in the previous sections, the $TOP_3(p_1)$ set of user preference p_1 are the hotels h_1, h_3 and h_2. In addition, the $RTOP_3(h_1)$ set of the hotel h_1 are the user preferences p_1 and p_2 and therefore, $I(h_1) = 2$. In the table 4.1 all the TOP_3 and $RTOP_3$ sets are shown for the example given.

Users	TOP_3
p_1	h_1, h_3, h_2
p_2	h_2, h_1, h_5
p_3	q, h_2, h_3

Hotels	$RTOP_3$	Influence Score
h_1	p_1, p_2	2
h_2	p_1, p_2, p_3	3
h_3	p_1, p_3	2
h_4		0
h_5	p_2	1
q	p_3	1

Table 4.1: Spatio-Textual TOP_k and $RTOP_k$ Queries Example

4.4 Best Term Definition

Given a set of spatio-textual objects O and a set of user preferences U, a problem that arises is how to increase the Influence Score of a specific spatio-textual object q, by enhancing it's description with at most b keywords, since the location cannot be changed. This problem is known as Best Terms.

Definition 6 : ***(Best Terms query)*** *Given a set of spatio-textual objects O, a set of terms $A = \bigcup_{o \in O} o.T$, a set of queries U, a scoring function f, an integer k, a spatio-textual object $q =< q.T, q.L >\in O$ and an integer, the set BT is a set of terms such that $BT \subseteq A$, $BT \cap q.T = \emptyset$, $|BT| \leq b$ and $\forall T \subseteq A - BT$, $|T| \leq b$ it holds that $I(q_1) \geq I(q_2)$ where $q_1 =< q.T \cup q.T, q.L >$ and $q_2 =< q.T \cup T, q.L >$.*

This problem is proven to be NP-hard by Gkorgkas et al. in [4]. Given this proof it is infeasible to find an exact solution, even for medium-sized datasets. The solutions provided in the following sections, Section 5 and Section 6, are approximate solutions.

In the example given in figure 4.1 the query object is the spatio-textual object q and the desired number of new keywords is $b = 1$. As shown in the table 4.1, the users that are in q's $RTOP_k$ set is the user p_3 and it's influence score of q is $I(q) = 1$. The q has the keyword pool, and the vocabulary A consists of the keywords bar, pool, restaurant. Since $b = 1$ one of the keywords bar or restaurant could be added to the q. In the case where the keyword bar is added to the q then in its $RTOP_k$ set user p_1 will be added and so its influence score will be $I(q) = 2$. In the case where the keyword restaurant is added to the q the user p_2 will be added to the $RTOP_k$ set and the influence score will be $I(q) = 2$. In this case where both options give the same increase in the influence score of the q, both results are acceptable.

Chapter 5

Baseline

This Chapter describes the two baseline algorithms that have already been introduced in earlier work along with their advantages and disadvantages. In the Section 5.1 the Best Term First algorithm is outlined and in the Section 5.2 the Graph-Based Term Selection, a more efficient algorithm, is detailed.

5.1 Best Term First

As already discussed in Chapter 4, the Best Terms problem is NP hard and so Gkorgkas et al. at [4] introduced a greedy algorithm, termed Best Term First (BTF), that provides an approximate solution to this problem. This algorithm uses two IR-trees, one for the spatio-textual objects, and one for the user preferences. It iterates and in each iteration a new term is added to the query item q, until b new terms are added to the q. The term in each iteration is chosen in such way that it increases the influence score of the query item $I(q)$.

This algorithm first creates a pseudo-preference q' that only uses spatial similarity, with $a = 1$, in order to access the user preferences in the IR-tree using only their distance from the query item. In other words, it exploits the IR-tree in order to sort the user preferences without the need of a sorting algorithm. In this way, the user preferences are accessed from the closest to the farthest from the query item q and this sorted access is used later in order not to perform some of the TOP_k queries for the user preferences.

Algorithm 1 Best Term First (BTF) Algorithm

Input: U: set of users, D: set of objects, q: query point, b: number of new terms

Output: BT: set of new terms

```
1: C ← ∅, buffer ← ∅
2: q' ← $< q.T, q.L, 1 >$
3: bestCandidate ←q
4: for $i = 0; i < b; i++$ do
5:     for all the $t \in A - q.T$ do
6:         C ← C ∪{<bestCandidate.T∪{t}, bestCandidate.L>}
7:     end for
8:     u ← next(U,q')
9:     while u ≠ null do
10:        $\tau \leftarrow \max_{p \in buffer}$ (f(p,u))
11:        if ∃ c ∈ C : f(c,u) ≤ $\tau$ then
12:            buffer ← $TOP_k(u)$
13:            $\tau \leftarrow \max_{p \in buffer}$ (f(p,u))
14:            for all the c ∈C do
15:                if f(c,u)≤ $\tau$ then
16:                    I(c) ← I(c)+1
17:                end if
18:            end for
19:        end if
20:        u ← next(U,q')
21:    end while
22:    bestCandidate ← $argmax_C(I(c))$
23: end for
24: BT ← bestCandidate.T-q.T
25: return BT
```

In each of the b iterations, the algorithm creates a candidate set of spatio-textual objects C, one for each term that can be added to the q where the size of the C is $|C| = |A - q.T|$, where A is the dictionary of all keywords existing in the user preferences and spatio-textual objects in the database. Each candidate has the location of q and as keyword terms all the terms of the q plus one extra term to be added to the q. For each user, the TOP_k query is performed and if a candidate object is included in the result its influence score is increased. A buffer is used in order not to perform all the TOP_k queries for all the user preferences. The buffer holds the result of the TOP_k query from the previous user preference and the score of each spatio-textua object is calculated for the current user. If no candidate object has better score from the worst score in the buffer, then the TOP_k query is not performed for the current user. Else the TOP_k query is performed and the new results are hold in the buffer.

When all users are checked, the candidate object with the best influence score chosen and its keyword term is added to the q. This process is repeated until b keyword terms are added to the q. The pseudocode of the algorithm is shown in algorithm 1.

Although this algorithm uses a greedy approach, it reduces the computational cost of performing all TOP_k queries using the pruning conditions of user preferences with the buffer. However, the number of b iterations is an important factor of the performance of this algorithm, since it forces the algorithm to access the user preferences multiple times.

5.2 Graph-Based Term Selection

Gkorgkas et al. at [4], apart from Best Term First algorithm 5.1, introduced a novel algorithm, termed Graph Based Term Selection. This algorithm, as the Best Term First algorithm, provides an approximate solution to the Best Terms problem and uses two IR-trees, one for the spatio-textual objects, and one for the user preferences.

As its name implies, Graph Based Term Selection uses a graph structure in order to estimate the gain on the influence score of each new term combination that is going to be added to the query object q. It consists of two algorithms, one 5.2.1 that creates a graph with the terms to be added to the q and one 5.2.2 that traverses the graph and selects the best combination of terms to be added to the q and increase its influence score.

5.2.1 Graph Construction

The Graph Construction algorithm, that is shown in algorithm 2, creates an weighted graph in which each node is one term that can be added to q. The weights on each edge depicts the increase on the influence score induced, if the respective set of terms is added to q. The algorithm only accessed the user preferences from which q is not visible and in maximum b terms need to be added to q in order to be visible to them. This subset of user preferences is $\widehat{U}(q)$ and $\widehat{U}(q) \subseteq U$.

Algorithm 2 Graph Construction (GC) Algorithm

```
Input: U: set of users, D: set of objects, q: query point, b: number of new terms
Output: G=(V,E): resulting graph
1:  V ← $\emptyset$, E ← $\emptyset$, buffer ← $\emptyset$, G ←(V, E)
2:  q' ←$< q.T, q.L, 1 >$
3:  u ← next(U,q')
4:  while u ≠ null do
5:     buffer ← $TOP_k(u)$
6:     $\tau \leftarrow \max_{p \in buffer}$ (f(p,u))
7:     if f(c,u)> $\tau$ then
8:        T ← u.T-q.T
9:        V ← V∪T
10:       $\lambda \leftarrow \max(1, \lceil (1 - \frac{\tau - \alpha\delta(q,u)}{1-\alpha})|u.T| - |q.T \cap u.T| \rceil)$
11:       if $\lambda = 1$ then
12:          E ← E∪{e=($t_i$,$t_i$, 1): $t_i \in$ T}
13:       else if $1 < \lambda \leq b$ then
14:          E ← E∪$\{e = (t_i, t_j, \frac{2}{\lambda(\lambda-1)}) : \forall t_i, t_j \in T$ and $t_i \neq t_j\}$
15:       end if
16:    end if
17:    u ← next(U,q')
18: end while
19: return G
```

In the beginning, the algorithm traverses all user preferences, checks if $u \in \widehat{U}(q)$ and adds to the graph G one node for each new term. The edges and the weights on them are determined by the number of terms λ that need to be added to the user preference u in order to be included in the $RTOP_k(q)$. The value of λ is calculated using the factor τ and the equation 5.1. The τ depicts the worst score thah q needs in order to be included in $TOP_k(u)$.

$$\tau = \alpha * \delta(q.L, u.L) + (1 - \alpha) * \frac{|q.T \cap u.T| + \lambda}{|u.T|} \tag{5.1}$$

In case where $\lambda \leq 1$, it means that only one term needs to be added and so the algorithm adds a loop edge with weight equal to 1 to each term that is not contained in q but contained in u. If an edge already exists, the already existing weight is increased by 1. If $\lambda > 1$, it means that more than one terms need to be added to q in order to be included in $TOP_k(u)$. In this case edges are added to the graph between all possible couple of terms that are not contained in q but contained in u. The weight of each edge is $w_e = \frac{2}{\lambda(\lambda-1)}$ that is the combination of all couple of terms added in order to make a sum of 1. If an edge already exists, the new weight is added to the weight in the edge.

This algorithm iterates only one time through user preferences, but in contrary to the Best Term First algorithm, it has to perform all TOP_k queries. In addition, the size of the graph depends on the number of distinct terms contained in $\widehat{U}(q)$. The major disadvantage of this algorithm is that the graph may not fit in the main memory if the distinct terms in $\widehat{U}(q)$ are numerous.

5.2.2 Best Subgraph Selection

After the creation of the graph with the terms and their estimation on the increase of the influence score, the Best Subgraph Selection algorithm, shown in algorithm 3, selects the best combination on terms that give the maximum increase in the influence score.

In the beginning, b nodes are chosen that have the highest degree and by them, b subgraphs are created with one node each. Then each subgraph is expanded with the node that has the highest degree, i.e. has the maximum sum of weights, and is subsequent to one of the already existing nodes in the subgraph. The recursion ends when the subgraph has b nodes, or if it cannot expand further. Finally, the subgraph which has the highest sum of edge weights is selected and the terms in its node are added to the q.

Algorithm 3 Best Subgraph Selection (BSS) Algorithm

Input: G=(V,E): graph, b: number of new terms

Output: BT: set of new terms

```
1: Q ← $\emptyset$, BT ← $\emptyset$
2: for $i = 0; i < b; i++$ do
3:     $t_i$ ← next node of G with the highest degree
4:     $G_{t_i}$ ← createSubgraph($t_i$)
5:     Q.add(sumOfWeights($G_{t_i}$), $G_{t_i}$)
6: end for
7: while |BT| ≤b do
8:     $G_S$ ← Q.pop()
9:     add to BT the b-|BT| highest degree nodes from $G_S$
10: end while
11: return BT
```

Chapter 6

Approach

In this Chapter the idea of this thesis is introduced in the Section 6.1. In the Section 6.2 the implemented algorithm based on the idea is described while in the Section **??** the design and the analysis of the code for this algorithm is documented.

6.1 The Idea

The idea of this algorithm is to try to avoid the disadvantages and keep the good aspects the two algorithms in Chapter 5 have and try to integrate them into one new algorithm. Algorithm Graph Based Term Selection 5.2 accesses only one time the user preferences. In addition, algorithm Best Term First 5.1 doesn't perform all the TOP_k queries for all user preferences. The search and the access of the user preferences on the IR-tree, and the performing of TOP_k query are the two most time consuming and resource demanding processes that need to be avoided. Consequently the new algorithm needs to integrate these two aspects from the two baseline algorithms.

Furthermore, intuitively, it needs less new terms to be added to q in order to add a user preference to the $RTOP_k(q)$ if this user preference is closer to q. In other words, the user preferences that are closest to q may have terms that when added to q help the influence score increase more than the terms of farthest user preferences. In addition, there was the observation that there are user preferences that are close to each other, but they belong to different leafs in the IR-tree. Consequently, some leaves are accessed more than one time which leads to more workload. Therefore, the idea of this algorithm is to visit the user preferences according to their leaf entries along with their distance from q, instead of visiting them only according their distance.

More specific, this algorithm, creates a candidate term set in each leaf-node of the user preferences' IR-tree it accesses and then chooses for the result the one that has the maximum increase on the influence score. First, it accesses the leaf-node of the user preferences' IR-tree that is closest to q and using the Graph Based Term Selection algorithm 5.2 finds a candidate term set. Then it examines all other nodes according to their distance from q and from their inverted index gets the b most frequent terms. If there are at least λ_{min} common terms with one of the candidate term sets then it continues examining the subtree of the specific node. If a leaf-node is reached and it has at least λ_{min} common terms with one of the candidate term sets, then a new term set is created with the Graph Based Term Selection algorithm, and the influence score of the already found term sets is been updated.

In case where a term set in the list exists that matches the b most frequent terms, then this term set is considered the best for this node and an estimation of the increase on the influence score is added to this term set without further examination of this node. In other cases, where no term set is found that has at least λ_{min} common terms with the b most frequent terms of a specific node, the node is considered that it doesn't have terms that are potentially in the best term set and so it doesn't need to be examined and is pruned.

6.2 Algorithm

This new algorithm is named Tree Based Selection, shown in algorithm 4, is divided in four other algorithms that are following described. The main algorithm initializes the list with the candidate term sets C and sets it as a parameter to the next algorithm. In the end it chooses the best candidate term set for the result.

The next algorithm is the Best Terms Non Leaf, shown in algorithm 5. This algorithm examines all non-leaf nodes of the user preferences' IR-tree until it reaches to a leaf node. At first run, where the list with the candidate term sets C is empty it goes directly to

Algorithm 4 Tree Based Selection (TBS) Algorithm

```
Input: root: root of IR-tree of users, b: number of new terms
Output: BT: set of new terms
1: C ← $\emptyset$
2: bestTermsNonLeaf(root,C,b,0)
3: BT ← $argmax_{T \in C}$(T.DeltaInf)
4: return BT
```

the leaf node that is closest to q and then calls the next algorithm that processes the leaf nodes. If the candidate term sets list C is not empty, then the set T_{TF} is selected from the inverted index of the node. The T_{TF} set contains the b most frequent terms of a specific node and is compared with the candidate term sets from the C list. If there are at least λ_{min} common terms between any term set and the T_{TF} set then the algorithm continues recursively to the children of this node. The value of λ_{min} is estimated based on the last accessed node and represents the minimum number of terms that were necessary in order to add at least one user preference in the $RTOP_k(q)$ in the last accessed node.

The leaf nodes are processed by the Best Terms Leaf algorithm that is shown in algorithm 6. This algorithm creates the candidate term set of a specific leaf node and adds it to the C list. If C is empty, the Graph Based Term Selection 5.2 algorithm is used to calculate a candidate term set. If C is not empty and exists a term set in the list that matches the T_{TF} set of terms, then this term set is considered the best term set for this specific leaf node and an estimation of the influence score is made if the Estimate Delta Inf algorithm and then added to the term set without further process of the leaf node. On the other hand, if no term set matches the b most frequent terms, a new term set must be created with the Graph Based Term Selection algorithm.

The estimation of the increase on the influence score in a term set of a specific leaf is calculated with the influence scores of the other term sets. More specific, as shown in the algorithm 7, the estimated increase is a combination of the gain in the influence score of the last term set and the average gain in the influence score of all term sets found so far.

One of the advantages of this new algorithm is that any heuristic algorithm can be used

Algorithm 5 Best Terms Non Leaf (BTNL) Algorithm

Input: node: IR-tree node, C: candidate term-sets, b: number of new terms, λ_{min}: min number of terms

Output: λ_{min}: min number of terms

```
1: proceed = false
2: if C=$\emptyset$ then
3:    proceed = true
4: else
5:    $T_{TF}$ = the b terms of the node that have the highest TF
6:    if $\exists T \in C$ such that $|T \cap T_{TF}| \geq \lambda_{min}$ then
7:       proceed = true
8:    end if
9: end if
10: if proceed then
11:    while node has more children do
12:       child = node.getNearestChild()
13:       if child is parent to leaves then
14:          $\lambda_{min}$ = bestTermsLeaf(child,C,b,$\lambda_{min}$)
15:       else
16:          $\lambda_{min}$ = bestTermsNonLeaf(child,C,b,$\lambda_{min}$)
17:       end if
18:    end while
19: end if
20: return $\lambda_{min}$
```

in order to calculate new term sets in each leaf node, instead of the Graph Based Term Selection. In addition, the pruning conditions on the user preferences' IR-tree reduces the cost in searching and accessing nodes of the IR-tree, as long as the cost of performing TOP_k queries in these nodes.

The implementation of Tree Based Selection and all its algorithms was made in Java. The xxL library was imported and extended for the use of the IR-tree index. In addition, Hash Maps and Hash Sets were also included in the implementation, where the use of a

Algorithm 6 Best Terms Leaf (BTL) Algorithm

```
Input: leaf: leaf of IR-tree of users, C: candidate term-sets, b: number of new terms,
    $\lambda_{min}$: min number of terms
Output: C: candidate term-sets, $\lambda_{min}$: min number of terms
 1: if C=$\emptyset$ then
 2:     RES = GBTS(leaf,C,b)
 3:     $\lambda_{min}$ = RES.getMinLambda()
 4:     return C, $\lambda_{min}$
 5: end if
 6: $T_{TF}$ = the set of b terms with the highest TF
 7: if $\exists T \in C$ such that $\sum_{t \in T} TF(t) = \sum_{t \in T_{TF}} TF(t)$ then
 8:     $\Delta I$ = estimateDeltaInf(T,leaf)
 9:     T.estimateDeltaInf($\Delta I$)
10:     C.updateTermSet(T)
11:     return C, $\lambda_{min}$
12: else
13:     RES = GBTS(leaf,C,b)
14:     $\lambda_{min}$ = RES.getMinLambda()
15:     return C, $\lambda_{min}$
16: end if
```

Algorithm 7 Estimate Delta Inf (EDI) Algorithm

```
Input: T: Term-set, leaf: IR-tree leaf
Output: $\Delta I$: score estimation
 1: $\Delta I = 0.5 * last\_gain + 0.5 * avg\_gain$
 2: return $\Delta I$
```

fast searching data structure was needed. For statistics, Tally library was integrated into the project, that could count different types of statistics, and create easy to see results. Furthermore, the basic Graph-Based Term Selection algorithm and the Best Term First algorithm, were already implemented by Gkorgkas et al. [4].

6.2.1 Variations

For experimental reasons, a variation of Graph-Based Term Selection was made. In this variation, the selection of keyword terms from the graph is made by selecting the adjacent node that has the maximum of the maximum weights on its edges. In the original algorithm, the selection was made by the adjacent node that has the maximum summation of weights on its edges. This variation is named Graph-Based Term Selection Max (GBTSM) and is also used in the Tree Based Selection, named Tree Based Selection Max. Experiments were made by using both two variations of the two algorithms.

Chapter 7

Experimental Results

The experimental results are presented in the Chapter. A brief description about the experiments made in this thesis is outlined in the Section 7.1, while the results of the experiments are presented in the Section 7.2.

7.1 Experiments Description

The experiments were performed on the server comidor01.idi.ntnu.no which has 2.60GHz processor, 32GB of RAM and 2TB of disk. In order to run the experiments, a dataset of approximate 740K hotel descriptions from Booking.com was used. This dataset contained a vocabulary of 958 features. The user queries that were generated had random preferred location within the area of the hotels. The keyword terms were selected as a random set from the available features in 100 hotels around the preferred location. In addition, both datasets D and U were indexed using an IR-tree where the maximum capacity of each node was 100 entries.

There were experiments with all five agorithms, the Graph-based Term Selection, the Graph-based Term Selection with Max weight, the Best Term First, the Tree Based Selection and the Tree Based Selection with Max weight. For all experiments the varying parameters were the following:

- Data cardinality |D| = 10K, 20K, 50K, 100K
- User preferences |U| = 100K, 200K, 500K, 1M

- k parameter k = 3, 5, 10
- Number of new terms b = 1, 3, 5, 8
- α parameter α = random, 0.1, 0.3, 0.5, 0.9
- Max preference size = 2, 3, 5

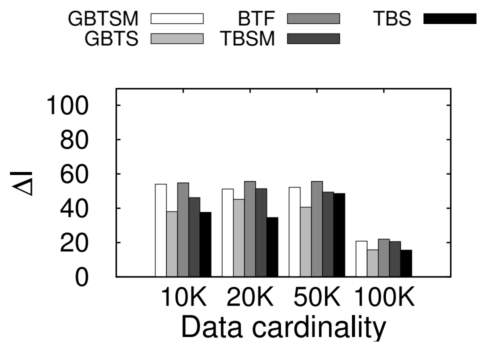
The metrics for evaluation the algorithms were the increase in the influence score ΔI, the number of I/O's performed by each algorithm, the processing time and in the two variations of Tree Based Selection the number of nodes pruned.

7.2 Experimental Results

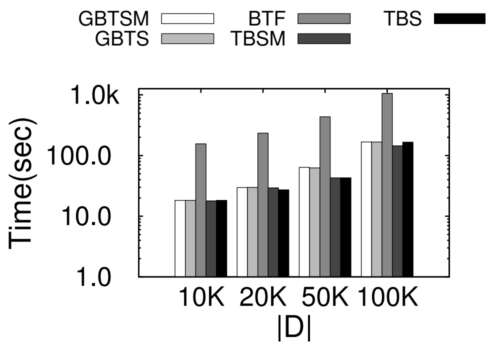
The Best Term First algorithm requires high processing cost, and so the comparison with this algorithm was made with smaller datasets. The default setting for this series of experiments was |D|=10K, |U| = 10K, k = 5 and b=3. Figure 7.1 depicts the results of this series of experiments, where the increase in the influence score ΔI is shown with varying the data cardinality 7.1a, number of new terms 7.1c and k parameter 7.1e, and the process time with varying the data cardinality 7.1b, number of new terms 7.1d and k parameter 7.1f. It is obvious that BTF is the most time consuming algorithm, but in terms of increase in the influence score it is quite close to the two variations of GBTS, while the two variations of TBS achieve a smaller increase in the influence score.

7.2.1 Varying Data Cardinality

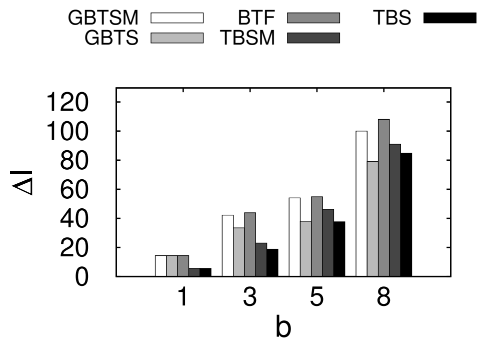
Figure 7.2 depicts the performance of the algorithms according to the number of spatio-textual objects. Figure 7.2a shows that all algorithms perform similarly with respect to the increase of the influence score. As the number of objects increase, the gain in the influence score drops as more spatio-textual objects compete for the same number of user-preferences and therefore it becomes harder for a query object to increase its influence score. In addition, as the number of objects increase, the number of I/Os performed and the process time increases too, as shown in figure 7.2b and figure 7.2d. As a dataset size increases, the cost of a single TOP_k query increases as well and therefore all algorithms are affected. The effect on the variations of GBTS is greater than the variations of TBS,



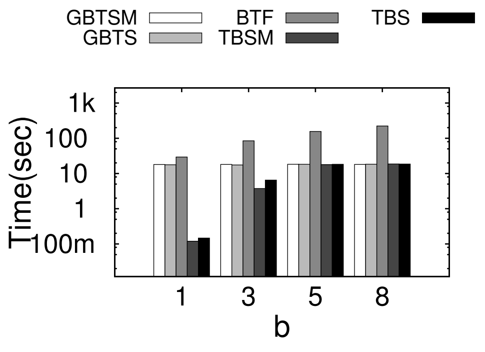
(a) ΔI varying |D|



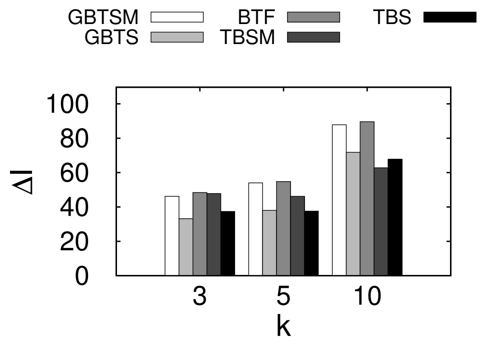
(b) time varying |D|



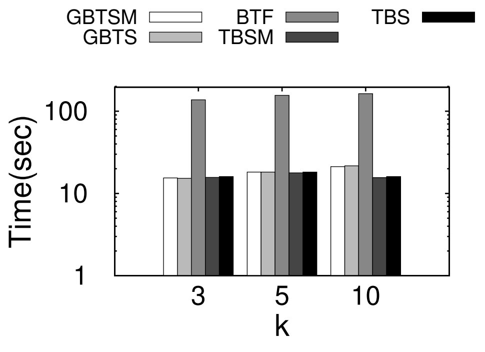
(c) ΔI varying b



(d) time varying b



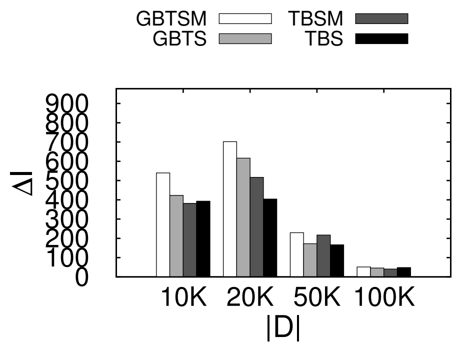
(e) ΔI varying k



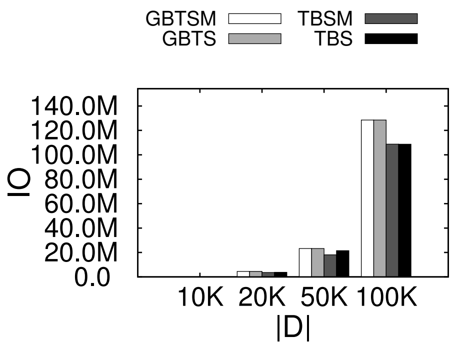
(f) time varying k

Figure 7.1: Evaluating the quality of results

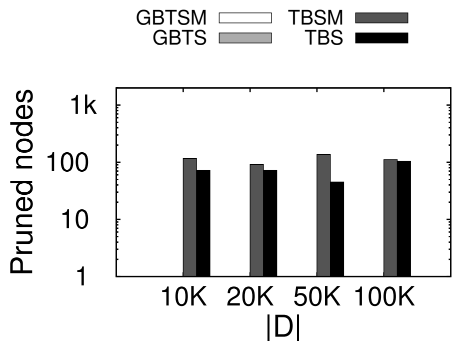
since TBS uses some pruning and estimation conditions in order not to compute some of the TOP_k queries needed, as shown in figure 7.2c.



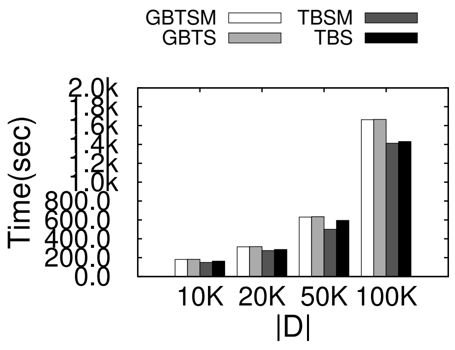
(a) ΔI



(b) I/O



(c) pruned nodes

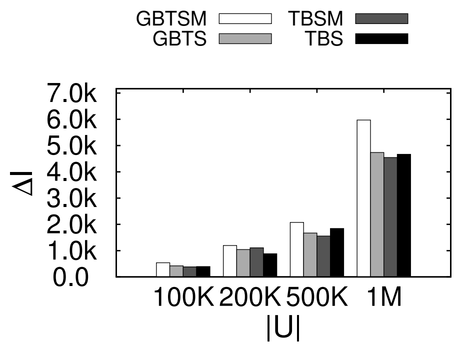


(d) time

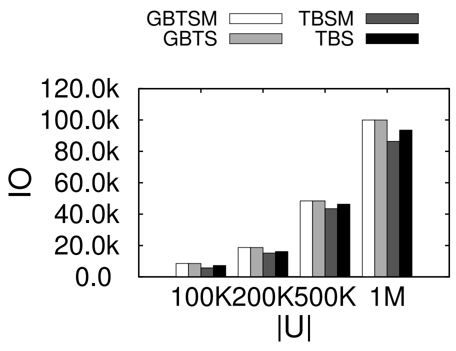
Figure 7.2: Varying Data cardinality

7.2.2 Varying User Preferences

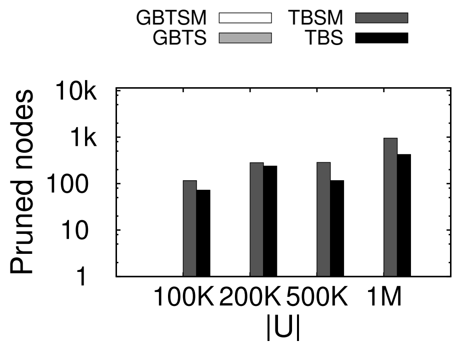
Figure 7.3 illustrates the performance of the algorithms according to the number of user preferences. In the figure 7.3a it is shown that, again, all algorithms perform similarly with respect to the increase of the influence score. As the number of user preferences increase, the gain in the influence score increases as more user preferences can be added to the $RTOP_k$ set of an object with an addition of a new set of terms. Furthermore, as the number of user preferences increase, the number of I/Os performed and the process time increases too, as shown in figure 7.3b and figure 7.3d, since more TOP_k queries need to be performed. As in the case of varying the data cardinality, in this case also, the processing time and the number of I/Os performed are greater on the variations of GBTS than the variations of TBS, since TBS doesn't compute some of the TOP_k queries, and the number of pruned nodes increases as the user preferences increase, as shown in



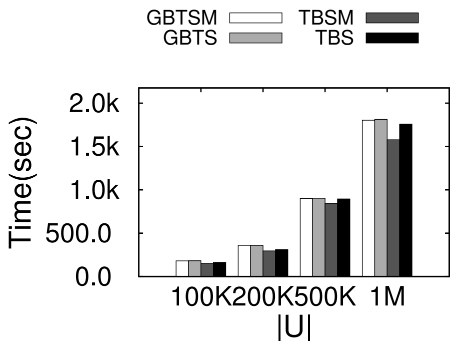
(a) ΔI



(b) I/O



(c) pruned nodes



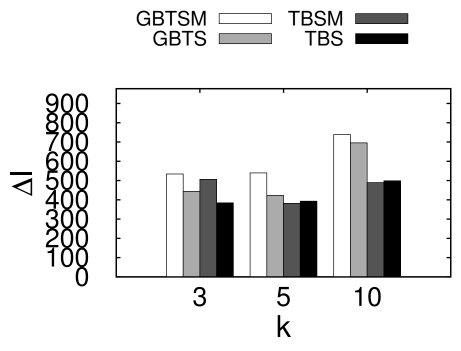
(d) time

Figure 7.3: Varying User Preferences cardinality

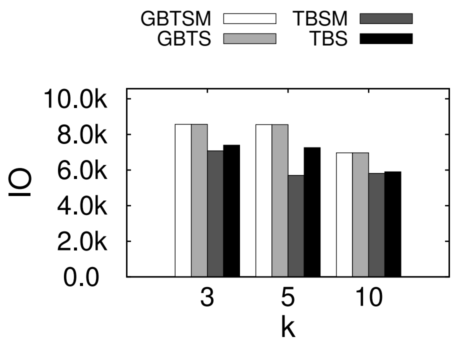
figure 7.3c.

7.2.3 Varying k Parameter

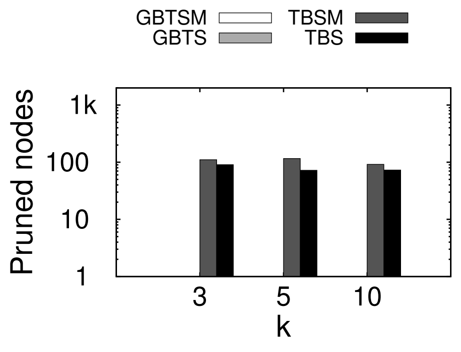
In figure 7.4 it is depicted the performance of the algorithms according to the number of k parameter. As shown in the figure 7.4a, as k parameter is increased, the gain in the influence score is increased, since more results could be found in the TOP_k set and it is easier for an object to be included in the TOP_k set of a user preference. In addition more processing time is needed, as shown in figure 7.4d, since more results need to be in the result set, and so more objects need to be processed. The number of pruned nodes in the variations of TBS, is almost stable as shown in figure 7.4c, since the condition to prune a node depends on the b frequent terms of a node and the candidate term sets.



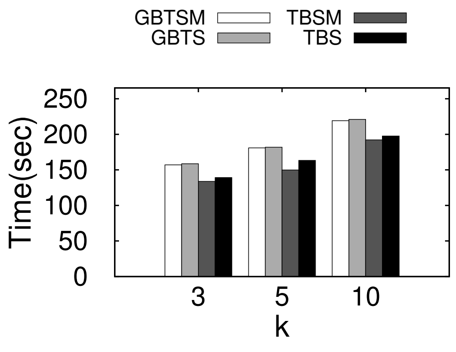
(a) ΔI



(b) I/O



(c) pruned nodes

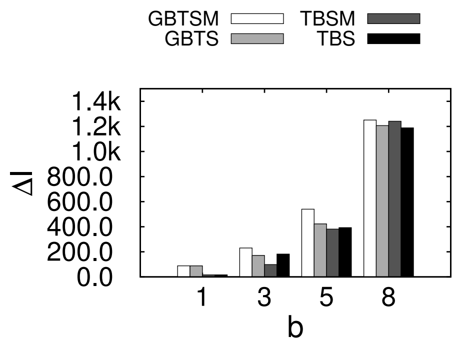


(d) time

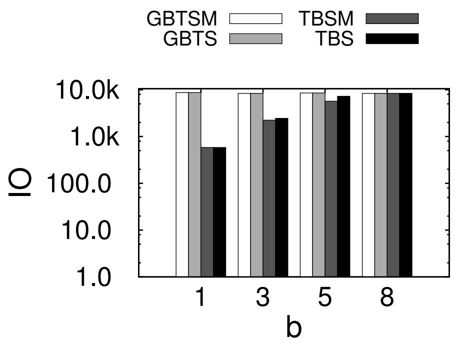
Figure 7.4: Varying k

7.2.4 Varying Number of New Terms

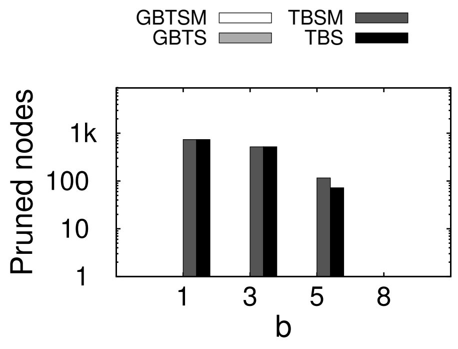
Figure 7.5 shows the performance of the algorithms according to the number of new terms. As it is expected and shown in figure 7.5a, increasing the number of new terms to be added to an object, the gain in the influence score is increased, since the more keywords an object has, the more users could attract. The number of I/Os performed and the processing time increases also, as depicted in figure 7.5b and figure 7.5d, since more terms need to be found. Figure 7.5c shows that the number of pruned nodes decreases as the number of new terms increases. This happens because the condition to prune a node examines the b most frequent terms of each node and the candidate term sets. Since more terms are included in these two sets, it is less likely for a candidate term set to be worse than the b most frequent terms of a node, and so less nodes are pruned.



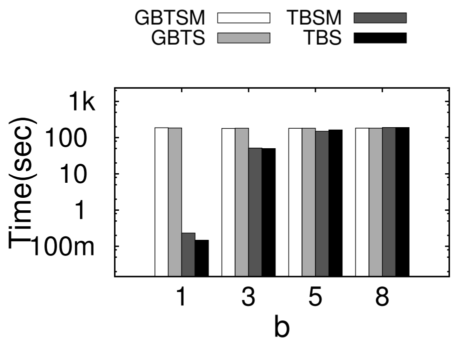
(a) ΔI



(b) I/O



(c) pruned nodes

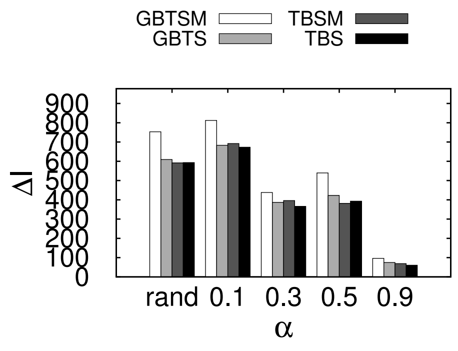


(d) time

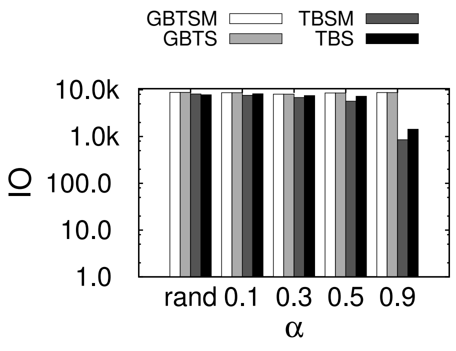
Figure 7.5: Varying b

7.2.5 Varying α Parameter

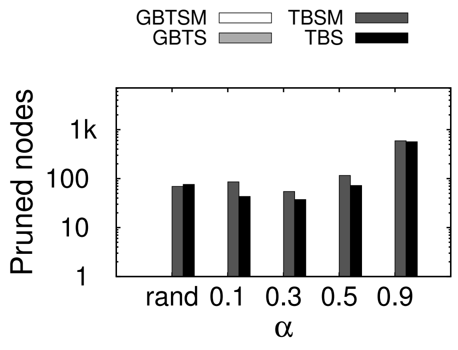
In figure 7.6 the performance of the algorithms is shown according to the number of α parameter. In figure 7.6a it is depicted that the increase of α parameter has a decreasing effect to the gain of influence score, while the random α parameter in each user preference has a medium gain. The parameter α indicates the importance of distance over the match of keyword terms, and so the greater the α, the greater the importance of distance is. Consequently, it is easier to gain more users if the keyword matching is more important, since the distance of an object cannot change in contrary to the keywords. The number of I/Os and the processing time are almost stable, as shown in figure 7.6b and figure 7.6d, since the number of data that need to be processed is not affected with the changes in the α parameter.



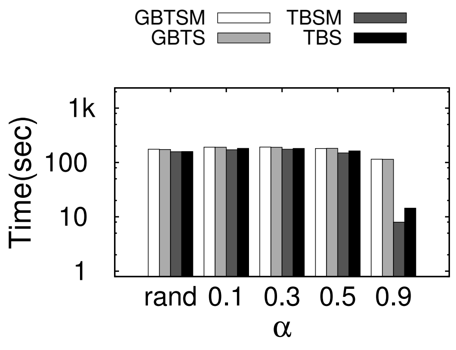
(a) ΔI



(b) I/O



(c) pruned nodes

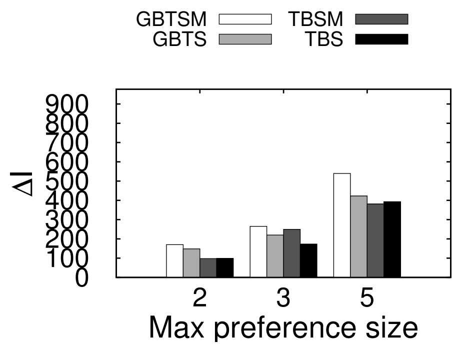


(d) time

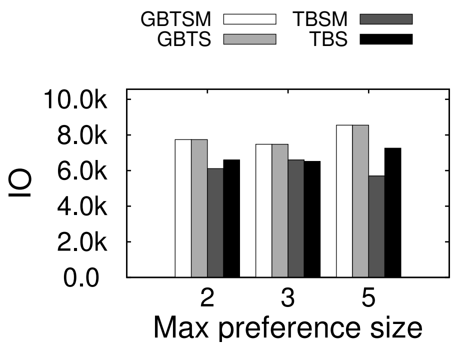
Figure 7.6: Varying α

7.2.6 Varying Maximum Preference Size

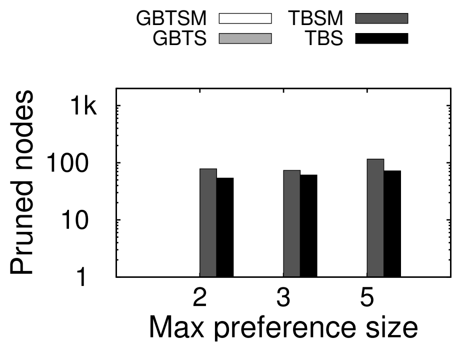
Figure 7.7 illustrates the performance of the algorithm according to the number of maximum preference size. As the maximum preference size increases, the possible gain of influence score decreases, as shown in figure 7.7a. This happens because, for a large user preference, more terms are required to be added to an object for it to be included in the TOP_k set of the user. This also affects the number of I/Os and the processing time, since the larger the query is the more complex it is to be processed, as shown in figure 7.7b and figure 7.7d. The number of pruned nodes is not affected, as shown in figure 7.7c since the pruning conditions do not depend on the size of the queries.



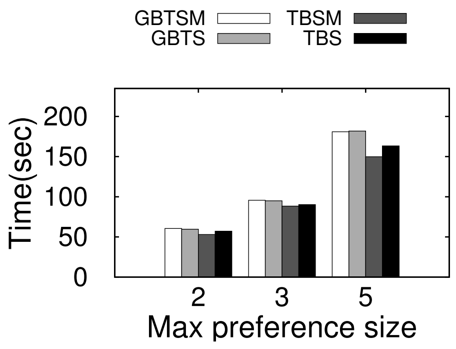
(a) ΔI



(b) I/O



(c) pruned nodes



(d) time

Figure 7.7: Varying Maximum Preference Size

Chapter 8

Conclusion

This Chapter concludes this thesis, with a brief discussion on its contribution and the results that came out of this work in Section 8.1. Furthermore some proposal about the future work possibilities are presented in the Section 8.2.

8.1 Conclusion

This work addresses the problem of increasing the influence score of a spatio-textual object, by enriching its textual description with at most b selected keywords, named Best Terms. It provides an approximate algorithm, that using the concepts of TOP_k and $RTOP_k$ queries, solves this problem. Experimental results are demonstrated, comparing the already existing algorithms with the new proposed algorithm. Furthermore, variations of the existing algorithms and the new algorithm is also compared. It is shown that the new proposed algorithms are an efficient and scalable solution to the Best Terms problem, that is less time consuming and resource demanding than the already existing algorithms.

8.2 Future Work

In future work, the aim is to examine more pruning conditions, other than the ones used in this work. It is also possible to experiment with the traversal of the IR-tree with the user preferences and the spatio-textual objects. In addition, more datasets could be examined in terms of user preferences, that may follow different distribution, or some real user queries. Last but not least, another interesting scenario would be to examine the user preferences together with the spatio-textual object, without computing the TOP_k queries.

References

[1] Yuan-Chi Chang, Lawrence Bergman, Vittorio Castelli, Chung-Sheng Li, Ming-Ling Lo, and John R. Smith. The onion technique: Indexing for linear optimization queries. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, 2000.

[2] Gao Cong, Christian S. Jensen, and Dingming Wu. Efficient retrieval of the top-k most relevant spatial web objects. *Proc. VLDB Endow.*, 2(1), Aug 2009.

[3] Ariel Fuxman, Panayiotis Tsaparas, Kannan Achan, and Rakesh Agrawal. Using the wisdom of the crowds for keyword generation. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, 2008.

[4] Orestis Gkorgkas, Akrivi Vlachou, Christos Doulkeridis, and Kjetil Nørvåg. Maximizing influence of spatio-textual objects based on keyword selection. In *Advances in Spatial and Temporal Databases*. 2015.

[5] Yannis Manolopoulos, Alexandros Nanopoulos, Apostolos N. Papadoloulos, and Yannis Theodoridis. *R-trees: Theory and Applications*. Springer, 2006.

[6] Sujith Ravi, Andrei Broder, Evgeniy Gabrilovich, Vanja Josifovski, Sandeep Pandey, and Bo Pang. Automatic generation of bid phrases for online advertising. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, 2010.

[7] Yufei Tao, Vagelis Hristidis, Dimitris Papadias, and Yannis Papakonstantinou. Branch-and-bound processing of ranked queries. *Inf. Syst.*, 32(3), May 2007.

[8] Akrivi Vlachou, Christos Doulkeridis, Yannis Kotidis, and Kjetil Nørvåg. Reverse top-k queries. In *Proceedings of Conference on Data Engineering (ICDE)*, 2010.

[9] Wikipedia. Convex hull — wikipedia, the free encyclopedia, 2015. `https://en.wikipedia.org/wiki/Convex_hull`.

[10] Weinan Zhang, Dingquan Wang, Gui-Rong Xue, and Hongyuan Zha. Advertising keywords recommendation for short-text web pages using wikipedia. *ACM Trans. Intell. Syst. Technol.*, 3(2), Feb 2012.