Efficiency-Effectiveness Trade-offs for Location Aware Search

Joel Mackenzie
joel.mackenzie@rmit.edu.au

Supervisor: J. Shane Culpepper
shane.culpepper@rmit.edu.au

Honours Thesis

School of Computer Science and Information Technology
RMIT University
Melbourne, Australia

October 2015

Abstract

With GPS data becoming commonplace through smartphones and other such devices, the need to support systems that are location-aware continues to increase. Just over half of all search queries from mobile devices have local intent [1], making location-aware search an increasingly important problem. Traditional Information Retrieval systems are able to return documents based on keywords, but this is insufficient for geographically motivated search tasks. This thesis investigates the efficiency and effectiveness trade-offs between two general types of geographical search queries, range queries and k nearest neighbour queries, for common web search tasks. We test and analyse state-of-the-art spatial-textual indexing and search algorithms for both query types across two large datasets. Finally, a rank-safe dynamic pruning algorithm is presented, which is shown to outperform the current tightly-coupled indexes used for top-k location-aware queries.

Keywords: Spatial textual indexing; location-aware search; geospatial information retrieval; experimentation; evaluation; performance

Contents

1 Introduction

As mobile devices continue to gain popularity, location-aware search is becoming increasingly popular. Mobile users are often interested in finding information about entities that are near their current location, such as stores, events, and attractions. Recent studies have shown that over 50% of searches in the Bing search engine have local intent [1]. One explanation for this phenomena is how readily available GPS has become on personal devices, and the increased amount of geotagged data associated with webpages. However, location-aware search is not limited to mobile devices. A study from Google has shown that 84% of computer users have searched with local intent, as compared to 88% of mobile users [19]. Therefore, location oriented search is popular regardless of the device of choice, which facilitates the need to efficiently and effectively support location-aware queries.

In order to successfully support these search tasks, Information Retrieval (IR) systems cannot depend solely on their textual component, as results may be suboptimal, leading to poor user satisfaction. To combat the issues that traditional IR systems face with location oriented tasks, spatial data structures can be applied to the IR system to support the geospatial relevance component that is necessary in location-aware search. These spatial data structures can be applied to an existing index to form a loosely coupled hybrid system. Popular spatial-only structures such as the R-Tree and variants [4, 20, 34] and the kd-Tree [5] can be used on top of an existing index to support the spatial component, similar to the work done by Christoforaki et al. [12]. Alternatively, hybrid structures such as the IR-Tree family [14, 25], the Integrated Inverted Index (I^3) [38] or the Spatial Inverted Index (S2I) [32] can be built, tightly combining both the spatial and textual components into a single, coherent index. Many of the existing structures vary greatly in areas such as memory hierarchy, data representation, and query support, serving varying needs.

One popular approach for spatially constraining a query is to use a "range constraint". Given a location p, and a maximum travel distance, a *range query* returns all items whose location is within the given distance from p. For example, a user may wish to find every petrol station within 5 km of their current location. Another approach is to use a *k nearest neighbour (knn) query*, which returns the k closest items to location p.

Intuitively, each of these spatial query types can be combined with textual relevance metrics to include textual relevance as well as spatial relevance. A user will express their intent through a series of keywords, as well as a location. Many different methods are possible; Keywords can be used conjunctively or disjunctively, and both range or knn queries could be utilised for the spatial relevancy. These queries are generally ranked as top-k lists, which are ranked using various metrics. A commonly used scoring function is a linear combination of a text-based measure (TF-IDF, Language model, BM25, etc.) and a geographic score – usually based on the Euclidean distance between the query location and the document location.

Another type of query known as the *Top-k Nearest Neighbour* has been explored in the literature [14, 25, 32, 38]. This query returns the top-k items with the best combined spatial and textual relevancy. This is different to the aforementioned techniques, which will simply find the spatially relevant documents, and then rank them appropriately.

This leads to the research questions that will be the focus of this work.

1.1 Research Questions

RQ 1:. *Which indexing approaches provide the best efficiency trade-offs for bag-of-words, top-k knn search in large document collections?*

RQ 2:. *What are the efficiency and effectiveness trade-offs for the two most common spatially constrained query types when applied to bag-of-words, location-aware search?*

1.2 Contributions

In this work, two alternative approaches for efficiently processing location-aware, bag-of-words queries are explored. Firstly, we introduce a variation of the *Weak AND* (WAND) [8] dynamic pruning algorithm, which incorporates upper bound estimates of the distance between documents and queries in order to support rank-safe dynamic pruning, and efficiently solve the *Top-k Nearest Neighbour Query* problem. Secondly, we re-examine the recent work of Christoforaki et al. [12] which efficiently solves the *Boolean Conjunctive Top-k Range Query* problem. Specifically, we explore the subtle differences between disjunctive bag-of-words top-k nearest neighbour queries, and disjunctive top-k range queries, with respect to both efficiency and effectiveness. Finally, an experimental evaluation of these systems is presented.

1.3 Organisation

The rest of the thesis is organised as follows.
Section 2 describes previous work in the area, from textual indexing to spatial data structures. Section 3 takes a more detailed look at the relevant literature around our research questions. Section 4 describes current filter-based approaches, where we propose another method of filter-based querying. Section 5 focuses on our main contribution, GEOWAND, and the details of our implementation. Section 6 outlines the experimental setup, approach, and the experiments performed, as well as an analysis of the results. In Section 7, the research is concluded, and Section 8 discusses some interesting research ideas that have been brought about from our work.

2 Preliminaries

2.1 Background

Information retrieval has evolved over the last 40 years. In particular, search systems have become increasingly complex as hardware continues to become less restrictive, with systems containing more memory and CPU's than ever before. It is often impractical to store indexes on-disk, as disk storage solutions are typically orders of magnitude slower than the equivalent in-memory representation due to the non-random access methods that disks utilise [36]. With these hardware improvements, in-memory top-k bag-of-words retrieval has become very popular, with most large-scale web-search engines, such as Google and Bing, utilising this as the default query format [36]. IR systems are concerned in efficiently returning effective results. That is, an IR system should quickly and cheaply return relevant results which satisfy a users information need.

Often, users of such system will enter a query where one or more of the query terms is a location. This implies that a location-aware query is desired by the user, so it is important to efficiently and effectively support such queries. The idea behind location-aware search is simple: A user can provide a regular bag-of-words query, and additionally, a location. The ranking function will return documents based both on textual and geographical relevance. For example, consider the query "`best coffee melbourne`" – presumably, the user is seeking the best coffee shops in the city of Melbourne. The (spatial) IR system would then return the top-k documents, ranked using a combination of both their textual relevance to the supplied query terms, and their spatial distance to the supplied query location. Mobile technology allows users to send their location with a query implicitly, through the use of the phone GPS. Note that other query types are possible. For example, the top-k results (textually) that fall within a certain distance of the given location is another method of satisfying such information needs.

Another aspect of IR that has gained popularity in the research community is the notion of cascading retrieval architectures. Large-scale IR systems, such as those used commercially, are thought to implement their retrieval as a collection of sequential stages. Early stages are often cheap, and used to filter the collection to retrieve a set of candidate documents. The later stages are typically more expensive, such as a Learning-To-Rank (LTR) stage, which may provide the final document ordering. Details of such systems are left for section 2.2.5.

2.2 Information Retrieval Systems

Information Retrieval Systems are the backbone of web search. Billions of search tasks are issued on the internet every day, and this number continues to grow, as more users begin to utilise web search systems. Of the many applications that IR systems have, web search is arguably the most popular. Here, the IR framework is described briefly.

2.2.1 Terms, Documents and Parsing

Basics. An IR system is concerned with efficiently returning relevant *documents* to users. In order to query an IR system, a user will enter some *keywords* into the system. Generally, a document will have an overall theme or topic, which is expressed through a number of terms. A *query* can also be seen as a set of terms, but queries are generally much shorter than documents. Formally, an IR system will index a collection of documents $\mathcal{D}$. For each document $d \in \mathcal{D}$, there will be a number of associated terms, $\{t_1, t_2, \cdots t_n\} \in d$. Each document is identified by an ordinal identifier, known as a document identifier (docID). A system may also contain a set of incoming queries $\mathcal{Q}$. For each query $q \in \mathcal{Q}$, there are a number of query terms $\{t_1, t_2, \cdots, t_n\} \in q$.

Normalisation. An inherent issue with IR systems in how individual terms can be indexed. For example, consider the term '`non-relevant`'. Is this indeed a single term, or are these two words in fact separate? Would '`non-relevant`' and '`non relevant`' be considered the same words? Due to this ambiguity, the IR system must introduce some rules regarding the way it indexes terms.

The most basic normalisation that is commonly used is known as *case folding*. Case folding merely involves normalising all of the text such that it is lower case.

Another simple normalisation of documents is to provide a *stop list*. A stop list is simply a list of terms that should be ignored during the indexing phase. Generally, common words are not useful in ranking documents, as they appear very often and add no value to the ranking process. Stopped indexes are more efficient, as less postings lists must be stored. The downside to stopping a collection is that phrase queries will not be supported. Note that if an index has been stopped, then the input queries must too be stopped. Therefore, queries such as "`The Who`" would fail, as both of the query terms are usually stopped.

Finally, consider the distinct terms '`cat`' and '`cats`'. For the sake of searching, a user does not want to re-run a query for every plural of a word. Therefore, words must be normalised to ensure that only a single version of any given word exists. This process is known as *stemming*. There are many different stemming algorithms, each with their own set of rules. Some commonly used stemming algorithms are the Krovetz [23] and the Porter [30] algorithms.

2.2.2 Indexing and Architecture

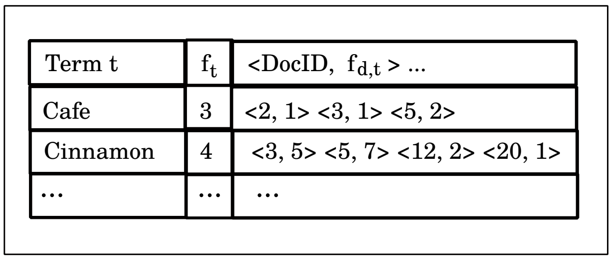
Indexing refers to the process of converting the collection $\mathcal{D}$ into an efficient representation. The most common index is known as the *Inverted Index* [41]. Although other representations exist, the focus of this work is aimed at working with traditional, inverted index-based IR systems.

Inverted Index components. After the collection has been parsed, the inverted index structure can be created. Essentially, at the point of completing the parsing, all of the information for the inverted index is already acquired, so it just needs to be dumped appropriately. However, there are a few separate components of the inverted index, so they will be discussed briefly. The *lexicon*, or *dictionary* simply stores a term t, and the number of unique documents the term appears in (the f_t value). There may also be some other metadata stored, such as the offset to where a posting list begins and ends (for each term). The *postings lists* encompass the statistical information about terms across the collection. Postings are generally stored as compressed integers, but have the basic format of the docID (for document d) followed by $f_{d,t}$, an integer which stores the amount of times term t appears in document d. Figure 1 shows the general layout of an inverted index.

Architecture and Implementation. The data structures used to store inverted indexes is dependent on implementation. Typically, the lexicon would be stored in a hash-table or B-Tree, which gives access to both the offsets to the corresponding postings, and the global term information. Postings lists are generally stored as compressed arrays of integers. Optimisations allow more efficient storage and retrieval of such structures, which are outlined in Section 2.2.6.

2.2.3 Query Processing

In the underlying IR system, there are many steps taken to answer a query. For example, the query terms must be parsed, normalised, and then sent to the retrieval system. The retrieval system is



Term t	f_t	<DocID, $f_{d,t}$ > ...
Cafe	3	<2, 1> <3, 1> <5, 2>
Cinnamon	4	<3, 5> <5, 7> <12, 2> <20, 1>
...	...	...

Figure 1: A typical inverted index organisation. f_t is the number of documents that term t appears in across the entire collection. Additional information may also be stored, such as the total number of times term t appears across every document.

concerned with finding documents that match the query – all documents that match the query will be ranked. In order to find these matching documents, the posting lists are used, as they contain the relevant information about terms and documents. There are two processing methods that can be used to find matching documents, *Document-at-a-Time* (DAAT), and *Term-at-a-Time* (TAAT). These methods will be shown through an example bag-of-words query "`best coffee melbourne`".

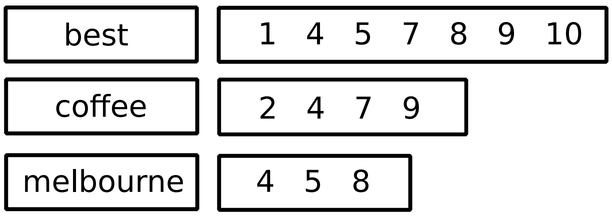


Figure 2: A simplified set of postings lists for 3 query terms. The postings lists show the document ID of each document containing the term.

Firstly, the IR system will retrieve the postings lists that correspond to the query terms. Using these postings lists, the retrieval algorithm can find the documents that contain the terms, and score those that match.

Document-at-a-Time. In a DAAT traversal, a cursor is kept on each posting list at once. Then, each posting list is traversed concurrently. For example, consider Figure 2. Initially, a cursor would point to the first document in each postings list. The list with the smallest document ID will be considered, which is the `best` list in the example. Since document 1 only appears in this list, the score will be calculated for the term 'best' in document 1, and the heap will be updated if necessary. Then, the cursor on the `best` list will be forwarded. The document with the smallest document ID will be found and scored next. This is document 2, in the `coffee` list. Once again, this is the only list in which document 2 appears, so it is scored, the heap is updated, and the cursor is forwarded. At this stage, the cursor on each postings list points to document 4. Therefore, the score for document 4 is calculated by adding the score for each term together. Then, the heap is updated if necessary, and the cursor on each postings list will be forwarded. Execution will continue until all of the cursors are exhausted. At this point, the heap contains the top-k documents.

Term-at-a-Time. In a TAAT traversal, a single posting list is processed at a time. Therefore, an additional structure is required to keep track of the partial scores of each document, known as an *accumulator*. An accumulator will simply store the document ID, and the current score of this document. Again, consider the example in Figure 2. The first list examined may be the list for the term `best`. Firstly, the score for this term in document 1 will be calculated, and stored in an accumulator with ID 1. Next, the score for this term in document 4 will be calculated, and stored in an accumulator with ID 4. Once the list for the term `best` has been processed, the next postings list will be processed. In this case, that is the `coffee` list. The same process occurs, but if a document already has an accumulator, the score will be added to the corresponding accumulator. For example, since document 4 appeared in the `best` list, when document 4 is processed in the `coffee` list, the score for the term 'coffee' in this document is simply added to the accumulator with ID 4. This process continues until all of the postings lists have been processed. At the end of processing, a set of accumulators will hold the total score for each document which contained one or more query terms. Finally, this set of accumulators can be sorted, and the top-k documents can be returned to the user.

There are advantages and disadvantages to both methods. For example, DAAT does not need to store the additional accumulators, as an entire document will be scored at once. On the other hand, DAAT postings lists must be ordered by document ID. Since TAAT indexes can be stored in any arbitrary order, additional enhancements can be made, such as placing the highest scoring documents at the front of each list.

2.2.4 Ranking

When it comes time for an IR system to return relevant documents to queries, the system must define a metric in which to measure the 'goodness' of the documents with respect to the given query. Typically, ranking must be both efficient and effective in order to satisfy a user. However, the concept of relevance is highly subjective. Due to this, many different ranking algorithms exist [41]. Here, we provide a brief overview of the most common ranking models.

Formal problem. Given a query q and a collection $\mathcal{D}$, a ranker will attempt to return the top-k documents $\mathcal{R} = d_1, \ldots, d_k$ such that d_k is the kth most relevant document with respect to q.

Methods. In order to both efficiently and effectively rank documents, most of the 'hard' work is done at indexing time. The reason this is possible is because the majority of ranking functions use statistical properties of the language of the documents that have been indexed. For example, words that appear in very few documents will be very valuable in ranking a query that contains such words, and hence should carry a higher relevance score than words that appear commonly in many documents. More formally, there are a few particular statistical properties that are of wide interest to most relevance functions:

1. Term Frequency (TF): The more a term t appears in document d, the more relevant d is to a query q containing t.
2. Inverse Document Frequency (IDF): The less a documents term t appears in the collection, the more valuable the term for ranking.
3. Inverse Document Length (IDL): The longer a document is, the more terms it will contain. Therefore, all documents should be fairly weighted based on length.

A common method of relevance judgement is known as *probabalistic* judgement. Although there are many formulations of probabalistic ranking, the idea is the same: If the probability of document d being relevant to query q can be accurately calculated, then the result listing will be ranked in the optimal order if sorted by probability. Where the many models of probabalistic ranking differ is in their probability calculations. Perhaps the most commonly used probabalistic ranking model is *Okapi BM25*, which stands for 'Best Match'. It was proposed by Robertson et al. [31] in 1994.

In the experiments ran in this paper, a variation of the Okapi BM25 ranking function was used, and is defined as follows:
Given a query q made up of terms $t_1, \ldots, t_n$, and a document d,

$$\text{BM25}(d, q) = \sum_{t \in q} w_t \frac{(k_1 + 1) f_{d,t}}{K + f_{d,t}} \cdot \frac{(k_3 + 1) f_{q,t}}{k_3 + f_{q,t}} \text{ where:}$$

$$\begin{aligned}
w_t &= \ln \frac{N - f_t + 0.5}{f_t + 0.5}, \\
K &= k_1((1 - b) + \frac{b - L_d}{L_{average}}, \\
f_{d,t} &\text{ is the number of occurrences of } t \text{ in the document } d, \\
f_{q,t} &\text{ is the number of occurrences of } t \text{ in the query } q, \\
f_t &\text{ is the number of documents containing } t, \\
N &\text{ is the number of documents in the collection,} \\
L_d &\text{ is the length of document } d, \\
L_{average} &\text{ is the average document length,} \\
k_1, k_3, \text{ and } b &\text{ are parameters that effect the weight of the distinct} \\
&\text{ components of the ranking function.}
\end{aligned}$$

Note that often $k_3 = 0$, meaning that a repeated term in a query will not increase the weight that this term provides to the ranking. In this case, BM25 will simplify to:

$$\text{BM25}(d, q) = \sum_{t \in q} w_t \frac{(k_1 + 1) f_{d,t}}{K + f_{d,t}}$$

Score Safety. Another important concept in ranking is known as score-safety, or safe-to-k. If a retrieval algorithm $\mathcal{A}$ that uses ranking function $S()$ returns a top-k list of documents $\mathcal{R}$, and an exhaustive retrieval run using $S()$ yields a top-k list $\mathcal{Z}$, then $\mathcal{A}$ is safe-to-k if and only if $\mathcal{R} = \mathcal{Z}$. This property must hold for every query. Therefore, a safe-to-k algorithm is guaranteed to return the optimal results list with respect to a given ranking function. In some IR systems, ranking efficiency is increased by finding approximate results, rather than the optimal top-k results. This is an example of trading effectiveness for efficiency – approximate results are likely to be less effective, but they are generally returned much faster.

2.2.5 Cascaded Ranking

In large-scale web search systems, millions of documents typically need to be examined for a given query. To efficiently return the top-k documents to a user, a cascaded approach can be used. The general cascaded ranking approach is as follows. Firstly, a simple filter will be used to find a set of candidate documents that meet some constraint. These 'stage 0' filters are often fast and simple, such as a Boolean conjunction or disjunction. This filter will generally (but not always) return $k' > k$ documents. This set of candidate documents is then passed to the next stage. In stage 1, the candidate documents will be ranked using some ranker such as BM25. Again, this is a relatively cheap operation, and can be seen as a reordering of the k' candidates. The top-k documents from this stage will be used as the new candidate set for stage 2. In stage 2, the top-k documents recieved will be reranked such that they are ordered based on Learning-To-Rank or higher dependency models [2, 3, 26, 27]. This is typically an expensive operation, which is why it is important to send only the top-k candidates to this stage. Finally, the reranked top-k document list is returned to the user. This process is illustrated in Figure 3.

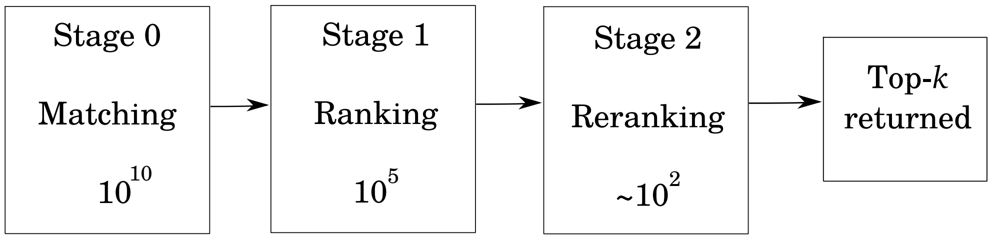


Figure 3: An example cascaded ranking system. Stage 0 finds documents that match with the query, and passes these documents to stage 1. Stage 1 will rank these documents, then pass them to stage 2. Stage 2 is a machine-learning step, which aims to reorder the candidate documents such that the top-k can be returned to the user. Stage 2 is much more expensive than earlier stages, which is why it is important to filter the document set before sending it to stage 2. The numbers below each stage indicate the magnitude of documents that will be examined by that particular stage.

Although our example cascaded system utilises a 3-stage approach, there may be many more stages, depending on the system. In addition, many different stage 0 filters could be used to filter the k' documents that are passed on to stage 1. Common stage 0 filters include Boolean conjunction or disjunction, but other properties of documents could be used, such as the geospatial location, or the spam-score of the document [15].

2.2.6 Optimisation

Previous research has involved experimentation with data structures and algorithms in order to improve efficiency, such as early termination algorithms [8, 16] and compression techniques [24, 41]. The current state-of-the-art techniques for top-k retrieval use *document-at-a-time* (DAAT) processing and a variation of the WAND algorithm, proposed by Broder et al. [8]. These state-of-the-art techniques are very efficient with respect to both time and space, due to a large focus of research in this area [8, 16, 16, 24, 29].

Compression. Compression allows for a smaller storage footprint, and also results in faster retrieval. In an IR system, the postings lists are generally compressed. This is because postings are simply integers, and integers are much easier to compress than textual data. The compression of postings lists is done at index time.

Although many integer compression algorithms exists [24], the most popular techniques involve compressing blocks of postings lists rather than the entire list as one block. The advantage of block compression is that random access to a given block is still possible, and only that particular block needs to be decompressed to retrieve the postings within. If the entire posting list was compressed, the entire list would need to be decompressed in order to access it. Block compression is usually done in cache-aligned block sizes, such as 64 or 128 entries per block.

Generally the time taken to retrieve and decompress a compressed block is extremely small. Compression also improves cache coherence of data structures, as less memory must be traversed to access the data.

Weak-AND. "Weak" or "weighted" AND (WAND) was first introduced by Broder et al. [8], and has been shown to improve efficiency in DAAT query processing without losing effectiveness [8, 16, 29]. The main idea behind WAND is to do a cheap 'estimate' of the highest score a given document can achieve. If the estimated score is higher than the score of the lowest scoring document in the top-k heap, then the current document could possibly make it into the top-k heap, and will be scored correctly. However, if the documents estimated score is below the threshold, the document cannot make it into the heap, and does not need to be scored.

At index time, the upper-bound score for any given term t is stored in the index as $U[t]$. A query q will enter the system, with terms $t_1, \cdots, t_n$. WAND will store a threshold value, θ. If k documents have not been scored, $\theta = -\infty$. As the kth document is scored, and the heap is updated, θ is set to the score of the lowest scoring document in the heap. For each query term t_i that is in document d, the upper-bound score $U[t_i]$ is added to an upper-bound *estimate* variable. If this estimate is larger than θ, then document d must be scored completely, as there is a chance that d will be in the top-k documents. However, if the estimate is lower than θ, document d could not possibly make the top-k heap, so it is ignored, and the next candidate document is examined.

As more documents are scored, the value of θ increases. This means that more documents are skipped, as there is less likelihood that *estimate* $> \theta$. Obviously the first k documents will be placed in the heap, as $\theta = -\infty$ until k documents are present in the heap. WAND has been shown to be safe-to-k, which allows a more efficient processing of queries without any loss of effectiveness. If desired, the value of θ can be overestimated such that WAND prunes more documents, which will improve efficiency at the cost of score safety.

In formal terms, WAND can be thought of as a Boolean operator. The WAND operator takes a list of Boolean variables $X_1, X_2, \cdots, X_t$, a list of associated positive weights $w_1, w_2, \cdots, w_t$, and a threshold θ, such that:

$$\mathsf{WAND}(X_1, w_1, X_2, w_2, \cdots, X_t, w_t, \theta) \text{ is } \textit{true}\text{, if, and only if}$$

$$\sum_{1 \le i \le t} x_i w_i \ge \theta$$

$$\text{where } x_i \text{ is the indicator variable for} X_i \text{, that is}$$

$$x_i = \begin{cases} 1 & \text{if } X_i \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

Therefore, WAND can act like an AND query through $\mathsf{WAND}(X_1, 1, X_2, 1, \cdots, X_t, t)$, or an OR query through $\mathsf{WAND}(X_1, 1, X_2, 1, \cdots, X_t, 1)$. The value of θ governs how close to an AND or an OR query WAND will act. By setting w_t to $U[t]$, and X_t to true if term t appears in the current document, the WAND operator will only return true for documents that have a total estimated upper-bound score greater than θ.

Other versions of WAND exist, such as BLOCK-MAX WAND, which stores an upper-bound score for each block of postings list entries, which allows even more effective pruning. However, this is out of the scope of the work presented here.

TAAT optimisation. *Term-at-a-time* (TAAT) processing is also very well researched, including many optimisations. However, TAAT processing is not considered in this work, as the vast majority of recent work suggests that DAAT processing is the better solution.

2.2.7 Evaluation

In order to test how well an IR system performs, the results returned by the system must be evaluated. There are two important aspects of a system that must be evaluated – efficiency, and effectiveness. Efficiency evaluation involves measuring the expenses of a system, for example, the time taken to service a query. On the other hand, effectiveness evaluation involves calculating how well the returned results satisfy a users' information need. This discussion will focus on effectiveness evaluation.

Effectiveness is evaluated through a range of methods derived from precision (P) and recall (R), such as average precision (AP), discounted cumulative gain (DCG), expected reciprocal rank (ERR), rank-biased precision (RBP) and so on [9, 21, 28].

Precision and Recall. Perhaps the simplest methods of evaluating an IR system are precision and recall. Both are very intuitive, and are formulated as follows.

$$\mathsf{P} = \frac{|\{\textit{relevant}\} \cap \{\textit{retrieved}\}|}{|\{\textit{retrieved}\}|}$$

$$\mathsf{R} = \frac{|\{\textit{relevant}\} \cap \{\textit{retrieved}\}|}{|\{\textit{relevant}\}|}$$

Precision calculates the percentage of documents returned that are relevant to a given query, whereas recall calculates the percentage of all relevant documents in the collection that are returned by the IR system. For example, if $\mathsf{P} = 1$, all retrieved documents are relevant to the query. Conversely, $\mathsf{P} = 0$ implies that no retrieved documents are relevant to the query. On the other hand, $\mathsf{R} = 1$ would imply that every single relevant document in the collection was returned, and $\mathsf{R} = 0$ means that no relevant documents were returned. Although intuitive, there are inherent problems with these metrics. Higher recall can be achieved by simply returning more documents, as there is a higher chance that relevant documents will be returned. However, this can decrease precision. Additionally, if a query has few relevant documents, the value for P will be small even if the system performs well.

User Model. Another major problem with the aforementioned metrics is that a trade-off exists between them. As mentioned, R can be increased trivially while causing a decreased P value. Additionally, both P and R do not account for the typical behaviour of a user. Generally, IR systems return the top-k documents as a ranked list. Therefore, users are shown what the system regards as the 'best' documents first. This is an important consideration, as the effectiveness of a system should be based on how users interact with it. This user model gives rise to a new breed of evaluation metrics that are formulated to consider the behaviour of a user.

Rank-Biased Precision. RBP is an evaluation metric that measures the rate of utility gained by a user [28].

Given a ranked relevance vector to depth d: $\mathcal{R} = \langle r_i \mid i = 1, 2, \cdots, d\rangle$ where $r_i = 0$ (not relevant) or $r_i = 1$ (relevant), and a user persistence value $p \in [0.0, 1.0]$,

$$\mathsf{RBP} = (1 - p) \cdot \sum_{i=1}^{d} r_i \cdot p^{i-1}$$

The value of p emulates how persistent a user will be. At higher values of p, the user is more likely to look further down the results list for a relevant document – the user is more persistent. As p decreases, the user will become less peristent (and evaluate less of the returned results). To illustrate this, at $p = 0.95$, there is a 60% chance that the user will look at the second page (of 10) returned documents. At $p = 0.5$, there is only a 0.1% chance of the user entering the second page of results. At the least persistent value of $p = 0.0$, the user will only evaluate the first document that was returned.

Expected Reciprocal Rank. ERR is another evaluation metric that emulates user behaviour [9]. The basis for ERR is the cascaded model of search, which makes the assumption that a user will iterate through the ranked result list in order, and evaluate whether the document satisfies the given query. If it does, the user will stop searching. ERR aims to calculate the expectation of the reciprocal of the position in the ranking list in which the user stops searching. Therefore, an ERR value of 1.0 is a perfect score, and 0.0 is a poor score.

Given a vector of documents $\langle d_1, d_2, \cdots, d_n \rangle$ which have been ranked with respect to query q such that d_n is the nth most relevant document,

$$\text{ERR} = \sum_{i=1}^{n} \frac{1}{i} \cdot p(q, d_i) \cdot \prod_{j=1}^{n-1} (1 - p(q, d_j)), \text{ where}$$

$$p(q, d_i) = \text{ the probability of document } d_i \text{ satisfying } q$$

Maximised Effectiveness Difference. In order to compare two different filtering methods, maximised effectiveness difference (MED) [37] can be used. MED is used in conjunction with a relevance metric, such as RBP or ERR. Given the output of 2 different systems $\mathcal{A}$ and $\mathcal{B}$, and a relevance metric M, MED identifies a set of relevance judgements that will seperate $\mathcal{A}$ and $\mathcal{B}$ maximally. Therefore, the output from a MED run will be a coefficient between 0 and 1, where 0 means that system $\mathcal{A}$ and $\mathcal{B}$ have no difference in effectiveness, and 1 means that $\mathcal{A}$ and $\mathcal{B}$ are completely differing in their results lists. A value of 0.2 is considered negligable, so any MED value below 0.2 would indicate that $\mathcal{A}$ and $\mathcal{B}$ are more or less at the same effectiveness level. Generally, $\mathcal{A}$ will provide a set of candidate documents that passed through a given filter, and $\mathcal{B}$ will be a "gold-standard" list, which are the true top-k documents that should be returned. In this case, MED will compare the relative effectiveness of $\mathcal{A}$ to the true result list. Clarke et al. [13] provide an in-depth analysis of Multi-stage IR systems using MED as their comparison method. A large advantage with MED is that no relevance judgements are required to compute the result, making it suitable for query and document sets that are not judged. Note that a good, "known" system is required to get the gold-standard list which is used in calculating MED.

2.3 Spatial Indexing and Retrieval

Spatial data structures have been around for over 3 decades, with the kd-Tree first introduced in 1975 by Bentley [5]. Spatial data structures are designed to efficiently support querying of spatial objects.

2.3.1 Spatial Objects

There are two simple types of spatial data objects that can be stored by spatial data structures, the *point* and the *region*. Regions are also known in the literature as *Minimum Bounding Rectangles (MBRs)* or *Minimum Bounding Boxes (MBBs)*. For this work, only 2-dimensional objects will be considered due to their simplicity and low storage overhead. However, both n-dimensional point and region objects can be applied to most spatial data structures. The 2-dimensional point object is defined as follows:

- A 2-dimensional **point** object p is a tuple of 1-dimensional coordinates: $p = \langle c_1, c_2 \rangle : c_1 \in (-90, 90), c_2 \in (-180, 180)$. Note that latitude comes before longitude. For example, a point

p on the Earth's surface can be defined as $p = \langle -12.3, 103.7\rangle$, where -12.3 and 103.7 are the latitude and longitude of p.

Additionally, the 2-dimensional region object is defined as follows:

- A 2-dimensional **region** object r can be defined as a set of points: $r = \{p_1, p_2, p_3, p_4\} : p_k = (c_1, c_2), c_1 \in (-90, 90), c_2 \in (-180, 180)$. For example, a region r on the Earth's surface can be defined as $r = \{p_1, p_2, p_3, p_4\} : p_1 = \langle 0.0, 0.0\rangle, p_2 = \langle 1.0, 0.0\rangle, p_3 = \langle 1.0, 1.0\rangle, p_4 = \langle 0.0, 1.0\rangle$, resulting in a bounding rectangle. Due to the definition of the region being rectangular, it is only necessary to store the lower-left and upper-right points. Therefore, a region r can also be defined as a tuple of points: $r = \langle p_1, p_2\rangle : p_k = \langle c_1, c_2\rangle, c_1 \in (-90, 90), c_2 \in (-180, 180)$, p_1 is the lower-left point, and p_2 is the upper-right point. Note that r is not a set in this definition, but a tuple, as ordering is explicit.

Quantisation. Often, latitudes and longitudes can be mapped into integers without losing precision. This process is done by appropriately scaling the float values, then transforming the scaled space into the set of positive integers. This can be used to improve compression, such as in the work done by Brisaboa et al. [6] or Kim et al. [22]. An obvious advantage of this representation is the improved efficiency of processing unsigned integers rather than conducting floating point arithmetic. However, it is important to notice that some precision is generally lost, which may result in false positives or false negatives when querying across quantised space. Generally, the quantised representation can not be used to recover the original representation.

2.3.2 Spatial Queries

Many different query types exist for spatial objects [10]. Often, the support of such queries is dependent on the data structure of choice. Assume that a user has provided a location point p, and a distance n. The two main spatial queries that are supported are:

- **k-Nearest-Neighbour query:** Retrieve the k nearest objects to p.
- **Range query:** Retrieve all objects within distance n of p.

In order to efficiently store and query these spatial objects, a number of spatial data structures have been proposed.

2.3.3 Spatial Data Structures

kd-tree. The kd-Tree [5] was one of the first spatial data structures to support knn queries. The kd-Tree is a space partitioning tree that efficiently organises points in k-dimensional space. At construction time, the algorithm cycles through the dimensions, splitting along the next dimension for a given level of the tree. The split occurs such that half of the points in dimension d occur to one side of the split line, and the other half occur on the other side of the split line. For example, a 2 dimensional kd-Tree would be created by first splitting data points such that half of the points are to the left of the split line, and the other half of the points are on the right of the split line. This implies that the values on the left of the split will be in the left sub-tree, and so on. Next, the splitting would occur in a horizontal direction at each sub-tree. Figure 4 shows a 2-dimensional kd-Tree and associated data set. The kd-Tree is simply a type of multidimensional binary search tree.

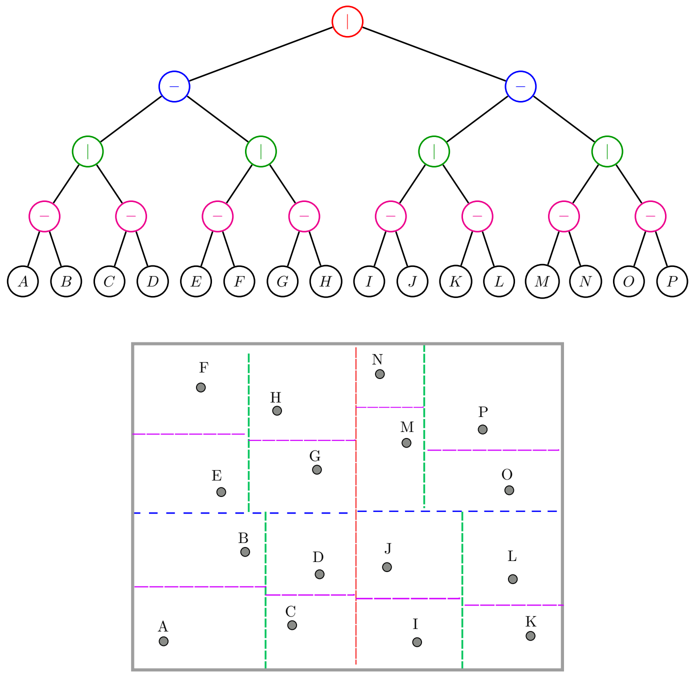


Figure 4: An example k-d Tree of dimension 2.

One limitation with the kd-Tree is that only point data can be efficiently stored. If MBRs are required, then a different data structure, such as the R-Tree, is a more suitable choice.

R-tree and variants. Proposed by Guttman [20] in 1984, the R-Tree is a commonly used spatial data structure for storing both point or MBR objects. The idea behind the R-Tree is to allow spatial pruning through the use of MBRs. At each internal node of an R-Tree, an MBR is stored. The MBR contains geographically 'close' objects. The closer to the bottom of the tree, the smaller each MBR is, and hence, less objects are stored within the MBR. In each leaf node of the tree, a group of spatial objects are stored. Therefore, the R-Tree essentially holds coarse approximations of data objects in the upper layers, and more granular approximations of data objects close to the leaves. Querying on an R-Tree allows spatial pruning – If a coarse (outer) MBR does not intersect with a query location, none of the subsequent (internal) MBRs need to be searched. However, it is important to notice that the efficiency of such pruning depends on how well the MBRs were initially partitioned. Due to this result, different types of R-Tree variants exist, each of which partitions the dataset in a different way. The most popular variants of the R-Tree are the R+-Tree, and the R*-Tree, proposed by Sellis et al. [34] and Beckmann et al. [4] respectively.

The R+-Tree is very similar to the R-Tree, but slight differences do exist. For example, the R+-Tree aims to minimise MBR overlap of internal nodes by (possibly) putting a spatial object in multiple leaves of the tree. This allows the searching a:lgorithm to consider fewer regions than in an equivalent R-Tree. The cost of this is obvious – The R+-Tree may store redundant data, making it spatially more costly, but more efficient to query.

Similarly, the R*-Tree is closely derived from the standard R-Tree. However, the R*-Tree uses a more complex construction process which involves node reinsertion and optimisation. The outcome of this process is that the R*-Tree will be faster to query than the equivalent R-Tree, at the cost of a longer construction time. Figure 5 shows an example R-Tree.

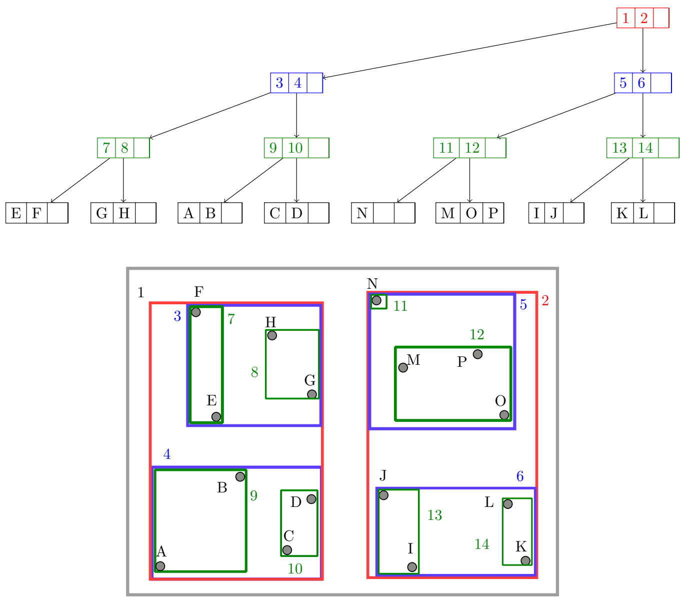


Figure 5: An example R-Tree with a fanout of 3, and minimum of 1 entry per node.

Finally, there exists a compressed version of the basic R-Tree, known as the CR-Tree [22]. The caveat is that in order to compress the tree, the MBRs must be quantized, turning them into *Quantised Relative Representation of Minimum Bounding Rectangles* (QRMBRs). The QRMBR implementation ensures that is no false-negatives, although false-positives are possible. In this context, a false-negative is when a QRMBR a is thought to not overlap with QRMBR b, but in fact, they do overlap. Conversely, a false-positive is when a is thought to overlap with b, but it does not. This tree performs $2.5\times$ better than the standard R-Tree in a disk-resident situation, while saving up to 60% of memory usage. It is important to note that these results are likely due to the optimised disk-access that the CR-Tree provides.

3 Spatial Information Retrieval

Spatial Information Retrieval refers to the problem of efficiently retrieving relevant documents with respect to both textual and spatial relevancy to a user query. In order to efficiently support spatial IR, traditional spatial and textual data structures must be combined together to produce hybrids that are capable of satisfying both spatial and textual constraints.

3.1 Current Approaches

Hybrid approaches tie together the support for both textual and spatial querying. That is, a hybrid data structure can efficiently support both aspects of a spatially-aware IR task. Hybrid structures are capable of answering more advanced queries than their spatial counterparts. Assume that a user has provided a location point p, distance n, and a query q. The following hybrid query types may be answered:

- **Conjunctive k-Nearest-Neighbour query:** Retrieve the k nearest objects to p that contain all terms $t \in q$.
- **Conjunctive Range query:** Retrieve all objects within distance n of p that contain all terms $t \in q$.
- **Disjunctive k-Nearest-Neighbour query:** Retrieve the k nearest objects to p that contain one or more terms $t \in q$.
- **Disjunctive Range query:** Retrieve all objects within distance n of p that contain one or more terms $t \in q$.
- **Top-k k-Nearest-Neighbour query:** Retrieve the top-k objects with respect to combined distance and textual relevance.

For each Boolean query, the results list would usually be ranked by textual relevancy. Note that this is different to the Top-k knn query, which will find the top-k items according to a ranking function based on a linear combination of spatial and textual aspects of each document. It is obvious that hybrid structures are capable of answering location-aware queries without the aid of any other system, unlike the spatial-only structures. However, spatial and textual data structures can be combined together to form *loose-combination* hybrid indexes, which are different to *tightly-coupled* hybrids. There are different trade-offs between these representations.

Loose combination. Loose combination hybrid indexes involve separate data structures for the textual and spatial components. These structures are not 'pure' hybrid structures, but better described as a 'segmented' structure. Due to the loose coupling of the spatial and textual structures, it is simple to implement them separately, and allows a 'mix-and-match' approach in which different structures can be easily coupled together. An example of this could be using an R-Tree to filter documents spatially, and passing the result into the IR system to complete the query. This is known as *space-first* processing, where the spatial component returns the spatially relevant objects to the textual component, which then filters the textually relevant objects from this candidate set. Conversely, *text-first* involves processing the textual query through an IR system, and passing the textually relevant objects into a spatial structure to filter out the spatially relevant results. However, Christoforaki et al. [12] show that a spatial structure is not necessary for text-first processing – A simple array can be used to look up the geo location of a document, and the document can either be added to the results list, or discarded if the location constraint was not met. The efficiency differences in text-first and space-first would largely depend on the size of the given textual and spatial datasets. For example, a text-first pass over a very large textual collection may take much longer than the space-first pass, as the space-first pass would allow a small subset of candidate documents to be ranked, whereas the text-first pass may score thousands or millions of spatially-irrelevant documents before filtering the spatially relevant documents. Conversely, text-first will certainly outperform space-first processing when there are few textually relevant documents to process for a given query. Zhou et al. [40] combine the R*-Tree with the inverted file in three different ways. Firstly, a 'double'-index is created, which is simply a spatial R*-Tree and a separate inverted index. In order to query the structure, the spatial and textual results are retrieved from their corresponding systems and the results are merged. Secondly, a text-first index was implemented, and finally, a space-first index was implemented. The text-first implementation performed slightly better than the space-first, which was a large improvement over the double-index.

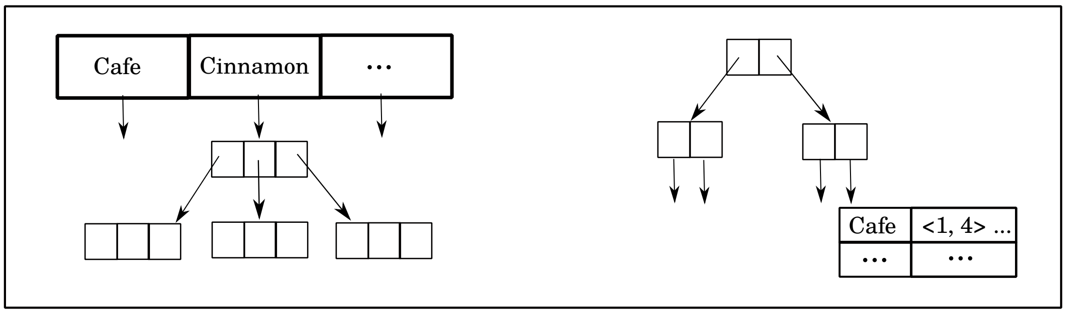


Figure 6: Two simple examples of loose combination hybrid index formats. The leftmost version is a text-first organisation, with an R-Tree (or similar) underneath each term in the postings list. The rightmost version is the space-first method, where each object in the leaf of the spatial tree has postings lists associated [40].

Tight combination. Tight combination hybrid indexes are data structures in which both textual and spatial information is held. Tight combination indexes are hybrids between spatial data structures and textual indexes, which allows a more seamless processing of location-aware queries. Tight combination structures have the ability to process both spatial and textual information at the same time, due to the organisation of the data structure. One such example of this is the IR-Tree, proposed by Cong et al. [14].

IR-Tree. The IR-Tree structure is based on that of the R-Tree, but additional textual information is stored in the branches of the tree, known as psuedo-nodes. The psuedo-node at a given internal node of the tree will store the maximum score of every term that occurs in the sub-tree rooted at the internal node. Figure 7 shows an example psuedo node for the term 'cinnamon' in a small subset of an IR-Tree. This allows simultaneous pruning of both spatially and textually irrelevant branches, reducing the size of the search space quickly. The general search algorithm proposed with the IR-Tree and similar structures is the best-first search method, otherwise known as 'greedy' search, in which the search algorithm will travel toward the most 'promising' objects before examining the less hopeful items. The greedy algorithm is outlined in Algorithm 1.

Other IR-Tree variants, such as the CIR-Tree, DIR-Tree and CDIR-Tree [14] follow a similar algorithm for top-k spatial bag-of-words retrieval. The key difference of these data structures is the organisation of data in the internal and leaf nodes – clustering techniques are used to build the tree such that both spatially and textually similar documents occur together. This results in an improved

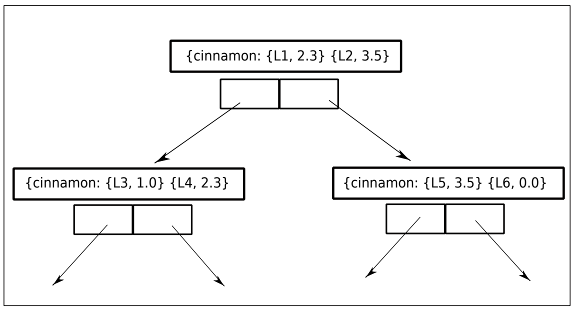

{cinnamon: {L1, 2.3} {L2, 3.5}
{cinnamon: {L3, 1.0} {L4, 2.3}
{cinnamon: {L5, 3.5} {L6, 0.0}

Figure 7: An example IR-Tree and pseudo-nodes for the term '`cinnamon`'. Note that all terms that appear in the tree below the current node will appear in the pseudo-node, a single word is shown here for simplicity. The score 0.0 in the node L6 shows that the term does not appear in any documents in the sub-tree of node L6.

Algorithm 1: Best-First IR-Tree traversal

```
1.1  function GETTOPk($q$, $IR$ , $k$)
1.2  $Ans \leftarrow \emptyset$
1.3  $Queue \leftarrow$ INSERT($root(IR), 0$)
1.4  while $Queue \neq \varnothing$ do
1.5      $Node \leftarrow$ NEXT_BEST_ELEMENT($Queue$)
1.6      if $Node$ is an Internal Node then
1.7          for $Child \in Node$ do
1.8              $Queue \leftarrow$ INSERT($Child$, SCORE($Child, q$))
1.9      else if $Node$ is a Leaf Node then
1.10         for $Object \in Node$ do
1.11             $Queue \leftarrow$ INSERT($Object$, SCORE($Object, q$))
1.12     else
1.13         $Ans \leftarrow$ INSERT($Object$, SCORE($Object, q$))
1.14         if $|\ Ans\ | = k$ then
1.15             return $Ans$
```

pruning of the search-space, at the cost of a much longer construction time. Li et al. [25] independently proposed a similar data structure called an IR-Tree (referred to as the Li-IR-Tree from here on). The key difference between the Li-IR-Tree and the family of trees proposed by Cong et al. [14] is that psuedo-nodes are stored in a concatenated form with the postings lists in the Li-IR-Tree, whereas the psuedo documents are stored separately in the IR-Tree. It is important to note that all of the mentioned IR-Tree variants are designed for on-disk use. Therefore, an in-memory Li-IR-Tree performs quite similarly to an in-memory IR-Tree.

S2I. Spatial Inverted Index (S2I) [32], is another type of tightly-coupled hybrid index. S2I uses a combination of inverted files and R-Trees to achieve good query times. Perhaps the most interesting aspect of S2I is that frequent items are stored seperately than infrequent items. In particular, S2I maps all objects containing a frequent term t to an aggregate R-Tree. This tree stores the maximum impact (the textual relevance score) of the objects in any subtree, rooted by the node of the subtree. In simple terms, the aggregate R-Tree is simply an IR-Tree for a single term. Infrequent items are stored in block-based postings lists. When a query is ran, each term t has a value v associated. If the v is above a precomputed threshold, then t is in the frequent items storage (an aggregate R-Tree). Otherwise, the infrequent items storage (postings lists) houses the information for term t. S2I is designed to answer top-k knn queries. Although S2I is shown to outperform IR-Tree, S2I uses significantly more space than the IR-Tree [10, 32].

SFC methods. Christoforaki et al. [12] proposed several hybrid indexes, each designed to answer ranked conjunctive Boolean range queries. The most relevant are described here.

The Text-First algorithm is a simple baseline that is shown to outperform R-Tree based approaches. Text-First does not rely on any spatial data structure, but rather stores a simple vector of the geolocation of each document. The query process is implemented on a DAAT IR system with block compression and skipping. A query simply retrieves the docID of any document containing all terms, checks if the document falls within the given query range, then scores the document using BM25 if it is indeed spatially relevant. Note that the final document ordering is ranked purely on textual relevance.

The first SFC algorithm from Christoforaki et al. [12] explored is the "space-filling curve, quad-tree" (SFC-Quad) algorithm. Firstly, SFC-Quad reorders the document set such that it corresponds to the order of a Z-curve [33]. This reordering clusters spatially similar documents together in a block-compressed inverted index. Next, a Quad-Tree [17] is built over the document set. At query time, the Quad-Tree is used to determine a set of ranges that relevant documents may fall within. Then, each range is traversed using a similar approach to Text-First.

Finally, the "space-filling curve, skip" (SFC-SKIP) algorithm is introduced. Similar to the SFC-Quad algorithm, documents are reordered by a Z-curve. Rather than using a Quad-Tree, however, SFC-SKIP does away with spatial data structures completely. An MBR is stored for each block of the compressed postings list, which encompasses the documents within the block. At query time, only blocks in which the MBR intersects with the query location are considered, and these are again scored in the Text-First manner. MBRs can also be stored for groups or larger blocks, which may promote hierarchial skipping. Both SFC-Quad and SFC-SKIP algorithms have similar time efficiency, but SFC-Quad was shown to be more space efficient.

3.2 Challenges in Location-Aware Search

Location-aware search has many unique challenges that must be met in order to have a successful system for location based queries.

3.2.1 Location ambiguity

Given a location `melbourne`, is the user referring to Melbourne in Australia, or Melbourne in the United States? A few remedies exist for such ambiguity. The system could simply return the most relevant documents for all places that match the given name. Another simple option is to give the user an option to select which `melbourne` they are referring to. For our dataset, the most populous location that matches the name is the assumed location. Further discussion about such solutions is out of the scope of this thesis, although it is worth noting that many smart devices automatically give the users current location via GPS. In this case, there is no ambiguity about the users' location.

3.2.2 Textual vs Spatial Relevance

Another challenge for such systems is balancing textual and spatial relevance. Given a set of documents $\mathcal{D}$ and a set of queries $\mathcal{Q}$, an example relevance judgement for document $d \in \mathcal{D}$ over query $q \in \mathcal{Q}$, where the location of $d = \ell_d$ and the location of $q = \ell_q$ could be defined as $f(q,d) = \alpha \cdot \beta(\ell_q, \ell_d) + (1-\alpha) \cdot \delta(q,d)$ where $\beta(q,d)$ is the spatial relevance component, $\delta(q,d)$

is the textual relevance component, and $\alpha \in (0, 1)$ is a parameter that can be used to add weight to either the spatial or textual component. Using a formulation such that α gives weight to either spatial or textual relevance is quite common in the literature [14, 25, 32, 38, 39]. Theoretically, values of α will influence both the time efficiency and the effectiveness of the retrieval run, as more influence to either component may allow spatial or textual pruning. However, this clearly depends on the type of structure in question. Both Li et al. [25] and Rocha-Junior et al. [32] found that α did not have noticeable effects on efficiency, whereas Cong et al. [14] found large differences in run-time with respect to the value of α and the given algorithms. Arguably, α should be set such that documents within a reasonable distance are returned. This is difficult, as a query may be in a very sparse or dense location, so α is hard to select before this is known. It is also worth noting that many formulations exist for $\beta(\ell_q, \ell_d)$ and $\delta(q, d)$, and these should be carefully selected based on the needs of the system.

3.3 Problems with Current Approaches

One of the biggest problems with the current approaches of location-aware search algorithms is that many are designed for use on disk [10, 12, 14, 25, 32, 38]. In-memory and disk organisations differ vastly, and this is very likely to effect the performance of current algorithms when converted to equivalent in-memory representations. This is particularly true with respect to the tree-based methods – Many of these approaches are heavily reliant on tree-traversal, which is known to be cache unfriendly. However, this is not true for all proposed structures, and largely depends on implementation details.

Another problem with some of the approaches is the memory usage. For example, both S2I and IR-Tree are shown to use a substantial amount of memory [10, 14, 25, 32], typically much more than the original text collection size. As such, these structures are not suitable for in-memory retrieval, as the space requirements far outweigh the plausible amount of memory a modern system contains for any reasonably large index.

Some approaches are based on spatial grids. That is, a grid is created such that each geographic object falls into only one cell, and generally, each cell may only contain a single object. However, the grid resolution would need to be incredibly high to accomodate each point into a unique cell, especially in highly dense locations such as a city centre. In turn, this leads to the grid structure having a very large number of cells, which can add additional space overhead.

Range-query based structures and knn based structures both have an inherent problem – How is the size of the MBR (or the value of k) chosen for a given query? An obvious solution is to have the user input the given value, but it is well known that users are not interested in "advanced" tasks. For example, Spink et al. [35] found that less than 5% of users made use of the advanced features they provided. Therefore, the system must either make an assumption, or use some formulation to decide on these values. This is not optimal, as it is impossible to tell how far a user is likely to travel without interacting with them.

Finally, many structures are focused on making the spatial retrieval aspect of the system highly efficient, and not considering the textual retrieval aspect. However, it has been suggested that the textual computation is where focus should be placed, as this is where the most computation must take place [12].

4 Filtering Techniques

A common approach for top-k retrieval in large search systems involves employing a cascaded retrieval approach, in which early filters are used to identify and refine a set of candidate documents, which can then be ranked using LTR or higher dependency models [2, 3, 26, 27]. The filtering approaches explored here are aligned with such cascaded retrieval systems, making them practical for use in large scale web systems.

4.1 Nearest Neighbour Filtering

Our baseline approach is a heuristic filter based on knn. Our approach differs from previous loosely coupled scoring methods [11, 12, 40] by guaranteeing that the initial k' documents derived from the spatial index are the k' nearest document neighbors for each query. Previous work used a fixed sized MBR to identify all documents within a pre-defined range, and considered only Boolean conjunctive queries between terms.

4.1.1 Algorithm

In this approach, a spatial tree is used to find the k' nearest neighbours to the query location ℓ_q, along with the distance of each candidate from the query point, which yields a candidate set $\mathcal{C}$, where $|\mathcal{C}| = k'$. Next, the candidate set $\mathcal{C}$ is passed to a modified DAAT query processor which runs as follows: For each candidate $c \in \mathcal{C}$, the combined spatial and textual score for query q is computed as $s(q, c)$. This process can be seen as a reordering of $\mathcal{C}$. Finally, the top-k items can be simply taken from the reordering and returned. There are two versions of this approach which we refer to as W-R* and W-kd. W-R* uses a bulk loaded R*-tree for knn queries, whereas W-kd uses a kd-tree for the knn filtering. Note that various k' values were tested, namely $k' = 1.5 \times k$, $2 \times k$ and $2.5 \times k$. However, only $k' = 2.5 \times k$ results are shown, as they are more effective than the other k' settings, with a negligable loss of efficiency.

Algorithm 2: Stage-One KNN WAND filter

```
2.1  function KNNFilterAndScore($q$, $\mathcal{S}$, $\mathcal{I}$, $k$, $k'$)
2.2  Ans ← ∅
2.3  Candidates ← GET_KNN($\mathcal{S}$, $\ell_q$, $k'$)
2.4  for $c \in$ Candidates do
2.5      score ← $\alpha \cdot \beta(\ell_q, \ell_c)$
2.6      score ← score + $(1 - \alpha) \cdot$ BM25$(q, c)$
2.7      Ans ← INSERT($c$, score)
2.8      if | Ans | > $k$ then
2.9          Ans ← DELETE_SMALLEST(Ans)
2.10 return Ans
```

4.2 Range Filtering

We also compare and contrast this approach with the more common filter-based approach which uses a range query to define the subset of documents that must be scored in the final text ranking stage. While not explicitly explored in previous work [11, 12, 40], it is relatively easy to support disjunctive, top-k, bag-of-words search queries using MBR constraints.

4.2.1 Algorithm

Firstly, an R*-tree is used to issue a range query, in which all documents within the given range are returned as a set of candidates, $\mathcal{C}$. Next, $\mathcal{C}$ is passed to a DAAT processor. This processor then evaluates all candidate documents and returns the top-k documents based on the textual score, BM25. For example, given an MBR of 10 km covering a query centroid, it is assumed that a user is willing to travel anywhere within this MBR, so the distance to the centroid is not relevant.

Three different MBR sizes were tested, namely MBR-1, MBR-10 and MBR-100, which uses MBRs sized at 1 km, 10 km and 100 km respectively, each centered over the query location.

Algorithm 3: Stage-One MBR WAND filter

```
3.1 function MBRFilterAndScore($q$, $\mathcal{S}$, $\mathcal{I}$, $k$, $MBR$)
3.2 $Ans \leftarrow \emptyset$
3.3 $Candidates \leftarrow$ GET_IN_RANGE($\mathcal{S}$, $MBR$)
3.4 for $c \in$ Candidates do
3.5     $score \leftarrow score + (1 - \alpha) \cdot \text{BM25}(q, c)$
3.6     $Ans \leftarrow$ INSERT($c$, $score$)
3.7     if $|\,Ans\,| > k$ then
3.8         $Ans \leftarrow$ DELETE_SMALLEST($Ans$)
3.9 return $Ans$
```

4.3 KNN vs Range filtering

Both knn and range oriented approaches have advantages and disadvantages with respect to each one another. Within these approaches, the different spatial systems are also relatively different, and should be selected depending on the needs of the IR system. These trade-offs are explored here.

Spatial Bounding Problems. As discussed in Section 3.3, both knn and MBR based structures generally make assumptions of the value of k and the MBR size, respectively. Consequentially, these assumptions make a large difference to efficiency times, due to spatial clustering. For example, let query q_1 = "`water fountain in sahara desert`". The likely result for using a generic MBR, say 100km^2, for q_1 would be 0 returned candidates – the Sahara desert is extremely large, and is unlikely to have any objects within hundreds of kilometers. However, the knn filter would simply return the k nearest results. Clearly, in this situation, the knn query is much more beneficial for a user – at least k' candidates can be reordered, yielding k results, which is better for the user than 0 results. Now, consider query q_2 = "`coffee shop new york city`". Let us assume a more strict MBR of 10km^2, and a large k value of 1000 (with $k' = 2.5 \times k$). There is likely to be thousands of documents located within this boundary. However, as before, the knn query will simply return the k' nearest documents. Clearly, in this situation, the reordered results from the MBR query are much more likely to suit the information need of the user. The knn query, on the other hand, is much less likely to find relevant textual documents from the 2,500 candidates it searched.

This issue is largely due to how clustered data is. It is likely that MBR queries are more suitable in highly clustered locations (at the expense of time efficiency), whereas knn queries have the advantage in sparse locations.

5 GeoWAND

A major limitation of the filtering techniques is that they rely on a heuristic method to filter candidate documents before a final ranking is computed. Consequentially, these approaches are not guaranteed to return the optimal top-k ranking. However, a carefully modified WAND traversal can guarantee score safety. The GEOWAND algorithm proposed here does provide this desirable property, while maintaining the ability to skip through many irrelevant documents without needing to score them. The general idea behind WAND and hence GEOWAND is to minimize the total number of documents scored. Only documents that can make the top-k based on the maximum score approximation will be scored. The primary difference to WAND and GEOWAND is that GEOWAND must also track the spatial relevance dynamically to determine if a document must be scored.

5.1 Algorithm

In addition to a standard inverted index, GEOWAND needs to maintain a list of geospatial coordinates for each document. These can be simply stored in a vector, which has a relatively small overhead with respect to the rest of the index. However, any efficient storage method will suffice.

Algorithm 4 shows the key steps involved in upper bounding the weighted spatial scoring estimate as documents are considered for final scoring, and a second score refinement stage once a candidate document is selected. Given a query q, with terms $t \in q$ and a location ℓ_q, the processing is as follows. In the initial candidate selection phase two contributions must be tracked, the *spatial_limit* and the *textual_limit*. The upper bound for the *spatial_limit* occurs when $\ell_q = \ell_d$. Each term *pivot* in document c_{pivot} has a global maximum textual score added to *textual_limit*. To keep the two scores normalized with respect to α, this upper bound is also added to *spatial_limit* on Line 4.14. In each iteration of the loop, the combined and normalized upper-bound textual and spatial scores are checked against the score of the lowest scoring document in a top-k min-heap, tracked with θ. If the *potential* total score for the current document exceeds θ, then the candidate document must be scored. Otherwise, the loop continues until the postings lists are exhausted, or when no more documents can score above θ.

When a candidate document must be scored, the true score of the candidate document is incrementally refined. First, the true spatial score contribution is updated for c_{pivot}.

Next, the textual contribution for each term in c_{pivot} is refined. The upper bound $U[t]$ for each term is iteratively substituted for the true textual score $\delta(t, d)$ of the term on Line 4.23. If the refined score of the candidate document is less than θ at any stage of this iteration, the document cannot make it into the heap, and execution is returned to the candidate generation loop.

Algorithm 4: WAND processing with geospatial weights

```
4.1  function GeoWAND($q, \mathcal{I}, k$)
4.2  for $t \leftarrow 0$ to $|q| - 1$ do
4.3      $U[t] \leftarrow \max_d\{w_d \mid (d, w_d) \in \mathcal{I}_t\}$
4.4      $(c_t, w_t) \leftarrow$ first_posting($\mathcal{I}_t$)
4.5  $\theta \leftarrow -\infty$                                          // Initial threshold value
4.6  Ans $\leftarrow \{\}$                                            // Empty top-$k$ heap
4.7  while the set of candidates $(c_t, w_t)$ is non-empty do
4.8      permute the candidates such that $c_0 \leq c_1 \leq \cdots c_{|q|-1}$
4.9      text_limit $\leftarrow 0$
4.10     spatial_limit $\leftarrow 0$
4.11     pivot $\leftarrow 0$
4.12     while pivot $< |q| - 1$ do
4.13         text_limit $\leftarrow$ text_limit $+ (1 - \alpha) \cdot U[$pivot$]$     // Set the textual upper-bound for term pivot
4.14         spatial_limit $\leftarrow$ spatial_limit $+ \alpha \cdot U[$pivot$]$     // Set the partial spatial upper-bound
4.15         if text_limit $+$ spatial_limit $> \theta$ then
4.16             $s \leftarrow$ text_limit $+$ spatial_limit
4.17             break, and continue from Step 4.19.     // The document can possibly make it into the heap
4.18         pivot $\leftarrow$ pivot $+ 1$
4.19     if $c_0 = c_{\text{pivot}}$ then
4.20         $t \leftarrow 0$                                   // Score document $c_{pivot}$
4.21         $s \leftarrow s -$ spatial_limit $+ \alpha \cdot \beta(\ell_q, \ell_{pivot})$     // Remove spatial estimate, add correct spatial score
4.22         while $t < |q| - 1$ and $c_0 = c_{\text{pivot}}$ do
4.23             $s \leftarrow s - (1 - \alpha) \cdot U[t] + (1 - \alpha) \cdot \delta(t, d)$     // Remove estimate for term $t$, add correct score
4.24             if $s < \theta$ then
4.25                 break, and reset all pointers to $c_{pivot} + 1$.
4.26         if $s > \theta$ then
4.27             Ans $\leftarrow$ insert(Ans, $(c_{pivot}, s)$)               // Insert $c_{pivot}$ to the top-$k$ heap
4.28             if |Ans| $> k$ then
4.29                 Ans $\leftarrow$ delete_smallest(Ans)
4.30                 $\theta \leftarrow$ minimum(Ans)
4.31         else
4.32             for $t \leftarrow 0$ to pivot $- 1$ do
4.33                 $(c_t, w_t) \leftarrow$ seek_to_document($\mathcal{I}_t, c_{pivot}$)     // forward pointers to $c_{pivot}$ or greater
4.34 return Ans
```

5.2 Document Scoring

To do the textual scoring, GEOWAND acts the same as WAND. Additionally, the coordinates are used to compute the normalised distance score for each candidate document that must be scored. The scoring operation that occurs for each candidate document is also slightly more complex. WAND must only score the textual relevancy, whereas GEOWAND must compute the normalised sum of both the textual and spatial scores before determining if the candidate document can be added to the top-k heap. This implies that a scoring operation in GEOWAND is typically more involved than its textual equivalent, and hence, it is important to harness skipping to improve efficiency without modifying effectiveness.

In our GEOWAND implementation, BM25 was used for textual ranking, but GEOWAND is not limited by this. In fact, most ranking models are suitable for use with GEOWAND. One difficulty that BM25 causes in the scoring computation is that BM25 is not a bounded score. This makes normalisation more difficult than in other approaches, where a score may be bounded by some maximum value. In any case, a ranking model can be used with GEOWAND so long as the textual and spatial relevance scores can be appropriately normalised with respect to one another. This ensures that the weightings provided by α are correct. As with WAND, the textual ranker must be additive with respect to terms, as this is how WAND allows skipping.

Formally, given a document d and a query q, the ranking function used in our GEOWAND implementation is described as:

$$\begin{aligned}
f(q,d) &= \alpha \cdot \beta(\ell_q, \ell_d) + (1-\alpha) \cdot \delta(q,d) \text{ where:} \\
\beta(\ell_q, \ell_d) &= \left(1 - \frac{\mathit{Euclidean}(\ell_q, \ell_d)}{d_{\text{max}}}\right) \cdot \left(\sum_{t \in q} U[t]\right), \\
\delta(q,d) &= \text{BM25}(q,d) \text{ as described in Section 2.2.4}, \\
\alpha &\in (0,1), \\
\mathit{Euclidean}(\ell_q, \ell_d) &\text{ is the Euclidean distance between } \ell_q \text{ and } \ell_d, \\
d_{\text{max}} &= \text{ a distance normalisation integer, and} \\
U[t] &= \text{ the upper-bound score for term } t \text{ across the collection}
\end{aligned}$$

Note that d_{max} will be explained further in the following section, as it is calculated differently depending on the version of GEOWAND that is used. Also note how α is used to weight the importance of the spatial and textual relevancy.

5.3 Global vs Local

In the initial implementation of GEOWAND, known as GEOWAND-global, the spatial estimation was found to be inducing additional and unneccesary postings lists to be examined, slowing the algorithm down. This bottleneck was identified, and as such, GEOWAND-local was introduced, which did improve the efficiency of GEOWAND. The following discussion refers to the formulation of $\beta(q, d)$ in Section 5.2.

In the case of GEOWAND-global, d_{max} is set to the largest Euclidean distance between any two documents in the spatial region. However, in the case of GEOWAND-local, d_{max} is set as the maximum Euclidean distance between ℓ_q and any other document in the collection. The outcome of this optimisation is that the value of β will converge to 0 more quickly than with the larger, global value. When a candidate document is selected for scoring, the true score of the candidate document is incrementally refined[1]. This incremental refinement allows early termination of the scoring loop on Line 4.24 of the psuedocode. So, GEOWAND-local essentially increases the likelihood of early termination, which means less postings lists must be examined. This spatial refinement takes place on Line 4.21.

5.4 Advantages and Disadvantages

Advantages. A large advantage of using GEOWAND is that it is very easy to add to an existing index, without the need of reindexing. The only addition to the index that is necessary is the spatial map, which stores the coordinates of each document in the collection. In our particular implementation of GEOWAND, we used a simple vector data structure. This data structure is shown to have negligable space overhead with respect to the size of the rest of the GEOWAND index. This simple representation also allows for $\mathcal{O}(1)$ lookup time for any latitude or longitude.

Another interesting and useful property is that as α approaches 0, GEOWAND reduces to WAND. Therefore, the GEOWAND algorithm is practical in that it can be used to efficiently and effectively solve both location-aware and non-location oriented bag-of-words queries without the need to change the algorithm. For example, consider an IR-system which services both spatial and textual queries. By modifying the value of α at runtime, better results can be provided, according to what sort of query was input.

Finally, and perhaps most importantly, GEOWAND harnesses the same skipping that the original WAND algorithm is known for. Due to the organisation of the WAND (and GEOWAND) indexes, it is incredibly efficient to traverse the postings lists. For example, cache invalidation is not an issue, due to the block-compression and the linear nature of the WAND traversal.

Disadvantages. A major disadvantage of systems that use the linear weighted scoring model that GEOWAND utilises is selection of α. Large values of α weight very spatially, which will degrade the Top-k knn search to more of a knn search, which will return close, but potentially irrelevant documents. On the other hand, weighting α to be very low will have the opposite effect – the query will degrade to a textual Top-k bag-of-words search. Many textually relevant documents will be returned, but they may be located in locations that are not close to the user. In addition to this problem, there is not likely to be a single value of α that is effective for every query. Therefore, the current 'fixed' value of α may not be optimal.

In addition to the problem of α selection, the ranking method may be less intuitive to a user. For example, with the MBR filter, a user is guaranteed that the results supplied are within the MBR. However, with GEOWAND, the documents returned are query dependent, and may not be relevant.

[1]The observation that text scoring can be incrementally refined was originally made by Gog and Petri [18]. Here we extend this idea to improve pruning for non-textual features.

5.5 Example Execution

Let query q contain 2 terms, t_1 and t_2, where $U[t_1] = 3.2$ and $U[t_2] = 5.8$. We now score 2 documents, d_1 and then d_2. Assume that d_1 contains only t_1, and d_2 contains both query terms, where $\delta(t_1, d_1) = 1.2$, $\delta(t_1, d_2) = 2.4$ and $\delta(t_2, d_2) = 4.2$. Let $\beta(\ell_q, \ell_{d_1}) = 1.2$ and $\beta(\ell_q, \ell_{d_2}) = 2.6$. Finally, let threshold $\theta = 2.9$, and $\alpha = 0.5$. We will now show the steps taken in processing both d_1 and d_2 sequentially.

1. Get the next candidate document, d_1.
2. Set the *spatial_limit* and *textual_limit* as the normalised upper-bound textual score, then set s. *spatial_limit* $= 0.5 \cdot 3.2 = 1.6$, *textual_limit* $= (1 - 0.5) \cdot 3.2 = 1.6$, and $s =$ *spatial_limit* $+$ *textual_limit* $= 1.6 + 1.6 = 3.2$
3. Check to see if the sum of the spatial and textual estimates outweighs the weight of the threshold: $3.2 > 2.9$. Therefore, we must score this document.
4. Firstly, refine the spatial score. $s = s -$ *spatial_limit* $+ \alpha \cdot \beta(\ell_q, \ell_{d_1}) = 3.2 - 1.6 + 0.5 \cdot 1.2 = 2.2$.
5. Since $s < \theta$, this document can not make the final results list, so we break out of the scoring block, and score our next candidate document.
6. Get the next candidate document, d_2.
7. Set the spatial limit as the upper-bound textual score, as well as the text limit. In this case, we set both the spatial and textual limit as the normalised sum of the upper-bound scores for t_1 and t_2, since both terms appear in document d_2: *spatial_limit* $= 0.5 \cdot (3.2 + 5.8) = 4.5$, *textual_limit* $= (1 - 0.5) \cdot (3.2 + 5.8) = 4.5$, and $s =$ *spatial_limit* $+$ *textual_limit* $= 4.5 + 4.5 = 9.0$
8. Check to see if the sum of the spatial and textual estimates outweighs the weight of the threshold: $9.0 > 2.9$. We must score this document completely.
9. Firstly, refine the spatial score: $s = s -$ *spatial_limit* $+ \alpha \cdot \beta(\ell_q, \ell_{d_2}) = 9.0 - 4.5 + 0.5 \cdot 2.6 = 5.8$.
10. Since $s > \theta$, we continue processing this document, as it can still make the results heap.
11. Now, we refine the textual score for each term. t_1 refinement: $s = s - (1 - \alpha) \cdot U[t_1] + (1 - \alpha) \cdot \delta(t_1, d_2) = 5.8 - 0.5 \cdot 3.2 + 0.5 \cdot 2.4 = 5.4$
12. Since $s > \theta$, we continue, as d_2 can still make the heap.
13. t_2 refinement: $s = s - (1 - \alpha) \cdot U[t_2] + (1 - \alpha) \cdot \delta(t_2, d_2) = 5.4 - 0.5 \cdot 5.8 + 0.5 \cdot 4.2 = 4.6$
14. Now, we have refined the score entirely. Since $s > \theta$, document d_2 will make the top-k heap, as it scores higher than the lowest scoring document that is currently in the heap.
15. Remove the lowest scoring value from the heap, and add d_2.
16. Update the heap theshold θ for the next document.
17. Continue, until all candidates have been searched.

This execution outlines the early termination that GEOWAND utilises – Since the refinement steps can only decrease the score of the candidate document, then it becomes clear that d_1 cannot possibly score high enough to enter the heap in step number 5. of our example. At this point, we need not process this document any more, so we move on to process the next document.

6 Experiments

6.1 Experimental Setup

All experiments were executed on a 24-core Intel Xeon E5-2630 with 256 GB of RAM hosting Red-Hat RHEL-v6.3. Postings lists were generated using Indri, stopping, and Krovetz stemming. Each posting list was then extracted, compressed, and stored in blocks of 128 entries using the FastPFOR library [24] to support skipping. For ranking, a WAND-based variant of BM25 was used with with $k_1 = 0.9$ and $b = 0.4$[2]. All algorithms were implemented in C++ and compiled with `-O3` flags, and all experiments were ran entirely in-memory. Unless otherwise stated, we report the mean of 10 consecutive runs of each query. For all experiments reporting GEOWAND timings, the default algorithm used is GEOWAND-local.

Search accuracy is measured using Maximized Effectiveness Difference (MED), and scored with RBP and a persistence $p = 0.95$ ($\text{MED}_{\text{RBP0.95}}$) [13, 37]. This approach allows us to quantify the effectiveness differences between runs without requiring relevance judgments.

6.2 Dataset

Experiments are conducted using a subset of the TREC[3] ClueWebB collection, known as CW09BGEO. Since this collection is not geotagged, the geotagging process is outlined here. Firstly, a Redis[4] database was created which stored tuples of the form `(location_name, latitude_longitude)`. These tuples were taken from GeoNames[5], and ordered such that the most populous locations would take preference over other locations with the same name. Secondly, location names in each document were extracted from the Freebase annotations of the ClueWeb Corpora[6], and were used to retrieve geographic coordinates that are associated with the location name from the Redis DB. Finally, these coordinates were then associated with the given DocID. The query set used is the 2009 Million Query Track, and the query set was geotagged in the same way as the document collection. A single location is used where multiple are specified for a document. In queries that have multiple locations specified, the query has been replicated as a one-location query for each specified location. For example, consider the query "`beaumont port arthur airport`". This query would be broken into two separate queries, with the location of `beaumont` supplied for one, and the location of `port arthur` supplied for the other. This resulted in 24 million unique, geotagged documents, and 5,315 unique geo-queries (Figure 8). Since some documents contained multiple locations, only the first location has been considered for each.

An issue with the TREC collection is the granularity of location data. Since document locations were found using natural language processing methods, many documents will have the same location name associated. Hence, the database lookup for such location names will result in documents having exactly the same coordinates. For example, there could be 100 documents with the location "*melbourne*" associated. All 100 of these documents would consequentially have the same latitude and longitude data associated to them. This is clearly not representative of geotagged documents, as documents would generally have high granularity geotags, placing them at a particular point inside

[2]The values for b and k_1 are different than the defaults reported by Robertson et al. [31]. These parameter choices were reported for Atire and Lucene in the 2015 IR-Reproducibility Challenge, see `github.com/lintool/IR-Reproducibility` for further details.

[3]trec.nist.gov

[4]redis.io

[5]download.geonames.org/export/dump/

[6]lemurproject.org/clueweb09/FACC1

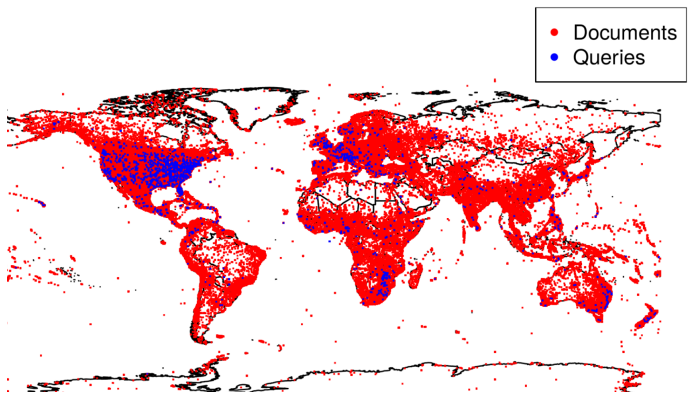


Figure 8: The graphical representation of the document set $\mathcal{D}$ and Query set $\mathcal{Q}$ in the CW09BGEO dataset. Note the clustering of both documents and queries around major cities and populous areas.

the city of Melbourne, for example. Another problem with such data is that nearest-neighbour queries may process much more than the desired k points, depending on the tie breaking scheme. For example, some implementations will return $n \gg k$ values if it happens that there are objects with a tied distance from the query location. Table 1 outlines this issue across real knn queries in the CWB-geo dataset. In order to make data more realistic, and to solve the knn problem, a simple 'scrambling' function was used to modify the geotags for each document, so that most documents have completely unique geotags. This scrambling essentially shifts a point randomly by no more than a few meters, which allows many more unique locations without modifying the results. See Algorithm 5 for further details.

k	Number Returned	Number Unused
100	8,000	7,900
1,000	9,000	8,000
10,000	20,000	10,000

Table 1: Some examples of the average number of candidate documents returned using the R*-Tree k-nearest-neighbour query over the CW09BGEO dataset. Note that a traditional k-nearest-neighbour query would return only k values, but with many duplicates, the algorithm may return $n \gg k$ values, which is undesirable. Results are rounded to the nearest hundred.

Before "scrambling", there were 150,000 unique locations across the document set, which was expanded to 23,413,629 unique locations. Queries were left unscrambled, which resulted in 2,000 unique query locations across the 5,315 queries.

A second dataset was created for use in preliminary testing. This smaller dataset is a subset of the CW09BGEO dataset, and is simply the first 1 million documents from the CW09BGEO dataset ordered by spam-score [15]. This dataset is referred to as WIKIGEO since a large number of geotagged Wikipedia documents are in the collection as these documents are often the highest ranked documents by spam score. A summary of the datasets can be found in Table 2.

Algorithm 5: Scramble coordinate

```
5.1 function SCRAMBLEPOINT($p$)
5.2 $flip \leftarrow$ RANDOM_BOOL()
5.3 $new \leftarrow$ RANDOM_FLOAT_IN_RANGE$(0, 0.0000000001)$
5.4 if $flip$ is true then
5.5     $p \leftarrow p + new$
5.6 else
5.7     $p \leftarrow p - new$
5.8 return $p$
```

Dataset	Docs	Unique terms	Unique locations
CW09BGEO	23,781,142	53,523,166	23,413,629
WIKIGEO	1,000,000	2,530,440	999,079

Table 2: Statistical summary of the two collections used in the experiments.

6.3 Implementation

The STR-R*-Tree used in experiments is a modified version of the libspatialindex R*-tree[7]. This version was used by Brisaboa et al. [6] and is provided on their website[8]. Some modifications were made to the algorithms used by the R*-Tree to act more favourably for the tasks we required. The kd-Tree used is implemented based on the Nanoflann library[9]. The kd-Tree is optimised for in-memory usage with 2D data. The IR-Tree was implemented using the C++ STL, based on the Java IR-Tree code that was used in the evaluation by Chen et al. [10]. Finally, the WAND and GEOWAND indexing and retrieval systems were based on the Succinct Retrieval Framework (SURF)[10].

6.3.1 Nomenclature

Table 3 outlines each system, and its experimental name in the following experiments.

6.4 Parameter Sweeps

Before any time or space readings can be taken from the tree-based data structures, the correct parameters must be selected. This ensures that the tree-based structures have the best performance in efficiently answering the provided stream of queries. The kd-Tree, R*-Tree and the IR-Tree were tested on the WIKIGEO dataset, and the kd-Tree and R*-Tree were tested across the CW09BGEO dataset to find out the optimal fanout for each tree and dataset, respectively. For the spatial-only trees, knn queries were tested on just the coordinates from the query set. For the IR-Tree, the full spatial-textual query set was used, α was set to an even spatial-textual weighting of 0.5, and the top-k items were returned for each query. Values of k tested were 100, 1,000, and 10,000. Note that $k = 10{,}000$ was not used in any further experimentation, as it was decided that $k = 10$ was a more suitable and practical value to test. However, subsequent tests showed that the chosen parameter settings were still optimal

[7]libspatialindex.github.io
[8]lbd.udc.es/research/serangequerying
[9]github.com/jlblancoc/nanoflann
[10]github.com/simongog/surf

System	Experiment Abbreviation
Standard WAND (text only)	W
GeoWAND	G-W
IR-Tree	IR
R*-Tree (knn queries)	W-R*
kd-Tree (knn queries)	W-kd
R*-Tree (1km MBR queries)	MBR-1
R*-Tree (10km MBR queries)	MBR-10
R*-Tree (100km MBR queries)	MBR-100

Table 3: Summary of the names of data structures that were used in experiments. Note that the W-R*, W-kd, and MBR-k methods all use a WAND index to complete queries once the spatial filtering has taken place. It is also important to realise that W-R* and MBR-k are the same data structures performing different query tasks.

for $k = 10$. Every tree was tested with fanout settings of 5, 10, 15, 20, 25, 30, 50, 100, 150, 200, 250 and 300. Only the most competitive fanout timings are shown for each test run.

6.4.1 WikiGeo

Figures 9, 10 and 11 show the results for the R*-Tree, kd-Tree and IR-Tree respectively. For the R*-Tree, there is very little difference for any of the tested fanouts. The kd-Tree had even less variation in the tested fanouts, though it was optimal for much larger fanout values than the R*-Tree. This is likely due to the different splitting and grouping policies that the trees use. It is also worth noting that the kd-Tree and R*-Tree are slightly more spatially efficient as the fanout size increases, but it is almost negligable. Finally, the IR-Tree timings are shown to be the least consistent with respect to the fanout values. The larger differences are due to the fact that the IR-Tree is structured such that the fanout value makes a large difference on the storage of the psuedo-nodes. As the IR-Tree fanout becomes larger, the number of psuedo-nodes that must be stored is decreased, but this leads to many more branches being available for searching, so there is a time-space trade-off with respect to the tree capacity. As we are optimising for time, we selected the capacity with the best time across the differing values of k.

6.4.2 CW09BGeo

Figures 12, and 13 show the results for the R*-Tree and kd-Tree. Notice that the IR-Tree results are not shown here. This is because the IR-Tree could not be built for the larger CW09BGEO collection, as the space usage was too great. This is discussed further in Section 6.7.2. Similar to the results in Section 6.4.1, the R*-Tree performed best with smaller fanout values than the kd-Tree, which favoured larger values. Again, this is simply due to the layout of such trees in memory, as their splitting policies and organisation are very different. Table 14 shows the selected fanout for each tree across the tested datasets. These are the fanouts used in each subsequent experiment.

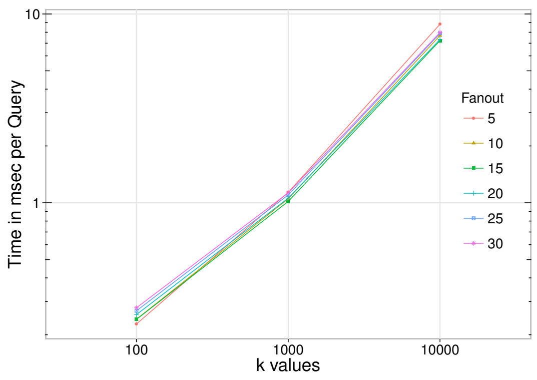


Figure 9: R*-Tree timings for differing fanouts over the WIKIGEO collection.

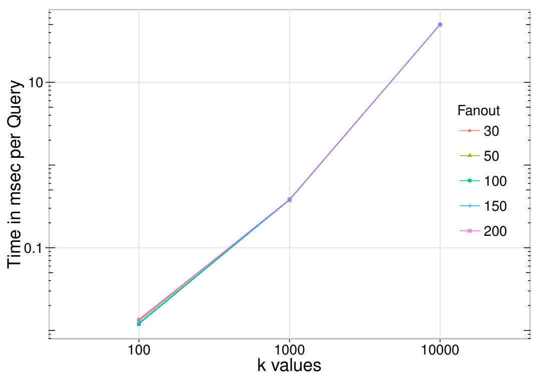


Figure 10: kd-Tree timings for differing fanouts over the WIKIGEO collection.

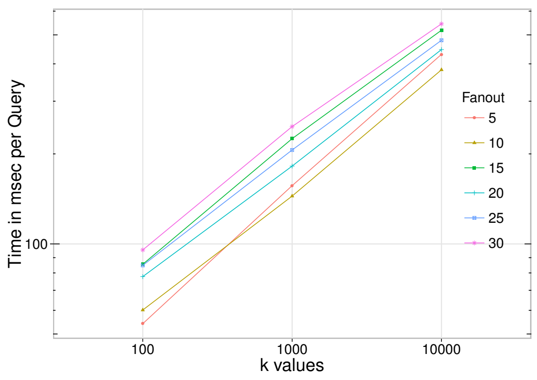


Figure 11: IR-Tree timings for differing fanouts over the WIKIGEO collection for $\alpha = 0.5$.

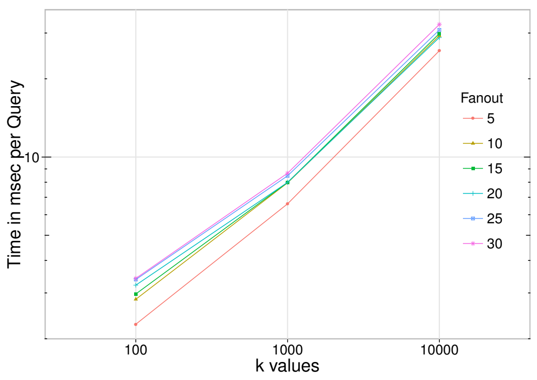


Figure 12: R*-Tree timings for differing fanouts over the CW09BGEO collection.

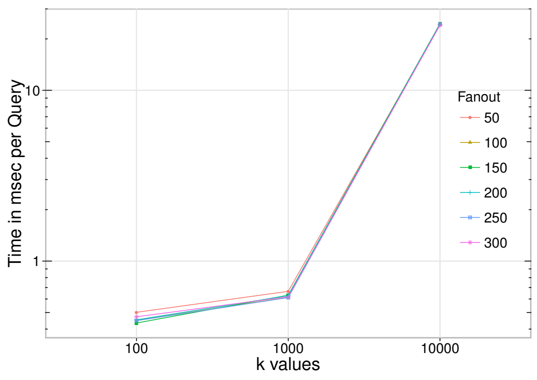


Figure 13: kd-Tree timings for differing fanouts over the CW09BGEO collection.

System	Dataset	Fanout
kd-Tree	WIKIGEO	100
R*-Tree	WIKIGEO	15
IR-Tree	WIKIGEO	10
kd-Tree	CW09BGEO	150
R*-Tree	CW09BGEO	5

Figure 14: Summary of the best tree fanout for each spatial index – collection combination.

6.5 Time-Space Trade-offs

The first set of experiments are aimed at answering RQ 1. That is, we are interested in the time and space trade-off for each data structure. The scalability of the IR-Tree is of particular interest, as previous research has shown the approach to have a significant space overhead [10], but to be efficient for small values of k. GEOWAND and the knn filtering efficiency is also of particular interest, as these methods are yet to be explored in the literature. The time and space experiments were ran over both the WIKIGEO collection, as well as the larger CW09BGEO collection.

6.5.1 WikiGeo

Space Efficiency. Figure 15 shows the relative spatial storage costs for each indexing approach for the WIKIGEO collection. We show the different space costs of the IR-Tree as a function of the tuning parameter `fanout`. Note that the total size of the original uncompressed document collection is around 9 GB. So the space overhead for the IR-Tree is significant, approaching 2x the size of the original collection, and 20x the size of the compressed WAND index. In fact, the space overhead for the IR-Tree is so substantial, we were unable to construct the index for the full CW09BGEO collection. It is clear that more focus should be placed on using well known methods for text indexing and compression in future hybrid indexing experimental comparisons, as the current methods are neither scalable nor friendly to in-memory traversal.

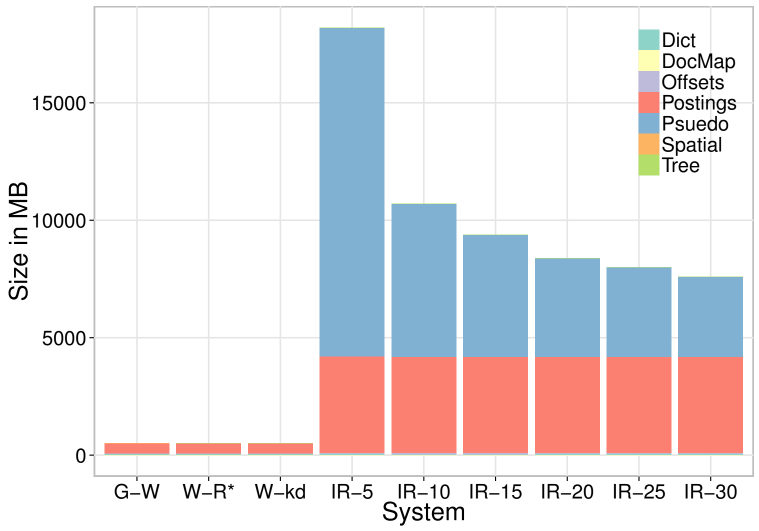


Figure 15: A comparison of the space usage for a GEOWAND inverted index, a loosely coupled WAND inverted index combined with an R*-Tree and kd-Tree, and IR-Tree of varying fanouts across the WIKIGEO collection. The space difference between the different WAND and GEOWAND indexes are negligible. Recall that W-R* is the same data structure as the MBR-k structures, so the space usage is the same.

Time Efficiency. Figure 16 shows the average processing time per query for $k = 10$, 100, and 1,000. Note that the MBR and knn filtering methods MBR-1, MBR-10, MBR-100, W-R*, and W-kd are not rank-safe. So, despite the clear efficiency advantages, some loss of effectiveness does occur. Effectiveness will be explored further in the next section. The IR-Tree (IR) and GEOWAND (G-W) methods are both safe-to-k. We can see that the efficiency gap between GEOWAND and IR-Tree grows as k increases. At $k = 1{,}000$, there is an order of magnitude difference between the running time of the two algorithms.

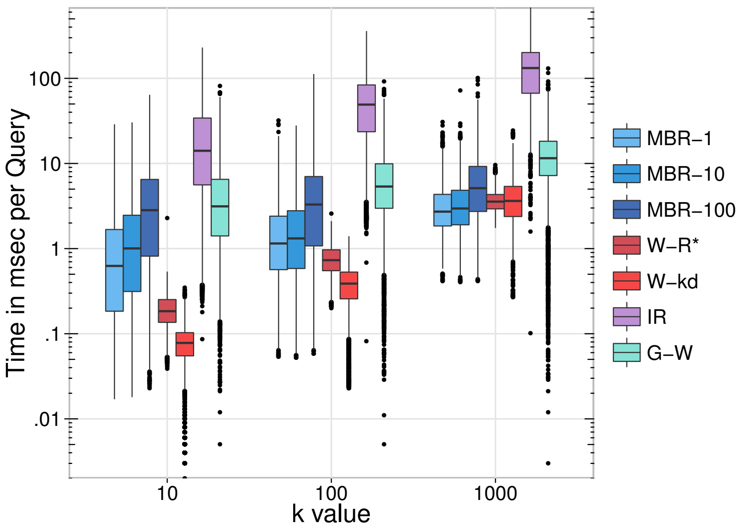


Figure 16: The mean running time per query as a function of k for all query type and index combinations on the WIKIGEO dataset, with $\alpha = 0.5$.

Effect of Alpha. Figure 17 shows the effect that α has on GEOWAND and IR-Tree indexes. As α approaches 1, GEOWAND becomes increasingly less efficient. This is because the added estimation of the spatial component makes it more difficult to skip documents, and the more weight given to β, the less likely skipping or early-termination is to occur. Similarly, the IR-Tree became less efficient as α approaches 1, but the extent in which the efficiency dropped is less for each 0.1 step increase in α. Note the anomaly at $\alpha = 0.8, k = 1{,}000$ for the IR-Tree. This is likely caused by another process starving the IR-Tree experiment of resources.

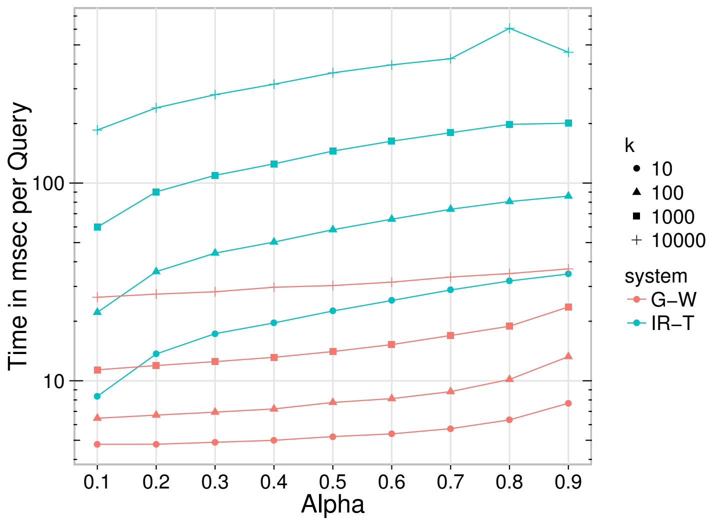


Figure 17: The effect of α on GEOWAND and the IR-Tree for the WIKIGEO dataset.

6.5.2 CW09BGeo

Note that the IR-Tree is no longer considered. As previously mentioned, the space overhead is too large to build an index for the CW09BGEO collection.

Space Efficiency. The space usage for the 3 different index types is shown in Figure 18. Each index utilises the same textual support data, but each has a differing spatial data structure. In this regard, it is seen that GEOWAND stores a smaller spatial component than the R*-Tree on top of the WAND index. However, the kd-Tree is smaller than the spatial vector used in GEOWAND. There is little difference between the size of each spatial structure, which confirms that most focus should be applied on ensuring the textual index is efficient. Recall that although the R*-Tree index is larger, it has the benefit of allowing both MBR and knn queries, whereas the kd-Tree can only support knn queries. In the GEOWAND index, the `Spatial` component can almost be combined with the `DocMap`, as the spatial data is stored in the docmap on disk. However, seperate figures are shown, as the structures differ when loaded into memory.

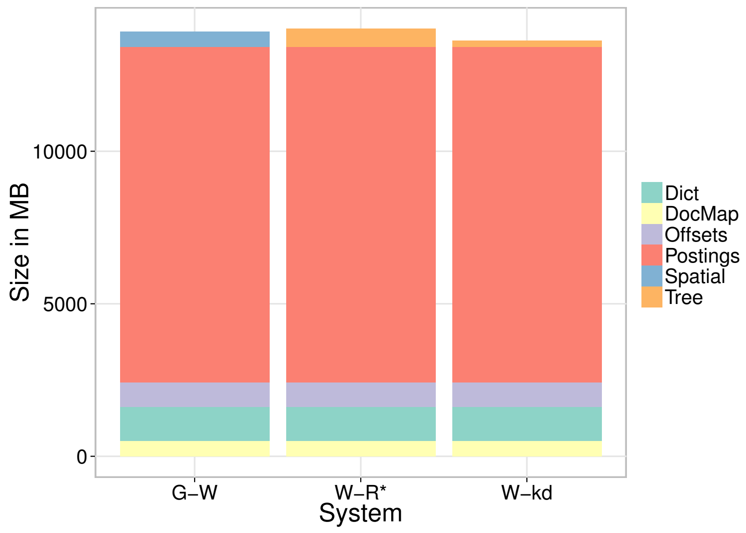


Figure 18: A comparison of the space usage for a GEOWAND inverted index, and a loosely coupled WAND inverted index combined with an R*-Tree and kd-Tree across the CW09BGEO collection. Recall that W-R* is the same data structure as the MBR-k structures, so the space usage is the same.

Time Efficiency. In the larger dataset, the relative efficiency of each system was very similar to that of the WIKIGEO dataset. Notice that a standard WAND run was included in the results, to show how the spatial runs compare to a regular query run. As k increases, GEOWAND and WAND have less difference in efficiency. The knn filtering methods are generally an order of magnitude faster than GEOWAND, and more efficient than the Range query for all values of k, even across the small MBR of 1km^2. Again, this is likely due to the MBR methods being centered over very populous areas, where there are many candidate documents that must be examined to find the top-k documents within that range.

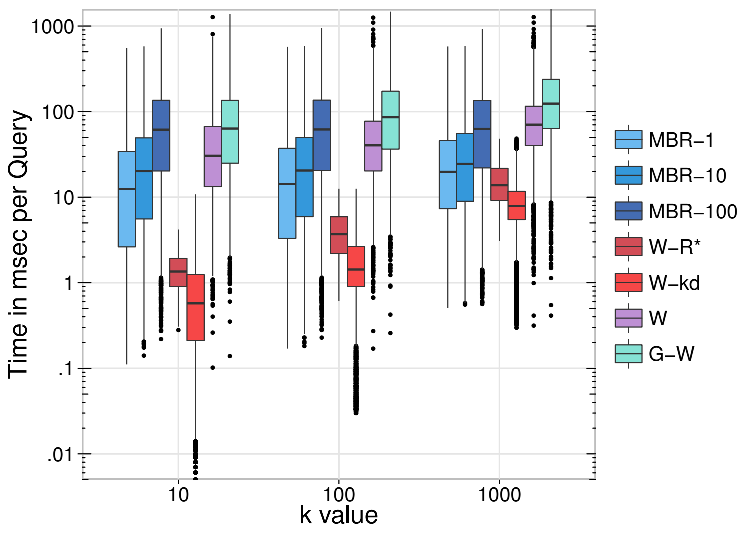


Figure 19: A comparison of the time with respect to k for filtering and GEOWAND systems on the CW09BGEO dataset. Note that $\alpha = 0.5$ in this experiment, and that W represents a standard textual WAND timing run across the textual query only.

6.6 Efficiency-Effectiveness Trade-offs

We now turn our attention to Research Question 2, and the effectiveness of the filter-based queries. Recall that two major types of queries dominate the literature for location-aware search – Range, and knn spatial-textual queries. Both of these query types can be processed conjunctively or disjunctively. Here we focus on only bag-of-words disjunctive top-k querying.

As shown in Figure 16 and 19, filtered Range queries are efficient, but there is inevitably some loss in effectiveness. This is due to the diversity in queries that are observed in large query streams. Some queries cover very densely populated areas, while other query locations may originate in sparsely populated areas. However, many potential documents close to the query origin do not guarantee that these documents also match the keywords in the query – this is clearly a disadvantage to the knn based filters. Conversely, the highest scoring documents for the combined textual-spatial score could be just outside of the MBR range which is fixed at query time. This makes automatically bounding the size of the MBR difficult in practice.

Table 4 shows the Maximum Expected Difference (MED) for two common utility-based effectiveness metrics, ERR and RBP. For both metrics, we see that the effectiveness loss for all of the filtering methods is substantial when compared to a rank-safe knn approach. When comparing the MBR Range Query methods, we see two important trends. First, the overall effectiveness is never less than 0.2, even when the queries are heavily biased towards the spatial score. This is largely an artifact of query diversity. Some queries have many thousands of potential documents that fall within the query range, while other queries have only a few. Even when there are many potential documents, there is no guarantee that the documents contain the the keywords with the highest impact. Even fewer would be conjunctive Boolean matches.

The second important trend is that the Range queries tend to perform better than the loosely coupled knn query approach, even with a conservative k' of $2.5 \times k$. This is because it is even less likely that the documents very close to a query are textually relevant in a dense document region. For

α	MBRF 1 km	MBRF 10 km	MBRF 100 km	KNNF $k = 10$	KNNF $k = 100$	KNNF $k = 1000$
			$\text{MED}_{\text{RBP0.95}}$			
0.2	0.7164	0.6978	0.6601	0.9947	0.9936	0.9703
0.5	0.5991	0.5746	0.5246	0.9921	0.9902	0.9551
0.8	0.4412	0.4085	0.3432	0.9880	0.9849	0.9319
			$\text{MED}_{\text{ERR@20}}$			
0.2	0.6844	0.6646	0.6264	0.9655	0.8892	0.8746
0.5	0.5589	0.5332	0.4825	0.9473	0.8354	0.8149
0.8	0.4021	0.3682	0.3042	0.9220	0.7625	0.7406

Table 4: The Maximum Effectiveness Difference (MED) as measured with $\text{MED}_{\text{RBP0.95}}$ and $\text{MED}_{\text{ERR@20}}$. A MED value less than 0.2 is considered negligible. The higher the value, the more likely there is a noticeable effectiveness difference.

example, a $1km$ MBR may return $100{,}000$ documents in a densely populated area, whereas a knn query with $k' = 2.5 \times k$ and $k = 1000$ would only examine the $2{,}500$ nearest documents. Therefore, the Range query is much more likely to return satisfactory results in this situation.

These two observations further strengthen our belief that rank-safe scoring methods such as GEOWAND are in fact the simplest and most intuitive approach to ranking documents with more than one type of weighting constraint. The importance of location can be easily determined independent of the locational document clustering. Furthermore, the rank-safe scoring methods need not select a k' or MBR size at query time, making the query process much easier and consistent.

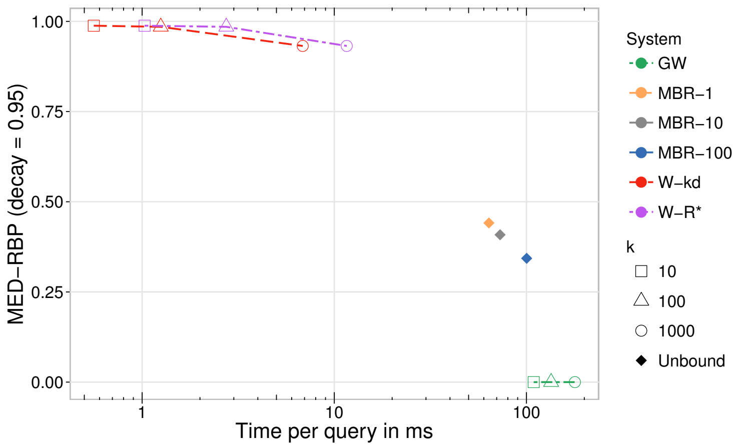


Figure 20: Effectiveness versus Efficiency for the CW09BGEO collection for $\alpha = 0.8$. The IR-Tree would appear on the same y coordinate as GEOWAND since it is safe-to-k, but it would be further right (as it would run slower than GEOWAND).

6.7 Evaluation

In this section, the filtering methods, IR-Tree, and GEOWAND are analysed and evaluated based on the experimental results accumulated in this work.

6.7.1 Filter Methods

In the mentioned versions of our filtering, different queries are supported. For example, the kd-Tree can only support knn filtering methods. On the other hand, the R*-Tree based filter can support both knn and MBR approaches, making it more flexible. More importantly, any reasonably efficient spatial structure can be utilised – If flexibility is required, the R*-Tree or any other structure that supports both query formats should be utilised. If the query type is fixed, then an appropriate structure for that particular query should be selected.

Although highly efficient, the filter based methods were not effective. In the worst case, the knn filter achieved very poor similarity to the results given by the safe-to-k, top-k knn approaches such as GEOWAND. If a knn filter is desired, the kd-Tree gives more varied, but overall more efficient results than the R*-Tree. Since the R*-Tree is more flexible, so if both MBR and knn queries are desired, the R*-Tree or a variant is the best way to go. Alternatively, MBRs can be calculated and stored independently from a spatial data structure, similar to the SFC methods [12].

6.7.2 IR-Tree

One good quality of the IR-Tree is that it is safe-to-k. That is, every query is guaranteed to have the top-k documents returned with respect to the ranking function. Therefore, the effectiveness of the IR-Tree is optimal with respect to the ranking function used, and better than any of the filtering methods which are not rank-safe, at the cost of both time and space efficiency.

One obvious disadvantage of the IR-Tree with respect to in-memory processing is the number of cache misses that occur. Since the tree structure is stored separately from the pseudo-nodes, which are in turn also separate from the postings lists, a single iteration of the best-first search may access all three data structures, which may be located far apart in RAM. This is thought to be the main bottleneck in the tree-based traversal techniques.

Another issue with the IR-Tree is the necessity of precomputing the textual scores for each term in each document. This leads to a large memory footprint, and compression of the pseudo-nodes is not possible without quantisation. Note that postings lists were stored in a similar format to the pseudo-nodes, which explains the increased size of the postings lists as compared to the compressed WAND postings. Another problem with precomputing the scores is that it makes changing the scoring approach very difficult. For example, if a different textual relevance metric was to be used, or even a single parameter changed (such as b in BM25), the entire IR-Tree would need to be rebuilt, which is a very time consuming process. Therefore, the IR-Tree is not particularly friendly to modifications.

As we saw in Figure 16, the hybrid IR-Tree index is less efficient than all of other approaches we tested, and the performance gap grows with respect to k. To better understand why the algorithm performed so poorly, we computed the number of branches in the IR-Tree that must be evaluated to determine the final top-k list. Figure 21 shows the number of branches as a function of k and queries processed. We can see that the number of branches traversed is significant for values of k greater than $1{,}000$, which are commonly used in multi-stage retrieval scenarios.

So, despite several previous papers reporting that IR-Tree is an attractive time and space trade-off for top-k knn queries, our experiments do not reach the same conclusion. The IR-Tree does not appear to be a competitive search algorithm for in-memory location-aware search in real-world web

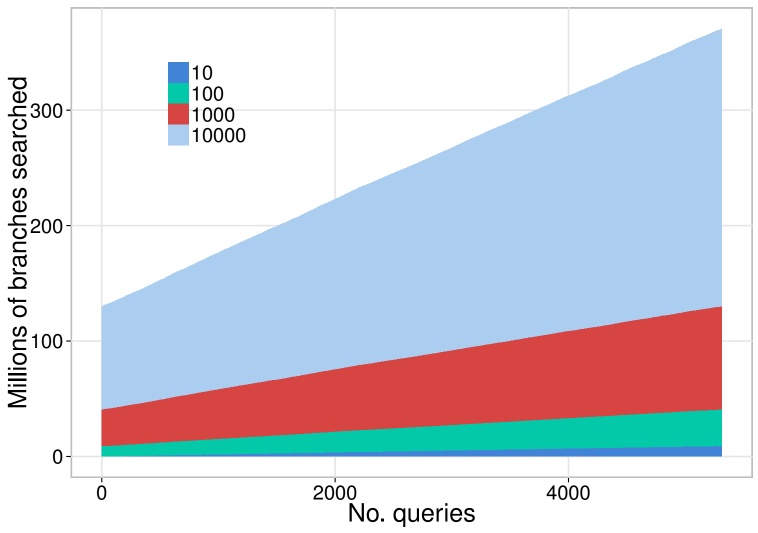


Figure 21: The cumulative number of branches evaluated for the IR-Tree traversal for $\alpha = 0.5$ on the WIKIGEO collection. The best-first search must search many more nodes as k increases, which results in a significant increase in the number of cache misses incurred when walking the tree.

indexes. It should be re-iterated that prior experimental studies of the IR-Tree reported times that were disk-based and not memory resident, and the textual objects were considerably smaller than the documents in our collection, as well as the selected values of k. So, the poor performance we have observed could be an artifact of the problem we are trying to solve, which may not match the original intent of the IR-Tree.

6.7.3 GeoWAND

GEOWAND has a few clear advantages over any of the filtered query approaches evaluated, as well as the IR-Tree. Firstly, GEOWAND allows a user to submit a location-aware query without the need to input any distance parameters. This is more intuitive than using the knn or MBR filters, in which a k' or MBR size must be selected by the user or decided a priori by the retrieval system. The obvious caveat here is that the α normalization must be carefully selected in order to provide the correct spatial and textual relevancy levels. Secondly, GEOWAND can be applied to an existing index without much effort. Reindexing is not necessary, and the only addition that must be made to the system is an efficient representation of the document locations. Thirdly, a GEOWAND index can be used to service standard textual queries without a loss of runtime efficiency or effectiveness compared to a plain WAND index. Finally, GEOWAND scales similarly to WAND, so it can be applied to much larger indexes efficiently.

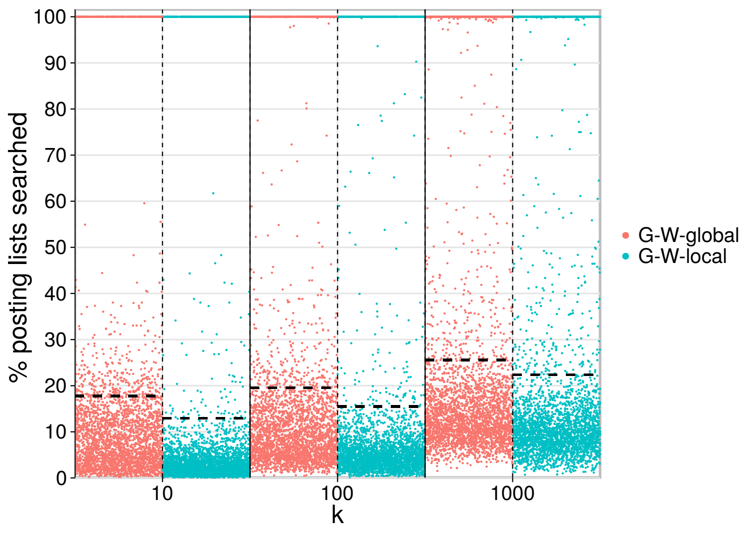


Figure 22: This graph shows the percentage of postings lists that GEOWAND evaluated with respect to an exhaustive run over all queries and k values for the CW09BGEO dataset with $\alpha = 0.5$. G-W-global refers to the version of GEOWAND that normalises each document with respect to a perfect spatial score. G-W-local is the improved version, which normalises each document based only the maximum possible score with respect to the query location, improving early termination. It is clear that GEOWAND continues to skip well with respect to k. The dotted line shows the average percentage of postings lists evaluated for each k.

Figure 22 shows the number of postings examined for both GEOWAND methods. Clearly GEOWAND benefits from the dynamic refinement of the spatial bound when $d_{\max}$ is query specific. This enhancement results in fewer candidate documents being fully scored since β decreases faster as the distance between the document and the query increases. We hypothesize that the efficiency could be improved further by reordering the documents spatially.

As previously mentioned, GEOWAND places no restriction in the method of retrieving the spatial coordinates of any document. In our implementation, a simple vector was used, which stored the latitude and longitude of each document. In this case, the added space requirement (in Bytes) for any arbitrary collection C can be computed as:

$$s = 2 \cdot \mathit{sizeof}(\mathit{double}) \cdot |C|$$
$$s = 16 \cdot |C|$$

It is important to notice that different representations may provide a more efficient storage solution, possibly at the cost of lookup time. For the purpose of this work, the vector based solution was sufficient.

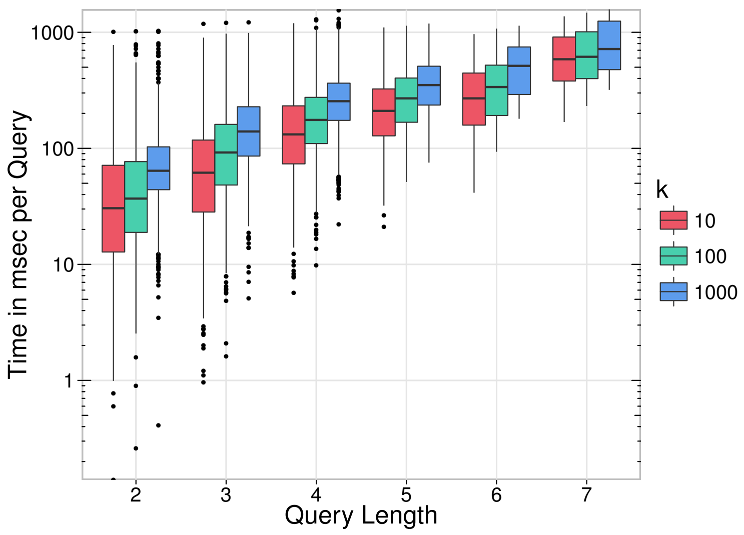


Figure 23: This graph shows the effect that the query length has on GEOWAND across the CW09BGEO dataset. GEOWAND still performs reasonably well for very long queries. Note that queries with length 1 are not considered, as these queries would be of geographical nature only with no textual keywords.

Finally, Figure 23 shows the relative performance of GEOWAND with respect to the length of a query. Shorter queries perform better, as less postings lists must be processed. One word queries are not shown, as these are queries where only a location is provided, but no true textual keywords are provided. Such queries would generally be handled in a geographic-only method, or a bag-of-words method, rather than the location oriented approach that GEOWAND applies. In the worst case, where $k = 1000$ for a 7 term query, GEOWAND takes around 700 milliseconds, which is quite reasonable, given how large the CW09BGEO dataset is. In the best case, GEOWAND is able to return the top 10 results for a 2 word query in 30 milliseconds, which is a very efficient result. Similarly, a top 1000 result list can be found in around 60 milliseconds. It is clear that the number of query terms has a much larger effect on efficiency than the value of k.

7 Conclusion

In this study, various approaches of location-aware web search were examined with respect to efficiency and effectiveness for knn and range-based filtering methods, and safe-to-k, top-k knn methods. Our experimental study has shown that while range-based methods are very efficient, the relative effectiveness of such systems is highly dependent on the properties of the spatial layout of documents and queries. Conversely, safe-to-k methods derived from top-k knn spatial-textual queries are capable of finding relevant documents regardless of the collection properties. This experimental evaluation answers our second research question.

We presented a variation of the textual WAND processing algorithm called GEOWAND which utilises spatial relevancy as well as textual relevancy to dynamically prune documents at retrieval time. GEOWAND is shown to be efficient, easy to implement and scalable, providing an attractive solution to research question 1. GEOWAND is shown to be an order of magnitude faster than a tightly coupled IR-Tree index, while being up to 15 to 20 times smaller.

One important optimisation that was not explored in this work is the document reordering using location based on space-filling-curves. Using an SFC or similar methods to reorder documents spatially has been shown to further increase the performance of location-aware search. However, we believe that these efficiency gains are orthogonal to our experimental study, as each system that was tested would benefit similarly from this enhancement.

8 Future Work

Clearly, cache-coherent, inverted index based structures provide a much more efficient representation for in-memory location-aware retrieval than the tree-based, spatially oriented data structures. However, there is still more room for improvement. For example, document reordering based on space filling curves has been shown to improve efficiency [12]. This could be applied to GEOWAND as well as other techniques.

An important addition to the experimental evaluation shown here is the SFC-Quad, and SFC-SKIP algorithms [12]. There was not adequate time to implement and test these algorithms, but they are shown to be very competitive. In future work, these should be implemented and compared with the systems shown here. We would expect these systems to be faster, but not any more effective than the MBR-based methods explored in this work.

Another interesting addition for GEOWAND is to extend it into a BLOCK-MAX version, based on the BLOCK-MAX WAND algorithm, proposed by Ding and Suel [16]. BLOCK-MAX WAND is shown to improve the performance of WAND substantially, and such performance gains are quite likely for GEOWAND too.

Additionally, further work could be completed on the spatial-filtering methods that were explored. For example, other types of queries could be explored, such as iteratively-deepening range queries (where the MBR is expanded a number of times). Conjunctive queries could also be explored.

Finally, succinct spatial data structures could hold the key to improving the performance of in-memory hybrid spatial trees. For example, the data structures explored by Brisaboa et al. [7] may be applicable to both the cascading and hybrid data structures explored here.

9 Acknowledgements

Firstly, I would like to extend my most sincere thanks to my advisor Shane, for his guidance, support, and patience over the course of my honours year, and for inspiring me to take part in IR research. I also must thank Shane for providing me with the funding that allowed me to complete this thesis. Secondly, I thank Farhana Choudhury for her vast knowledge of the literature in which this thesis was built upon, her help was invaluable. I also extend a thanks to all of the other students and friends for the various discussions across the semester, it was very helpful to interact with others working on research projects. Finally, I would like to thank my friends, family, and my partner, for providing endless support throughout my studies.

References

[1] Microsoft: 53 percent of mobile searches have local intent. `http://searchengineland.com/microsoft-53-percent-of-mobile-searches-have-local-intent-55556`. Accessed: 2015-09-03.

[2] N. Asadi and J. Lin. Document vector representations for feature extraction in multi-stage document ranking. *Information Retrieval*, 16(6):747–768, 2013.

[3] N. Asadi and J. Lin. Effectiveness/efficiency tradeoffs for candidate generation in multi-stage retrieval architectures. In *Proc. SIGIR*, pages 997–1000, 2013.

[4] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. *ACM SIGMOD Record*, 19(2):322–331, 1990.

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

[6] N. R. Brisaboa, M. R. Luaces, G. Navarro, and D. Seco. Range queries over a compact representation of minimum bounding rectangles. In *Proc. ER*, pages 33–42, 2010.

[7] N. R. Brisaboa, M. R. Luaces, G. Navarro, and D. Seco. Space-efficient representations of rectangle datasets supporting orthogonal range querying. *Information Systems*, 38(5):635–655, 2013.

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

[9] O. Chapelle, D. Metlzer, Y. Zhang, and P. Grinspan. Expected reciprocal rank for graded relevance. In *Proc. CIKM*, pages 621–630, 2009.

[10] L. Chen, G. Cong, C. S. Jensen, and D. Wu. Spatial keyword query processing: an experimental evaluation. In *Proc. VLDB*, pages 217–228, 2013.

[11] Y.-Y. Chen, T. Suel, and A. Markowetz. Efficient query processing in geographic web search engines. In *Proc. SIGMOD*, pages 277–288, 2006.

[12] M. Christoforaki, J. He, C. Dimopoulos, A. Markowetz, and T. Suel. Text vs. space: Efficient geo-search query processing. In *Proc. CIKM*, pages 423–432, 2011.

[13] C. L. A. Clarke, J. S. Culpepper, and A. Moffat. Assessing efficiency-effectiveness tradeoffs in multi-stage retrieval systems without using relevance judgements. `http://arxiv.org/abs/1506.00717`, 2015.

[14] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *Proc. VLDB*, 2(1):337–348, Aug. 2009.

[15] G. V. Cormack, M. D. Smucker, and C. L. A. Clarke. Efficient and effective spam filtering and re-ranking for large web datasets. *Information Retrieval*, 14(5):441–465, Oct. 2011.

[16] S. Ding and T. Suel. Faster top-k retrieval using block-max indexes. In *Proc. SIGIR*, pages 993–1002, 2011.

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

[18] S. Gog and M. Petri. Compact indexes for flexible top-k retrieval. In *Proc. CPM*, pages 207–218, 2015.

[19] Google. Understanding consumers local search behavior. `https://think.storage.googleapis.com/docs/how-advertisers-can-extend-their-relevance-with-search_research-studies.pdf`.

[20] A. Guttman. R-trees: A Dynamic Index Structure for Spatial Searching. In *Proc. SIGMOD*, pages 47–57, 1984.

[21] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Information Systems*, 20(4):422–446, 2002.

[22] K. Kim, S. K. Cha, and K. Kwon. Optimizing multidimensional index trees for main memory access. *ACM SIGMOD Record*, 30(2):139–150, 2001.

[23] R. Krovetz. Viewing morphology as an inference process. In *Proc. SIGIR*, pages 191–202, June 1993.

[24] D. Lemire and L. Boytsov. Decoding billions of integers per second through vectorization. *Soft. Prac. & Exp.*, 41(1):1–29, 2015.

[25] Z. Li, K. C. K. Lee, B. Zheng, W.-C. Lee, D. Lee, and X. Wang. IR-tree: An efficient index for geographic document search. *TKDE*, 23(4):585–599, Apr. 2011.

[26] C. Macdonald, R. L. T. Santos, and I. Ounis. The whens and hows of learning to rank for web search. *Information Retrieval*, 16(5):584–628, 2013.

[27] C. Macdonald, R. L. T. Santos, I. Ounis, and B. He. About learning models with multiple query-dependent features. *ACM Trans. Information Systems*, 31(3):1–39, 2013.

[28] A. Moffat and J. Zobel. Rank-biased precision for measurement of retrieval effectiveness. *ACM Trans. Information Systems*, 27(1):2, 2008.

[29] M. Petri, J. S. Culpepper, and A. Moffat. Exploring the magic of WAND. In *Proc. ADCS*, pages 58–65, 2013.

[30] M. F. Porter. An algorithm for suffix stripping. *Readings in Information Retrieval*, pages 313–316, 1997.

[31] S. E. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In *Proc. TREC-3*, 1994.

[32] J. a. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg. Efficient processing of top-k spatial keyword queries. In *Proc. SSTD*, pages 205–222, 2011.

[33] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc., 2005.

[34] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-tree: A dynamic index for multi-dimensional objects. *Proc. VLDB*, pages 507–518, 1987.

[35] A. Spink, B. J. Jansen, and H. C. Ozmultu. Use of query reformulation and relevance feedback by excite users. *Internet Research*, 10(4):317–328, 2000.

[36] T. Strohman and W. B. Croft. Efficient document retrieval in main memory. In *Proc. SIGIR*, pages 175–182, 2007.

[37] L. Tan and C. L. A. Clarke. A family of rank similarity measures based on maximized effectiveness difference. `http://arxiv.org/abs/1408.3587`, 2014.

[38] D. Zhang, K.-L. Tan, and A. K. H. Tung. Scalable top-k spatial keyword search. In *Proc. EDBT*, pages 359–370, 2013.

[39] D. Zhang, C.-Y. Chan, and K.-L. Tan. Processing spatial keyword query as a top-k aggregation query. In *Proc. SIGIR*, pages 355–364, 2014.

[40] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.-Y. Ma. Hybrid index structures for location-based web search. In *Proc. CIKM*, pages 155–162, 2005.

[41] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(2): 6–1 – 6–56, 2006.