Efficient and Effective Large-scale Search

Anagha Kulkarni

CMU-LTI-13-003

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh PA 15213
www.lti.cs.cmu.edu

Thesis Committee:
Jamie Callan, Chair
Jaime Carbonell
Christos Faloutsos
Alistair Moffat (The University of Melbourne)

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in Language and Information Technologies

Abstract

Search engine indexes for large document collections are often divided into *shards* that are distributed across multiple computers and searched in parallel to provide rapid interactive search. Typically, *all* index shards are searched for each query. For organizations with modest computing resources the high query processing cost of this exhaustive search setup can be a deterrent to working with large collections. This thesis questions the necessity of exhaustive search and investigates *distributed selective search* as an alternative where only a few shards are searched for each query.

For selective search to be as effective as exhaustive search it is important for the chosen shards to contain the majority of the relevant documents. Toward this goal, we study three types of document allocation policies that organize the collection into shards using different strategies for concentrating the relevant documents for a particular query into a few shards. Empirical evaluation with three large datasets indicates that topical partitioning of the collection provides the most cost-effective selective search setup. We further develop the best performing allocation policy to control for the distribution of sizes of the constructed shards. The resulting shards exhibit nearly uniform size distribution, and support comparable selective search performance. Analyses of several other variables of the shard creation process demonstrate that selective search is not highly sensitive to different parameterizations of these variables, confirming the reliability and consistency of the observed trends.

The set of shards that are searched for a query directly influence the effectiveness of selective search. We test the efficacy of a widely used resource ranking algorithm for the task of shard selection. The results indicate that a (nearly) off-the-shelf resource ranking algorithm can be employed to support a highly efficient selective search approach. We also propose a family of three shard ranking algorithms that use a joint formulation to model the two interdependent problems of shard ranking and rank cutoff estimation. The empirical results demonstrate that the proposed algorithms along with the query-specific predictor of rank cutoff provide a substantially higher search efficiency than the off-the-shelf resource ranking algorithm and provide comparable search effectiveness.

A comparative analysis of query runtime in a low-resource environment demonstrates that distributed selective search is much faster than distributed exhaustive search. Analysis of the shard ranking runtime suggests two ways of reducing this overhead. Topical homogeneity of the shards can be exploited to reduce the sample size used by the sample-based shard ranking algorithms to substantially lower the runtime overhead of this step. Experiments with a well-established query optimization technique indicate that the query runtime with selective search is much shorter than with query optimization alone.

Acknowledgments

First and foremost I would like to thank my advisor, Jamie Callan, for his support and advice throughout my academic journey toward this milestone. My committee members, Alistair Moffat, Christos Faloutsos, and Jaime Carbonell have been generous with their time and advice. I would like to thank them for the same.

Jaime Teevan, who was one of my mentors during an internship at MSR has inspired me in more ways than she realizes. I would like to thank her, and my two other awesome mentors from that internship, Krysta Svore, and Susan Dumais for a wonderful summer. I have been fortunate to find student collaborators that I have actually enjoyed working with. Yubin Kim, and Almer Tigelaar are two of those people, and I want to thank them for being no-nonsense collaborators.

My parents have blessed me with many things, but the one thing that I am most thankful for is the grit that they instilled in me which has been critical to reaching here. The unwavering support and love that my parents and my mother-in-law have showered on me has kept me going in the most challenging times, and I am eternally indebted to them for that. My sweetest 'thank you' goes to my son, Sohum, who brightens every day with his innocent smile. None of this would have come to pass was it not for the love and encouragement that I have received from my best friend and husband, Mahesh. M: this thesis is dedicated to you, and our family.

——Om Shri Sairam——

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Information retrieval (IR) as a field of research is relatively young compared to many other disciplines of science. We have however engaged in the act of storing, organizing, and searching information for a very long time. And yet IR has never been as important and essential as it is in this age of *big data*. We have witnessed IR applications being brought out of the confines of libraries to our personal computers, laptops, tablets and mobile phones for everyday usage. The volume of search requests that commercial search engines service daily is an attestation of our increasing search requirements. For the largest search engine by volume, the figures are in the order of few billion queries per day.

This increased reliance and need for search has been driven by many factors, one of which is the growing availability of large, information-rich, and search-friendly collections. The Web is its most prominent example[1] but it's hardly the only one. More and more organizations and businesses are digitizing all types of internal data in order to make it searchable. The information that can be mined from these large collections is often invaluable to the organizations.

The research community has also responded to the trend of increasing collection sizes by compiling, and using progressively larger datasets in their research. Shared IR evaluation efforts such as TREC[2] have adopted increasingly larger datasets over years. For instance, the largest dataset that was widely used at TREC 2000, WT10g, consisted of 1.69 million pages (10GB uncompressed). By TREC 2010 the dataset size had grown by more than 250 times. The dataset that was used by many TREC participants in 2010, ClueWeb09 CategoryA English, consisted of half a billion documents (15TB uncompressed).

The other factor that has contributed to the widespread use of search systems has been the advancements in retrieval algorithms employed by modern search systems. The current state-of-the-art retrieval algorithms are able to provide much more accurate search results than their previous manifestations. The early retrieval algorithms typically inferred a document's

[1] In 2000 Google's index consisted of 1 billion pages. By 2010 this figure had crossed the mark of 30 billion pages.
[2] http://trec.nist.gov/

relevance based on the presence (or absence) of the query terms, and provided an unranked result set. Whereas the current search systems might employ a multi-layered retrieval strategy that operates different algorithms at each level, and make use of a slew of information, such as, the document quality, authority, and time-stamps, in addition to the user query.

When the task of information search was reserved for librarians the expected shortest response time for an information need might have been minutes. In contrast the current search systems are pushing the state-of-the-art at sub-second query response time. Through Web search more and more users are being trained to expect instant search ability on datasets of enormous size. Search engines have lead by example and have changed user expectations over time.

All these factors, together, have made search systems more user-friendly, effective, and popular, in general. The search service which was often perceived in the past as being cumbersome, and slow, has been transformed into an easy and extremely useful feature. For many services and establishments the ability to search has become quintessential to their business operations. The above trends have also however placed unprecedented amounts of computational demands on the underlying search system. Processing large collections is in itself computationally expensive, but the sophisticated retrieval algorithms further add to the complexity.

1.1 Distributed exhaustive search

Commercial search providers have kept up with the growing computational demand by periodically expanding their computing and network infrastructure. The large document collections are typically partitioned into *shards* which are then distributed across a large number of computers and searched in parallel to provide rapid interactive search. Typically, *all* index shards are searched for each query. This solution, referred to as *distributed exhaustive search*, (depicted in Figure 1.1), assumes availability of large computing clusters. As such, the data centers deployed by the commercial search engines are the quintessential components of this search architecture.

1.2 Distributed selective search

For most mid-sized and small organizations, however, deploying a large computing infrastructure is impossible. As a result, such resource-constrained organizations often have to settle for working with smaller document collections, and also avoid employing computationally intense state-of-the-art retrieval models. This thesis offers a search solution for such organizations that does not require them to make such compromises. More generally, this thesis proposes a search architecture that can support efficient and effective large-scale search using few computing resources. To achieve this goal we propose an architecture that partitions the collection such that only a few shards need to be searched for a query. We refer to this approach as *distributed*

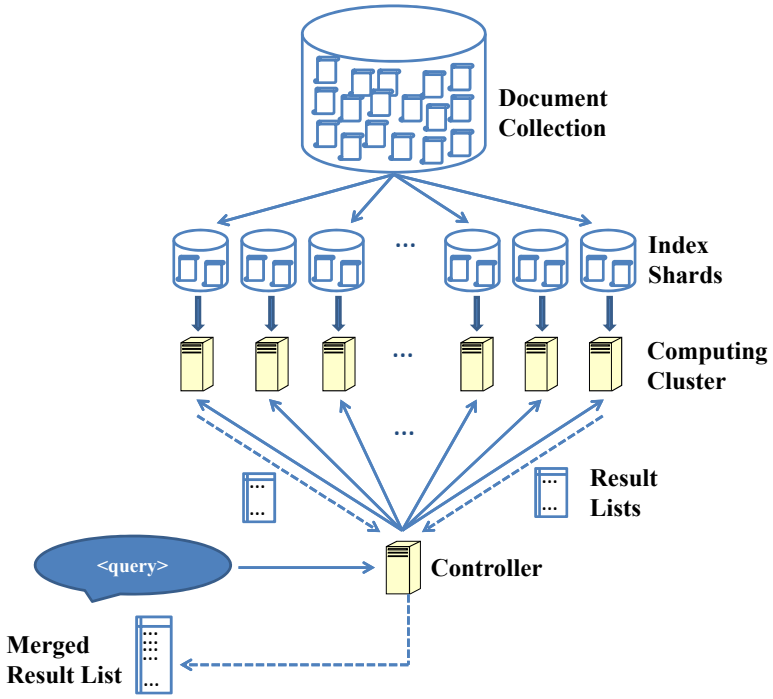


Figure 1.1: Distributed exhaustive search.

selective search and demonstrate that it reduces the amount of search effort needed per query dramatically while providing competitive search effectiveness.

1.2.1 Distributed selective search: Efficiency

Distributed selective search brings together insights from several different research areas of IR. In fact, we build upon two key ideas from existing large-scale search approaches: *parallelism*, and *partial search*. The task of estimating the relevance of a document for a query can proceed independently of any other document's relevance estimate. As such, query evaluation on large collections can be easily parallelized. Like distributed exhaustive search, distributed selective search also exploits this property by partitioning the collection into smaller shards which can be searched in parallel.

Secondly, for most search applications every document that contains a query term does not need to be evaluated in order to provide accurate search results. Large-scale search systems often leverage this property and deploy *partial search* approaches to improve query processing efficiency. *Tiering* techniques are instances of this category that divide the collection into a small number of tiers based on document properties such as, quality and geographic source [5, 18, 63].

During query processing only the *primary tier* is searched by default. The secondary tier(s) are searched if the primary tier fails to return sufficient results. Tiering techniques exploit the observation that it is not necessary to evaluate every document that contains a query term in order to provide accurate search results. However, the primary tier is also typically quite large. As a result often it is further partitioned into smaller shards in order to facilitate distributed exhaustive search.

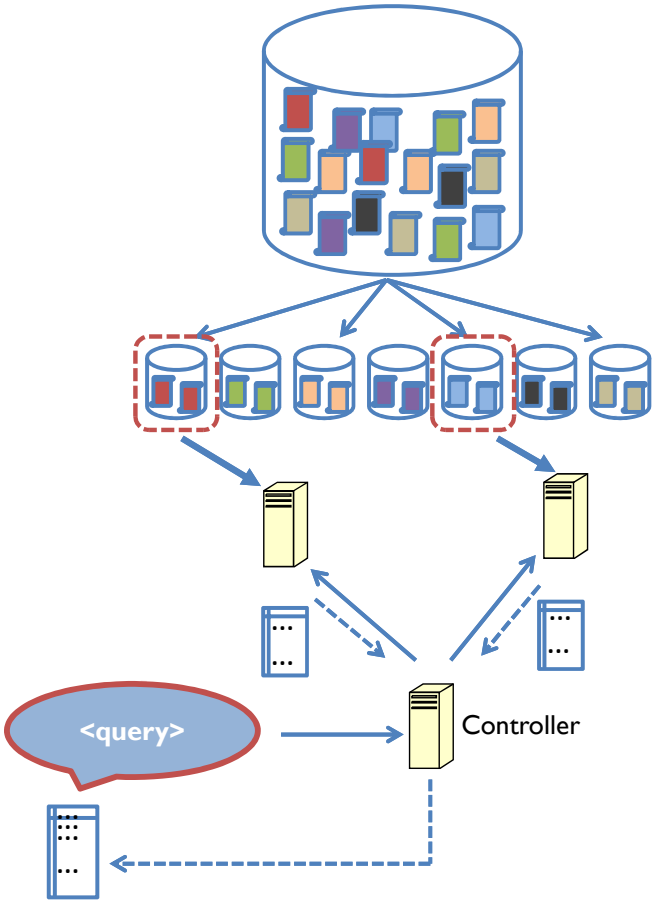


Figure 1.2: Distributed selective search.

In this thesis we claim that both the above properties can be better leveraged using distributed selective search by employing a more sophisticated collection partitioning technique, and by searching only a small subset of these partitions for each query. The proposed approach

in effect transforms a large-scale distributed task (as performed by exhaustive search) into a smaller distributed task. Naturally, the demand for computing resources reduces dramatically, as is illustrated by the schematic diagram of selective search in Figure 1.2.

So far we have focused on the efficiency of distributed selective search, and on its low resource requirements. In the following section the focus shifts to search effectiveness. We present the center pieces of the selective search architecture that make it possible to support search that is as accurate as exhaustive search while being substantially more efficient.

1.2.2 Distributed selective search: Effectiveness

The distributed selective search approach is based on the following two-part hypothesis where we conjecture that:

1. **A document collection can be partitioned such that the majority of the relevant documents for a query are concentrated in a few shards without any prior knowledge of the query-set; and**
2. **When a collection is thus partitioned three things follow:**
   - **(a) the shards can be ranked based on their relevance to the query;**
   - **(b) the minimum number of top shards in this ranking that need to be searched for the query can be estimated; and**
   - **(c) the merged results from the searched shards can be just as accurate as results from exhaustive search.**

The first part requires the collection to be organized in a way that is conducive to selective search. If the distribution of relevant documents across shards is skewed then the search can be restricted to the relevant shards without any loss in search effectiveness. The challenge though is to provide a skewed distribution of relevant documents for every query that the search system receives without any prior knowledge of the query stream, and without re-partitioning the collection often. We identify the *Cluster Hypothesis* as a potential solution to this problem [77]. As per the Cluster Hypothesis *closely associated documents tend to be relevant to the same requests*. This suggests that if similar documents are clustered together then the relevant documents for a query would also be grouped together. By extension, a document similarity-based partitioning technique would divide a collection such that, (a) the distribution of relevant documents across the created shards is skewed for any (or many) given query, and (b) each shard consists of similar, and thus topically homogeneous documents (*topical shards*). Developing a collection partitioning technique that satisfies the above requirements, and also scales to large collection sizes is one of the goals of this thesis.

The second part of the hypothesis recognizes that for each query the set of relevant shards

might be different. As a first step, the ability to rank the shards based on their estimated relevance to the given query is needed. Next, in order to search as few shards as possible without degrading effectiveness, a rank cutoff on the estimated shard ranking has to be computed. Proposing accurate, efficient and scalable solutions to these problems is the other goal of this thesis. We conjecture that these two tasks, shard ranking, and shard rank cutoff estimation, are interdependent and propose a joint formulation that provides query-specific predictions for both the problems.

Finally, the above hypothesis states that a search architecture that satisfies all of the above requirements can balance search effectiveness, and efficiency, while demanding few computing resources. A thorough validation of this hypothesis using some of the largest available datasets, and using the appropriate evaluation metrics is the bigger goal of this thesis.

1.3 Contributions of thesis research

This thesis brings together learnings from research areas of large-scale search, cluster-based retrieval, and federated search, to propose a search architecture that is efficient and effective even in low-resource environments. Search approaches for large collections have always employed distributed query processing in order to achieve fast response time [4, 5, 7, 21, 63]. However, these approaches have typically partitioned the collection such that the majority of the shards need to be searched for each query. The search approach proposed in this thesis also employs distributed query processing, however, by controlling the partitioning of the collection, the search is restricted to only a few shards for each query. The consequence of this is substantially lower search effort, and fewer computing requirements than those of other large-scale search approaches. Thus one of the key insights from this work is the importance of the collection organization employed by a distributed search system.

One of the key elements of this thesis, the Cluster Hypothesis, comes from the research area of cluster-based retrieval [24, 27, 64, 79, 82, 85]. Our contribution toward this research area is in proposing a collection partitioning technique that, in addition to modeling the Cluster Hypothesis, offers several advantages over the traditional approaches. First, it is scalable. Large collections can be efficiently divided into topical shards using the approach proposed in this thesis. The computational complexity of the approach is linear in collection size. Second, it can be parallelized. The process of assigning a document to a shard can proceed in parallel for multiple documents. Third, it is general. Since the proposed approach does not require knowledge of the query traffic, and does not use any external resources, it can be applied to any collections. Finally, it is effective. Through empirical validation we demonstrate that the proposed approach creates shards that concentrate the relevant documents for a query into a few shards. Most existing collection partitioning techniques lack some or all of these properties. The other contribution of this thesis is its potential to revive the research area of cluster-based

retrieval which went dormant more than a decade ago due to lack of consistent and stable empirical results.

The field of federated search has investigated the problem of ranking collections of documents (*resources*) for the past two decades [2, 14, 15, 32, 40, 65, 66, 69, 74]. This research problem is similar to the shard ranking problem that we study in this thesis. Our contributions to this line of work are two-fold. Nearly all existing resource ranking algorithms model the task as consisting of a single goal: estimating the distribution of relevant documents across resources for the query. Instead, we argue that it consists of two interdependent goals: (a) shard ranking, and (b) shard rank cutoff estimation. We propose a joint formulation of these tasks, and demonstrate that a query-specific ranking, and cutoff estimation improves search efficiency. This thesis is also one of the first to study the efficiency of the resource (or shard) ranking algorithms. Previous work has focused exclusively on the effectiveness of these algorithms. This investigation leads to our other contribution. We renew the research interest in the older class of resource ranking algorithms, popularly known as the *big document* approaches or *model-based* algorithms [14, 32]. Although, these algorithms have fallen out of favor because of their lower effectiveness, we demonstrate that they offer unparalleled efficiency. This suggests that reviving the research on more accurate model-based algorithms might be worthwhile.

Providing a good balance between the competing requirements of search efficiency and search effectiveness is a challenge for any search approach. Most existing techniques that improve search efficiency can claim competitive search effectiveness only for the top ranks of the search results [23, 31, 47, 74, 84, 85]. Users and applications, such as, legal search, that care about effectiveness at deeper ranks cannot benefit from such techniques. As a result the impact of these search approaches is limited. We addresses this limitation by employing some of the most comprehensive metrics that evaluate the effectiveness of the search results at much deeper ranks. The results from the empirical evaluation demonstrate that selective search provides competitive effectiveness even on these metrics, and the corresponding search cost is still substantially lower than that of exhaustive search. To the best of our knowledge no other search technique can provide such a balance of efficiency and effectiveness.

The amount of search effort needed per query has a direct impact on the operational costs of a search system. In addition to the computing resources used by the search system, the cooling systems used to maintain the temperature of the computing facilities also consume enormous amount of energy. As such, the savings in search effort offered by selective search also translate into financial, and environmental savings. Overall, we believe that the proposed approach is a *greener* search process, that also lowers operational and maintenance costs. As such, the distributed selective search technique can potentially be useful even to large organizations that are not constrained in terms of the available computing resources but would like to lower their energy requirements.

In summary, there are several rationales, based on both search efficiency and effectiveness,

based on historic relevance, and on its potential to change the current search paradigm, that justify a careful and systematic study of distributed selective search.

1.4 Overview of dissertation organization

This thesis is organized as follows. The related prior work that forms the basis of the work in this thesis is described in Chapter 2. The high-level architecture of the distributed selective search approach is provided in Chapter 3 along with a reference system. Chapters 4 and 5 describe the two center-pieces of the proposed search architecture – shards creation, and shard selection. An in-depth treatment of the efficiency of selective search, as measured in query runtime, is provided in Chapter 6. In Chapter 7 we conclude the dissertation with the summary of the research, its contributions, and its possible future directions.

Chapter 2

Related Work

Three lines of research provide the context for the work in this dissertation. The problem of efficiently searching a large collection is at the core of any large-scale search system. We review this literature in the first section of this chapter. The second line of related work, *cluster-based retrieval*, takes us back four decades where the initial goal was to improve search efficiency but later shifted to improved effectiveness. The literature from the *federated search* field is also closely related to the work in this thesis, especially because of the shared research problem of ranking collections of documents for a query.

2.1 Large-scale search

Commercial search providers care deeply about the price-performance tradeoff that the deployed system can provide [4, 5, 7, 21, 63]. As a result, most search providers optimize for multiple operational requirements, such as, throughput, response-time, resource utilization, hardware costs, and infrastructure costs. A host of strategies are typically employed to meet these operational requirements.

Distributed query processing is among the most widely used strategy for achieving short query response time. Here the large collection is divided into smaller partitions, *shards*, that are searched in parallel using large computing clusters. This strategy exploits the inherent parallelism in the query evaluation process in order to reduce query latency. Another popular technique is that of index replication where data redundancy is employed to achieve the needed throughput and fault-tolerance. Many search systems also use *tiering* which divides the collection at multiple levels with the goal of evaluating the query against only a subset of the collection [5, 63].

Notice that the first two strategies, index partitioning and index replication, do not offer any savings in the average search cost or in the number of resources needed for the system, but tiering can. We thus review tiering strategies in more details in Section 2.1.1. Techniques

categorized as *index pruning* algorithms also aim for reducing the average amount of search effort needed per query, and are frequently used by commercial search engines. We review this literature in Section 2.1.2.

In addition to the above techniques caching at several levels of the search system architecture is routinely employed to reduce the disk and network activities, both of which are almost always the costliest components of query processing.

2.1.1 Tier-based retrieval

A *document quality* based tiering approach separates the high quality documents into the first tier, and the remaining documents are delegated to the lower tier(s). The document quality can be defined as a function of various document properties, such as, its spam score, click probability, and number of inlinks. In this architecture most queries are served by the first tier which contains a relatively small subset of the collection and the lower tiers act as fall-back options which are searched only if the higher tier fails to generate sufficient search results. Every query that is successfully served by the first tier, effectively avoids searching a large portion of the collection thus decreasing the search cost.

For Web search, it has been observed that often search intents tend to be geographically localized. A tiering technique that exploits this property can be used to drive down the search cost. *Geographical tiering* divides the document collection into tiers based on the geographical source of the documents [18]. Cambazoglu et al. maintain a goal of constructing search results in a geographically tiered search system that are identical to those generated by complete search. Each incoming query is processed against the local tier by default and the highest possible document score that could be computed for the query at each of the remote tiers is estimated using statistics obtained from offline queries. The highest document score estimates are compared to the score of the kth document retrieved from the local tier to predict if the remote tier would contribute document(s) to the top k ranks. The query is forwarded to all the remote tier which are predicted to contribute one or more documents.

It is important to note that the tiers created by any of the above strategies are still quite large. In fact, because of their size each tier is typically further divided into multiple smaller shards that are evaluated in parallel for each query. For search environments with limited computing resources the cost of searching even just the primary tier can be high.

2.1.2 Index pruning

There is a large body of prior work that investigates the problem of improving search efficiency by manipulating the *inverted index* of the document collection. These approaches are commonly categorized as either performing *dynamic index pruning* or *static index pruning* depending upon when the reduction in index happens.

Static index pruning

Static index pruning techniques explore the space of *lossy compression* techniques where search efficiency is improved by permanently excluding certain portions of the inverted index such that these omissions would have minimal adverse effect on the search effectiveness.

Carmel et al. [19] propose two *term-centric* pruning techniques that work by reducing the entries in the term postings lists. Shorter posting lists reduce the volume of data that needs to be read and transfered from the disk. The first technique, *uniform*, prunes the postings list of each term in the index using a single fixed threshold that is defined on the document scores for the term. In the second technique a term specific threshold is computed for each term by defining the threshold as a function of the k^{th} highest document score for the term. The experimental results demonstrate that term specific thresholding performs better than the uniform thresholding approach. However, even with term specific pruning the average precision degrades significantly when only 10% of the index is pruned. The drop in P@10 values is more gradual as the index is pruned more aggressively. Also the overlap between the top 10 original results and those obtained with the pruned index is high (about 0.7 or higher) for 35% or lower index pruning. Overall, this approach is a good contender if only precision at early ranks is of interest.

de Moura et al. [25] extended the above work by proposing a pruning approach that can effectively handle conjunctive and phrasal queries in addition to disjunctive queries. The set of sentences that contain at least one of the terms identified by Carmel et al. [19] as important to the document is identified in the first step. This set of sentences is ranked based on the frequency of these terms. All the terms in the top n sentences are included in the pruned index. This approach retains not just the important terms but also the terms that co-occur with those terms in the document. The authors hypothesize that these terms are also likely to co-occur in the conjunctive and phrasal queries. Single term queries are excluded from the evaluation query set because their performance under the proposed approach, and the baseline system (Carmel et al. [19]) is very similar. The experimental results demonstrate that at 60% index pruning the locality based technique can provide comparable average precision, and precision-recall curve to that with unpruned index. The query runtime with the 60% pruned index is 65% of that with unpruned index. Also the overlap between the top 20 original results and those obtained with the 60% pruned index is about 0.7. Overall, the locality based approach provides substantial improvements over Carmel et al. across the board.

Büttcher and Clarke [12] experiment with *document-centric* pruning that works by indexing only a subset of terms from each document. Kullback-Liebler divergence computed between the unigram language model of a document and the collection is used to rank the terms in the document based on their contribution to the document. The top k terms or the top n% of the terms in the document, that is, the terms that are most unique to the document, are selected to be indexed from each document. In addition to this in-memory pruned index the complete index

is also searched in parallel for every query. The results from the unpruned index are referred to whenever postings cannot be found in the pruned index for a query term. The experimental results demonstrate that by indexing only the top 10% terms from each document, the pruned index size is 12% of the original index, and the query processing time reduces by 87%. The corresponding search Precision degrades by 2.5% for the top 10 documents, by 3.4% for the top 20 documents, and MAP degrades by 16%. Nguyen [55] propose a posting-based approach that combines term-centric and document-centric approaches to compute a score for each postings list entry $< t, d >$. The pruning is performed by thresholding on these scores to selectively include postings list entries. The experimental results show that document-centric pruning does as well or better than the posting-based approach.

Static index pruning is most useful when the pruned index of a collection can fit entirely in the main memory. For large collections this might be difficult even when index is pruned aggressively. For instance, the largest dataset used in this dissertation consists of 0.5 billion documents (Indri index is 5.4 TB). A 10% subset of the complete index would be about 54 GB for this dataset. As such, the usability of these techniques for large datasets is questionable and remains to be evaluated. Also, the performance of these techniques in absence of the on-disk unpruned index has not been studied. Recall that in this thesis we assume that only limited computational resources are available to the search system. Thus a setup that needs to consult the complete index of a large dataset, even occasionally, would be unrealistic. Since static index pruning employs lossy compression it is important to study individual query performance. This is especially true for higher compression rates, such as those recommended by de Moura (60% and higher), and for higher Recall levels where search Precision performance can potentially vary substantial across queries. Unfortunately, none of the studies provide stability or robustness analysis that can quantify the fraction of evaluation query that were adversely affected by pruning.

Dynamic index pruning

Dynamic index pruning is a lossless query optimization technique that aims at reducing the amount of computations needed for a query, on average.

Smeaton and van Rijsbergen [70] proposed one of the early approaches that explored *dynamic index pruning*. The main idea was that not all query terms are equally important for obtaining document ranking that is similar to the *perfect ranking* and thus excluding these query terms from the query evaluation process can provide efficiency gains without degrading search quality. To operationalize this they process the query terms in the descending order of their *inverse document frequency* (idf) weights and accumulate partial similarity scores for documents in the postings lists of these terms. After a postings list for a query term is processed in its entirety the upper bound on the similarity score that is obtainable from the documents not yet in the accumulators is computed. If this value is smaller than the highest partial similarity score in the accumulators

then the query evaluation is terminated.

Wong and Lee [83] studied two variants of this approach. The first method processes the query terms in the descending order of their $tf_{max} * idf$ weights where tf_{max} is the highest within document frequency observed for the term. The second approach takes this a step further by ordering the entries in the postings list for each term based on the tf weight of the term in the document. Sorting of the postings lists ensures that documents that would contribute more toward the partial document scores are processed before the documents that would make small contributions.

Moffat and Zobel [54] suggested thresholding the number of accumulators that hold the partial document scores during the query evaluation as a way of increasing the efficiency. They demonstrated that using only 1% of the collection size as the upper limit on the number of accumulators could lead to search Precision that was comparable to the one obtained using an unlimited number of accumulators when their method was executed in the *continue* mode. In the *quit* mode the processing of the postings lists and the query evaluation stops once the upper limit on number of accumulators is reached but in the *continue* mode the postings lists continue to be processed and the partial scores for the documents already in the accumulators are updated. Thus the continue mode does provide savings in terms of the memory used to hold the accumulators however it does not reduce the search space.

Persin et al. [58, 59] pioneered a line of research that leveraged different orderings of posting list entries in order to improve query evaluation efficiency. Persin recommended sorting the posting lists based on document term frequency instead of document number, and provided mechanisms to query and compress such frequency sorted indexes. Anh et al. [1] improved on this approach using *impact sorted index* where the entries in the postings lists are sorted based on the normalized $tf.idf$ values. Query evaluation is terminated after examining the top k postings list entries. The empirical results demonstrate that the search effectiveness does not degrade significantly. Garcia et al [29] introduced another ordering policy for postings lists where the entries in the postings lists (documents) are sorted by the frequency with which they were ranked highly for training queries. These techniques are also commonly referred to as the *early termination optimization* or *inexact top k retrieval* techniques.

Turtle and Flood [76] proposed an optimization technique that is *rank safe* for the top r results – it is guaranteed to produce the same document ranking for the top r results as would be provided by a non-optimized system. The first step of the algorithm orders the query terms (and their posting lists) according to their inverse-document-frequencies (idf). Less frequent (more important) terms are ordered before more frequent terms. The first r documents provided by this ordering of the posting lists are evaluated and ranked. The lowest document score in this ranking is referred to as the *max-score*. For every document after the first r documents the following scoring strategy is used. The query terms are sorted based on their idfs. The score contribution of the query term t for the current document is computed and added to

the document score. For the remaining query terms their highest possible contributions[1] are assumed and temporarily added to the document score. If the resulting document score is less than the max-score then the evaluation for this document is terminated. However, if the score is higher than the max-score then the next query term is evaluated similarly. If the document is scored for all the query terms and the final document score is higher than the max-score then the document is inserted into the current top r ranking and the max-score value is updated. As the evaluation progresses the acceptance threshold placed by the increasingly higher max-score value becomes harder to satisfy and the evaluation for many documents terminates before all the terms are scored.

Brown [11] proposed an optimization technique based on the use of *topdocs* lists which are partial posting lists that are sorted on term frequency (instead of document number). In addition to the term posting lists, the topdocs list are maintained for terms with long posting lists (low idf). During query evaluation all the documents for terms with short posting lists are added to the *candidate document set* but for terms with long posting lists only the documents in the topdocs are added to the set. All the documents in the candidate set are evaluated and ranked to provide the final results. Unlike the max-score approach, the optimization proposed by Brown [11] is not rank or score safe.

The *term bounded max-score* (TBMS) algorithm is an extension to the *max-score* algorithm proposed by Turtle and Flood [76] for top k retrieval [72]. The TBMS algorithm employs the topdocs lists, proposed by Brown [11], to further improve the efficiency of the max-score algorithm. However, instead of using term frequency to sort the topdocs lists, TBMS employs document weight which is the ratio of term frequency and document length. In the first step of the query evaluation process all the entries from the topdoc lists of each of the query terms are added to the candidate document set, D. The documents in the set D are scored and ranked in the next step to provide the first version of the top k results for the query. The smallest document score from set D is used as the threshold for the max-score algorithm. Specifically, when a document outside of the candidate set is considered for evaluation, first an upper bound on the document's score is estimated and compared to the threshold. If the upper bound is not higher than the threshold then the document is guaranteed to be not in the top k ranks and thus is not evaluated. TBMS uses this methodology to skip over posting lists entries and also query terms to reduce the number of computations needed for each query, on average.

In general, large-scale search systems almost always employ a dynamic index pruning technique to improve query processing efficiency. We thus compare the approach proposed in this thesis with a baseline search system that is optimized using a dynamic index pruning technique in Chapter 6. However, before we conclude this section, the following limitations of dynamic index pruning techniques are worth noting.

The effectiveness of some of the techniques, such as those by Smeaton and van Rijsber-

[1]The highest possible contribution of a term is a collection specific value that can be precomputed.

gen [70], and Wong and Lee [83], can be dependent on the query length. These techniques provide a simple and an effective way to optimize the evaluation of long queries. However, it is difficult for these techniques to achieve a good balance between effectiveness and efficiency for short keyword queries where discarding a term would have a significant impact on the search effectiveness. While other techniques, such as those by Brown [11], and Strohman et al.[72], need additional data structures (topdocs) for their operation. More importantly though, dynamic index pruning can offer only limited reduction in the amount of disk activity needed for query evaluation. This is because the complete posting lists for the query terms must still be read from the disk, even if the individual posting list entries (and the corresponding document scoring) are skipped.

2.2 Cluster-based retrieval

The term *cluster-based retrieval* has been used for two different types of search approaches in prior work. The search approaches that cluster the complete collection as a preprocessing step, and select one or more clusters during query processing have been referred to as cluster-based retrieval methods [24, 42, 64, 78, 85]. Document retrieval models that cluster only the documents returned for the query with the goal of improving search effectiveness have also been referred to as cluster-based retrieval [38, 48]. The former is related to this thesis and is thus studied in detail below.

Retrieval models that employed clustering algorithms were much more popular a few decades ago than they have been in the recent past. For instance, in the early 1970s the use of inverted indexes for efficient access, and full-text retrieval were not common IR practices. One early work employed clustering of document collections to enable an efficient user-guided search [64]. The query response time had to be strictly controlled due to the interactive nature of the task. This was achieved by partitioning the collection containing 200 documents into smaller groups of similar documents using Rocchio's clustering algorithm [41]. For query processing the similarity between the query and the cluster centroids was computed and the most similar cluster was returned to the user. This approach reduced the number of similarity computations needed for each query thus improving the efficiency and query response time.

Jardine and van Rijsbergen [42] organized the collection of 200 documents into a hierarchy of progressively more similar document clusters. Specifically, the single-link clustering algorithm and the Dice coefficient as the similarity measure were applied. During query evaluation the single most similar cluster from the hierarchy was chosen for each query using a top-down tree traversal based cluster selection approach. The proposed approach was compared with the traditional ad hoc document retrieval model as the baseline and cluster-based retrieval with optimal cluster selection as the gold-standard benchmark. The experimental results demonstrated that the proposed approach performed at par with the best performing ad hoc retrieval setup

in terms of precision but not recall. Also, the proposed approach fared substantially worse than cluster-based retrieval without optimal cluster selection.

Although, the approach proposed by Jardine and van Rijsbergen [42] had a limited impact on the subsequent techniques proposed in this area, their paper remains an important landmark in cluster-based retrieval because it was the first work to formally define and make use of the *cluster hypothesis* which is the basis of all the cluster-based retrieval algorithms. The cluster hypothesis states that *similar documents tend to be relevant to the same request*. This implies that if a collection is organized such that the sets of similar documents are separated into their own clusters then the document retrieval task can be formulated as a function of the cluster retrieval problem. This transformation of the document retrieval problem is expected to improve its efficiency and effectiveness. The cluster hypothesis is one of the cornerstones of the work proposed in this thesis.

Croft [24] provided a more principled cluster selection technique by formulating the task in a probabilistic framework where the conditional probability of the cluster being relevant to the query was estimated using Bayes' rule as follows.

$$P(C_i|Q) = \frac{P(Q|C_i)P(C_i)}{P(Q)} \propto P(Q|C_i)P(C_i) \tag{2.1}$$

Here the prior probability, $P(C_i)$ is simply the size of the cluster, and $P(Q|C_i)$ is approximated by the product of individual query term generation probabilities.

$$P(Q|C_i) = \prod_{q_j \epsilon Q} P(q_j|C_i) \tag{2.2}$$

This cluster ranking model is similar to the widely used *query likelihood model* for document retrieval. An empirical evaluation of this model was conducted on a collection of 1400 documents from the Cranfield collection. The documents were clustered into a hierarchy of 37 levels and more than 400 leaf-node clusters using single-link clustering algorithm. Two variants of the collection selection algorithm based on the direction of the traversal, top-down or bottom-up, were evaluated. The results showed that as compared to the baseline, cosine similarity based document retrieval from the complete collection, the cluster-based retrieval with bottom-up cluster selection does marginally better when the evaluation metric weights precision twice as important as the recall. This is not completely surprising because the experimental setup used in this work allowed cluster-based retrieval to return only one cluster for every query and the clusters selected by the bottom-up strategy were almost always small (about 2 documents).

Subsequent work in cluster-based retrieval that attempted to reproduce the trends observed by Jardine and van Rijsbergen, and Croft, using a better evaluation setup that consisted of more datasets and improved metrics were unable to demonstrate consistent and useful gains in retrieval effectiveness over that of the traditional document retrieval [33, 79, 82]. In fact, Voorhees [79] concluded after an extensive comparison of hierarchical clustering algorithms,

and several cluster selection strategies that even the combination of the costliest clustering algorithm (complete linkage) and a costly top-down cluster selection approach could only provide marginal improvement over the baseline document retrieval approach. One of the last works in this line of research, by El-Hamdouchi and Willet [27], also concluded after a thorough comparative analysis that the non-clustered traditional retrieval on the complete collection was more effective than any of the hierarchical cluster-based retrieval approaches.

There are several potential causes of the observed inconsistent improvements with cluster-based retrieval methods, such as, ineffective document representation (using only document titles or abstracts), sub-optimal cluster selection algorithms, datasets with short documents, and small datasets. Overall, none of the studies could justify the additional cost of creating elaborate collection hierarchies by demonstrating consistent and substantial improvements in search effectiveness. Also, the scalability of the hierarchical approaches was becoming a serious limitation as larger document collections were starting to become available. As a result, this line of work that focused on hierarchical arrangement of datasets and on improving search effectiveness went dormant for about a decade.

The work by Xu and Croft [85] is related to cluster-based retrieval approaches because it uses the Cluster Hypothesis. However, it did not continue the tradition of creating document taxonomies, neither did it expect to improve search effectiveness over complete search. The approach proposed by Xu and Croft [85] offered a solution which consisted of a collection partitioning technique with linear time complexity, and a more effective cluster selection algorithm. Specifically, a two-pass *K*-means clustering algorithm and a Kullback-Liebler divergence based distance metric were employed to partition the complete collection into clusters of similar documents. The efficiency of the partitioning algorithm allowed for experimentation with collections containing over a half a million documents while using a full-text retrieval approach. For each query the relevant clusters were selected by computing the distance between the query's unigram language model and each cluster's unigram language model. The Kullback-Liebler divergence was used as the distance metric. The empirical evaluation compared the accuracy of the documents retrieved from the top 10 clusters to those retrieved from the complete collection. The effectiveness of the proposed approach was found to be comparable to that of the centralized search for all the three datasets evaluated. When compared to a second baseline system that organized the documents based on their sources, the proposed approach provided substantially more accurate retrieval at all of the top 30 ranks.

Because of the similarities of the search architecture of the above approach and cluster-based retrieval techniques, we reviewed this work in the current section. However, Xu and Croft had approached their work from the distributed IR and federated search point of view (Section 2.3) where the document collection might be already partitioned into groups of documents (resources). As a result, in addition to the above setup they also studied two other search environments which offered progressively less cooperation to the search system. The *local clus-*

tering scenario clustered only the documents within a resource according to their similarities. The least cooperative setup, *multi-topic representation* did not physically re-organize documents but simply estimated several topic models for each of the resources which were then used for improving resource selection. The experimental results demonstrated that at early ranks the Precision of the document retrieved with local clustering is comparable to those retrieved from a complete index. The Precision however degrades at deeper ranks (15, 20, and 30). It is not surprising that the least cooperative setup, multi-topic representation, does much worse than the performance with complete index. However, multi-topic representation does better than the distributed retrieval baseline of source-based organization.

The most cooperative setup studied by Xu and Croft is closely related to the search approach proposed in this thesis because of the similarities in the collection organization technique, and the search architecture.

More recently Puppin et al. [60] employed a co-clustering algorithm to partition a large document collection using query log data. The query log covered a period of time when exhaustive search was used for each query. These training queries and the documents that they retrieved were co-clustered to generate a set of *query clusters* and a set of corresponding *document clusters*. A collection of 6 million documents was partitioned into 17 clusters. Documents that were not retrieved by any of the training queries could not be clustered using the proposed technique. Such documents made up 52% of the collection. The overlap with results from a complete index was used as a search Precision measure, instead of human-labeled relevance judgments. Random partitioning of the dataset into 17 resources was employed as the baseline organization. Searching the top few clusters was found to be more accurate than searching the top few random partitions.

2.3 Federated search

The field of federated search concerns itself with servicing search requests in the presence of multiple (often autonomous) collections of documents (*resources*) [15, 66]. This thread of research is related to the two research areas reviewed earlier and to this thesis because of the similarities in their search architecture. However, there are two main distinguishing characteristics of federated search. First, the collections of documents, the resources, are predefined. The search system does not, and often cannot reorganize the documents. The other unique attribute is the level of cooperation that the resources offer to the federated search system. In many cases, minimal cooperation is available. Each resource is an autonomous entity that does not share any data with the federated search system and provides access to the documents only through a query interface.

In this environment it becomes necessary to learn a *profile* for each resource that can then be used to infer the ranking of resources for a given query. Query-based sampling (QBS) [16, 17]

has been widely used to acquire resource profiles in an uncooperative environment. QBS approximates random sampling of the resource contents by executing a small set of queries against the resource and downloading the top few documents retrieved for each of these sampling queries. The set of downloaded documents is then used to create a representation for the resource. Several variants of QBS have investigated its various parameters. The effects of different types and sizes of the sampling query sets have been studied [23, 39, 68]. Also, the impact of different types of retrieved sets (documents, snippets, document blocks) and their sizes have also been analyzed in literature [6, 67]. The other research problem in federated search that has received much attention is the resource selection problem where the goal is to infer a ranking of resources based on their estimated relevance to the query. This is closely related to the problem of *shard ranking* studied in this thesis. Previous work in resource ranking algorithms can be classified into three families: model-based, sample-based, and feature-based algorithms.

2.3.1 Model-based algorithms

Nearly all of the early resource ranking algorithms adopted a model-based approach where they learned a representative model for each resource and used it to infer a ranking of resource for a query.

Gravano et al. proposed a series of three algorithms [30, 31, 32], *bGlOSS, vGlOSS*, and *hGlOSS*, all of which assumed a cooperative search setup where each resource periodically shares collection statistics. The bGlOSS algorithm assumed a boolean model where a resource C is scored for the query Q using the following function.

$$gloss(Q, C_j) = \prod_{i=1}^{n} \frac{df(q_i, C_j)}{|C_j|} \tag{2.3}$$

where the function $df(q_i, C_j)$ returns the number of documents in resource C_j that contain an occurrence of the query term q_i. This approach assumed a cooperative environment where each resource shares all the necessary collection statistics with the resource ranker. The empirical evaluation using a search setup with six resources demonstrated that for 84% of the queries the top ranked resource is among the *best* resources for that query, and for 88% of the queries all the *best* resources were included in the set of resources ranked highly by GlOSS. The set of *best* resource for a query was defined based on the number of candidate documents in the resource for the query, not using human-labeled relevance judgments.

A later manifestations of GlOSS, vGlOSS [30], improved over the boolean version by adopting the vector-space model for document and query representation, and by proposing a more sophisticated resource scoring. At a high-level, vGloss ranks resources based on their *usefulness* for the query. The *usefulness* of a resource is defined as the number of documents for which the similarity to the query is above a user-defined threshold. In order to compute the usefulness

scores without executing the query against each resource, an approximate scoring function is proposed. For this function two resource-specific summary statistics are assumed to be available for every vocabulary item of the resource: cumulative term weight (for instance, collection term frequency), and document frequency. These statistics are used in two different ways to create vectors for hypothetical documents from each resource. Query vector and the document vectors are used to compute approximate similarity scores which are then thresholded and eventually provide the estimated *usefulness* score for the resource. An empirical evaluation using 53 resources and nearly 7000 queries demonstrated that the proposed approach when thresholded at similarity score of 0.2 provides good balance of recall and precision on the resource ranking task.

The next model in this line of research, *hGloss* [32], is an extension of vGloss (or bGloss) that operates in a hierarchy of Gloss servers as a meta-server. The goal is to direct the user query to the Gloss server that indexes resources containing relevant documents for the query. As such, the task is to rank the Gloss servers, instead of the resources. The proposed approach is similar to the scoring function employed by vGloss. The hGloss server uses server-level summary statistics, such as, the number of resources in the server that contain the query term, to infer a ranking of the Gloss servers. The user chooses a particular server(s) to visit based on this ranking and the vGloss scoring function is used next to rank the resources indexed by this server.

Callan et al. [14, 15] proposed a resource ranking approach, CORI, that did not assume any cooperation from the resources. The CORI algorithm represented each resource as one large document containing all the unique terms (and their document frequencies) in the resource. This transforms the resource ranking task into a document ranking problem. CORI employs an adaptation of the INQUERY [13] document ranking algorithm to the big documents to infer a resource ranking for the query. Specifically, the formulation used for ranking the big documents is as follows:

$$T(i,j) = \frac{df(i,j)}{df(i,j) + 50 + 150 * cw(j)/avg\_cw} \tag{2.4}$$

$$I(i) = \frac{log(\frac{K+0.5}{kf(i)})}{log(K + 1.0)} \tag{2.5}$$

$$p(q_i|s_j) = 0.4 + 0.6 \ * \ T(i,j) \ * \ I(i) \tag{2.6}$$

where $df(i,j)$ is the number of documents in resource s_j that contain query term q_i, $cw(j)$ is the number of indexing terms in resource s_j, avg_cw is the average number of indexing terms across resources, K is the total number of resources, $kf(i)$ is the number of resources that contain q_i and n is the query length in terms. For a query term q_i, the $I(i)$ component is constant for all the resources. The $T(i,j)$ component introduces a bias for small resources (small $cw(i)$) with many documents containing the query term (large $df(i,j)$). The probability estimates based on the individual query terms (Equation 2.6) are combined using one of the INQUERY operators, such

as, *sum* or *and*, to obtain a relevance score for the resource. The collection statistics required in the above formulation is approximated using the query-based sampling (QBS) technique. CORI has been empirically evaluated on datasets of different sizes and different number of total resources [15]. The results showed that for each dataset the top 10% of the resources contained about 60% of the relevant documents as that contained in the top 10% of the gold standard ranking. CORI's top 50% of the resources contained 90% or more of the relevant documents in the 50% of the gold standard ranking. The gold standard ranking was obtained by ordering the resources based on the number of relevant documents in a resource. The effect of incomplete collection statistics was studied by comparing CORI's performance with complete resource representations and with representations learned with QBS. The difference in CORI's performance was found to be small when the resource representations were learned from 3% to 6% sample of each resource.

2.3.2 Sample-based algorithms

The sample-based resource ranking algorithms subscribe to the philosophy that summary statistics from each resource do not model the ability of the resources to provide relevant documents. In order to capture that a sample of documents from each resource is used to construct a *central sample index* (CSI) which forms the basis of these sample-based approaches.

Si and Callan [69] leveraged the insight that the documents sampled from the resources (typically using the query-based sampling technique) to learn the resource descriptions (or profiles) could be re-purposed for resource ranking. Specifically, a *central sample index* (CSI) is created from the sampled documents, and the CSI is used as a proxy for the complete central index of the collection. The user query is run against the CSI and the top n retrieved documents are assumed to be relevant. If n_R is the number of documents in n that are mapped to shard R then a score s_R for each R is computed as:

$$\theta_R = n_R * w_R \tag{2.7}$$

where the shard weight w_R is the ratio of size of the shard ($|R|$) and the size of its sample ($|S_R|$). The shard scores θ_R are then normalized to obtain a valid probability distribution which is used to rank the shards. An empirical evaluation that compared ReDDE with CORI observed that ReDDE performed consistently better than CORI.

The SUSHI [74] algorithm also employed the central sample index (CSI) to predict a ranking of the shards for a given query. It uses the scores and adjusted ranks of the top 50 documents retrieved from the CSI for the query to fit curves for each resource represented in these documents. Three types of curves, linear, logarithmic, and exponential are tried. The curve with the best fit is used to interpolate scores for the top m ranks for each resource. The interpolated points from each resource are merged into a single ranking and scores are aggregated to compute a relevance score for the resources. The number of unique resources present in the top

R documents in this ranking is the predicted rank cutoff for the P@R metric. This estimator is evaluated using common federated search datasets. The results demonstrate that SUSHI predicts lower rank cutoffs on average as compared to a query-agnostic fixed cutoff of the 10 and its precision at rank 10 is comparable to that of the baseline approach for many of the datasets. Notice that in addition to resource ranking, the SUSHI algorithm also predicts the number of top ranked resources that should be searched for a query. Traditionally, a query-independent value that has been predefined is used for this cutoff. To the best of our knowledge, this is the first approach that proposes a query-specific predictor for the cutoff. We extend this research direction in this thesis by proposing a family of cutoff prediction algorithms (Section 5.3).

Ipeirotis and Gravano [40] organize the resources into a topical hierarchy that is constructed by clustering documents sampled from each resource. This hierarchy is static and query independent. The hierarchical resource selection is performed by evaluating the query against all of the resources at a particular level in the hierarchy using one of the *flat* resource ranking algorithms like CORI. The top ranked resource at this level is then selected to be searched and the sub-tree rooted at that node is further explored. The experimental results show that this hierarchical resource ranking approach enables higher average precision as compared to the flat resource ranking technique.

2.3.3 Feature-based algorithms

More recently Arguello et al. [2] took a classification based approach that develops classifier features that are functions of CORI's resource ranking, ReDDE's resource ranking, query category, and click-through data. For each resource a binary classifier is learned that estimates the relevance probability. The confidence score assigned for a prediction by each of the classifiers is used to rank the resources. The fixed rank cutoffs of 1–5 are evaluated and the results demonstrate that the non-content features together with the conventionally used content features yield a robust ranking approach. Of the different resource ranking algorithms reviewed here the above approach is most dependant on external knowledge resources such as query log data, and query categories.

2.4 Summary

This chapter reviewed three lines of research that are related to the work in this dissertation. Since one of the primary goals of this thesis is to enable efficient large-scale search, prior art in this research area was reviewed in Section 2.1 with special emphasis on the different techniques that are commonly employed to improve the efficiency of the search process. This survey highlights the absence of an approach that offers substantial reduction in search cost (effort, runtime, and computing resources), while providing search effectiveness that is on par with that of exhaustive search.

The other two research areas, cluster-based retrieval, and federated search, reviewed in this chapter are closely related to this thesis because we build upon some of the central ideas from this literature to propose the search architecture described in the next chapter.

Chapter 3

Distributed Selective Search

As search systems have evolved to work with larger document collections, several techniques have emerged that prevent the corresponding query runtime and/or the search effort from increasing. Some of these approaches exploit the inherent *parallelism* in the search process by distributing the document collection, and consequentially the search effort, across several computing nodes [5, 7, 21, 63]. These techniques offer improvements in query runtime that are typically proportional to the number of computing nodes allocated to the task. Since the complete collection (or the complete primary tier) is searched in a distributed mode for each query we refer to these approaches as the *distributed exhaustive search* (DES) techniques.

Other search approaches employ *partial search* in order to improve search efficiency. These techniques observe that most queries can be answered effectively without evaluating all the documents in which the query term(s) occur (Section 2.1.2). These techniques reduce the average amount of search effort needed per query by evaluating fewer documents. In this thesis we propose a search strategy that leverages both properties, parallelism and partial search, to propose a search strategy that is especially well suited for environments with limited computing resources. .

One of the goals of this chapter is to introduce the proposed search approach at a high-level. This enables us to provide the complete view of the studied search architecture before analyzing its individual components in the later chapters. Like any complex system with multiple components, the different parts of this architecture influence and interact with each other to ultimately impact the performance of the entire search system. This chapter provides the necessary grounding for the study of these dynamics in the remainder of the thesis. The other objective of this chapter is to introduce the baseline system and the evaluation methodology that will be used throughout the thesis. We start by describing the baseline approach in the next section.

3.1 Baseline system: Distributed exhaustive search (DES)[1]

The classic search systems consisted of a single monolithic inverted index for the entire document collection. But as the collection sizes increased the single inverted index was spread across several computing nodes in order to facilitate parallel search on the entire collection. The distributed exhaustive search, as instantiated in this thesis, consists of two phases. In the offline phase the document collection is divided into smaller partitions (shards). Typically, the total number of shards for a collection is decided based on the number of computing nodes available to the search system or the desired query response time. During the online phase of exhaustive search the query is forwarded to all the shards for processing. The ranked results retrieved at each shard are then merged and compiled into a final results list for the query.

The distributed exhaustive search has become the *de facto* search architecture for large collections. Many search systems also support some form of *query optimization techniques* (Section 2.1.2) in order to improve search efficiency. We also experiment with an optimized exhaustive search system in a later chapter of this dissertation and compare it with the proposed approach. For the remaining chapter, however, we select unoptimized distributed exhaustive search as a baseline due to its simplicity, and easy interpretation.

3.2 Proposed approach: Distributed selective search (DSS)[2]

We are interested in a search approach that can process a large document collection efficiently and effectively using few computational resources. Using distributed exhaustive search to achieve this goal is a challenge. By design exhaustive search is a resource-greedy approach. As such, an alternative approach that effectively balances search effectiveness and efficiency without placing enormous demands on computing resources is needed. In order to make this more specific we define a set of objectives that the new approach must satisfy.

- **Competitive search effectiveness**: The search results should be as accurate as those obtained with a strong baseline, on average.
- **Low search effort**: The average amount of search effort expended per query (θ_P) should be substantially lower than that needed by the baseline system (θ_B), $\theta_P << \theta_B$.
- **Short query runtime**: The query response time (t_P) should be substantially shorter than that with the baseline approach (t_B) given the same amount of computing resources, $t_P < t_B$.
- **Low resource requirements**: The proposed approach should be able to operate in low-resource environments where the number of available processing cores are relatively small

[1] We use the following terms interchangeably: distributed exhaustive search, DES, and exhaustive search.
[2] We use the following terms interchangeably: distributed selective search, DSS, and selective search.

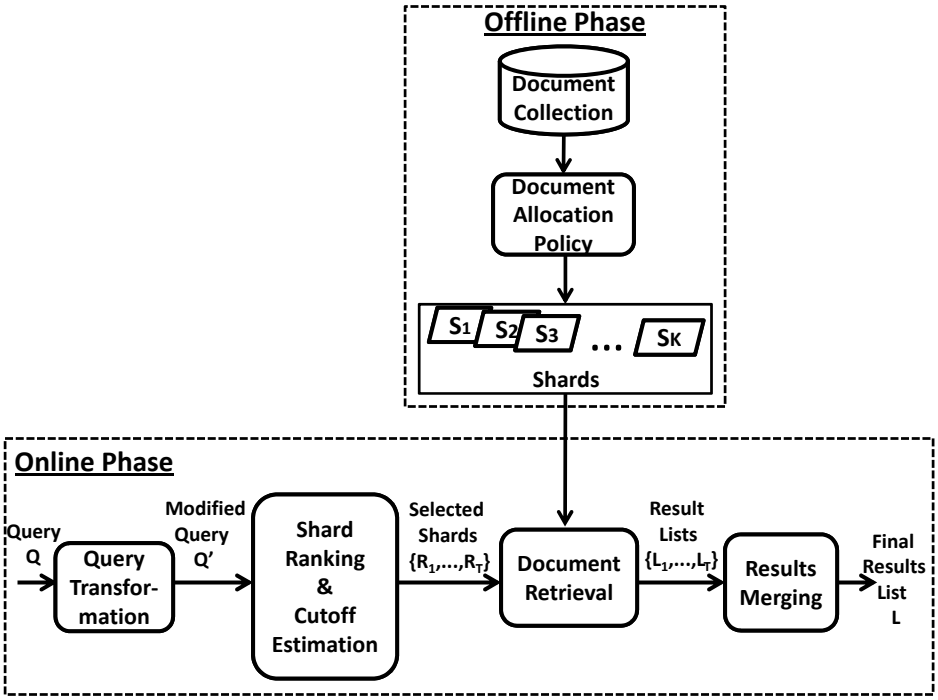


Figure 3.1: Schematic diagram of distributed selective search architecture.

(10–50 cores), and the main memory requirement is also low (8GB–32GB).

- **Scalability**: The search approach should be able to satisfy all of the above requirements for a wide range of collection sizes (1 million to 1 billion documents).

While developing an approach that meets the above objectives, we also require it to respect the following constraints and assumptions.

- We expect the approach to not use any external resources such as query logs, and labeled data. This allows the approach to be **widely applicable**.
- We assume that the collection to be searched consists only of **textual data**, that is, no images, maps, or audio files.
- We also assume a **cooperative environment** where the search system has a complete control over the organization, storage and search of the dataset.

We propose *distributed selective search* as the potential solution that meets the above objectives and requirements. The search architecture that supports selective search is first described in this section, followed by the details about the approach itself.

Like distributed exhaustive search, the selective search architecture, illustrated in Figure 3.1, also adopts a two-phase setup. In the offline phase the document collection is organized into

multiple smaller partitions (shards) using a *document allocation policy*. Query processing happens in the online phase where the query may be first transformed into a richer representation. In the next step of *shard ranking and cutoff estimation* the shards are ranked based on their estimated relevance to the query. The top few shards are searched in the *document retrieval* step, and the result lists from each of these shards are merged in the final component of this architecture.

The distributed selective search approach is based on two hypotheses. The first assumes that a document collection can be partitioned such that the majority of the relevant documents for a query are concentrated in a few shards without any prior knowledge of the query-set. The second hypothesis conjectures that when a collection is thus partitioned the search can be restricted to a few shards that have been identified as relevant for the query, and the retrieved results can be just as accurate as exhaustive search.

The first hypothesis is addressed with the *Cluster Hypothesis* which states that closely associated documents tend to be relevant to the same requests. We leverage this observation to develop a topic-based document allocation policy that partitions the collection based on document similarity. If Cluster Hypothesis holds then a topic-based document allocation would create shards that concentrate the relevant documents for a query into a few shards. For example, the majority of the relevant documents for the query *march madness* would be concentrated in the *sports* shard(s), and those for the query *barack obama on sequester* would be in the *politics* shard(s).

The second part of the selective search hypothesis recognizes that for each query a different set of shards might be relevant. As a result, the ability to rank the shards based on their estimated relevance to the given query is needed (shard ranking). Also, the number of top shards that should be searched for the query must be computed (shard rank cutoff estimation). We recognize the inter-dependent nature of these two problems and propose a family of three algorithms that provide a joint modeling of the shard ranking and shard rank cutoff estimation problems. For a given query each of these algorithms identifies the shards that are likely to contain the majority of the relevant documents for the query. During the document retrieval step the query is executed against each of the selected shards.

The results returned by each of the shards searched for the query are merged into a single result list in the final step of the selective search process. The merging of the ranked lists is straightforward due to the cooperative environment assumed for this search approach. The relevance scores of documents retrieved from different shards are comparable because the global statistics such as the idf (inverse document frequency of the complete collection) that are used for score normalization are compiled and shared with each of the shards in advance at partitioning time.

At a high level selective search attempts to strike a balance between the competing goals of high search accuracy and low search effort by controlling the partitioning of the collection into shards, and by reducing the search space for each query. Since selective search processes the

query against only a few shards it is easy to see why it would easily function in low-resource environment.

The ability of the selective search approach to provide competitive search effectiveness hinges on two components of this architecture. If topic-based document allocation is able to provide a skewed distribution of relevant documents across shards, and if the shard ranking algorithms are able to exploit this distribution, then the search can be restricted to just a few shards without hurting effectiveness. However, this might be more or less true for some queries than others. As such, a detailed study of the search effectiveness supported by selective search is one of the primary goals of this thesis.

Applying a sophisticated document allocation policy comes with a side-effect of higher computational cost of the offline phase. As such, one of the important research challenges is to design and develop a document allocation policy that is able to parallelize the partitioning process, is efficient, and scales sub-linearly in the collection size. While developing these approaches we also observe the restriction on using external resources in order to maintain a wide applicability of the proposed search approach. Although, shard creation is the single most computationally intense task of this architecture it is worth noting that it is executed only once for a static (or near-static) collection. Even for a more dynamic collection a complete re-partitioning would only be warranted if the topical structure of the dataset changes drastically.

In addition to achieving competitive search effectiveness, it is crucial to verify that selective search offers real and consistent savings in search effort. Since selective search processes the query against only a few shards, we would expect the corresponding search effort to be substantially smaller than that with the baseline, exhaustive search. However, for some queries this might not be true because the selected shards may contain all the candidate documents for the query. In this case, the search effort for both, selective search and exhaustive search, would be same. Validating that such cases are rare, and that on an average selective search is more efficient than the baseline is an important objective for this work.

A thorough investigation of each of these research problems is the goal of this thesis. A detailed treatment of every individual component of this architecture is provided in the following chapters. In the remainder of this chapter we describe and experiment with a *reference search system* that demonstrates an end-to-end functioning of the distributed selective search system. In the next section we start by introducing the datasets used through out this dissertation.

3.3 Datasets

Three of the largest available datasets were used in this work: GOV2, the CategoryB portion of ClueWeb09 (CW09-B), and the English portion of ClueWeb09 (CW09-Eng). These are summarized in Table 3.1. The GOV2 corpus [22] consists of 25 million documents from US gov-

Table 3.1: Datasets.

Dataset	Size (uncompressed) (GB)	Number of Documents	Number of Words (billion)	Vocabulary Size (million)	Average Document Length
GOV2	400	25,205,179	23.9	39.2	949
CW09-B	1500	50,220,423	46.1	96.1	918
CW09-Eng	15000	503,903,810	381.3	1,226.3	757

Table 3.2: Query sets.

Dataset	Query Set	Average Query Length	Average Number of Relevant Documents per Query	Number of Relevance Levels
GOV2	701-850	3.1	181 (± 149)	3
CW09-B	TREC09:1-100	2.1	81 (± 45)	5
CW09-Eng	TREC09:1-100	2.1	127 (± 67)	5

ernment domains, such as .gov and .us, and also from government related websites, such as, www.ncgov.com and www.youroklahoma.com[3]. TREC topics 701-850 were used for evaluation with this dataset and the statistics for these queries are provided in the Table 3.2.

ClueWeb09 is a newer dataset that consists of 1 billion web pages crawled in January and February 2009. Out of the 10 languages present in the dataset the English portion contributes about half of the pages. The two datasets used in this thesis were created from this English portion of ClueWeb09. The CW09-B dataset consists of the first 50 million English pages and the CW09-Eng consists of all the English pages in the dataset (over 500 million). For evaluation with both CW09-B and CW09-Eng datasets we use the 100 queries that were used in the Web track at TREC 2009 and 2010.

3.4 Evaluation Metrics

Three different aspects of retrieval performance are evaluated in this work — retrieval effectiveness, stability and search efficiency. The following sections describe the metrics we use in this thesis to model each of these three aspects.

[3]http://ir.dcs.gla.ac.uk/test_collections/GOV2-summary.htm

3.4.1 Search effectiveness

We measure search effectiveness using several Precision metrics. Specifically, we analyze results of P@{10,30,100} metrics. The P@n metric measures the fraction of relevant documents in the top n ranks. Typically, as the n increases search systems struggle to maintain the corresponding search precision. This motivates us to include metrics such as P@30 and P@100 to test the efficacy of distributed selective search at deeper ranks.

We also evaluate using the mean average precision (MAP) metric, which is one of the most commonly used summary metrics due to its perceived stability for comparing systems. MAP computes the arithmetic mean of *Average Precision* of all the evaluation queries. The Average Precision (AP) is computed by averaging the precision value at each relevant document in the result list until some rank. The standard TREC evaluation uses the top 1000 results for AP. Unlike the P@n metrics, MAP models precision and *Recall*, both in its formulation. The Recall for a result set is the fraction of total number of relevant documents for the query that could be retrieved. Providing competitive performance on metrics that model search Recall is especially challenging for systems that search only a subset of the collection. Although prior work almost never evaluated using MAP, we include this metric in order to understand the limits of the proposed search approach, and to provide a Recall-based evaluation.

One of the drawbacks of MAP is its inability to distinguish between different grades of relevance in the results list. To compensate for these known shortcomings we also experiment with the metric referred to as the Normalized Discounted Cumulative Gain at rank cutoff of 100 (NDCG@100) [43]. The NDCG metric measures the worth of a document based on its relevance and position in the results list.

The above metrics (P@10,30,100, MAP, and NDCG@100) together provide a thorough evaluation of search effectiveness across different Recall levels. Prior work has typically used only a subset of the above metrics. The difference between two values of these metrics was tested for statistical significance using the paired T-test for all of the experiments reported in this dissertation. When testing if an experimental configuration is significantly better than the baseline, we have used a stricter significance criterion ($p < 0.01$). For the experiments where the goal is to verify that the experimental configuration is not worse than the baseline, we have employed a significance criterion of $p < 0.05$, so that even relatively small degradations compared to the baseline are identified.

3.4.2 Stability

The search effectiveness metrics described above enable comparative analysis based on average-case retrieval performance. The objective of the *stability* analysis described in this section, however, is to quantify the performance of individual queries, specifically when contrasted with the performance of the baseline system, distributed exhaustive search (DES). The motivation for

stability analysis originates from the observation that a search system with good average-case accuracy might still exhibit high variance across queries and thus lead to a poor user experience.

For this analysis queries are categorized along two dimensions based on their individual search accuracies. One of the dimensions, *improvement levels*, measures the percentage of queries for which selective search accuracy is,

worse: $P@r(DSS) < P@r(DES)$,
equal: $P@r(DSS) = P@r(DES)$,
better: $P@r(DSS) > P@r(DES)$,

than the exhaustive search accuracy. These are exact comparisons with zero tolerance. In the best case, selective search would perform at least as well or improve over the exhaustive search accuracy for all the queries.

The other dimension, classifies queries into *difficulty levels* based on their performance with exhaustive search (DES). Specifically, a query is categorized as,

hard: if $P@r(DES) <= 0.2$,
moderate: if $0.2 < P@r(DES) <= 0.6$,
easy: $P@r(DES) > 0.6$

The thresholds were chosen to model our intuition of query difficulty and are not necessarily supported by any IR theory. The intent of these query difficulty levels is to provide information about the types of queries for which selective search is or is not effective. For instance, the *improvement levels* together with *difficulty levels* can quantify, the percentage of *hard* queries that do *better* versus the percentage *easy* queries that do *better* with selective search. We can also ask, for example, whether selective search is as effective for *hard* queries as it is for *easy* queries. The best case stability would be when the *worse* category is empty.

3.4.3 Search efficiency

We quantify search efficiency using two different types of metrics. The search cost metrics, that are described below, quantify the total amount of search effort expended for a query by the given search approach. For the most part of this dissertation we use the search cost metrics to model search efficiency, and the motivation for this is also discussed below. Later, in Chapter 6 we also quantify efficiency in terms of query runtime where the total search time for a query-set is reported.

3.4.4 Search effort: Cost-in-Documents (CiD)

Data transfer from the hard disks to memory is almost always the costliest component of query processing in a large-scale search system. The volume of data fetched from the disk for a query

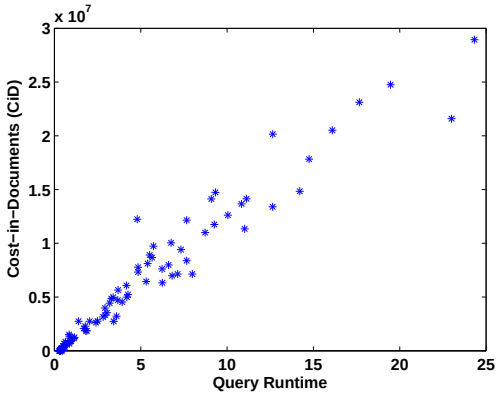


Figure 3.2: Query runtime versus the number of postings evaluated for query.

is proportional to the number of postings scored for a query (discounting caching effects). Also, the number of computations performed for a query is directly proportional to the number of postings evaluated for a query. The results in Figure 3.2 validate this relation by comparing query's runtime with the number of postings evaluated for it. For this analysis we used the first 100 queries from the TREC 2009 Million Query track. Each query was processed against a single Indri[4] index using a single processor. The document collection consisted of about 51 million documents from the CW09-Eng dataset.

The Pearson's correlation coefficient between the two variables, query runtime and number of postings evaluated for query, is 0.98. Prior work that studies search efficiency also employed posting list based efficiency metrics [10, 72]. These observations motivate the primary metric for search efficiency used in this dissertation, *cost-in-documents* (CiD). For selective search the CiD metric consists of two components. The total number of postings scored at all the selected shards for the query, CiD_R^q, is the first part of the metric.

The second component captures the cost of the shard selection step. The number of postings evaluated for the query by the shard ranking algorithm is used to model the cost of this step. For shard ranking algorithms such as ReDDE that adopt the *sample-based* model, the cost of this step is proportional to the number of postings evaluated for the query at the *sample index* (Section 2.3.2). For *model-based* approaches (section 2.3.1) such as CORI, the cost of shard ranking is proportional to the number of shard models evaluated for the query. In either case we refer to this quantity as CiD_{SS}^q, and the CiD^q for a query is simply the addition of CiD_R^q and CiD_{SS}^q. The CiD values reported in result tables and figures throughout this dissertation are the CiD^q averaged over all the evaluation queries. Since the shard selection step is not needed for exhaustive search, its CiD metric consists only of the CiD_R component.

[4]http://www.lemurproject.org/indri/

3.4.5 Search effort: Cost-in-Shards (CiS)

There are also peripheral costs associated with query processing and they can be non-trivial. In a distributed search system costs associated with operations, such as, disk seek are replicated at each node processing the query. In general the cost of such operations is proportional to the number of shards searched for the query. The network cost of sending the result lists to the server that merges the results is also a function of the number of shards searched for the query. We thus define a second cost metric, *cost-in-shards* (CiS), the average number of shards searched per query, as the other measure of the search system's efficiency.

3.4.6 Search effort: Motivation

For both the cost metrics we choose to abstract away from system-dependent variables, such as, the disk seek time, and the network cost. Instead we work with the above cost metrics which are not dependent on the particular instrumentation of the system. For similar reasons we choose to not model the effects of caching in the formulations of search cost described above. The efficacy of caching is strongly dependent on various factors, such as, the current query stream, query workload, the other processes on the machine, and system specifications (memory size). Instead the above cost metrics offer more manageable formulations that can be thought of as providing upper-bounds on the search cost.

The other motivation for quantifying efficiency using CiD and CiS is the following practical constraint. Many of the experiments in this dissertation were performed using a simulated search environment. Specifically, the partitioning of the datasets and the shard searches were simulated using a single or a small number of inverted indexes. The simulated search setup allowed us to explore a wide range of research problems efficiently. However, simulated search cannot provide accurate query runtime estimates. Instead the above described cost metrics model search efficiency more accurately in a simulated search environment. Although we use CiD and CiS as the primary efficiency metrics in this dissertation, we also provide a query runtime based efficiency analysis of distributed selective search in Chapter 6.

3.5 Summary

Overall, the three types of evaluation metrics described above, search effectiveness, stability, and search efficiency, together provide an evaluation framework that is sophisticated and thorough.

3.6 A reference system: Setup

In this section we present a fully specified reference search system that demonstrates the functioning of the complete selective search pipeline and its evaluation. We start by describing

Table 3.3: Distributed selective search versus distributed exhaustive search.

Dataset	Search Type	CiS	CiD (million)	P@10	P@30	P@100	NDCG @100	MAP
GOV2	Exhaustive	10 (/10)	3.63	0.58	0.52	0.42	0.47	0.32
	Selective	10 (/50)	0.87	0.47	0.37	0.21	0.25	0.07
CW09-B	Exhaustive	10 (/10)	5.37	0.27	0.26	0.21	0.12	0.18
	Selective	10 (/100)	0.76	0.21	0.14	0.06	0.10	0.03
CW09-Eng	Exhaustive	10 (/10)	51.29	0.13	0.13	0.12	0.11	0.07
	Selective	10 (/1000)	2.56	0.07	0.04	0.01	0.02	0.00

the experimental setup.

For the offline phase, we used a simple document allocation policy that assigns each document to one of the shards at random. We do not expect random shards to support accurate selective search but we use it for its simplicity which is important at this point in this dissertation. The three datasets, GOV2, CW09-B, and CW09-Eng were partitioned into 50, 100, and 1000 shards, respectively. Each shard contained about half a million documents.

During the online phase every query was first transformed into of *full-dependence model* representation [52] that explicitly asserts the dependencies among the query terms (Section 5.1). All of the following steps used the transformed query. Next, a *resource selection algorithm* from federated search, ReDDE, (Section 2.3.2) was used to rank the shards for the query. The *centralized sample index* needed for ReDDE was created by sampling 4% documents from each shard. For nearly all of the experiments in this dissertation that use ReDDE we have used the same sample size of 4%. This sample size was chosen based on the analysis presented by Callan, 2001 [15] where resource representations learned from 3% to 6% samples of each resource performed only slight worse than the complete resource representations.

In the document retrieval step the top 10 shards in the ranking proposed by ReDDE are searched for each query using Indri [51], an inference network and language modeling based retrieval algorithm. The number of shards searched for selective search was used to guide the number of shards that the dataset is divided into for distributed exhaustive search. Specifically, each dataset was partitioned into 10 random shards for exhaustive search. This strategy allocates the same number of computing resources to both search approaches, which is necessary for a fair appraisal. The same retrieval model, Indri, was used by distributed exhaustive search as well.

3.7 A reference system: Results

The search effectiveness and efficiency, expressed in terms of metrics described in the previous section are provided in Table 3.3. For all the three datasets the effectiveness of distributed selective search is substantially worse than that of exhaustive search across all the effectiveness metrics. The P@10, for example, is 19%, 22%, and 46% lower for GOV2, CW09-B, and CW09-Eng, respectively. The corresponding search efficiency, quantified in terms of the two cost metrics, is much better for selective search. For GOV2, the CiD cost for selective search is 76% lower than that for exhaustive search. Similarly for CW09-B and CW09-Eng the CiD is 86% and 95% lower, respectively.

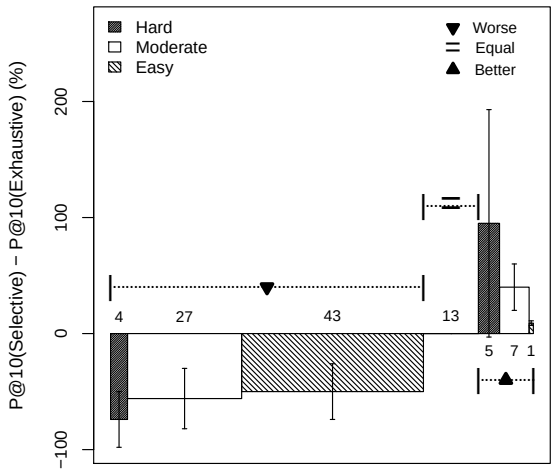
None of the above trends are surprising. The improved efficiency is explained by the fact that only 10% or fewer shards were searched for any of the datasets. It also explains in part the degradation in search effectiveness. However, the other important cause of the poor search effectiveness is the random document allocation policy used to create the shards. This allocation policy has an effect of spreading the relevant documents for a query across all the shards. As a result, the shard ranking step of selective search is not able to offer much. In fact, searching any 10 shards would lead to search effectiveness that is comparable to the one reported in Table 3.3. These results underscore the importance of the document allocation policy used for shard creation in case of selective search. Thus the next chapter starts with the analysis of this component of the proposed search architecture.

The stability analysis for P@10 metric is illustrated in the plots in Figure 3.3. The three improvement levels are along the X-axis with the span markers. The three query difficulty levels are represented as individual bars within the improvement spans. The magnitude of improvement is represented by the bar's height.

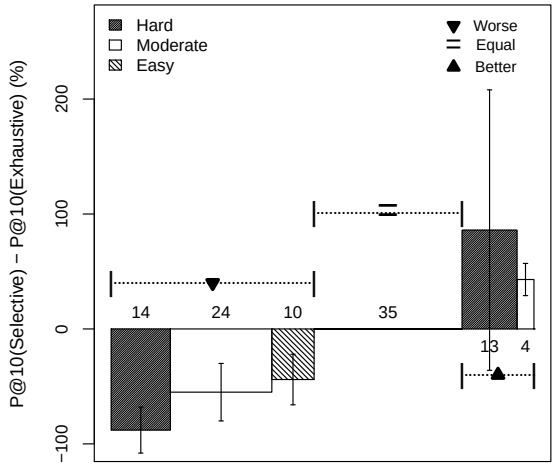
For the smallest dataset, GOV2, only 13% of the queries do the same as exhaustive. Whereas, for the largest dataset, CW09-Eng, nearly half of the queries (46%) are exactly as precise as with exhaustive search. This is remarkable given that only 1% of the shards (10 out of 1000) were searched by DSS for this dataset. We see a positive correlation between selective search effectiveness and collection size. These results suggest that selective search can offer better efficiency and effectiveness trade-off for larger collections.

Furthermore, for the same dataset, CW09-Eng, the precision of 21% of the queries improve by about 50% on average when only 1% of the collection is searched, instead of the complete collection. For all the datasets we see that a non-negligible fraction of queries improve when the search system can access only a limited portion of the collection. This is contrary to the common belief that exhaustive search provides the highest possible search effectiveness for a query.

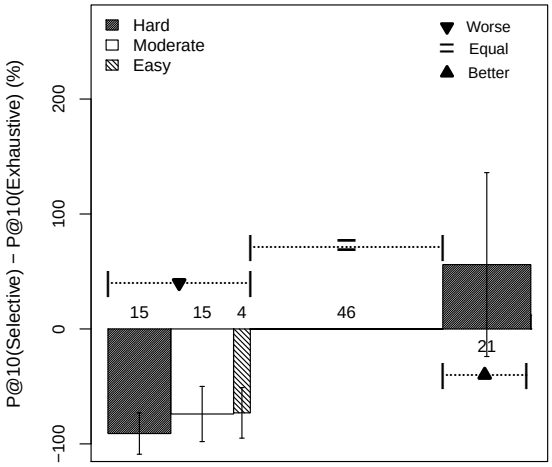
The improvements in effectiveness observed at query-level are not visible at the aggregate-level. As was observed in Table 3.3 the average-case precision is significantly lower. The



(a) GOV2 dataset. X-axis: % of queries, binned by improvement and difficulty levels.



(b) CW09-CatB dataset. X-axis: % of queries, binned by improvement and difficulty levels.



(c) CW09-CatA-Eng dataset. X-axis: % of queries, binned by improvement and difficulty levels.

Figure 3.3: Stability analysis for P@10

percentage of queries that do worse with selective search and the magnitude of the loss in precision are the primary cause of the lower overall performance. Although these results are specific to the P@10 metric, the stability trends for the remaining metrics are similar.

3.8 Summary

This chapter introduced a new search approach, distributed selective search, that exploits two inherent properties of large-scale search, parallelism and occurrence-redundancy, to achieve its goal. The proposed approach improves efficiency by searching only a small fraction of the collection, and maintains the search effectiveness by being clever about organizing the collection.

The overall architecture of distributed selective search was presented in this chapter and the individual components are described in detail in the following chapters. A reference system that demonstrates the workings of the complete architecture was also presented and evaluated against a baseline search system in this chapter. The datasets, evaluation methodology, and the baseline search system presented in this chapter are used through this dissertation. The next chapter is devoted to the offline task of partitioning the collection into shards for distributed selective search.

Chapter 4

Offline phase: Shard Creation

When partitioning a large document collection into shards for distributed selective search (DSS), several research challenges arise. First, the shard creation approach developed for this task needs to be highly efficient and scalable in order to be applicable to large collections. Second, it also needs to create shards that facilitate effective selective search. A thorough analysis of several shard creation approaches that satisfy these requirements to a varying degree is provided in the first section of this chapter. Other related research problems, such as, the number of shards that a collection should be partitioned into, and the size distribution of the created shards, are also studied in later sections.

4.1 Document allocation policies[1]

A document collection can be divided into a set of partitions using many different approaches. One of our requirements for the document allocation policy is that it be generally applicable. We thus restrict ourselves to only those *document allocation policies* that do not rely on external resources such as query-logs or categorization schemes, which were both used in previous work [47, 60]. Although such resources can provide valuable additional information that can be used in the allocation decision, they are not always available.

We also assume that the document allocation policy must operate on large document collections and might have access to limited computational resources. Efficiency of the approach is thus an important consideration. Algorithms that can be partially or fully parallelized are preferred. As such, algorithms that scale linearly or sub-linearly in collection size and can be parallelized would be best suited for this work.

The set of shards constructed by the document allocation policy needs to support competitive search effectiveness. When the relevant documents for a query are concentrated in a small number of shards the search can be restricted to those shards without compromising on the

[1]This work was published in LSDS-IR 2010 [45], and CIKM 2010 [44]

search effectiveness. Thus the allocation policy needs to be capable of achieving such an organization of the collection without any prior knowledge of the query stream that will be served by the search system.

We study three type of document allocation policies that satisfy the above requirements to varying extent: random, source-based and topic-based, which are described next.

4.1.1 Random document allocation

The random allocation policy assigns each document to one of the shards at random with equal probability. This policy has the advantages of being the most efficient, scalable, easy to implement, and applicable to any document collection. It also naturally lends itself to parallelization. Appending a new set of documents to an existing set of shards is also straightforward. It is thus not a surprise that commercial Web search engines such as Google [7] and Bing[2] employ this allocation policy for sharding. At query time, this allocation policy also offers the advantage of spreading the computational load evenly across the cluster.

One might not expect a random policy to be effective for distributed selective search since it is likely to spread the relevant documents evenly across multiple shards. Nonetheless it is a contender because of its advantages, its use in prior research [60], and its use by large-scale operational search systems.

4.1.2 Source-based document allocation

The information about the *source* of each document in the collection is typically available. Some of the examples of the source of the document would be the company department or the agency that created or maintains the document, the Web host that supplied the document [85], or the legal case that the document was filed under. This information can be used to organize the collection into disjoint shards. The source-based allocation policy was widely used in prior research in distributed IR [14, 17, 28, 80, 85] with the intention of replicating real world distributed search environments consisting of independent document sources.

If we assume that documents from the same source are similar then as per the *Cluster Hypothesis* [77] such documents would also be relevant to same information needs. This suggests that a source-based partitioning would create shards that exhibit a skewed distribution of relevant documents, and thus are conducive to selective search. However, the above assumption of similar documents being members of the same source would often be violated. For example, the source, *wikipedia*, provides access to pages that are not similar or related. Such exceptions would lead to noisy or less cohesive shards and might affect search performance. Nonetheless, we believe that source-based organization offers a better distribution of relevant documents than

[2]Personal communication.

random allocation. We test this conjecture by developing a source-based allocation technique that is described next.

In previous work that examined source-based allocation [47, 85], document URLs provided the *source* information. Each unique top-level hostname was assigned a separate shard and the documents were partitioned accordingly. However, for many collections this strategy would lead to a large number of very small partitions. Thus instead we propose the following strategy for the source-based allocation policy.

In the first step, the collection is sorted based on document URLs, which arranges documents from the same website consecutively. In the second step, groups of M/K consecutive documents are assigned to each shard where M is the total number of documents in the collection and K is the total number of shards to be created. However, this is not a strict policy. Splitting of websites across shards is avoided wherever possible.

As compared to the random allocation policy, source-based allocation is computationally more complex. The first step of sorting the documents based on the URLs has a complexity of $O(|M|log(|M|))$, where $|M|$ is the number of collection documents. Also, this step can only be partially parallelized (for example, using merge sort) and thus is less efficient than the random policy. It is possible to design a more efficient source-based policy by avoiding the sort step on the complete collection. A simple hash function on the document URLs can be used to group the documents based on the website. A comparative analysis of these two variants of the source-based allocation policy is left for future work.

Similar documents often use common terminology. As a result, posting lists for shards created using a source-based policy are longer than for random shards. Longer posting lists could imply longer I/O during query processing. On the other hand, posting lists for source-based shards could offer a higher compression factor than random shards. Posting lists are often compressed using techniques such as delta encoding where instead of the complete document identifiers the differences in the identifiers are stored. Smaller deltas offer better compression. Source-based sharding would organize a larger percentage of documents with similar vocabulary in a consecutive order than random allocation. As a result, the delta values for the postings lists are typically smaller for source-based than those with random allocation. The resulting higher compression could compensate to some extent for the longer posting lists and ameliorate longer I/O for source-based shards.

4.1.3 Topic-based document allocation

Like source-based, the topic-based allocation also appeals to the *Cluster Hypothesis* [77] in order to group similar (and thus relevant) documents together. However, instead of using the source of the document as the proxy for a similarity metric, topic-based allocation explicitly models the similarity between documents. In this work, we approximate *semantic similarity* with *lexical similarity*. Documents that exhibit affinity when evaluated using lexical similarity

metrics are grouped together. In such an organization of documents where each shard is composed of lexically and thus semantically coherent set of documents, each shard can be seen as representing a unique *topic*. This grouping of documents into topics can also be thought of as a dimensionality reduction process. Thus the task of organizing the collection into shards can also be recast into that of learning a set of topics from the collection. We operationalize this intuition using two different topic modeling techniques: *K*-means clustering and Latent Dirichlet Allocation.

K-means based document allocation

The time-tested *K*-means algorithm [49] is one of the obvious choices for grouping *topically* similar documents into clusters (shards). *K*-means is a simple version of the *Expectation-Maximization* algorithm [26] that starts by partitioning the dataset into *K* clusters using a set of *K* seed centroids. Following this the algorithm iteratively alternates between the Maximization step where the centroid estimates are updated using the current dataset partitions, and the Expectation step where the dataset is repartitioned using the updated centroids.

K-means was successfully used in prior research [47, 85]. In the next subsection we first describe the limitations of the approach proposed by Xu and Croft [85], which motivates us to propose the improved approach described in the following section.

Approach by Xu and Croft, 1999 Xu and Croft [85] showed that the *K*-means clustering algorithm can be used to partition small collections into distributed indexes that support effective partial search. However, problems of scale and accuracy emerge when applying Xu and Croft's method to much larger datasets. Typically, a clustering algorithm is applied to the entire dataset in order to generate clusters. Xu and Croft also use the entire document collection to determine its final partitioning. Although the computational complexity of the *K*-means algorithm is linear in the number of documents ($|M|$), applying this algorithm to very large collections is still computationally expensive. Also, parallelization opportunities in the case of the standard *K*-means algorithm are limited to the *Expectation* step.

The *K*-means algorithm requires a similarity or a distance metric for document-to-centroid assignment during the expectation step. Xu and Croft [85] used the distance metric shown below.

$$dist(C^i, D) = \sum_{w \in D} \frac{c(w, D)}{|D|} \, log \frac{c(w, D)/|D|}{(c(w, D) + c(w, C^i))/(|D| + |C^i|)} \tag{4.1}$$

where $c(w, D)$ and $c(w, C^i)$ are the occurrence counts of word w in document D and centroid C^i, respectively. Equation 4.1 can be restated as shown below.

Algorithm 1 Sample-based K-means (SB K-means)

Input: Document collection $\boldsymbol{C}$, Sample size $|S|$, Number of shards K
Output: $\boldsymbol{R}_K$ Topical shards

1: $\boldsymbol{S} \leftarrow$ SAMPLE ($\boldsymbol{C}$, $|S|$) // Sample $\boldsymbol{S}$ documents from $\boldsymbol{C}$.

// Learn Phase
2: $\{\boldsymbol{CENT}_K, \boldsymbol{R}_K^S\} \leftarrow K$-MEANS ($\boldsymbol{S}$, K) // Cluster $\boldsymbol{S}$ documents into K sample-shards $\{R_1^S, \cdots, R_k^S\}$, with cluster centroids $\{CENT_1, \cdots, CENT_k\}$.

// Infer Phase
3: $\boldsymbol{R}_K \leftarrow$ PROJECT ($\boldsymbol{C}$, $\boldsymbol{CENT}_K$) // Use the K centroids to project the complete collection into K shards $\{R_1, \cdots, R_k\}$.

$$dist(C^i, D) = \sum_{w \in D} \frac{c(w, D)}{|D|} \, log \frac{c(w, D) \cdot (|D| + |C^i|)}{(c(w, D) + c(w, C^i)) \cdot |D|} \tag{4.2}$$

This formulation of the distance metric has three drawbacks. Firstly, the metric is biased toward centroids that are shorter in length. This is an artifact of the presence of $|C^i|$ term in the numerator (Equation 4.2). Secondly, this formulation allows for unequal contribution from the document and the centroid models by placing the second term inside a logarithm function. The terms that are important only to the document but not to the centroid can make a large contribution toward the distance value. Lastly, the term weighting that is provided by inverse collection or inverse document frequency is not available in this metric. Absence of any form of smoothing for the term estimates must also make those estimates unstable.

Sample-based K-means (SB K-means) We address the scalability problem using a simple modification to the standard K-means algorithm. We assume that for the purposes of distributed selective search topical clusters defined by a subset of the collection (instead of the entire collection) are sufficient. We propose a sample-based K-means (SB K-means) approach that applies the standard K-means algorithm to only a small sample of documents from the collection. The *cluster definitions* thus learned are then used to infer the topical assignment for each document. The two steps of SB K-means, *learn* and *infer*, are described in detail below using the pseudo code in Algorithm 1.

For the first step of the Algorithm 1, sampling a subset (S) from the collection, we employ simple random sampling (SRS). The SRS strategy chooses documents at random from the complete collection (without replacement). The size of the sample $|S|$ is simply determined based on the operational requirements. For example, $|S|$ can be chosen such that the next step of the algorithm, *Learn*, fits in the memory. The effects of this sample size selection strategy are

analyzed in Section 4.1.3. Keeping in mind the high efficiency requirement of the algorithm we note that the sampling step can be parallelized by splitting the collection into P disjoint sets of documents and then sampling $|S|/P$ unique documents from each set in parallel. The complexity of this step is $O(|S|)$.

In the *learn* step, the standard K-means clustering algorithm is applied to the sample S to generate K clusters (or sample-shards). There are several important parameters of the standard K-means algorithm that need to set for effective clustering of the sample. We perform a five-pass K-means where the centroids are updated at the end of each iteration. The set of centroids from the last iteration are used by the inference step. Using a fixed number of iterations of K-means allows us to limit the runtime of the algorithm and thus maintains its efficiency. Also, it is often the case that the clustering solution obtained using a fixed number of iterations is similar in quality to the one obtained using other convergence criterion such as stable document-to-cluster assignment [50]. The algorithm used to select the seed centroids for the K-means algorithm is also an important design choice [3, 35, 53, 75]. We study several existing algorithms and also propose a seed centroid selection in Section 4.3.

Instead of the distance metric (Equation 4.1) used by Xu and Croft we employ a symmetric version of negative Kullback-Liebler divergence (Equation 4.3) that computes the similarity between a document D and a centroid C^i. The first component in the equation computes $KL(C^i||D)$ which emphasizes the terms that are important to the centroid while the contribution of the terms in the document is dampened by the logarithm function. However, the second component compensates for this bias by emphasizing the terms that are important to the document.

$$sim(C^i, D) = \sum_{w \in C^i \bigcap D} p_{c^i}(w)\ log \frac{p_d(w)}{\lambda\ p_B(w)} + \sum_{w \in C^i \bigcap D} p_d(w)\ log \frac{p_{c^i}(w)}{\lambda\ p_B(w)} \tag{4.3}$$

$p^i_c(w)$ and $p_d(w)$ are the unigram language models of the cluster centroid C^i and the document D, respectively. $p_B(w)$ is the probability of the term w in the background model which is the arithmetic mean of the K centroid models. λ is the smoothing parameter.

Using the maximum likelihood estimation (MLE), the cluster centroid language model is,

$$p^i_c(w) = \frac{c(w, C^i)}{\sum_{w'} c(w', C^i)} \tag{4.4}$$

where $c(w, C^i)$ is the occurrence count of w in C^i. Following Zhai and Lafferty [86], we estimate $p_d(w)$ using MLE with Jelinek-Mercer smoothing which gives

$$p_d(w) = (1 - \lambda)\ \frac{c(w, D)}{\sum_{w'} c(w', D)} + \lambda\ p_B(w) \tag{4.5}$$

As such, the presence of $p_B(w)$ in the denominator plays an important role of incorporating the inverse collection frequency of the term into the metric which Zhai and Lafferty [86] found to

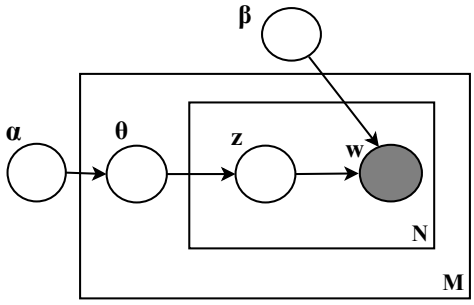


Figure 4.1: Graphical Model for Latent Dirichlet Allocation.

behave similar to the traditional inverse document frequency (IDF) statistic[3].

The standard *K*-means algorithm cannot be completely parallelized due to the Maximization step. However the *learn* step of the SB *K*-means algorithm operates only on a small subset of the collection (S) and is efficient as long as the size of S and the size of the corresponding vocabulary (V) is not too large. The complexity of the *learn* step is $O(|S||V|K)$.

The last step of the SB *K*-means algorithm infers the shard membership for each document of the collection. The KL divergence based similarity metric in the Equation 4.3 is used to compute the similarity between the document and the K shard centroids learned in the previous step. The document is assigned to the shard with which it exhibits highest similarity, and any ties are broken randomly.

Note that the above inference process for each document is independent of the inference for any other document. As a result, this step naturally lends itself to parallelization. In theory, the shard assignment for all the collection documents can commence in parallel. In practice, the number of available computing nodes upper bounds the parallelism supported by this step. Nevertheless, even massive collections can be efficiently partitioned into topical shards using this approximate version of the *K*-means algorithm. The overall complexity of this step is $O(|M||V|K)$ where $|M|$ is the number of documents in the collection, and V is the vocabulary.

The process of appending a new set of documents to an existing set of shards is no different from that of shard assignment in the *infer* step and thus can be performed very efficiently.

Sample-based Latent Dirichlet Allocation (SB-LDA)

Latent Dirichlet Allocation (LDA) [8] is a widely used text modeling approach. It operates in a generative probabilistic framework where the process of corpus generation is modeled at three levels - corpus, document and word. A corpus is assumed to be composed of multiple documents and each document is assumed to be a mixture of *latent topics* where each topic

[3]Please refer to [56] for the transformation of negative KL-divergence with smoothing into a similarity metric as used in Equation 4.3.

is defined by a distribution over words. The generative process for every document D in the corpus, in the presence of K topics can be stated as follows.

1. Sample a distribution over topics (θ, a K dimensional vector) from a Dirichlet distribution parameterized by the corpus parameter α, also a K dimensional vector.
2. For each word w in the document D:
   (a) Sample a topic z_w from a Multionomial distribution parameterized by θ which is a distribution over topics that was sampled in step 1; and
   (b) Sample a word w from $p(w|z_w, \beta)$ which is a multionomial distribution over words conditioned by the topic z_w.

where β is a k x V matrix, k is the number of topics and V is the vocabulary. Thus each row in the β matrix is a topic model defined over the words in V. Figure 4.1 provides *plate* representation of this model where the boxes (plates) represent the steps that are repeated. For example, the corpus consists of M documents and thus the document generation process of this model is repeated M times. The filled circle indicates the entity in the model that is observable, which in our case is the words.

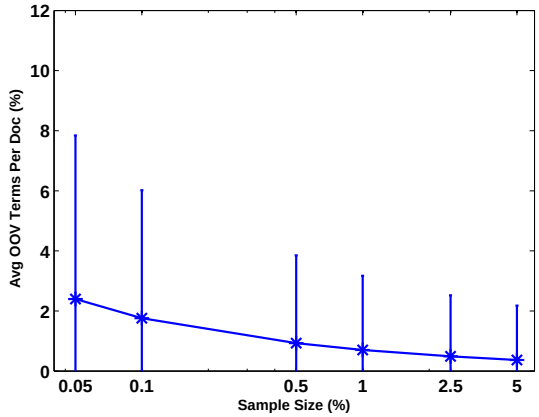
We use an implementation of LDA[4] that estimates the corpus parameters α and β, using variational approximation. The computational cost of parameter estimation for LDA is quite high, specifically $O(|M|N^2K)$, where $|M|$ is the number of documents in the collection, N is the average document length and K is the number of latent topics. We are interested in large document collections in this work, as a result, the value of the variable $|M|$ dominates the computational cost, that is $|M| >> N^2$. To control the complexity of parameter estimation, we perform estimation on a small sample S of M where $|S| << |M|$, like we did for SB K-means allocation policy. This reduces the computational complexity of this step to $O(|S|N^2K)$.

Once the K topic models are learned, the collection is partitioned into as many shards using LDA inference. For each document the LDA inference consists of estimating a distribution over the K topics using the learned topic models. This distribution is an estimate of the document's generation probability from each of the topics. The computational complexity of this step is $O(|M|NK)$. The document is assigned to the topic with highest generation probability. Note that the inference process for any document is independent of all other documents. As a result, the inference process can be easily parallelized.

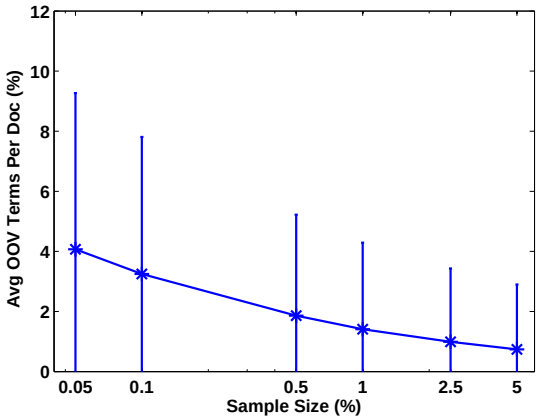
Sample-size and OOV terms

Using a subset (S) instead of the entire collection to learn topical clusters reduces the computational cost and makes the topic-based allocation technique efficient and scalable. However, it also introduces the issue of out-of-vocabulary (OOV) terms during inference. The remaining

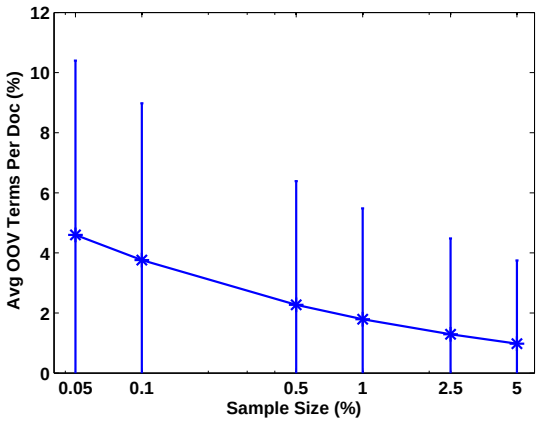
[4]http://www.cs.princeton.edu/~blei/lda-c/index.html



(a) GOV2 Dataset.



(b) CW09-B Dataset.



(c) CW09-Eng Dataset.

Figure 4.2: Sample size vs. percentage of OOV terms per document, on average. (Error bars indicate one standard deviation of the mean.)

documents in the collection are bound to contain terms that were not observed in S and thus are absent from the learned topic models. Inference must proceed using the seen terms and ignore the OOV terms. However, the inference quality can potentially degrade because of the discounting of the OOV terms. It may be important to select a sample size that leads to a small percentage of OOV terms per document. Note that inference occurs at the document level and thus as long as the percentage of OOV terms per document is small – even if the overall percentage of words that are out-of-vocabulary is high – the inference quality for each document would not be affected severely.

We know from Heaps' law [37] that when examining a corpus, the rate at which vocabulary is discovered tapers off as the examination continues. Based on this we hypothesize that using a relatively small sample might be sufficient to obtain an acceptably low percentage of OOV terms per document, on average. We verify this hypothesis empirically for each of the three datasets used in this dissertation.

Figure 4.2 (X-axis in log domain) plots the sample size versus the average percentage of OOV terms per document for GOV2, CW09-B and CW09-Eng datasets. A sample size as small as 0.05% of the collection is sufficient to discover more than 95% of the document vocabulary, on average. Also note that the drop in the average values is sub-linear in the sample size. This is in accordance with Heaps' law. Thus after a certain point there is little benefit in increasing the sample size because additional documents do not reduce the percentage of OOV terms significantly.

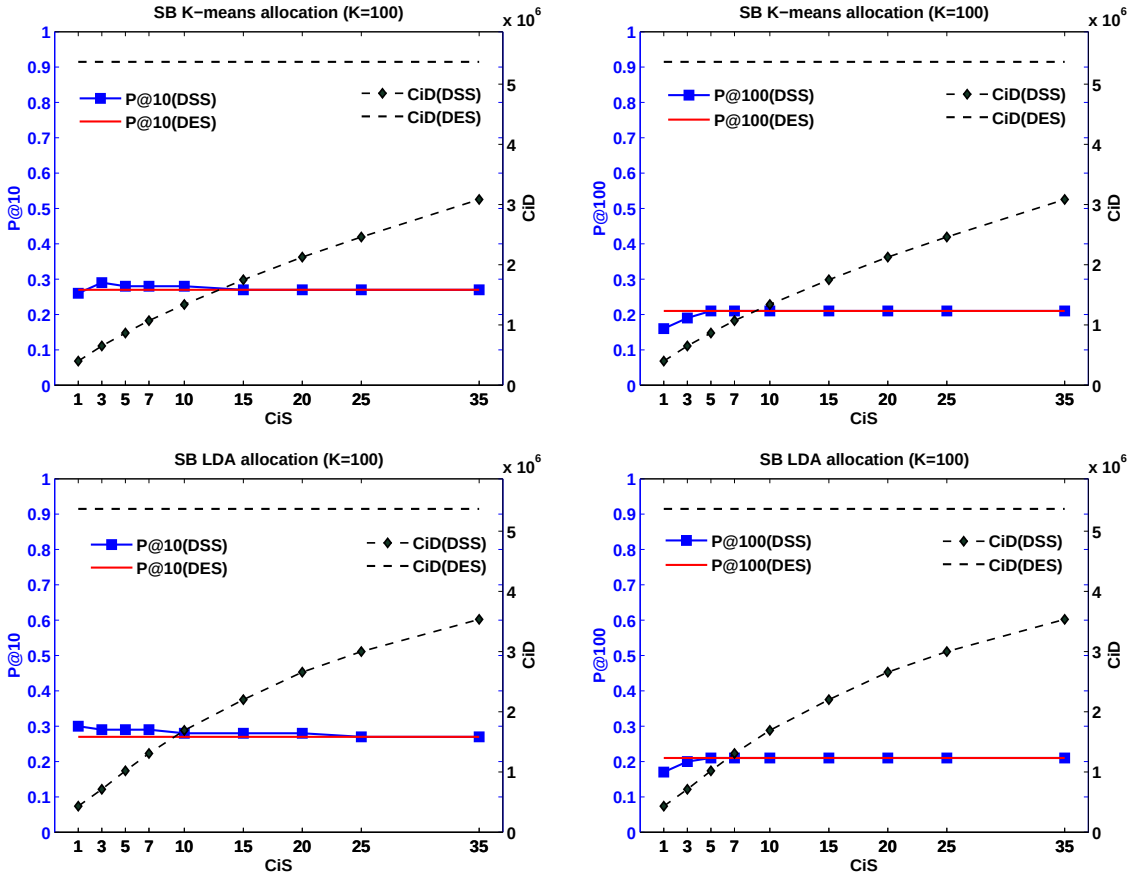
We leverage these observations to make our experimental methodology efficient. For GOV2 and CW09-B datasets we sample 1% (250K and 500K documents) and for CW09-Eng dataset we sample 0.1% (500K documents) of the entire collection using uniform sampling. These samples are used by the topic-based document allocation techniques to learn the cluster centroids.

4.1.4 Experimental results: Search effectiveness and efficiency

An empirical comparative analysis of the four document allocation policies described earlier is the focus of this section. We start with comparing the two topic-based allocation approaches — SB K-means and SB-LDA.

SB K-means versus SB-LDA

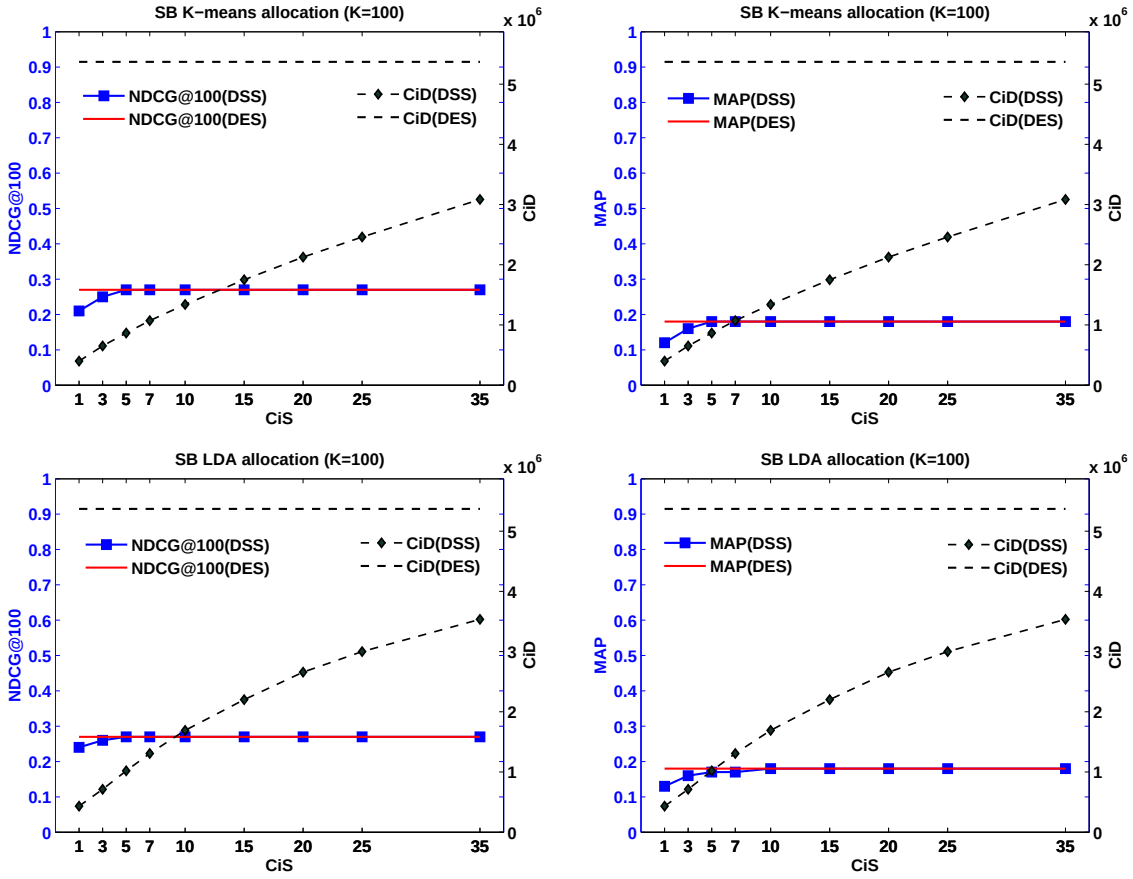
For this evaluation the CW09-B dataset was partitioned into 100 shards using both of the topic-based techniques. Everything else in the selective search framework was held constant across the two setups. Specifically, for both the experiments the full-dependence model query representation was used, the resource selection algorithm, ReDDE with 4% sample (described in Section 2.3.2), was employed for shard ranking, and inference network and language modeling based retrieval algorithm, Indri [51] was used for document retrieval at each searched shard.



(a) (b)

Figure 4.3: Distributed Selective Search with SB *K*-means and SB-LDA. Dataset: CW09-B. Metrics: P@10 and P@100. Distributed Exhaustive Search: CiD=5.37M.

As reported in Section 4.1.3 the sample-size chosen for CW09-B dataset was 1%, or about 500K documents (vocabulary: about 1M). The SB *K*-means algorithm operated on this sample. For efficiency reasons only 50K documents (0.1%, vocabulary: about 230K) could be used to learn the topic models with SB-LDA. In spite of its smaller sample size the SB-LDA based approach took 16 times longer than *K*-means for learning the topics. The time to learn the 100 topic models with LDA was more than 50 hours (wall clock time) while that with *K*-means was about 3 hours. Later, in Section 6.2 we report the *K*-means runtime for all the three datasets. The *K*-means implementation used for all the experiments in this dissertation was developed in-house. A production-grade implementation would be more efficient. Also, other optimization techniques such as *centroid pruning*, and use of efficient data-structures such as *kd*-trees [57], could achieve further speedup in *K*-means runtime. In general, *K*-means has a definite advantage over LDA in term efficiency.



(a) (b)

Figure 4.4: Distributed Selective Search with SB *K*-means and SB-LDA. Dataset: CW09-B. Metrics: NDCG@100 and MAP. Distributed Exhaustive Search: CiD=5.37M.

Figures 4.3 and 4.4 present the selective search results for the two topical shard creation techniques. The columns in the plot grid are different effectiveness metrics and the two methods are along the rows. Each plot in the grid reports three metrics. The left Y-axis is one of the search effectiveness metrics. The right Y-axis provides the primary cost, cost-in-documents (CiD). The secondary cost which is simply the number of shards searched, cost-in-shards (CiS), is along the X-axis. For convenient comparison the effectiveness with distributed exhaustive search (DES) is also reported in each of the plots. The exhaustive search costs are reported in the figure captions.

When comparing the topic-based allocation approaches with the baseline, exhaustive search, we see that both the topic-based techniques support selective search that is as precise or better than exhaustive search. Selective search of the single top ranked shard provides competitive accuracy at early ranks as measured by P@10 and NDCG@10. The corresponding cost in CiD

is more than an order of magnitude smaller than that with DES. When more than the top ranked shard is searched DSS supports higher effectiveness than that of exhaustive with both the allocation approaches. Some of these improvements are statistically significant for both SB *K*-means and SB-LDA. For precision at deeper ranks (P@100 and MAP) more top shards need to be searched for competitive performance. In case of SB *K*-means searching the top 5 shards is sufficient and the CiD is about a sixth of that of exhaustive. For SB-LDA the CiS cost is double of that of SB *K*-means, the top 10 shards need to be searched for competitive MAP and the CiD is about a third of that of exhaustive.

These are among the first results that empirically validate the potential of distributed selective search. These results demonstrate using a range of effectiveness metrics that selective search can be as effective as exhaustive search without evaluating every candidate document in the collection for the query. In fact, we see that searching only a small fraction of the collection is sufficient. In the following section these results are confirmed using two additional datasets.

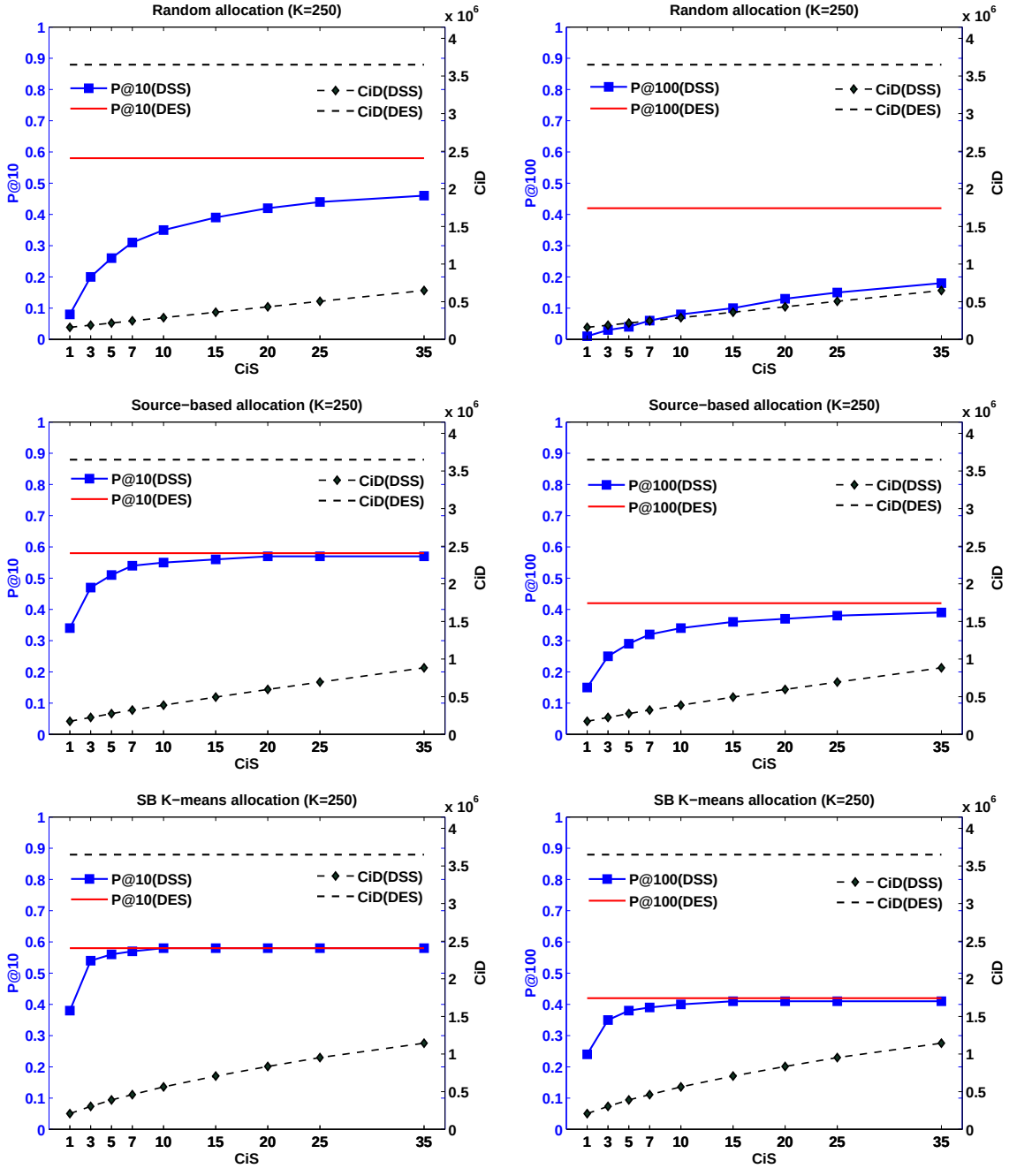
Recall that the samples used to learn the topic models using LDA and *K*-means in these experiments were a very small subset of the collection. These results demonstrate that an exact clustering solution that uses the entire collection is not necessary for selective search to perform at par with the exhaustive search. An efficient approximation to topic-based techniques can partition large collections effectively and facilitate selective search.

When comparing the two topic-based techniques, SB *K*-means and SB-LDA, we conclude based on the observed trends that during query processing the two techniques provide fairly comparable search performance. However, based on the difference in their computational complexities for shard creation we chose SB *K*-means for the remaining experiments reported in this dissertation. Henceforth when we refer to topical shards we imply shards created using the SB *K*-means approach. Next we present a detailed empirical comparison between the random, source-based and topic-based allocation policies using the three datasets described in Section 3.3.

Random, Source-based and Topic-based Allocation Policies

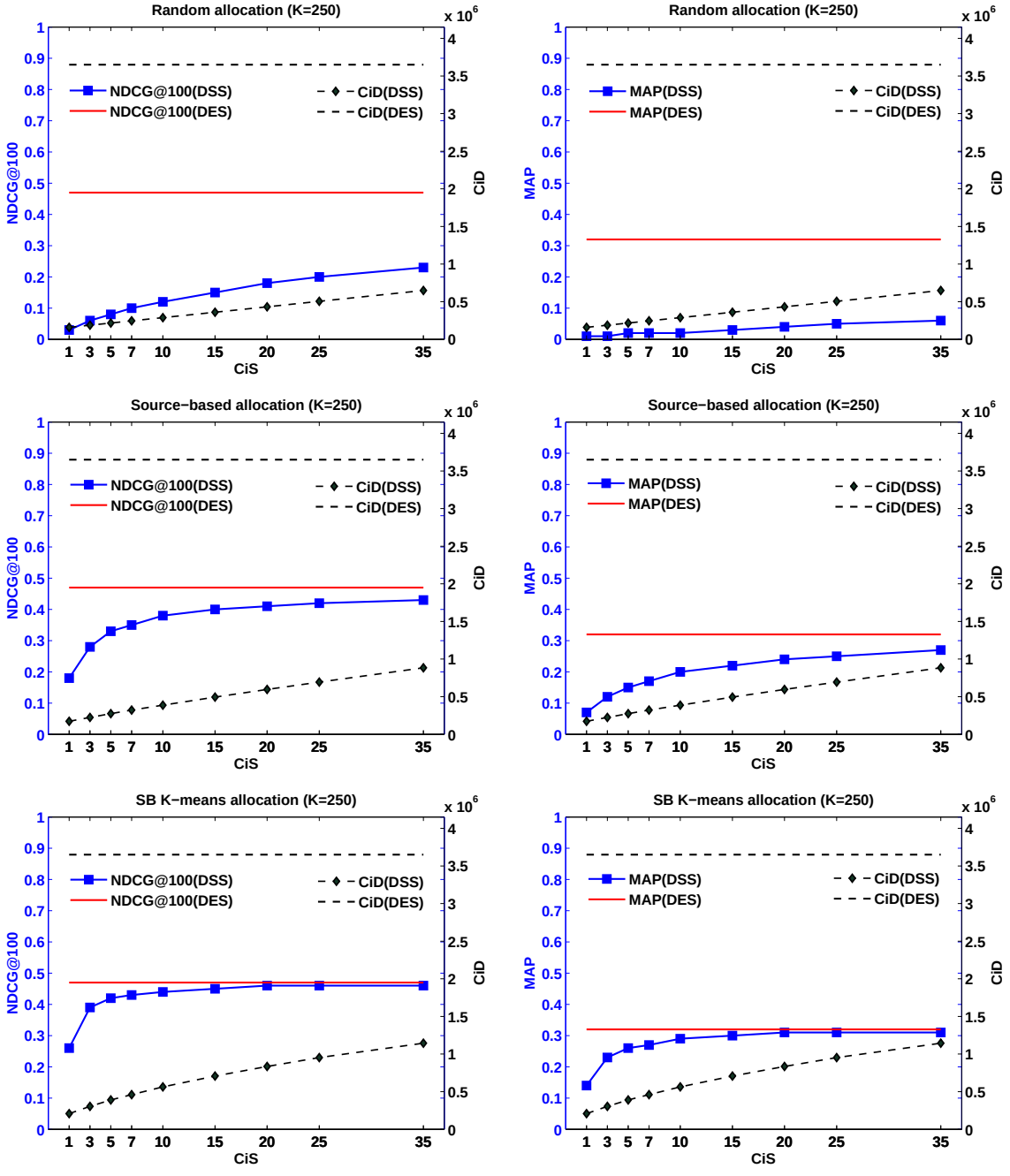
The ability of the different allocation policies to support competitive selective search is evaluated in this section. This analysis also tests the necessity of the *Cluster Hypothesis* for the success of selective search. The three allocation polices studied in this section conform to the Cluster Hypothesis at different levels, the random allocation being the least and SB *K*-means being the most complying method.

To enable this study the GOV2 dataset was partitioned into 250 shards using each of the three allocation policies, the CW09-B dataset was partitioned into 100 shards and CW09-Eng was divided into 1000 shards. The choice of number of shards for each dataset was guided by the analysis, described later in Section 4.2, that studies the impact of this parameter on selective search performance.



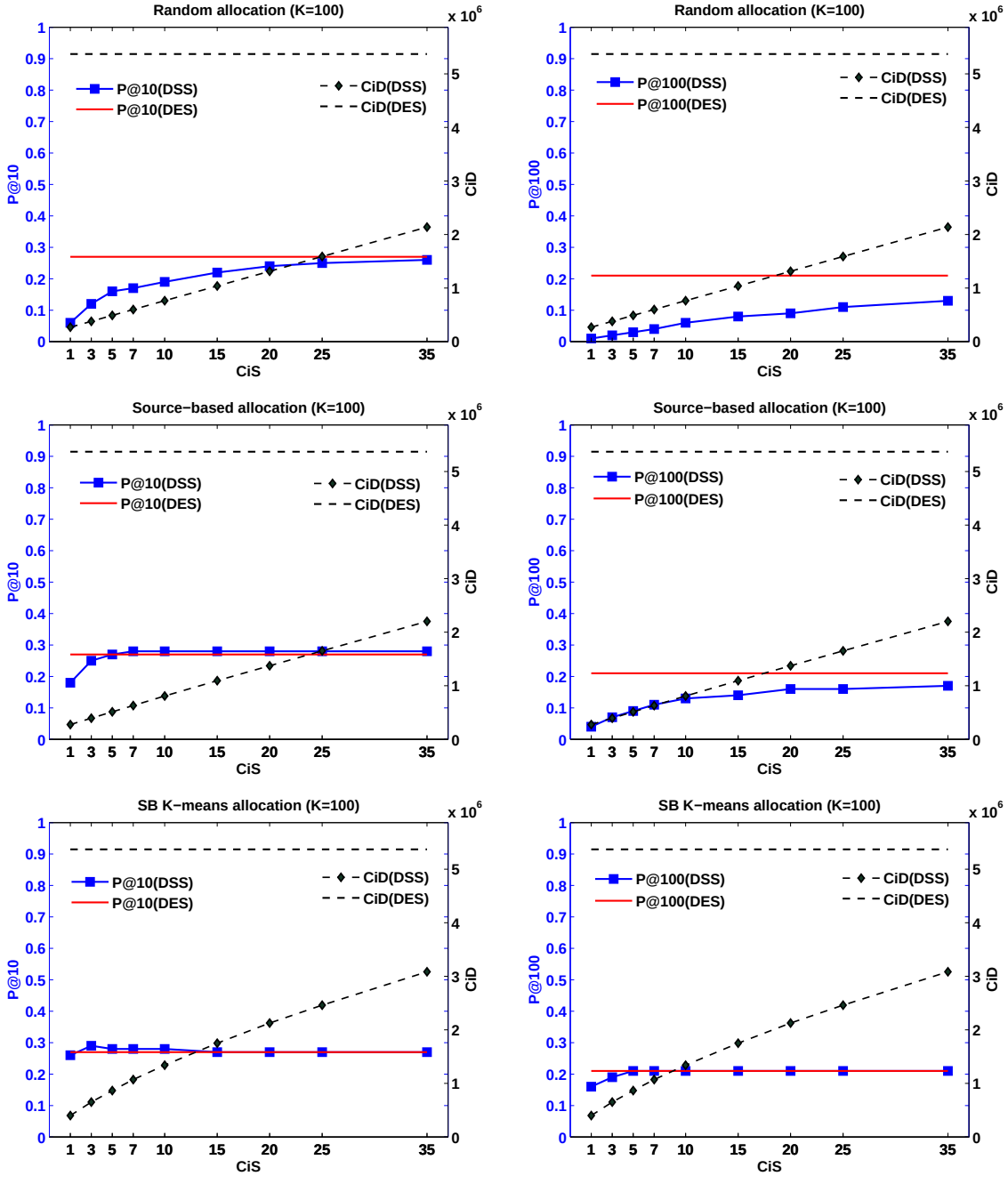
(a) (b)

Figure 4.5: Distributed Selective Search with Random, Source-based, and Topic-based shards. Dataset: GOV2. Metrics: P@10 and P@100. Distributed Exhaustive Search: CiD=3.63M.



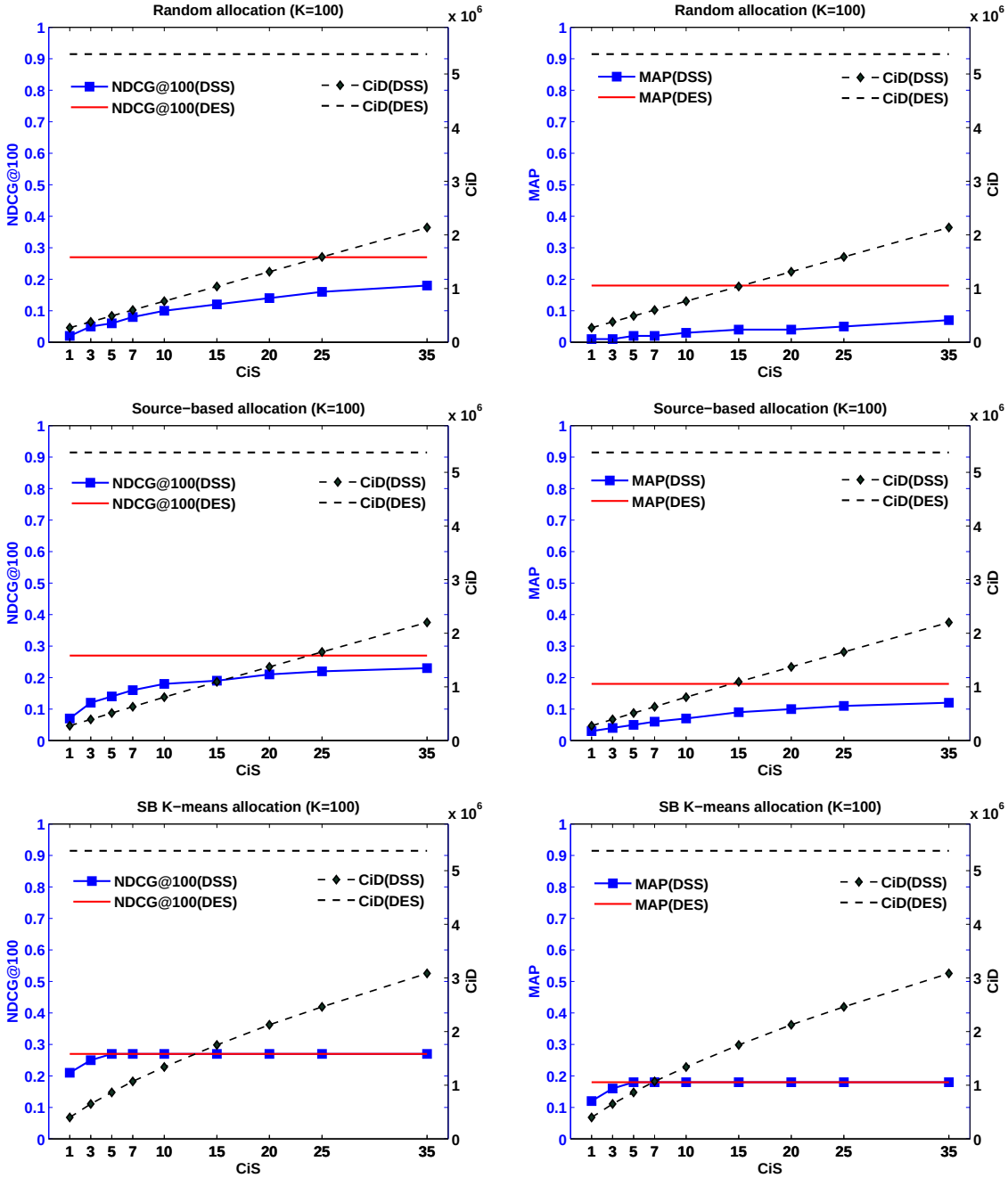
(a) (b)

Figure 4.6: Distributed Selective Search with Random, Source-based, and Topic-based shards. Dataset: GOV2. Metrics: NDCG@100 and MAP. Distributed Exhaustive Search: CiD=3.63M.



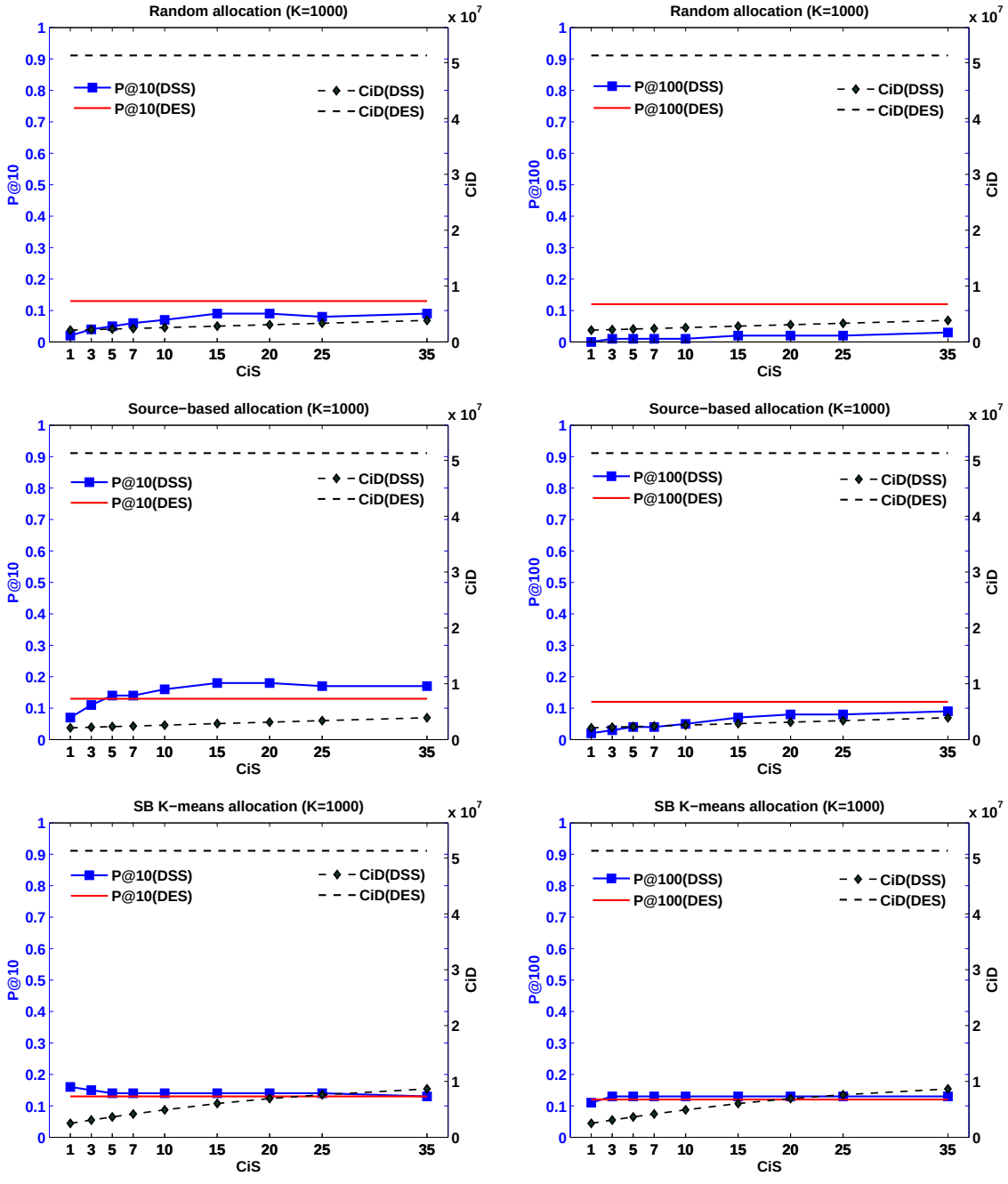
(a) (b)

Figure 4.7: Distributed Selective Search with Random, Source-based, and Topic-based shards. Dataset: CW09-B. Metrics: P@10 and P@100. Distributed Exhaustive Search: CiD=5.37M.



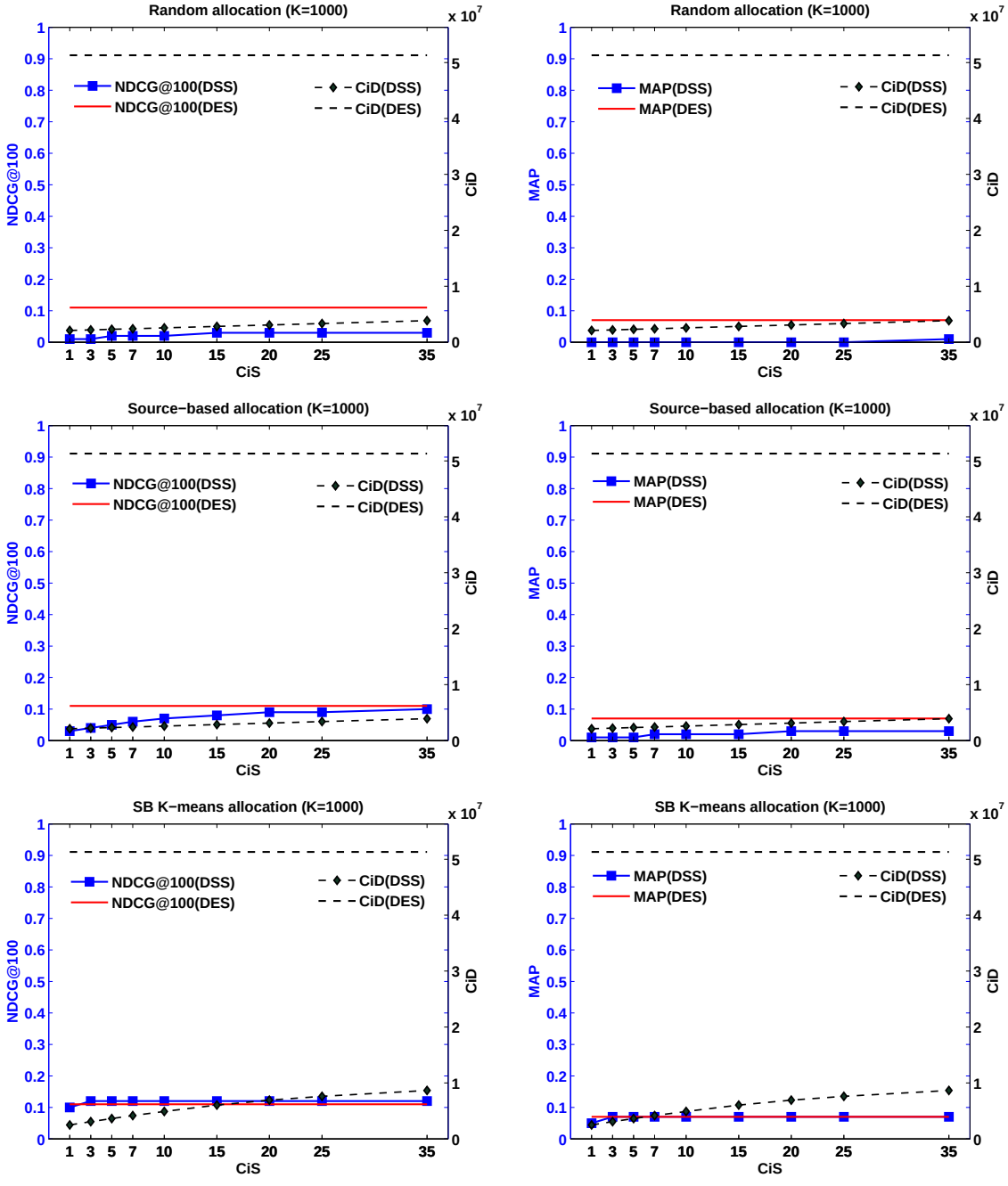
(a) (b)

Figure 4.8: Distributed Selective Search with Random, Source-based, and Topic-based shards. Dataset: CW09-B. Metrics: NDCG@100 and MAP. Distributed Exhaustive Search: CiD=5.37M.



(a) (b)

Figure 4.9: Distributed Selective Search with Random, Source-based, and Topic-based shards. Dataset: CW09-Eng. Metrics: P@10 and NDCG@10. Distributed Exhaustive Search: CiD=51.29M.



(a) (b)

Figure 4.10: Distributed Selective Search with Random, Source-based, and Topic-based shards. Dataset: CW09-Eng. Metrics: P@100 and MAP. Distributed Exhaustive Search: CiD=51.29M.

As before, the ReDDE algorithm was employed for shard ranking, and the Indri algorithm was used for document retrieval in all the experiments.

Figures 4.5 through 4.10 provide the selective search results for the three datasets. The columns in the plot grid are different effectiveness metrics and the three allocation policies are along the rows. Each plot in the grid reports three metrics. The left Y-axis is one of the search effectiveness metrics. The right Y-axis provides the primary cost, cost-in-documents (CiD). The secondary cost which is simply the number of shards searched, cost-in-shards (CiS), is along the X-axis. For ease of comparison the effectiveness with distributed exhaustive search (DES) is also reported along with the effectiveness for distributed selective search (DSS). The exhaustive search costs are reported in the figure captions.

Several trends emerge from these results. When comparing the three allocation policies (across rows of plots), we see that the source-based approach is able to converge to exhaustive search accuracy faster than random allocation, and topic-based policy is in turn able to converge faster than the source-based, for all the evaluation metrics and across all the datasets. Recall that smaller CiS value implies fewer active shards which translates to fewer disk seeks and network costs for a query, and smaller CiD indicates fewer disk transfers and fewer computational cycles per query. Also, recall that the CiD also includes the cost of shard ranking.

For the metrics that evaluate only until early ranks (P@10), the efficiency and effectiveness trade-off offered by source-based is comparable to that provided by the topic-based shards, especially for the larger collections. The random partitioning also performs surprisingly well at early ranks for the larger collections, especially CW09-B. This might be because a large portion of the CW09-B dataset consists of Wikipedia articles many of which are relevant documents. For the metrics that evaluate precision at deeper ranks (P@100, NDCG@100 and MAP) only the topic-based policy can continue to provide competitive performance irrespective of the collection size. These results indicate the rate at which selective search converges to exhaustive search performance is also dependent on the metric being optimized.

Even for NDCG@100, a metric that is sensitive to both the position and the relevance level of the documents in the results list, selective search with topical shards performs on par with exhaustive search for all the datasets. This is especially remarkable for the ClueWeb09 datasets which used evaluation query-sets with 5 grades of relevance judgments.

When analyzing the partitioning techniques individually, for the random policy we see that searching more shards improves the search effectiveness rapidly early on but the rate of improvement tapers off for metrics such as P@100, NDCG@100 and MAP. This is not surprising since these metrics model the Recall component. As such, we would expect them to exhibit a linear, positive correlation with number of shards searched. As expected, the search cost shows a linear increase with the number of shards searched. These results demonstrate that the random document allocation policy is not the ideal choice for selective search, especially for applications where reduction in search effectiveness is not acceptable. It is thus not surprising

that the commercial Web search engines (for example, Google and Bing) which are know to use random partitioning search all the shards.

Sharding a collection using source-based allocation policy offers much improvement over the random allocation policy for all the datasets. This policy is especially well suited for applications such as Web retrieval where the collection size is large and the most important metrics might only be P@10 and NDCG@10. Selective search with source-based shards would offer an economical alternative to exhaustive search for such applications.

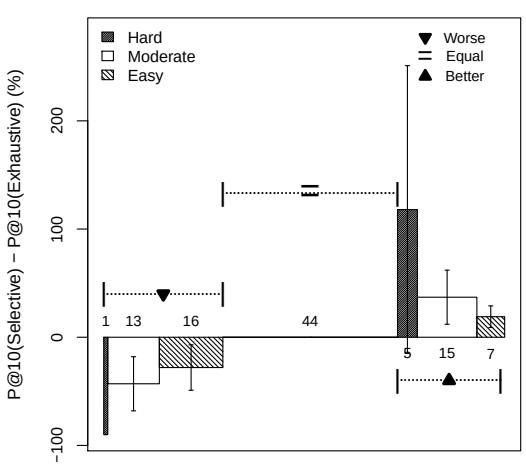
The ability to lower both the search costs even when optimizing for *comprehensive* metrics such as P@100 and MAP, is the distinctive strength of the topic-based allocation policy. For the GOV2 dataset the selective search with topical shards becomes comparable to exhaustive search on all the metrics analyzed here when the top 20 shards are searched (CiS), and the corresponding CiD is 0.8M documents. The CiD is 23% of that of exhaustive. Similarly, for CW09-B, the convergence occurs after the top 5 shards have been searched. The corresponding CiD is 16% of that of exhaustive. For CW09-Eng searching the top three shards is enough when working with topical shards. This equivalent to 6% of CiD for exhaustive.

Section 3.2 outlined a set of objectives that the proposed search approach needs to satisfy. The above results demonstrate that *Competitive search effectiveness* objective is met successfully for all the effectiveness metrics, and all the three datasets. The *Low search effort* objective is also clearly satisfied. The search effort expended by topic-based selective search is 77%, 84%, and 94% lower than that with exhaustive search. This trend indicates that selective search's ability to reduce the search cost improves with collection size. This is an useful property that makes selective search an especially appealing solution for large-scale search. Also, these results satisfies the *Scalability* objective. The *Low resource requirements* requirement is also easily met because only a small fraction of the total shards are searched for each query. For Gov2 8% of the shards are searched, while for CW09-B and CW09-Eng only 5% and 0.3% of the shards need to be searched for each query.

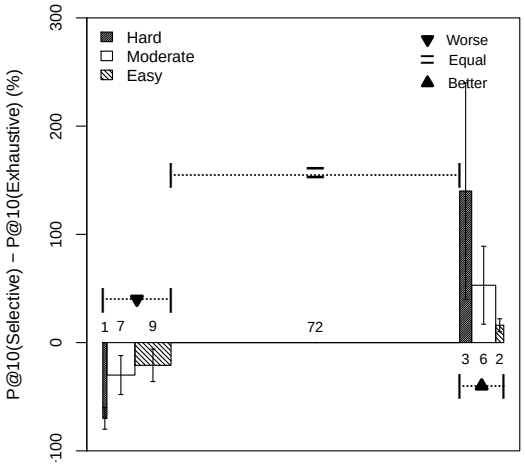
Overall, these experimental results demonstrate that the three document allocation policies have different potentials in terms of their ability to support selective search. Each of the document allocation policies, more or less, converges to the exhaustive search performance, however, at different paces. Topic-based shards provide the most consistent and cost-effective solution as compared to the source-based and random shards.

4.1.5 Experimental results: Stability analysis

The results in the previous section establish that selective search with topic-based and source-based shards can provide *average-case* precision that is comparable to that of exhaustive search and also offer substantial savings in search costs. Ideally we would expect these average-case trends to hold for each individual query. However we verify this empirically using a query-level stability analysis focused on the two allocation policies, source-based and topic-based.

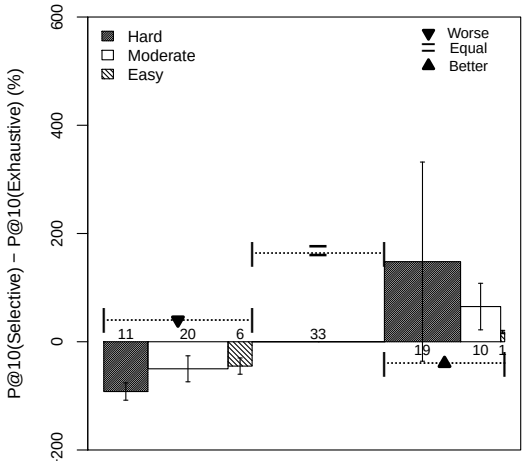


(a) Stability analysis of source-based shards. CiD=0.60M, CiS=20.

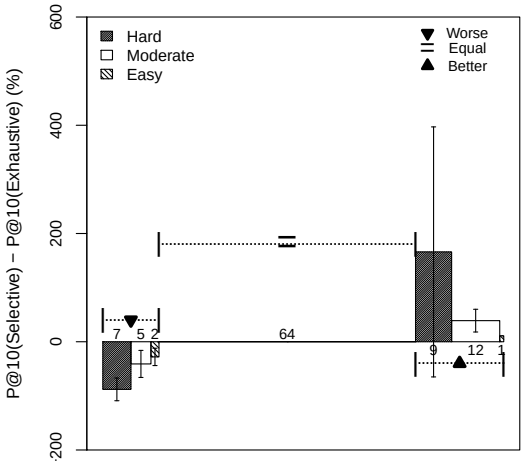


(b) Stability analysis of topic-based shards. CiD=0.56M, CiS=10.

Figure 4.11: Stability analysis. Metric: P@10. Dataset: GOV2. (X-axis: % of queries, binned by improvement and difficulty levels.)

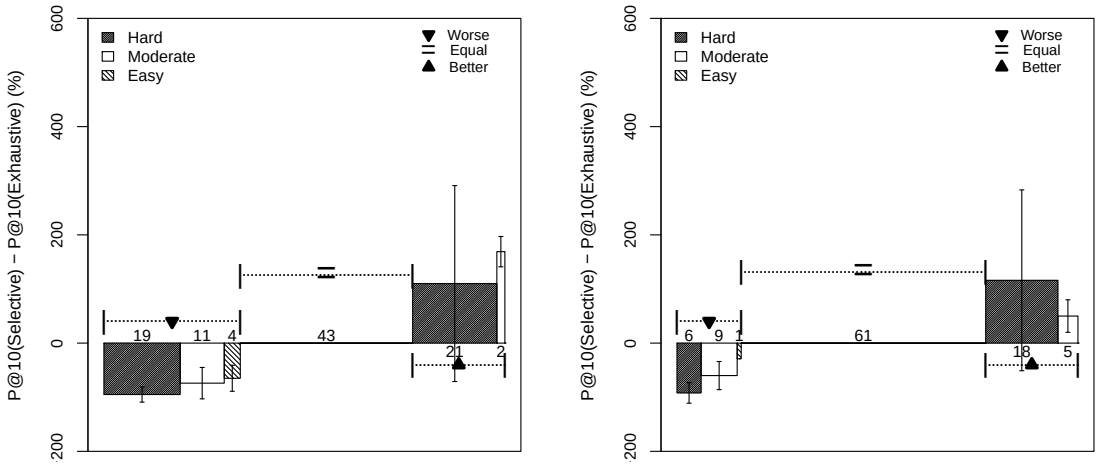


(a) Stability analysis of source-based shards. CiD=0.51M, CiS=5.



(b) Stability analysis of topic-based shards. CiD=0.65M, CiS=3.

Figure 4.12: Stability analysis. Metric: P@10. Dataset: CW09-B. (X-axis: % of queries, binned by improvement and difficulty levels.)



(a) Stability analysis of source-based shards. CiD=2.32M, CiS=5.

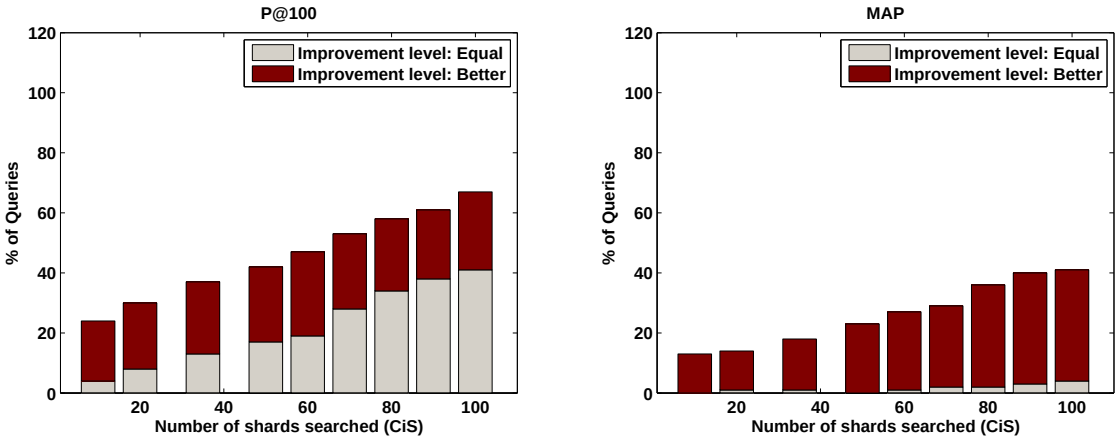
(b) Stability analysis of topic-based shards. CiD=2.49M, CiS=1.

Figure 4.13: Stability analysis. Metric: P@10. Dataset: CW09-Eng. (X-axis: % of queries, binned by improvement and difficulty levels.)

Recall that the stability analysis categorizes the evaluation queries along two dimensions (Section 3.4.2). The categorization based on the *improvement levels* (worse, equal, and better) group together the queries based on their selective search effectiveness relative to exhaustive search. Each of these groups are further sub-divided based on the *difficulty levels* (hard, moderate, and easy) which is measured by query's exhaustive search effectiveness.

Use of Oracle for CiS

One of the parameters that need to be set in order to operationalize selective search is the number of shards to search for a query (CiS). For many of the experiments in this dissertation we have used an *oracle* to set CiS. The oracle for CiS chooses the smallest value for which the aggregate selective search effectiveness is comparable to the aggregate exhaustive search effectiveness. For instance, if the search effectiveness is measured using the MAP metric, then the CiS value for which the aggregate MAP with selective search is comparable to the aggregate MAP with exhaustive search is the value set by the oracle. A difference of at most 5% in the two search effectiveness values is tolerated. Since this methodology indirectly uses relevance judgments to set the CiS parameter we refer to it as an oracle estimator (or CiS oracle). The CiS parameter influences the effectiveness as well as the efficiency of selective search. A large CiS value might improve search effectiveness since more number of shards are searched, and would degrade efficiency due to the same reason. Using the oracle to set this parameter provides the best possible balance between search effectiveness and efficiency. In practice, finding this point might be difficult since the CiS oracle needs relevance judgments for the queries, which



(a) (b)

Figure 4.14: Stability analysis of source-based shards. Metrics: P@100 and MAP. Dataset: GOV2.

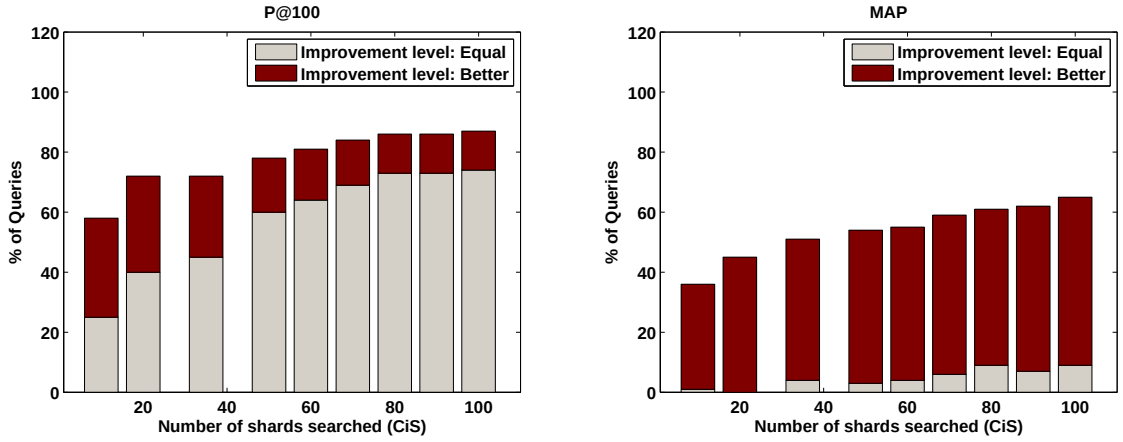
are rarely available for the query sets of interest. As such, the selective search performance obtained with CiS set by an oracle provides an upper bound on the search performance.

In the first part of this section the search effectiveness is measured using P@10 metric. Using this strategy, the top 20, 5, and 5 source-based shards were searched for each query for the GOV2, CW09-B, and CW09-Eng datasets, respectively. With topic-based shards the top 10, 3, and 1 shards were searched for each query for the GOV2, CW09-B, and CW09-Eng datasets, respectively. The corresponding CiD costs are specified in the figure captions.

The stability results for selective search with source-based and topic-based shards are provided in Figures 4.11 through 4.13. When comparing the two partitioning approaches in terms of the fraction of queries that degrade with selective search, we observe consistent trends across the three datasets. With source-based shards selective search degrades 30-37% of the queries. While with topical shards a smaller fraction of queries, 14-17% degrade with selective search. Recall that only the top 10 out 250, 3 out of 100, and 1 out of 1000 topic-based shards were searched in these experiments for GOV2, CW09-B, and CW09-Eng, respectively. When analyzing the stability over a range of shard cutoffs (results or figures not included), we see a monotonic improvement in the stability as more shards are searched because selective search becomes progressively more similar to exhaustive search.

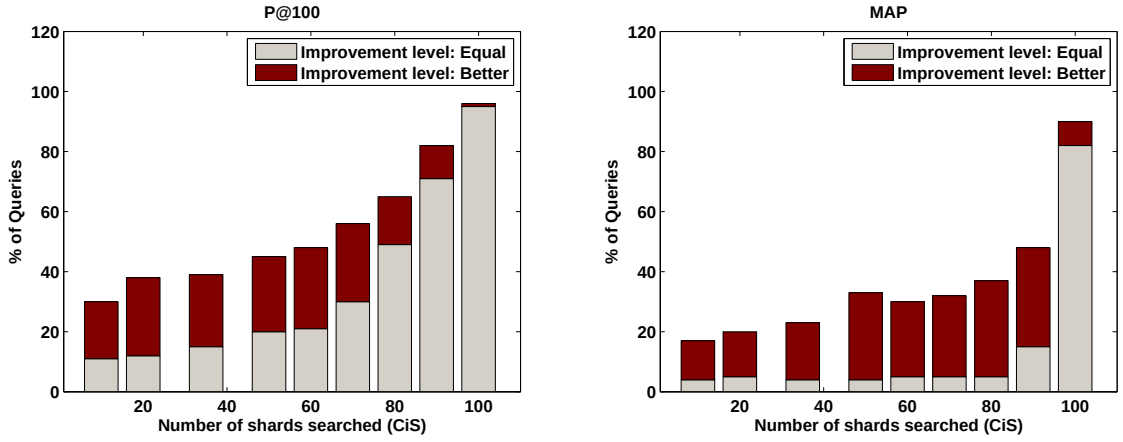
The corresponding values for the primary search cost metric (CiD) are more or less comparable for selective search with both the sharding techniques. The secondary cost, CiS, however, is consistently lower for selective search with topic-based shards.

For all the three datasets we see that a large fraction of the queries (> 75%) fall in the equal and better improvement levels for the topic-based selective search results. Queries from all the three difficulty levels improve as well as degrade with topic-based selective search. We expected



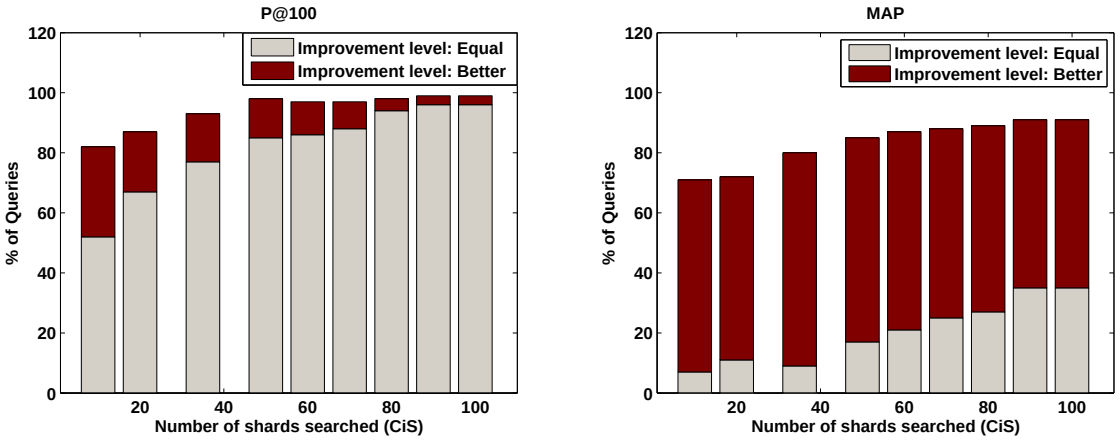
(a) (b)

Figure 4.15: Stability analysis of topic-based shards. Metrics: P@100 and MAP. Dataset: GOV2.



(a) (b)

Figure 4.16: Stability analysis of source-based shards. Metrics: P@100 and MAP. Dataset: CW09-B.



(a) (b)

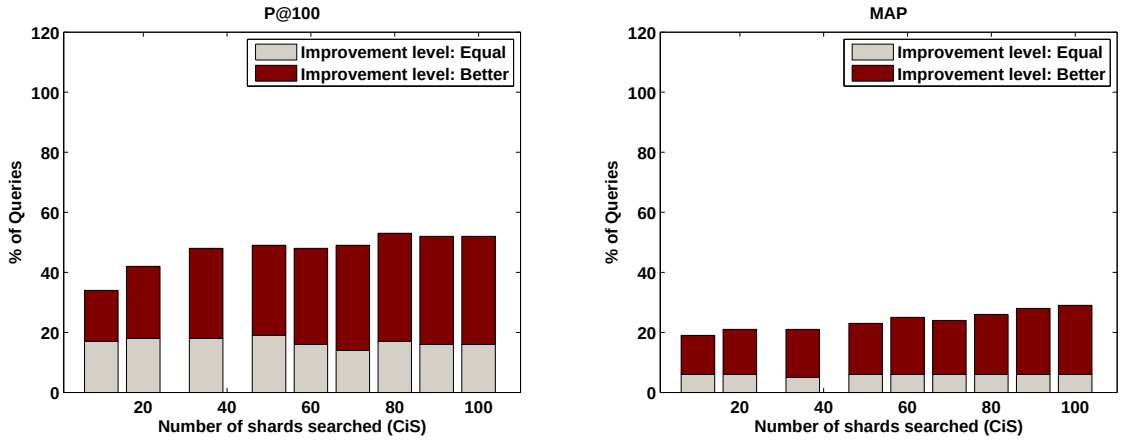
Figure 4.17: Stability analysis of topic-based shards. Metrics: P@100 and MAP. Dataset: CW09-B.

the hard queries (P@10(Exh) < 0.2) to be difficult for selective search. We see that although some hard queries degrade with selective search a non-negligible fraction also performs better with selective search, especially for the larger datasets.

One of the conclusions of the previous section was that the rate at which selective search converges to exhaustive search accuracy is dependent on the metric being optimized. The metrics that evaluate only at early ranks (P@10) demonstrated markedly different trends than metrics, such as, P@100 and MAP. For this reason, we present a simplified version of stability analysis for P@100 and MAP next.

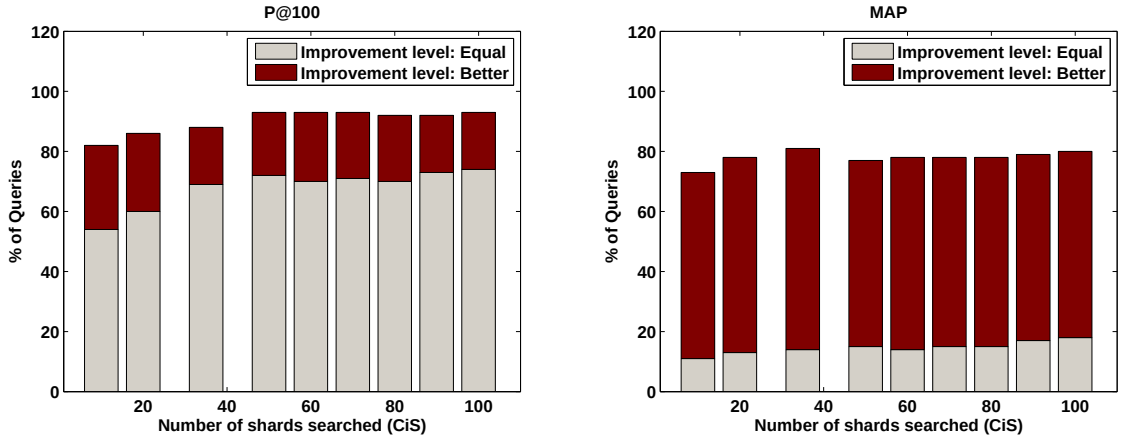
Figures 4.14 through 4.19 present a stability analysis where only the two improvement levels of *equal* and *better* are specified. We see that the topic-based shards support substantially more stable search than source-based shards for all the datasets and for both the metrics analyzed, P@100 and MAP. The differences in the respective stabilities of topic-based and source-based are larger for P@100 and MAP than P@10. This complies with the trends observed for the average-case results in the previous section where the converge speed for P@100 and MAP was much slower than that for P@10.

Because of its formulation the MAP metric is more sensitive to changes in the ranking of relevant documents than the P@n metrics. As a result, reproducing the exact values as that of exhaustive search is harder for MAP. This is one of the reasons why fewer queries categorize into the *equal* improvement level when analyzing the stability for MAP.



(a) (b)

Figure 4.18: Stability analysis of source-based shards. Metrics: P@100 and MAP. Dataset: CW09-Eng.



(a) (b)

Figure 4.19: Stability analysis of topic-based shards. Metrics: P@100 and MAP. Dataset: CW09-Eng.

4.1.6 Experimental results: Relevance density distribution

The previous two sections demonstrated empirically that shards created using topic-based allocation can support selective search that is consistently more effective and efficient than source-based and random allocation. This section studies the cause of the differences observed in the performance of the three shard creation techniques.

We believe that the distribution of relevant documents across shards needs to be skewed in order for selective search to provide competitive effectiveness. Based on the Cluster Hypothesis we expect the shards created with topic-based allocation to best satisfy this requirement, and the shards created with random to conform the least. This hypothesis can be validated by analyzing the relevance distribution of shards created using the three techniques. Instead, however, we normalize the relevance distribution of shards with their sizes because of the following reasons.

As is shown in Figure 4.20, the distribution of shard sizes is skewed for topical shards for all the datasets. Whereas, for source-based it is less skewed, and for random it is uniform. Normalizing the distribution of relevant documents with the distribution of shard sizes eliminates any biases toward larger shards. Furthermore, a larger shard is likely to result in higher search cost (CiD) than a smaller shard. Thus given two shards with same number of relevant documents for a query, the smaller is preferable to the larger for efficient search.

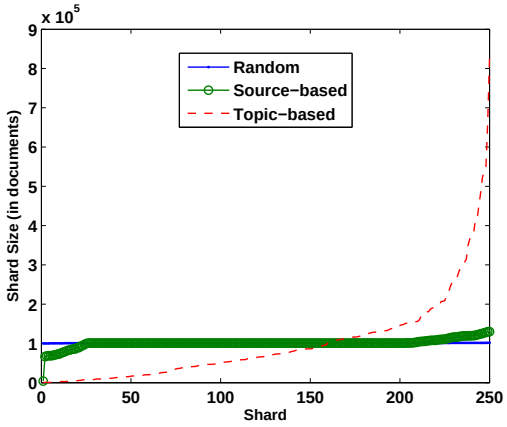
In summary, modeling both shard relevance and size provides a more accurate and complete picture of the ability of an allocation policy to support effective and efficient selective search. We thus analyze the size normalized relevance distributions, referred to as the *relevance density distribution* (ρ) in this section. The relevance density of each query q and shard R pair, ρ_R^q, is defined as:

$$\rho_R^q = \frac{\mu_R^q}{|R|} \tag{4.6}$$

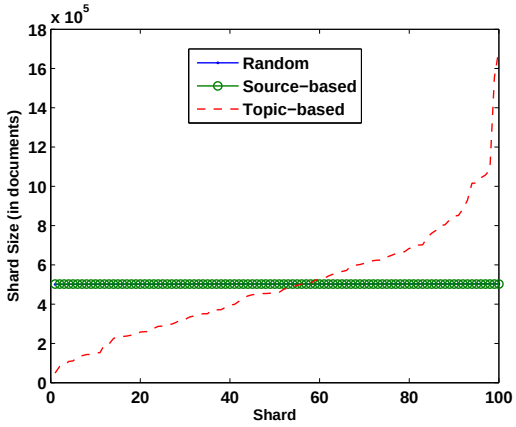
where μ_R^q is the set of documents in R that were judged relevant for query q, and $|R|$ is the total number of documents in shard R. We use the formulation in Equation 4.6 to compute the relevance density distribution for a given set of shards. First, the relevance density for each query-shard pair (ρ_R^q) is computed and the shards are ranked based on their density value (in descending order) for the query. These ranked relevance density values are then averaged across all the evaluation queries to generate the relevance density distribution.

Figure 4.21 provides a grid of plots where the different allocation policies are along the columns and the three datasets are along the rows. The figures report the average relevance density for the top 30 shards. Note that for each query this could be a different set of 30 shards.

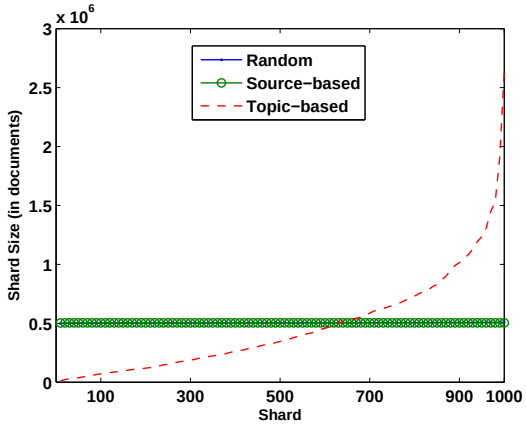
We see very similar trends across the three datasets. Although the total number of shards (K) for each dataset is different, the distributions exhibit nearly identical trends. The relevance density distribution for topical shards is much more skewed than that for the shards created with the other two partitioning techniques. The relevance density of the highest ranked topical



(a) Dataset: GOV2. Number of topical shards (K) = 250. Average shard size = 100,821. Complete dataset size = 25,205,179 documents.



(b) Dataset: CW09-B. Number of topical shards (K) = 100. Average shard size = 502,204. Complete dataset size = 50,220,423 documents.



(c) Dataset: CW09-Eng. Number of topical shards (K) = 1000. Average shard size = 503,904. Complete dataset size = 503,903,810 documents.

Figure 4.20: Size distribution of random, source-based, and topic-based shards.

shard is more than twice that of the topical shard at the next rank. If the top ranked topical shard is set aside then the remaining distribution for the topical shards is very similar to that of source-based for all the datasets. This explains the high search precision results at early ranks but low precision results for deeper ranks that source-based shards provide in Section 4.1.4.

The formulation for relevance density of an individual shard in Equation 4.6 can be extended to the complete collection where the numerator specifies the number of relevant documents for the query in the collection and the denominator is simply the collection size. These relevance density values averaged across queries for the GOV2, CW09-B, and CW09-Eng are $0.7 \cdot 10^{-}3(\pm 0.6 \cdot 10^{-}3)$, $0.02 \cdot 10^{-}4(\pm 0.01 \cdot 10^{-}4)$, and $0.002 \cdot 10^{-}4(\pm 0.001 \cdot 10^{-}4)$, respectively. As compared to these values the relevance density of the top ranked topical shard is 1.43, 45, and 750, times larger for GOV2, CW09-B and CW09-Eng, respectively. Given that each of these datasets is larger than the previous one these numbers are not entirely surprising. However, they do explain the faster convergence to exhaustive search performance for the larger datasets that was observed in Section 4.1.4.

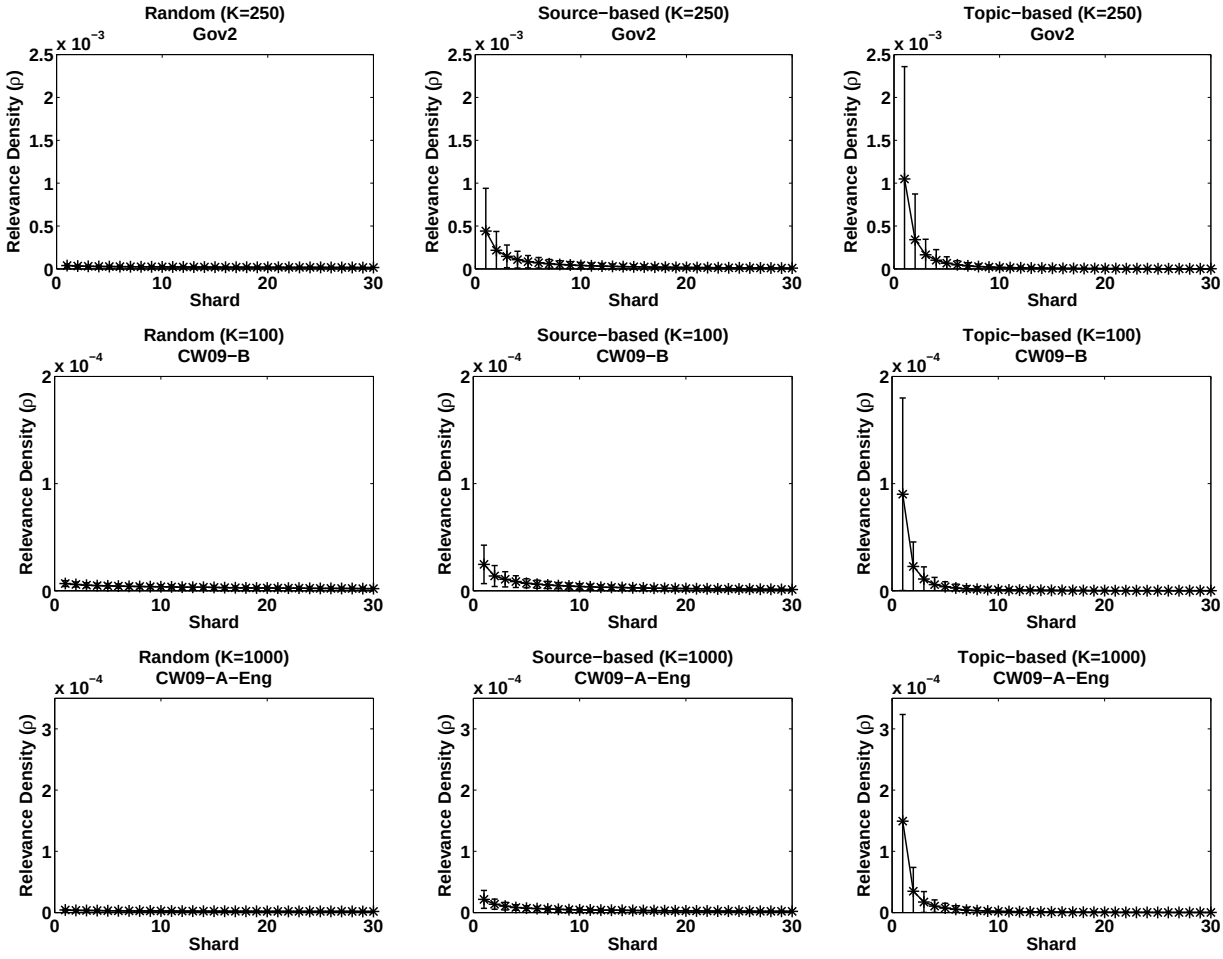
Overall, the relevance density analysis supports and strengthens the search effectiveness, efficiency and stability results provided in the earlier sections. Note that the results reported in this section also provide an empirical validation of the Cluster Hypothesis. Henceforth we study distributed selective search only with topic-based shards due to their higher performance.

4.1.7 Experimental results: Effect of query length on search performance

The length of the query could potentially influence the performance of selective search. For instance, a longer query could imply a more diverse information need than a shorter query. As a result, the search effectiveness for the longer query might be lower than the shorter query since the relevant documents for the former might be spread out across more shards. On the other hand, it could also be argued that a longer query might be less ambiguous (more focused) than an under-specified shorter query which would lead to lower search effectiveness for the latter. An investigation into the relation between query length and selective search performance that tests the above conjectures is the focus of this section.

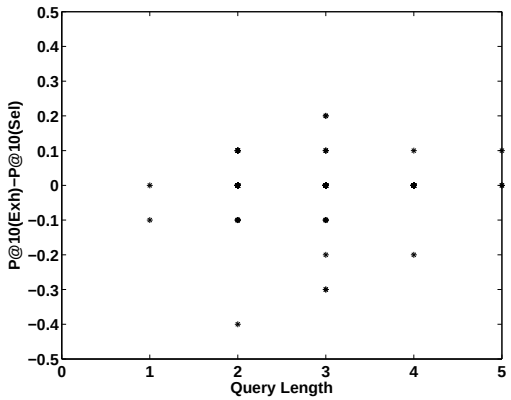
Figure 4.22 provides the necessary data for this analysis. The scatter plots for the three datasets, GOV2, CW09-B, and CW09-Eng, compare the query length on the X-axis with the difference in the retrieval effectiveness of exhaustive search and selective search on the Y-axis. The number of shards searched for each dataset in these experiments were provided by an oracle that optimized for overall search effectiveness. Specifically, the number of shards that provided comparable retrieval effectiveness to that of exhaustive search was chosen. This is similar to the methodology used earlier for the stability analysis, and described in more detail in Section 4.1.5. For a thorough analysis we study the relation between query length and selective search effectiveness using two metrics, P@10 and MAP.

None of the plots illustrate any systematic correlation between the two variables. The Pear-

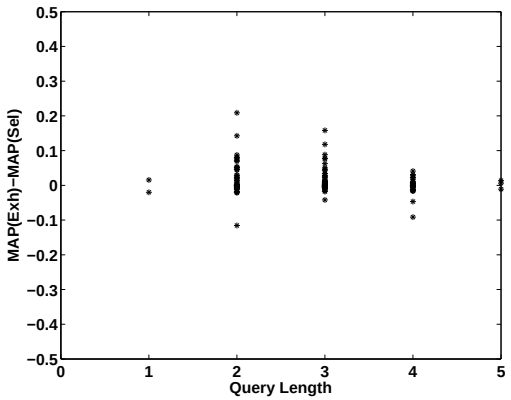


(a) (b) (c)

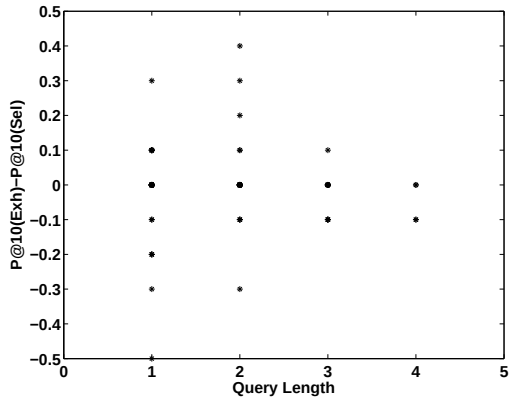
Figure 4.21: Relevance density distributions.



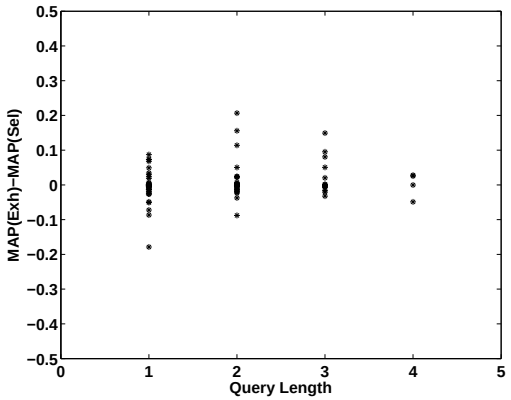
(a) Dataset: GOV2, Metric: P@10, Number of shards searched: 20, Pearson correlation coefficient: 0.04



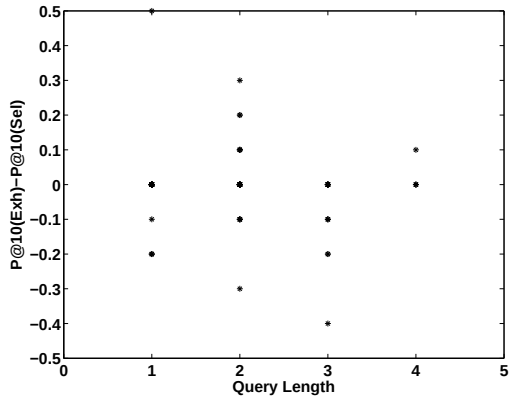
(b) Dataset: GOV2, Metric: MAP, Number of shards searched: 20, Pearson correlation coefficient: -0.20



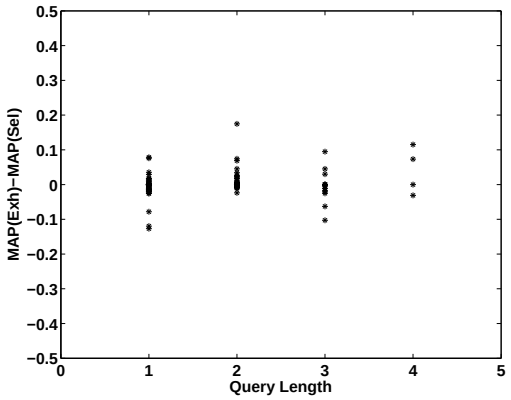
(c) Dataset: CW09-B, Metric: P@10, Number of shards searched: 5, Pearson correlation coefficient: -0.02



(d) Dataset: CW09-B, Metric: MAP, Number of shards searched: 5, Pearson correlation coefficient: 0.16



(e) Dataset: CW09-Eng, Metric: P@10, Number of shards searched: 3, Pearson correlation coefficient: -0.10



(f) Dataset: CW09-Eng, Metric: MAP, Number of shards searched: 3, Pearson correlation coefficient: 0.16

Figure 4.22: Effect of query length on selective search effectiveness.

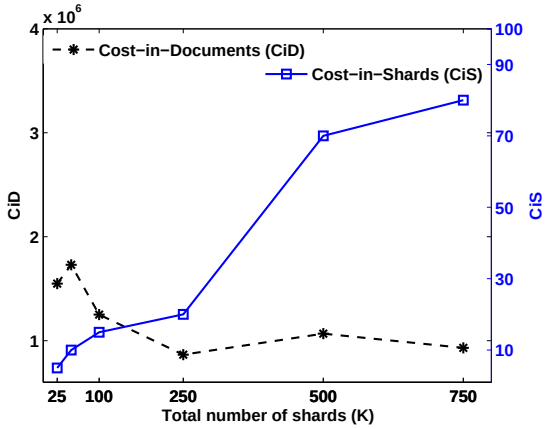
son correlation coefficient reported in the plot captions confirms absence of strong correlation between query length and selective search effectiveness. The MAP for one of the 4 term queries for the CW09-Eng dataset, *french lick resort and casino*, degrades by more than 50% with selective search, however, for another 4 term query, *uss yorktown charleston sc*, selective search improves MAP by more than 100%. Similar trends are observed for the shorter queries as well. The correlation coefficient for the GOV2 dataset and the MAP metric can be interpreted as a borderline weak relation. The negative polarity of this relation suggests that it is harder for selective search to provide competitive MAP scores for shorter queries than for longer queries for this dataset.

Overall however this data does not suggest that selective search's effectiveness is strongly influenced by the length of the query. It is also important to notice that the variation in query length is limited for the query sets employed here. A more extensive evaluation that experiments with a larger and more varied query set could provide additional insights. This might be especially useful if the search task of interest exhibits a wider range of query lengths than that analyzed in this section.

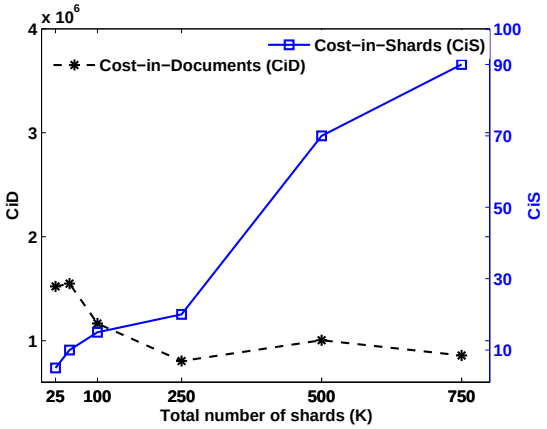
4.2 Number of topical shards for a collection (K)

The number of partitions (K) that a collection is divided into for a distributed search is typically determined by certain system specifications, such as, the number of available computing nodes, or the desired query response time. This parameterization strategy works well for the distributed exhaustive search. Dividing the collection into as many shards as the number of compute nodes allows for the maximal usage of the available resources while exploiting the inherent parallelism in the exhaustive search process. For selective search however we hypothesize that the parameterization of K needs to be collection-specific. The rationale behind this hypothesis is that if the collection is divided into as many partitions as the number of distinct and dominant topics in it then selective search can minimize both the search costs while maintaining competitive effectiveness. In this section we test the above hypothesis using different parameterizations of K.

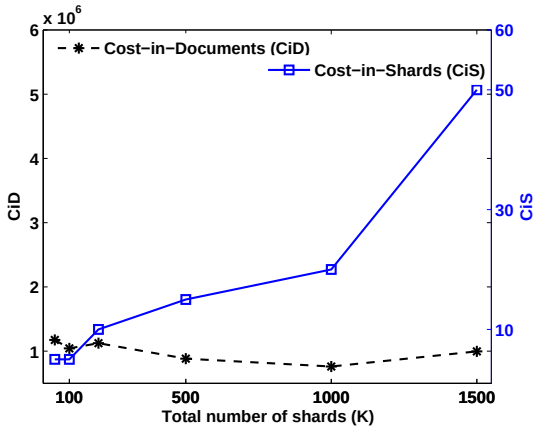
We note that the problem of estimating a collection-specific value for K is an instance of the *model complexity selection* problem which has been studied by many [62, 75]. Others have studied the problem of estimating K specifically in the context of K-means clustering. For instance, the X-means algorithm proposed by Pelleg and Moore [57] is a wrapper around the basic K-means algorithm where the initial clusters K_0 are iteratively split into 2. The decision about whether to split a cluster is based on the Bayesian Information Criterion (BIC) proposed by Kass and Wasserman (1995), which takes into account the model complexity of the models being compared, and the data likelihood with respect to each of them. The iterative procedure stops if no centroids are split in an iteration, or the number of clusters exceeds a user-specified maximum threshold.



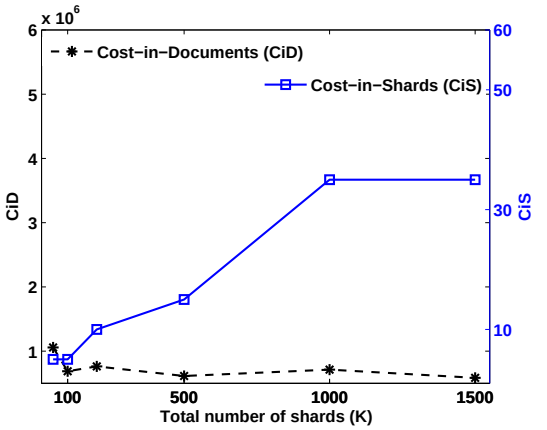
(a) Dataset: GOV2.
Tuning set 1: P@10=0.59, P@30=0.53, P@100=0.43, NDCG@100=0.45, MAP=0.30.



(b) Dataset: GOV2.
Tuning set 2: P@10=0.57, P@30=0.51, P@100=0.40, NDCG@100=0.48, MAP=0.33.



(c) Dataset: CW09-B. Tuning set 1. P@10=0.31, P@30=0.30, P@100=0.22, NDCG@100=0.29, MAP=0.18



(d) Dataset: CW09-B. Tuning set 2. P@10=0.25, P@30=0.25, P@100=0.19, NDCG@100=0.25, MAP=0.17

Figure 4.23: Effects of number of total shards (K) on the costs of distributed selective search. Query set: Tuning.

Another method by Hamerly and Elkan, named *G*-means [34], is also a wrapper around *K*-means which also starts with some initial number of clusters and iteratively splits each centroid into 2 child clusters subject to an evaluation criterion. The evaluation criterion used is a statistical test that decides whether the data that belongs to a given centroid is Gaussian or not. Empirically they show that *G*-means is better at estimating the number of clusters than *X*-means, which often overfits the data in their experiments leading to an overestimation of *K*.

Unfortunately, the computational cost of applying these methods to large-scale datasets is almost always prohibitively high. Developing efficient, scalable and effective model complexity selection algorithms is an open research problem. In this work we only study the parameter *K* in the context of selective search, specifically, we are interested in analyzing the sensitivity of selective search to this parameter. The estimation of this parameter is left for future work.

Experimental results

For this analysis we experiment with a range of values for *K*. For each of the *K* values the collection was repartitioned into those number of shards and selective search was performed. These experiments were performed with the GOV2 and CW09-B datasets and the values for *K* were chosen to maintain a comparable range of average shard-sizes across the two datasets. Due to its size we do not perform this analysis for the CW09-Eng dataset. Since CW09-B and CW09-Eng datasets are related to each other (Section 3.3) we extrapolate the conclusions from the CW09-B results to CW09-Eng. We acknowledge that this is not ideal and cannot replace empirical validation for the CW09-Eng dataset.

A two-fold cross-validation setup was employed for this analysis where 50% of the evaluation queries were used as a tuning set and the other 50% served as the test query set during each fold. The results for GOV2 and CW09-B on the tuning sets from each of the two folds are reported in Figure 4.23. The different parameterizations of *K* are specified along the X-axis. The number of top shards searches (CiS) for each configuration is specified on the right Y-axis. The CiS values for each experimental configuration was provided by an oracle which selects a value that provides search effectiveness that is comparable to that with exhaustive search. The oracle methodology is described in detail in Section 4.1.5. The left Y-axis specifies the search cost in terms of the average number of documents processed for each query (CiD). Because of the use of an oracle, the search effectiveness (specified in the figure caption) for each of the configurations is comparable to the corresponding exhaustive search performance.

A value for *K* that minimizes both the search costs, CiD and CiS, is desirable. However, as the plots show the two cost metrics are inversely proportional to each other. A smaller value of *K* creates larger shards which lead to more documents to be evaluated (higher CiD) at each of the shards. As a result of the more documents being evaluated at each shard, fewer shards need to be searched in order to perform on par with exhaustive search, thus reducing CiS. This trend is reversed when the collection is divided into larger number of shards.

Table 4.1: Effect of number of total shards (*K*) on selective search performance. Query set: Test.

Dataset	Search Type	Fold	P@10	P@30	P@100	NDCG @100	MAP	CiD (million)	CiS
GOV2	Exhaustive	Fold 1	0.57	0.50	0.40	0.48	0.34	3.54	20 (/20)
	Selective	Fold 1	0.57	0.51	0.40	0.48	0.33	0.80	20 (/250)
	Exhaustive	Fold 2	0.60	0.54	0.44	0.46	0.31	3.71	20 (/20)
	Selective	Fold 2	0.59	0.53	0.43	0.45	0.30	0.86	20 (/250)
CW09-B	Exhaustive	Fold 1	0.23	0.23	0.19	0.24	0.17	4.48	5 (/5)
	Selective	Fold 1	0.25	0.25	0.19	0.25	0.17	0.69	5 (/100)
	Exhaustive	Fold 2	0.30	0.29	0.23	0.30	0.19	6.27	5 (/5)
	Selective	Fold 2	0.31	0.30	0.22	0.29	0.18	1.04	5 (/100)

We see similar trends across the two tuning sets for both the datasets. Of the different values of *K* evaluated for GOV2, dividing the collection into 250 topical shards, and searching the top 20 shards for each query, offers a good balance between the two cost metrics on both the tuning sets. We evaluate this configuration on the test query set. The results reported in Table 4.1 show that selective search is as effective as exhaustive search on the test set.

For CW09-B dataset the value of 100 for the parameter *K* is recommended by both the tuning sets since it minimizes both the search costs (CiD and CiS) simultaneously. The performance of selective search on the test query set using this parametrization is reported in Table 4.1. The *K* values recommended for the two datasets by this analysis are neither the same nor proportional to the dataset sizes. The smaller dataset (GOV2) is divided into more shards than the larger dataset (CW09-B). This observation supports the above hypothesis that selective search benefits from a collection-specific parameterization of *K*. Other criterion for setting the value of *K*, such as the collection size, cannot provide the best selective search setup in terms of balancing search efficiency and effectiveness.

We also observe that partitioning the GOV2 dataset into 750 shards instead of 250 offers comparable CiD. Similarly for CW09-B, the CiD for *K* of 1000 and 1500 is lower or comparable to that of *K*=100. However, the corresponding CiS costs are higher for larger *K* values. For example, the top 25 shards need to be searched when CW09-B is divided into 1000 shards, as opposed to the top 5 shards when partitioned into 100 shards. The shard creation cost is also lower when the dataset is partitioned into fewer number of shards. For these reasons we choose *K*=250, and *K*=100 for CW09-B for remaining experiments.

However, it is also important to note the following merit of using larger *K* values. When a dataset is partitioned into larger number of shards there are more opportunities for parallelization during query processing. For example, in case of the CW09-B dataset, one could chose to

distribute the processing of 0.8M documents per query across 50 nodes (K=1500) instead of 5 nodes (K=100), thus achieving the same effectiveness but faster query response time. If the additional computational resources necessary to exploit this parallelization are available then the query run time can be further improved, albeit at higher cost (CiS). These results demonstrate that although a collection-specific K is recommended for selective search, the operational needs of a search system can also be accommodated.

Following the recommendations for K made here we partition GOV2 into 250 and CW09-B into 100 topical shards for the remaining experiments reported in this chapter. We extrapolate from the CW09-B results and choose K=1000 for the CW09-Eng dataset since the CW09-Eng dataset is an order of magnitude larger in size than CW09-B.

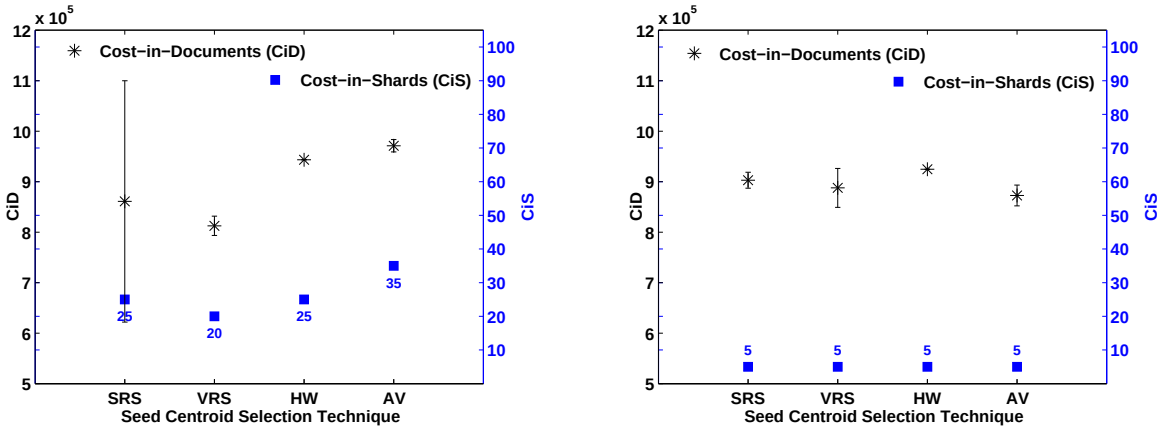
4.3 Seed centroid selection for topic-based allocation policy

One of the document allocation policies we propose for topic-based partitioning of the collection (Section 4.1.3) employs K-means clustering algorithm. The K-means algorithm is initiated using K seed centroids which are often simply the documents from the collection. A long line of research in clustering algorithms [3, 35, 53, 75] has demonstrated the strong influence of the seed centroid selection technique on the final clustering solution (shards). We also know from the earlier sections that selective search performance is inherently dependent on the collection shards. In this section we analyze the influence of this parameter on selective search performance. We experiment with four seed centroid selection techniques which are described next.

Recall that because of efficiency considerations the K-means clustering is applied only to a subset of the complete collection (Section 4.1.3). The seed centroids are consequentially also selected from the sample and not the collection. In the following sections this sample is denoted as S.

4.3.1 Simple Random Sampling (SRS):

The first technique that we experiment with is one of the most commonly employed centroid selection strategy. Here the seeds are sampled uniformly at random (without replacement) from all the data points. SRS is also one of the simplest and most efficient sampling technique with computational complexity of $O(K)$. However, we argue that SRS is not well-suited for this work due to the following two reasons. First, since SRS is designed to select a representative sample of the underlying distribution, it would sample multiple seeds from high-density regions. However, such seeds are quite likely to be similar to each and thus result in multiple similar shards which leads to inefficient selective search. Such errors are hard to catch and correct for in a non-hierarchical clustering algorithm such as K-means.



(a) Dataset: GOV2. P@10=0.58, P@30=0.52, P@100=0.42, NDCG@100=0.47, MAP=0.32

(b) Dataset: CW09-B. P@10=0.27, P@30=0.26, P@100=0.21, NDCG@100=0.27, MAP=0.18

Figure 4.24: Effect of seed centroid selection strategy on selective search performance.

Secondly, SRS is agnostic to the *quality* of the chosen seed centroids. For example, in the case of a collection of Web documents selecting a spam page as a seed centroid is unlikely to anchor a stable cluster. These observations direct us to also experiment with three other seeding techniques that correct for some or all of these problems of SRS. These alternative techniques are described next.

4.3.2 Vocabulary size based Rejection Sampling (VRS):

We experiment with a variant of SRS that addresses one of the problems noted above while retaining the simplicity and efficiency of SRS. The objective of the vocabulary size based rejection sampling (VRS) is to choose *high quality* documents as the seed centroids. This method uses the document vocabulary-size as a measure of document quality. A *rejection sampling* based methodology is applied where a candidate document is sampled uniformly at random from the complete collection which is then accepted if its vocabulary-size is above certain threshold τ else it is rejected. We set τ to be the average document vocabulary size of the set from which the centroid would be chosen. This approach is nearly as efficient as the random sampling technique with the computational complexity of about $O(K)$.

4.3.3 Hartigan and Wong (HW):

Hartigan and Wong [35] proposed a seed centroid selection technique with the goal of reducing the over and under-sampling errors of SRS. The HW technique selects a *diverse* set of seed centroids to achieve this goal. First, a *global centroid* is computed from all the data-points. Next, the data-points are ordered based on their similarity with the global centroid. In order to sample

K seed centroids every $[1 + (i-1) \cdot \lfloor S/K \rfloor]$th data-point from the ordered list is chosen where S is the subset of the collection on which the sample-based K-means will be applied and $i = 1..K$. The computational complexity of this technique is $O(|S||V|)$, where V is the vocabulary of the sample. The computational complexity for HW is higher than the previous two techniques. Note that unlike all the other seeding techniques studied here, HW is deterministic.

4.3.4 Arthur and Vissilvitskii technique (AV):

More recently, Arthur and Vassilvitskii [3] proposed a seed centroid selection technique that is similar in spirit to the Hartigan and Wong technique but provides theoretical guarantees and justifications. In this approach the first seed centroid is selected uniformly at random from the collection sample S. Each of the remaining centroids are sampled from a distribution where the sampling probability of a data-point is proportional to its shortest distance from any of the currently selected seed centroids. As a result, data-points that are further away from any of the current centroids are more likely to be sampled. The authors also recommend sampling multiple data-points and retaining the farthest as the new centroid. This has an effect of increasing the probability of sampling a farther away point while still maintaining a low probability of sampling outliers. Notice that this approach requires the sampling distribution to be recomputed after each new centroid is sampled. The computational complexity of this approach is the highest among the four techniques we study at $O(|S||V|K)$, where V is the vocabulary of the sample.

4.3.5 Experimental results

We follow the recommendations of the previous sections and partition the GOV2 and CW09-B datasets into 250 and 100 topical shards, respectively, using the K-means allocation policy described in Section 4.1.3. For the VRS technique the minimum vocabulary size (the acceptance threshold) which is the average vocabulary size of the collection sample was computed to be 63 terms per document for GOV2 and 83 for CW09-B. For the three non-deterministic techniques (SRS, VRS and AV) three restarts were performed to capture the variance. Figures 4.24a and 4.24b report the results where the X-axis specifies the seed centroid selection technique. The minimum number of top shards that provided comparable search effectiveness to that of exhaustive search was chosen for each dataset. This provided an upper bound on selective search effectiveness under the different seed centroid selection techniques. These values are specified along the right Y-axis, and the corresponding cost in terms of documents (CiD) is on the left Y-axis. The search effectiveness for each of the techniques is comparable to that of exhaustive search which is reported in the figure caption.

For both the datasets we see that the efficiency of selective search is sensitive to the seed centroid selection technique. However, the sensitivity is lower for the larger dataset. Also, the

Algorithm 2 Size bounded sampling-based K-means (SB^2 K-means)

Input: Document collection $\boldsymbol{C}$, Sample size $|S|$, Number of shards K, Targeted shard-size range $[\theta^L, \theta^U]$

Output: $\boldsymbol{R}$ Topical shards

// Initial Phase

1: $\boldsymbol{S} \leftarrow$ SAMPLE ($\boldsymbol{C}$, $|S|$) // Sample $\boldsymbol{S}$ documents from $\boldsymbol{C}$.

2: $\{\boldsymbol{CENT}_K, \boldsymbol{R}^S_K\} \leftarrow K$-MEANS ($\boldsymbol{S}$, K) // Cluster $\boldsymbol{S}$ documents into K sample-shards $\{R^S_1, \cdots, R^S_k\}$, with cluster centroids $\{CENT_1, \cdots, CENT_k\}$.

// SPLIT Phase: Identify and split the large sample-shards

3: $\{\boldsymbol{CENT}_{K_{SPLIT}}, \boldsymbol{R}^S_{K_{SPLIT}}\} \leftarrow$ SPLIT ($\boldsymbol{R}^S_K$, θ^U), where $\boldsymbol{R}_{K_{SPLIT}}$ is the total number of sample-shards after the split phase. $K_{SPLIT} \geq K$.

// PROJECT Phase

4: $\boldsymbol{R}_{K_{SPLIT}} \leftarrow$ PROJECT ($\boldsymbol{C}$, $\boldsymbol{CENT}_{K_{SPLIT}}$) // Use the centroids to project the complete collection into K_{SPLIT} shards $\{R_1, \cdots, R_{k_{SPLIT}}\}$.

// MERGE Phase: Identify and merge small shards

5: $\boldsymbol{R}_{K_{MERGE}} \leftarrow$ MERGE ($\boldsymbol{R}_{K_{SPLIT}}$, θ^L, θ^U), where $\boldsymbol{R}_{K_{MERGE}}$ is the total number of shards after the merge phase.

high variance in CiD observed for the smaller dataset does not carry over to the larger dataset. For both the datasets, however, VRS provides lowest or near lowest search costs (CiD and CiS, both). Also recall that the computational complexity of VRS is among the lowest of the four techniques. Based on these observations we choose to employ the VRS seed centroid selection technique for all the remaining experiments in this chapter.

4.4 Size bounded sampling-based K-means (SB^2 K-means)

Distributed search systems typically prefer to divide the collection into equal sized partitions which allows for better load balancing and also provides low variance in query run times. The topic-based allocation approach (described in Section 4.1.3), however, is not guaranteed to create shards with an uniform distribution of sizes. In fact, the inherent topical diversity of the collection determines the size distribution of its shards. Figure 4.20 reported the sizes of shards that were created using the sampling-based K-means allocation approach (along with random, and source-based) for the three datasets, GOV2, CW09-B and CW09-Eng. We see that the distribution of the shard sizes is highly skewed for each of the datasets. Only a small fraction of the shards are of size that is comparable to the average shard size. The largest topical shard

is eight, three, and twenty-four times bigger than the expected shard size for GOV2, CW09-B and CW09-Eng, respectively. The smallest shard is nearly empty for all the datasets. Overall, there is a large variance in the shard sizes for each of these datasets.

In this section we propose and test a topic-based document allocation approach that partitions the collection such that the majority of the resulting shards are of comparable size. The new approach, *size bounded sampling-based K-means (SB^2 K-means)* is a simple extension of the sampling-based *K*-means algorithm and thus retains its efficiency and scalability.

Algorithm 2 provides the pseudo code for the SB^2 *K*-means approach. The *Initial phase* for the SB^2 *K*-means starts with sampling a small set of documents from the complete collection. These documents are clustered into sample-shards and the sizes of the sample-shards are used to identify the *large* sample-shards. The *split* phase iteratively identifies and divides the large sample-shards until a certain stopping criteria is reached. The next step, *project*, is the same as that employed in sampling-based *K*-means where the complete collection is partitioned into topical shards using the sample-shards centroids. In the third and final step of the algorithm, *merge*, small shards from the previous step are identified and combined with other shards using a certain merging strategy. This step is also repeated until a stopping criteria is reached. The main motivation for the three step SB^2 *K*-means algorithm is its scalability. The specific parameterizations and other details for each of the steps are given in the following sections.

4.4.1 Initial phase

The first two steps of Algorithm 2 are similar to those in sampling-based *K*-means in Section 4.1. Specifically, the sample $\boldsymbol{S}$ is compiled using *simple random sampling* which selects documents uniformly at random (without replacement) from the complete collection. The sizes of the samples for GOV2, CW09-B and CW09-Eng datasets are 252K, 502K and 503K, respectively. The samples are clustered into 250, 100 and 1000 sample-shards for GOV2, CW09-B and CW09-Eng, respectively, using the KL divergence based *K*-means algorithm described in Section 4.1.3. The *K* seed centroids for the *K*-means algorithm are selected using the vocabulary size based rejection sampling (VRS) approach described in Section 4.3.

4.4.2 Split phase

The upper bound on the sample-shard size is set to 110% of the average sample-shard size. Each sample-shard that is larger than the upper bound is divided into m sample-shards using the sampling-based *K*-means algorithm. The value for parameter m is proportional to the size of the sample-shard. Specifically, it is the ratio of the sample-shard size and the average sample-shard size. The m new sample-shards are augmented to the existing set of sample-shards and the original large sample-shard is removed from the set. Some or all of the new sample-shards may be large sample-shards. Thus the split step is applied iteratively five times to the latest set of

large sample-shards or until there are no more large sample-shards, whichever occurs earlier. This stopping criteria allows us to bound the runtime of the algorithm. Initial experiments showed that five rounds of iterative splitting provided a good balance between efficiency and effectiveness of this step.

4.4.3 Project phase

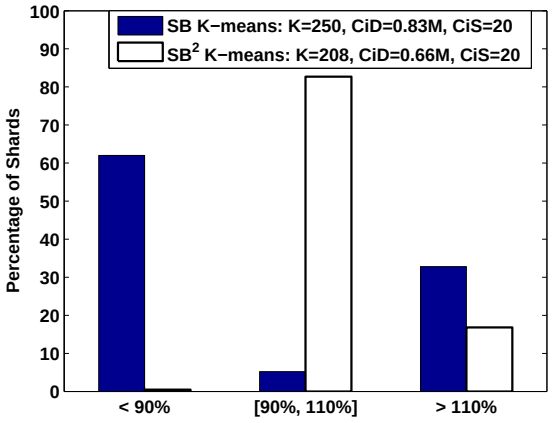
The parameterization employed for this step is exactly same as that used with the sampling-based K-means. The affinity metric given in equation 4.3 is used to infer the topical similarity between a document and each of the cluster centroids. The document is allocated to the cluster with the highest similarity. The output of this step is a disjoint partitioning of the document collection in topical shards.

4.4.4 Merge phase

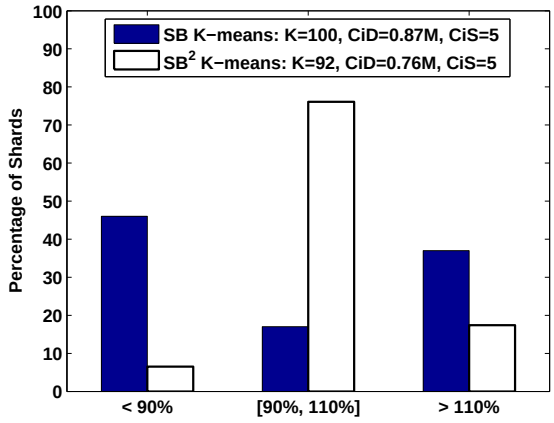
In the final step, the shards that are smaller than 90% of the average shard size are identified as the *source* shards. The shards that are not a large shard (size greater than 110% of the average shard size) are identified as the potential *sink* shards. Notice that the source and the sink sets are not mutually exclusive. A merging iteration starts with the largest sink shard and ends when the smallest sink shard has been considered. For each sink shard the largest possible source shard that can be merged without resulting in a large shard is combined. Like the split phase, the above merge process is repeated five times on the latest set of shards or until no shards can be merged. Due to the iterative nature of this approach more than two original shards can get merged.

Notice that the merging of shards is solely driven by the shard sizes. The topical similarity or dissimilarity of the shards being merged is not considered by the above approach. However, we did experiment with merging strategies that were functions of both, shard size and topical relatedness between shards. The experimental results for these strategies were often similar and sometimes inferior to the ones reported in this section. However, the computational cost of these strategies was much higher than the one used here because the topical similarity between shards had to be computed for each merge iteration.

The other design consideration that was explored was *when* to apply the merge phase. Unlike the split step which operates on the sample-shards the merge step operates on the shards in the above algorithm. We tested an alternative algorithm where the merge step was also performed on the sample-shards, right after the split phase. In this variant of the algorithm the project step uses cluster centroids that are products of two or more original centroids. We saw that this reduced the topical *fidelity* of the centroids and consequentially of the resulting centroid. Preliminary experiments demonstrated that this adversely affected selective search performance. As a result, the proposed algorithm employs the merge step only after the



(a) Dataset: GOV2.



(b) Dataset: CW09-B.

Figure 4.25: Shard size distribution for SB *K*-means and SB^2 *K*-means.

projection step.

4.4.5 Experimental results

The parameterization reported in the previous section results in 208 shards for GOV2, 92 shards for CW09-B, and 807 shards for the CW09-Eng dataset. The size distributions for each of these sets is compared to the ones obtained using the SB *K*-means approach in Figure 4.25. The Y-axis represents the percentage of shards assigned to each of the three shard size categories – small, targeted and large.

These plots demonstrate that for each dataset the SB^2 *K*-means algorithm creates substantially more shards that have size in the targeted range of [90%, 110%] of the average shard size. The SB^2 *K*-means is most effective for the GOV2 dataset for which 83% of the shards are of comparable size. CW09-B and CW09-Eng exhibit similar trends to each other where nearly 75% of the shards are within the targeted shard size range. As compared to the other two datasets CW09-Eng contains the smallest reduction in the number of large shards. Specifically, 21%, as compared to 48% for GOV2, and 54% for CW09-B. Recall that the same split iterations (5) were used for each dataset. It might be that for the CW09-Eng dataset more split iterations could have been useful. These results suggest that instead of using a collection-agnostic value for number of split iterations, a parameterization strategy that is a function of the collection size, and the number of original large shards might be more effective at reducing the number of large shards.

The effects of shard size manipulation on the efficiency of selective search are summarized in the legend of each plot in Figure 4.25. We see that the equal sized shards support more

efficient selective search. The cost in terms of documents (CiD) reduces by 20%, 12% and 12% for GOV2, CW09-B, and CW09-Eng, respectively. The ReDDE algorithm, and many other shard ranking algorithms, are inherently biased toward the larger shards. We see that a more uniform shard size distribution helps counterbalance this bias which in turn reduces the average number of documents evaluated per query. The CiS which is the number of top shards searched per query remains steady in spite of the changes in the shard. This indicates that the SB^2 *K*-means algorithm does not disperse the relevant documents for a query across shards. The search effectiveness results for both the allocation approaches, SB *K*-means and SB^2 *K*-means are the same and thus are not repeated here.

Overall, this analysis establishes SB^2 *K*-means as the recommended topic-based document allocation technique for selective search because of its higher efficiency and near-uniform size distribution of the created shards.

4.5 Summary

This chapter studied the offline phase of the selective search architecture where the goal is to efficiently partition large collections into shards that are able to support effective selective search. We developed and tested several shard creation techniques. The experimental results recommend source-based document allocation policy for its efficiency and reasonable average-case precision at early ranks. Source-based, however, scores low on query-level stability analysis. Also if effectiveness at deeper ranks is important then selective search with source-based shards is not recommended.

Instead, the topic-based policy proves to be an all-round sharding technique for distributed selective search. The offline cost of shard creation is higher for the topic-based policy than the other techniques studies here. However, during the online phase of query processing, topic-based shards support selective search that is consistently more effective and efficient for a range of metrics and dataset sizes. We also see a positive correlation between distributed selective search performance and collection size in the results presented in this chapter. Selective search is more effective and efficient when searching larger datasets.

This chapter also presented a *relevance density* analysis that compared the partitioning techniques based on size normalized distribution of relevant documents across shards. The topic-based shards exhibit the highest skew in the relevance density distribution, followed by source-based. The ability of the topic-based policy to concentrate the relevant documents for a query into a few shards is one of the primary reasons for its high effectiveness and efficiency. This analysis also brought to surface the skewness in the shard size distribution for topic-based shards.

We propose a variant of the topic-based policy that reduces the skewness in the distribution of shard sizes substantially. The new approach further increases the cost of partitioning a col-

lection into topical shards. However, the experimental results demonstrate that the additional offline cost pays off during the online phase of selective search. A set of more evenly sized topical shards is able to support selective search that is just as effective but more efficient than topical shards with a skewed size distribution.

This chapter also analyzed the effects of the total number of shards that a collection is partitioned into on selective search performance. As a general trend the results show that increasing the number of total shards decreases the cost-in-documents but increases the cost-in-shards. The parameterization that minimizes both the costs is collection-specific in the results presented here. We believe the optimal value for this parameter is a function of the latent topics in the collection. An efficient estimator for the number of latent topics in a large collection is an open research challenge.

Overall, this chapter establishes that distributed selective search with topic-based shards successfully satisfies the objectives laid out in Section 3.2: *Competitive search effectiveness*, *Low search effort*, *Low resource requirements*, and *Scalability*, while also complying with the specified constraints and assumptions. The query runtime objective is studied in Chapter 6.

Chapter 5

Online phase: Query processing

This chapter studies the online phase of distributed selective search where the incoming queries are processed against the shards created during the offline phase. The query processing task consists of three stages: query transformation, shard ranking and cutoff estimation, and retrieval and merging. The first two components are the focus of this chapter. For the final step of shard retrieval we employ an off-the-shelf document retrieval algorithm, Indri [51] implemented in the Lemur Toolkit[1]. The results merging task is straightforward in this cooperative search environment. The relevance scores of documents retrieved from different shards are comparable because the global statistics such as the idf (inverse document frequency of the complete collection) that are used for score normalization are compiled and shared with each of the shards at partitioning time.

5.1 Query representation: Bag-of-words versus Dependence model

The commonly used *bag-of-words* (BOW) query representation is an unstructured formulation of the query that ignores any potential relations between the query terms. Instead, using a query representation that explicitly asserts the dependencies among the query terms has been shown to improve adhoc retrieval performance [52] in an exhaustive search setup. The query representation proposed by Metzler and Croft [52] expresses term dependence using ordered and unordered proximity constraints constructed from the query terms. A full-dependence model query assumes each query term to be dependent on all the other query terms. An example of a full-dependence model query formulated using the Indri Query Language[2] is given in Figure 5.1. The relation between consecutive terms is expressed using the ordered distance operator #1, and the weaker dependence between distant terms is conveyed through the unordered window operator (#uwN) in this representation.

[1] www.lemurproject.org

[2] http://www.lemurproject.org/lemur/IndriQueryLanguage.php

Bag of words query: *obama family tree*

Full-dependence model query:

```
#weight(
    0.8 #combine(obama family tree)
    0.1 #combine(
            #1(obama family)
            #1(family tree)
            #1(obama tree)
        )
    0.1 #combine(
            #uw8(obama family)
            #uw8(family tree)
            #uw8(obama tree)
            #uw12(obama family tree)
        )
)
```

Figure 5.1: Query representation example.

Although Metzler and Croft concluded that dependence model query representations improve the search effectiveness, they also noted that the improvements are dependent on factors such as the collection size, homogeneity of the documents, proportion of noisy documents in the dataset, and query lengths. In our work when we partition a large collection into multiple topical shards, the characteristics of the individual shards are markedly different from those of the complete collection. For instance, a topical shard is smaller, topically more homogeneous, and less noisy than the complete collection. As such, it is not clear whether the trends observed by Metzler and Croft for exhaustive search would also hold for selective search. Also, the effect of query representation on search efficiency has not been explored. In order to answer these questions we compare the bag-of-words representation with the full-dependence model query representation in the context of distributed selective search, and also in the context of exhaustive search.

For this investigation a search setup where the three datasets, GOV2, CW09-B, and CW09-Eng had been partitioned into 50, 100, and 1000 topical shards, respectively, using the SB *K*-means technique (Section 4.1.3) was employed[3]. During the online phase the ReDDE [69] algorithm was used to rank the shards for each query. The TREC topics from the evaluation queries sets for each of the datasets were used as the bag-of-words baseline queries. The TREC topics were transformed into full-dependence model queries using the utility made available

[3]The work presented in this chapter was done before SB^2 *K*-means, the size-balanced shard creation approach presented in Section 4.4, was developed. As a result, the experiments presented in this chapter use SB *K*-means, and not SB^2 *K*-means.

Figure 5.2: Bag-of-words versus dependence model query representation. ▲ denotes significantly better search effectiveness with dependence model query representation than with BOW ($p < 0.01$).

(a) Dataset: GOV2.

Search Type	Query Rep	CiS	CiD (million)	P@10	P@30	P@100	NDCG @100	MAP
Exhaustive	BOW	10 (/10)	3.63	0.53	0.48	0.38	0.42	0.29
	Dep	10 (/10)	3.63	▲0.58 (+9%)	▲0.52 (+10%)	▲0.42 (+8%)	▲0.47 (+10%)	▲0.32 (+9%)
Selective	BOW	10 (/50)	1.59	0.53	0.48	0.37	0.42	0.28
	Dep	10 (/50)	1.62	▲0.59 (+10%)	▲0.52 (+10%)	▲0.41 (+8%)	▲0.47 (+10%)	▲0.32 (+10%)

(b) Dataset: CW09-B.

Search Type	Query Rep	CiS	CiD (million)	P@10	P@30	P@100	NDCG @100	MAP
Exhaustive	BOW	7 (/7)	5.37	0.24	0.22	0.19	0.24	0.16
	Dep	7 (/7)	5.37	▲0.27 (+11%)	▲0.26 (+5%)	0.21 (+15%)	▲0.27 (+10%)	▲0.18 (+11%)
Selective	BOW	7 (/100)	1.08	0.25	0.24	0.19	0.24	0.15
	Dep	7 (/100)	1.07	▲0.28 (+11%)	▲0.28 (+5%)	▲0.21 (+14%)	▲0.27 (+10%)	▲0.18 (+11%)

(c) Dataset: CW09-Eng.

Search Type	Query Rep	CiS	CiD (million)	P@10	P@30	P@100	NDCG @100	MAP
Exhaustive	BOW	3 (/3)	51.29	0.12	0.11	0.10	0.10	0.06
	Dep	3 (/3)	51.29	0.13 (+8%)	0.13 (+0%)	▲0.12 (+15%)	0.11 (+17%)	0.07 (+14%)
Selective	BOW	3 (/1000)	3.16	0.13	0.13	0.12	0.11	0.06
	Dep	3 (/1000)	3.16	0.14 (+7%)	0.15 (+11%)	0.13 (+13%)	0.12 (+8%)	0.07 (+14%)

by Metzler[4] for each of the datasets. The parameters recommended by Metzler and Croft in [52] for the transformation of the bag-of-words queries to full-dependence model representation were used for these experiments. Specifically, we use 0.8, 0.1, and 0.1 weights for the unigram, ordered n-gram, and unordered n-gram features, respectively.

For selective search the number of shards searched (CiS) for a query was decided using an oracle setup. A detailed description of the oracle based methodology for setting CiS is in Section 4.1.5. In summary, the CiS oracle chooses a value for this parameter that provides a good balance between the effectiveness and the efficiency of selective search. As such, the results presented in this section provided an upper bound on selective search performance under the two different query representations. These oracle values are reported in the result tables as the cost-in-shards (CiS) metric.

The dependence model query representation incurs additional processing cost since new posting lists need to be created on the fly for the multi-term features. The search efficiency metric, cost-in-documents (CiD), however only computes the number of documents evaluated for each query and thus does not model this overhead associated with dependence model query processing. However, since the focus of this section is on search effectiveness, we do not address this inaccuracy in the measurement of search cost here. Instead, the following chapter revisits search efficiency from the perspective of search time, and provides a more accurate characterization of the cost of selective search with the two query representations.

The results for the three datasets are reported in Table 5.2. The trends in search effectiveness observed for exhaustive search are closely replicated by selective search for all the three datasets. Many of the improvements in effectiveness with dependence model queries are statistically significant for the smaller datasets, GOV2 and CW09-B. For the largest dataset, although dependence model query representation improves search effectiveness, the improvements are not statistically significant in most cases. This is contrary to the observation made by Metzler and Croft that the improvements offered by dependence model query representation were larger for larger datasets. It is possible that the collection-agnostic parameter settings that were used in this work (as per the recommendation by Metzler and Croft) are the cause of this contradiction.

The average number of documents evaluated per query (CiD) are comparable for the two query representations for both the search approaches. However, the overhead costs of processing the dependence model queries are not included in the reported numbers for both, exhaustive search and selective search.

Overall, these results demonstrate that selective search is similar to exhaustive search in its ability to leverage richer query representation to improve search effectiveness.

[4]http://ciir.cs.umass.edu/ metzler/dm.pl

5.2 Shard ranking

The next step in the query processing pipeline of selective search is that of ranking the shards based on their estimated relevance to the query. The goal is to restrict the search to only the most relevant shards for the query.

The problem of ordering a collection of documents (resources) based on their query relevance has been extensively studied in the context of federated search [2, 14, 32, 40, 69, 74]. All the experimental evaluations reported until this point in the dissertation have employed a simple variant of a well-established resource ranking algorithm, ReDDE, for the purpose of ranking shards. In this section we analyze the effects of the choice of shard ranking algorithm on selective search performance. Specifically, we experiment with two resource ranking algorithms, CORI and ReDDE, and compare their ability to support selective search (both are described in detail in Section 2.3.1 and 2.3.2). In addition to being widely used these particular algorithms were chosen because they exhibit different characteristics and biases.

The CORI algorithm belongs to a class of algorithms that learn a representative model for each resource and use them as the basis for resource ranking. In contrast, the ReDDE algorithm adopts a *sample-based* approach where samples of documents from each shard are directly used as representatives of the shards' contents. The results obtained from running the query against the *sample index* are used as the basis for resource ranking.

The models learned for CORI typically only contain summary statistics such as the shard-specific *df*s for the shard vocabulary. Information about term frequency in individual documents is not available and is not used during shard scoring. As such, document level term weighting is not performed for CORI. In case of ReDDE, however, the document retrieval against the sample index provides an opportunity to use document term weights. This is important because the term weights model the ability of a shard to provide high relevance documents as opposed to low relevance or false-relevant documents. This limitation of CORI may be less of a problem for topical shards which inherently provide some amount of reduction in noise (false-relevant documents).

The other difference between the two ranking algorithms is in the formulation they employ for shard scoring. The shard size based normalization used by the CORI algorithm induces a bias for smaller shards containing many candidate documents. The ReDDE algorithm on the other hand is biased toward larger shards containing many candidate documents because it weights the shard scores based on the shard sizes. These differences among the algorithms make them good candidates for testing the sensitivity of selective search to the choice of shard ranking algorithm.

The CORI and ReDDE algorithms also differ in their computational costs. Although previous work does not provide any comparative analysis, we expect CORI to be more efficient than ReDDE due to the following reasons. Recall that CORI is a single step process where the

query (Q) is evaluated against each of the shard models that contain the query terms (worst-case computational complexity: $O(K|Q|)$, where K is the number of shards.). Whereas ReDDE consists of a two step process where the query is first executed against the central sample index (CSI) (worst-case computational complexity: $O(|CSI||Q|)$), and then a shard ranking is inferred from the retrieved results. The number of documents in the CSI are typically much larger than the number of models used by CORI ($|CSI| \gg K$). We thus expect selective search with ReDDE to be computationally costlier than selective search with CORI. The focus of this section is the analysis of selective search's effectiveness using the two shard ranking algorithms, however, we acknowledge that a careful study of the costs of these two algorithms is an important future work.

The ReDDE algorithm as proposed by Si and Callan [69] was described in Section 2.3.2. In this work we used a modified version of ReDDE that demonstrated better performance in preliminary experiments. We describe this variant of ReDDE next, and then report the experimental results comparing selective search performance with CORI and ReDDE.

5.2.1 Modified ReDDE

The resource ranking algorithm, ReDDE [69], uses a *central sample index*, *CSI*, to estimate the distribution of relevant documents across shards. Previous work has typically assumed an uncooperative environment and thus employed query-based-sampling [16] for creating the CSI. In this dissertation, however, we work with a cooperative setup and thus employ the more accurate sampling technique, simple random sampling. The original ReDDE algorithm and the variant used in this dissertation also differ in the formulation used for shard score computation.

ReDDE executes the query against the CSI and uses the shard membership of the top m retrieved documents, and the shard weights (which are ratio of resource size to sample size) to compute a score for each shard (Section 2.3.2). This formulation weights each retrieved document equally irrespective of its rank or score in the retrieved list. Instead we use the relevance score assigned by the retrieval model to weight the document. Also, we do not scale the shard scores with shard weights because that introduces a bias for larger shards. When these modifications are put together the resulting scoring function is as follows:

$$s_R = \sum_{i=1}^{m} score(D_R^i) \tag{5.1}$$

where s_R is the score computed for shard R, D_R^i is a document from shard R that was retrieved at rank i from CSI for the query. The $score(.)$ function returns the relevance score assigned by the ranking model. This modification to the formulation incorporates an aspect of shard relevance quality into the estimation process. The scores computed using this formulation are used to rank the shards for the query, and the top few shards are searched for the query.

Table 5.1: Selective search performance using CORI and ReDDE shard ranking algorithm. ▲ denotes significantly better search effectiveness than the other shard ranker ($p < 0.01$).

Dataset	Shard Ranker	CiS	CiD (million)	P@10	P@30	P@100	NDCG @100	MAP
GOV2	CORI	10 (/50)	1.63	0.54	0.48	0.37	0.42	0.22
	ReDDE	10 (/50)	1.59	0.53	0.48	0.37	0.42	▲0.28
CW09-B	CORI	7 (/100)	1.21	0.23	0.23	0.18	0.23	0.13
	ReDDE	7 (/100)	1.08	0.25	0.23	0.19	0.24	0.15
CW09-Eng	CORI	5 (/1000)	5.37	0.13	0.12	0.10	0.10	0.04
	ReDDE	3 (/1000)	3.16	0.13	0.13	0.12	0.11	0.06

5.2.2 Experimental setup: CORI versus ReDDE

The three datasets were partitioned into 50, 100 and 1000 topical shards. A central sample index (CSI) used by the ReDDE algorithm was created for each dataset using simple random sampling. From each shard 4% of the documents were sampled for the CSI. Recall, that the sample size was guided by the results from the analysis presented by Callan, 2001 [15] where resource representations learned from 3% to 6% of the resource documents were found to be only slightly worse than the complete resource representations. Many of the previous studies that used CORI learned the shard models from a subset of the document collection. This design choice was often enforced by the uncooperative search environment assumed in these studies. We need not operate within such constraints in this work and thus for CORI we use the complete collection to obtain accurate collection statistics.

CORI uses a unigram language model when learning the shard models. As a result, queries that need term co-occurrence information, such as the dependence model queries, cannot be evaluated accurately with this algorithm. As a result, the bag-of-words representation was used for experiments reported in this section. The Indri algorithm was employed for the CSI retrieval in case of ReDDE experiments, and for shard retrieval in both ReDDE and CORI experiments reported here. The number of shards searched (CiS) was decided by an oracle for all the selective search experiments reported in this section. Please refer to Section 4.1.5 for a detailed description of the oracle methodology for setting CiS. In short the CiS oracle chooses a value for this parameter that provides a good balance between the effectiveness and the efficiency of selective search. As such, the results presented in this section provided an upper bound on selective search performance under the two different shard ranking algorithms.

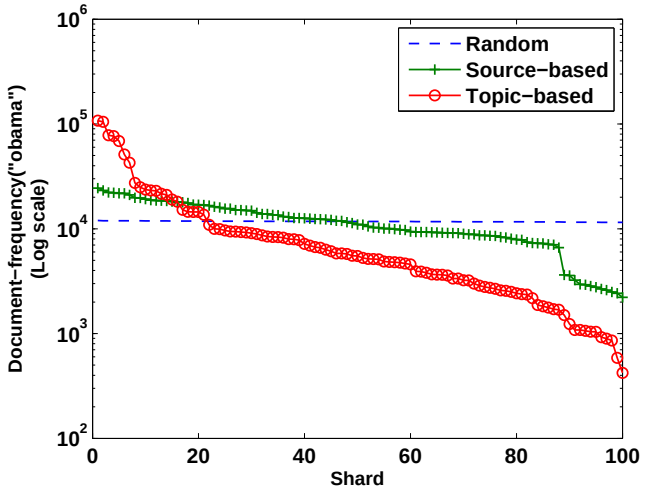


Figure 5.3: Distribution of the term *obama* across shards. Dataset: CW09-B.

5.2.3 Experimental results: CORI versus ReDDE

The results reported in Table 5.1 demonstrate that both the shard ranking algorithm support competitive selective search performance, especially at early ranks. CORI struggles with providing competitive effectiveness at deeper ranks, especially for the smallest dataset, GOV2. In terms of the search cost, CiD, ReDDE scores consistently better than CORI across all the datasets. Recall that CiS is the number of top shards that had to be searched in order to provide search effectiveness that is comparable to that of exhaustive search. We see that for the largest dataset more of the top shards need to be searched when they are ranked by CORI than by ReDDE. This suggests that the ranking proposed by CORI for this dataset is inferior to the one by ReDDE which is consistent with prior research.

Previous work that compared CORI and ReDDE have often reported larger difference in their performance [2, 69, 74] than that observed here. We believe that one of the reasons for this discrepancy is the difference in the accuracy of the collection statistics made available to CORI. Recall that in this work we use the complete collection to generate the shard models for CORI whereas many of the previous works were restricted only to a subset.

The other notable difference between this search setup and the ones used in other studies is the topic-based arrangement of the documents. The distribution of terms across shards is markedly different for topical shards than for random, or even source-based shards. An example is provided in Figure 5.3 where the document frequencies of the term *obama* in each of the 100 shards of the CW09-B dataset are plotted. The distribution of the term across shards is much more skewed for the topic-based shards than the other two types of shards. A shard ranking algorithm is likely to be more accurate when the distribution of the query terms across shards is skewed. This could be the other contributing factor for the higher performance of the

CORI algorithm here than that reported in previous work.

Overall, these results demonstrate that the performance of distributed selective search is not overly dependent on the choice of the shard ranking algorithm. We also see that model-based algorithms like CORI perform much better with topic-based shards and in cooperative search environment, than what they have in prior research.

5.3 Shard ranking and cutoff estimation: Sampling-based Hierarchical Relevance Estimation (SHiRE)[5]

In addition to experimenting with existing resource ranking algorithm for selective search we also propose a suite of algorithms that solve the two related problems of *shard ranking*, and *shard rank cutoff estimation* together. Although the problem of ranking collections of documents has received plenty of attention, the task of estimating the *number of shards* that should be searched for a given query has gone nearly unnoticed. Most previous work (including the experiments reported in this dissertation until now) used fixed cutoff values on the shard ranking that are query-agnostic [14, 60, 65, 69] or used other query independent criteria, such as, the load on the system, to determine the number of shards that would process the query [61].

We define the *optimal shard rank cutoff* as the smallest rank in a given shard ranking that provides search effectiveness that is on par with exhaustive search. Notice that the optimal cutoff is specific to a shard ranking. Two different shard rankings for the same query could have different optimal cutoff values. The cutoff also depends on the evaluation metric under consideration.

Figure 5.4 plots the optimal rank cutoffs for 150 evaluation queries used with the GOV2 dataset. The optimal rank cutoffs were computed for precision at rank 10 metric (P@10). The straight lines represent the commonly used query-agnostic fixed rank cutoffs. This figure illustrates that the optimal shard rank cutoffs (represented by asterisks) exhibit high variability across queries. A fixed cutoff leads to an under or overestimation error for many queries. A small fixed value, such as 1, would lead to a poor search effectiveness for 42% of the queries in this example, while a larger fixed value of 10, would lead to an unwarranted increase in the search cost for 99% of the queries. Neither of these error types are acceptable choices for most search environments but it is especially critical for search providers with limited computing resources to operationalize a search strategy that is efficient and also effective.

The task of estimating the optimal shard rank cutoff for a query is inherently dependent on the predicted ranking of the shards for the query. An ordering that places the relevant shards at the top enables an early shard rank cutoff. However, a poor shard ranking requires a much deeper search into that particular ranking. Solving these problems together allows for

[5]This work was performed in collaboration with researchers from the University of Twente, Dr. Almer Tigelaar and Dr. Djoerd Hiemstra, and was published in CIKM 2012 [46].

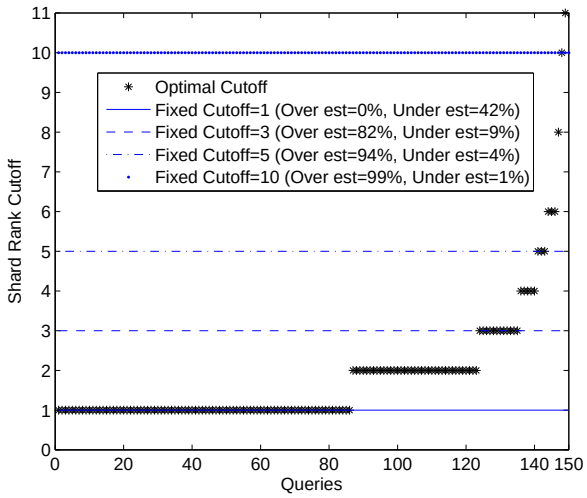


Figure 5.4: Optimal versus fixed shard rank cutoffs for ReDDE. Dataset: GOV2. Metric: P@10.

modeling the intrinsic dynamics between these two components. We thus propose a family of three algorithms that provide a ranking of shards for the given query and also estimate the optimal search cutoff in this ranking that supports competitive search effectiveness. We refer this family of algorithms together as *S*ampling-based *H*ierarchical *R*elevance *E*stimation (SHiRE). The individual algorithms are named as Lexical SHiRE (Lex-S), Rank SHiRE (Rank-S), and Connected SHiRE (Conn-S). The outline of the algorithm that is common to all the three members of the SHiRE family is provided in the form of a pseudo-code in Algorithm 3.

The first step of Algorithm 3 is common to all the three SHiRE algorithms. Each of the SHiRE algorithms starts by executing the query against the CSI using Indri and obtaining a ranked result list of documents. The information contained in this list, such as, the document ranks, the retrieval scores, the document contents, and the originating shards of the documents, is used by each algorithm in a unique way to construct a query-specific hierarchy in the second step. The specifics are provided in Sections 5.3.2 through 5.3.4. In the final step of the SHiRE algorithm the constructed hierarchy is traversed bottom-up starting from the node containing the top ranked CSI document. The traversal stops when the root node is visited. The final output of this traversal is a set of shards that are estimated to contain majority of the relevant documents for the query. The traversal and scoring function used in this last step is common to all the SHiRE algorithms, and is described in Section 5.3.1.

The rational behind overlaying a hierarchy on the *flat* CSI results for the query is to encode additional information about the documents and the originating shards. Often the sample used to create the CSI, and thus the shard ranking, is a very small portion of the complete collection. As such, the shard ranking algorithm is offered only a limited and incomplete view of the collection contents. Short queries can make the task of shard ranking further challenging.

Algorithm 3 Sampling-based **Hi**erarchical **R**elevance **E**stimation (SHiRE)

Input: Central Sample Index (CSI), and query Q.

Output: Set of shards, $\{R_1, .., R_P\}$, to search for the query Q.

// Execute the query Q against CSI and obtain the ranked list of documents retrieved for the query.

1: $\{D_1..D_n\} \leftarrow$ runQuery(CSI, Q)

// Organize the CSI results into a hierarchy using one of SHiRE algorithms: Lexical SHiRE (Algorithm 4) OR Rank SHiRE (Algorithm 5) OR Connected SHiRE (Algorithm 6).

2: tree.node(D_1) $\leftarrow$ transform($\{D_1..D_n\}$) // transform function returns a pointer to the node containing the top ranked CSI document.

// Perform a bottom-up traversal of the hierarchy starting at the node returned by the previous step (Section 5.3.1). The output of the traversal function is the set of shards that are predicted to be relevant to the query Q.

3: $\{R_1, .., R_P\} \leftarrow$ traversal(tree.node(D_1))

When the *flat* CSI results are transformed into an hierarchy it provides an opportunity to model the relations between the retrieved documents, in terms of their lexical similarities or shard memberships. The goal is to gleaning additional information for shard ranking and cutoff estimation.

Like ReDDE, the SHiRE algorithms are also instances of *sample-based ranking algorithms* since they make use of the *sample index* to represent the shard contents. This choice was motivated by the observation that the use of CSI for resource ranking has shown to be effective by several previous studies [2, 69, 74], and also in our experiments in Section 5.2. Also, note that the CSI creation happens during an offline phase that is not repeated for each query.

The details for steps two and three of the Algorithm 3 are provided next, starting with the third step of traversal and scoring of the hierarchy of the CSI documents.

5.3.1 SHiRE tree traversal and scoring function

Given a hierarchy where each leaf node is a document from the CSI result list for the query, the traversal starts from the leaf node that represents the top ranked CSI document. A bottom-up traversal starting from this node reveals more documents at each step up through the tree. All the documents that are directly attached to the current node or are indirectly attached through a sub-tree, are both considered revealed at this level. The documents uncovered at each step 'vote' for the shards that they represent. However, the votes are exponentially decayed at each level in this traversal. Specifically, a document d revealed at step U in the traversal contributes

a vote toward its parent shard as follows:

$$Vote(d) = W_d \cdot B^{-U} \tag{5.2}$$

where W_d is either the document score assigned by the retrieval model or is a unit weight, B is the base of the exponential function, and $U = 0, 1, 2, 3..$ is the traversal level.

The bottom-up traversal terminates once the root node is reached. The accumulated votes assigned to the shards during the traversal are used to obtain a ranking of the shards ($\{R_1..R_K\}$) for the query. This method of scoring the shards for the query has an effect of driving the scores to zero for shards that are represented only higher up in the tree. We leverage this property to define an estimator for shard rank cutoff. It predicts the rank (P) at which the corresponding shard score is close to zero as the cutoff for the query. As such, the top ranked shards up to the predicted cutoff ($\{R_1..R_P\}$) are returned as the final output of the SHiRE algorithms.

For the experiments presented in this chapter we interpret the score of 10^{-4} or lower as having converged to zero. In retrospect we realize that it would have been better to introduce an additional parameter (C) for this threshold. The values chosen for the parameters B and C together influence the cutoff prediction made by the SHiRE algorithm for a query. Introducing a new parameter C would facilitated a thorough study of these interactions. In the experiments presented in this dissertation, only a single value for C (10^{-4}) was tested; it was effective, so other values were not considered. However, other values could very well be better choices for the parameter C. This exploration was not conducted here but would be a useful future direction for the SHiRE algorithms.

By starting the tree traversal at the top ranked CSI document these algorithms assert that the shard represented by the first document in the CSI ranking is likely to be the most relevant shard for the query. The process of dampening the votes at each step models the intuition that longer path lengths from the anchor node implies less similarity with it which further implies lower likelihood of relevance to the query (*Cluster Hypothesis* [77]). If a tree is very shallow, few shards would accumulate a zero relevance score, whereas a very deep tree will result in many shards with zero scores.

This tree traversal and the scoring methodology is further illustrated with examples for each of the three SHiRE algorithms in the following sections.

5.3.2 Lexical SHiRE (Lex-S)

The *Lex-S* algorithm organizes the CSI documents retrieved for a query into a hierarchy based on their lexical similarities. The pseudo-code for *transform* function implemented by the Lex-S is provided in Figure 4. As a first step the Lex-S algorithm represents each of the CSI documents retrieved for the query as a vector of tf-idf values for the unique terms of the document. An off-the-shelf agglomerative hierarchical clustering algorithm is then applied to the document vectors in order to obtain a document similarity based tree. The Manhattan distance metric was

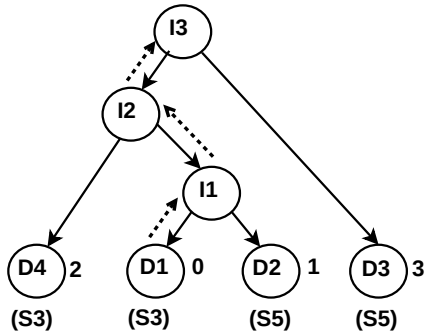


Figure 5.5: Lexical SHiRE. CSI ranking: $D1, D2, D3, D4$. Dashed arrows represent the path followed by the bottom-up traversal starting at $D1$. Numbers besides the leaf nodes specify the order of revelation of the documents. Parent shards specified in the brackets.

Algorithm 4 *transform* function by Lexical SHiRE (Lex-S)

Input: ranked results list from CSI for Q, $\{D_1..D_n\}$.
Output: tree node for the top ranked document, tree.node(D_1).

// Fetch the document vectors for $\{D_1..D_n\}$.
1: **for each** rank r in $\{1..n\}$ **do**
2: $DV_r \leftarrow$ fetchDocumentVector(D_r)
3: **end for**

// Apply hierarchical agglomerative clustering (HAC) to the document vectors
4: tree $\leftarrow$ HAC(Linkage_criterion:Ward, Distance_metric:Manhattan, $\{DV_1..DV_n\}$)

// Return the node in the resulting tree for the top ranked document D_1.
5: **return** tree.node(D_1)

used to compute pairwise lexical similarities between the document vectors and the Ward's method [81] provided the linkage criterion. The computational complexity of agglomerative clustering is $O(n^3)$ where n is the number of CSI documents retrieved for the query.

The motivation for organizing the documents based on their similarity is that such an hierarchy provides equal voting rights to all documents that are similar (attached to the same node in the tree) irrespective of their CSI ranking.

An example hierarchy for a toy set consisting of four CSI documents retrieved for a query is shown in Figure 5.5. Dn represents the document at rank n in the CSI results. Recall that the ranking of shards is derived by traversing up this tree, starting from the leaf node for the first document in the CSI ranking ($D1$). The next step in the traversal would reach the internal node $I1$ and thus reveal the document $D2$. $D4$ would be observed next and $D3$ would be

Algorithm 5 *transform* function by Rank SHiRE (Rank-S)

Input: ranked results list from CSI for Q, $\{D_1..D_n\}$.
Output: tree node for the top ranked document, tree.node(D_1).

```
    // Create a leaf node for the first document, D_1.
1:  L_1 ← createLeafNode(D_1)

    // Create a parent node for the first document
2:  I_1 ← createInternalNode()

    // Attach the leaf node as a left child of the parent
3:  makeLeftChild(I_1, L_1)

4:  for each rank r in {2..n − 1} do
5:      L_r ← createLeafNode(D_r)  // Create a new leaf node for the current document, D_r
6:      makeRightChild(I_{r−1}, L_r)  // Attach the leaf node as a right child of the last created parent
7:      I_r ← createInternalNode()  // Create a parent node for the next document
8:      makeLeftChild(I_r, I_{r−1})  // Attach the previous parent as a left child to the new parent node
9:  end for

    // Create a leaf node for the last document, D_n.
10: L_n ← createLeafNode(D_n)

    // Attach the leaf node as a right child of the last created parent
11: makeRightChild(I_{n−1}, L_n)

    // // Return the node in the resulting tree for the top ranked document D_1.
12: return L_1
```

found the last. If V_{D1} is the vote that $D1$ ascribes to its parent shard then $D1$ will contribute: $V_{D1} \cdot B^{-U}$, where U=0 is the number of steps traversed up the hierarchy, and B is the base of the exponential decay function. Similarly, $D2$, $D4$ and $D3$ will contribute: $V_{D4} \cdot B^{-1}$, $V_{D3} \cdot B^{-2}$, and, $V_{D4} \cdot B^{-3}$ respectively, to their parent shards. The shard $S3$ will accumulate votes from documents $D4$ and $D1$, and shard $S5$ will accumulate votes from documents $D2$ and $D3$. The resulting scores would be used to rank the shards $S3$ and $S5$. For more complex trees where a traversal step reveals a subtree instead of an individual node, all the documents attached to the sub-tree will be considered revealed at the same level. As a result, the B^{-U} component of the scoring equation will be the same for all these documents.

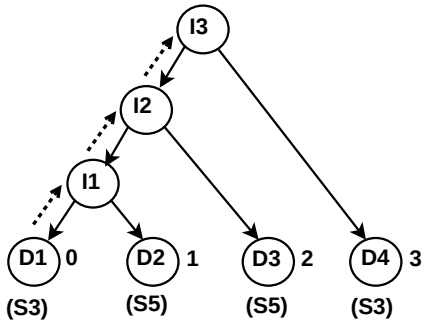


Figure 5.6: Rank SHiRE. CSI ranking: $D1, D2, D3, D4$. Dashed arrows represent the path followed by the bottom-up traversal starting at $D1$. Numbers besides the leaf nodes specify the order of revelation of the documents. Parent shards specified in the brackets.

5.3.3 Rank SHiRE (Rank-S)

The *Rank-S* algorithm uses only the rank information of the documents retrieved from CSI for the query to construct the hierarchy. A pseudo-code for this transformation is provided in Figure 5. The tree creation starts at the leftmost leaf node which is assigned to the top ranked document. All the remaining documents get attached as right leaf nodes in the resulting left-branching binary tree. An example tree structure created by the Rank-S transform function for a toy example is illustrated in Figure 5.6. The tree creation stops when the last document in the CSI ranking get attached to the highest node (root) in this tree.

The motivation for the Rank-S transformation is to make use of the document ranks to infer shard ranking and cutoff estimation. The goal is to prefer shards that are likely to provide highly ranked documents for the query.

The procedure for inferring a shard ranking from this tree is similar to that used by the Lex-S algorithm. For the toy example in Figure 5.6 the traversal would start at the leaf node, $D1$, and proceed to discover documents $D2$, $D3$, and $D4$, in that order. As a result, the parent shard of documents $D1$ and $D4$, shard $S3$ will be assigned a score of $(V_{D1} \cdot B^{-0}) + (V_{D4} \cdot B^{-3})$. The shard $S5$ which is represented by documents $D2$ and $D3$ will be assigned a score of $(V_{D2} \cdot B^{-1}) + (V_{D3} \cdot B^{-2})$.

The shard ranking and cutoff estimation achieved through the above method can also be replicated without the use of a tree structure, by simply walking down the ranked list, and using the document rank to set the variable U in equation 5.2. Instead, we use the tree structure to leverage the existing framework of the SHiRE algorithm.

The simplicity of the Rank-S algorithm makes it the most efficient algorithm of this family. This approach is most similar in spirit to the CRCS(e) ranking method [65]. However, other CSI based shard ranking methods, like ReDDE, can also be easily transformed to work with a left-branching binary tree of CSI documents where the votes are not decayed during the bottom-up traversal. As such, the computational complexity of Rank-S is very similar to that of ReDDE.

Algorithm 6 *transform* function by Connected SHiRE (Conn-S)

Input: ranked results list from CSI for Q, $\{D_1..D_n\}$.
Output: tree node for the top ranked document, tree.node(D_1).

// Create a leaf node for the first document, D_1.
1: $L_1 \leftarrow$ createLeafNode(D_1)

// Create a parent node for the first document
2: $I_1 \leftarrow$ createInternalNode()

// Attach the leaf node as a child of the parent
3: makeChild(I_1, L_1)

4: $j = 1$
5: **for each** rank r in $\{2..n\}$ **do**
6: $L_r \leftarrow$ createLeafNode(D_r) // Create a new leaf node for the current document, D_r
// Check if the current document, D_r, belongs to the same shard as the previous document.
7: **if** (shard(D_r) == shard(D_{r-1})) **then**
8: makeChild(parent(D_{r-1}),L_r) // Attach the leaf node as a child of the parent node of the previous document
9: **else**
10: j++
11: $I_j \leftarrow$ createInternalNode() // Create a new internal node
12: makeChild(I_j,parent(D_{r-1})) // Attach the previous parent as a child of the new parent
13: makeChild(I_j,L_r) // Attach the leaf node as a child of the new parent
14: **end if**
15: **end for**

// Return the node in the resulting tree for the top ranked document D_1.
16: **return** L_1

5.3.4 Connected SHiRE (Conn-S)

The *transform* function defined by the *Conn-S* approach uses the *connections* between the retrieved CSI documents, specifically their shard membership, to guide the construction of the hierarchy. The pseudo-code for the Conn-S function is specified in Figure 6. This approach is similar to the Rank-S tree construction approach. The top ranked document is at the deepest node in the hierarchy. The next document in the CSI ranking is either attached at the same level as the previous node, or it prompts creation of another level depending upon whether it

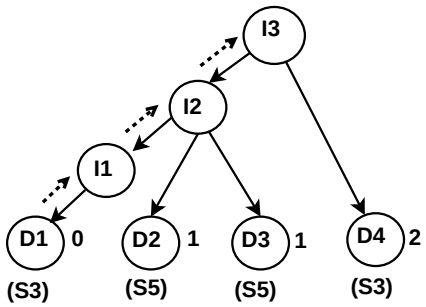


Figure 5.7: Connected SHiRE. CSI ranking: $D1, D2, D3, D4$. Dashed arrows represent the path followed by the bottom-up traversal starting at $D1$. Numbers besides the leaf nodes specify the order of revelation of the documents. Parent shards specified in the brackets.

Table 5.2: Selective search performance with different shard ranking algorithms. Dataset: GOV2. ▼ denotes significantly worse effectiveness than ReDDE ($p < 0.05$).

	CiS	CiD (million)	$\text{CiD}_{SHiRE} - \text{CiD}_{ReDDE}$	P@10	P@30	P@100	NDCG @100	MAP
Exhaustive	-	3.62	-	0.58	0.52	0.42	0.47	0.32
ReDDE (T=5)	5.0	1.29	-	0.58	0.52	0.42	0.47	0.32
SUSHI	3.4	0.65	-50%	▼0.51	▼0.41	▼0.32	▼0.35	▼0.22
Lex-S (B=3)	6.5	1.24	-4%	0.57	▼0.51	▼0.40	▼0.45	▼0.30
Rank-S (B=3)	3.4	0.81	-37%	0.58	0.52	0.41	▼0.46	▼0.31
Conn-S (B=3)	4.6	1.01	-22%	0.58	0.52	0.42	▼0.46	0.31

belongs to the same shard as the previous document. As per the Cluster Hypothesis, for topical shards, documents from the same shard are likely to be relevant to the same information need. Placing such documents at the same internal node provides them equal voting rights.

For the toy example (Figure 5.7), the Conn-S algorithm attaches documents $D1$ and $D2$ to different internal nodes since the documents originate from different shards. In contrast, $D2$ and $D3$ belong to the same shard, and thus $D3$ attaches to the same node as $D2$. Finally, $D4$ leads to a creation of another internal node because it belongs to a different shard ($S3$) than $D3$ ($S5$). This continues until every document in the CSI ranking is processed. A CSI ranking that does not place any pair of documents from the same shard in consecutive ranks results in a left-branching binary tree using this method. A shard ranking is estimated from the resulting hierarchy using the same approach as that used by Lex-S: bottom-up traversal with exponential decays at each level. Specifically, documents $D1$ and $D4$ assign votes of $V_{D1} \cdot B^{-0}$ and $V_{D4} \cdot B^{-2}$ to shard $S3$, respectively. Documents $D2$ and $D3$ assign votes of $V_{D2} \cdot B^{-1}$ and $V_{D3} \cdot B^{-1}$ to shard $S5$, respectively.

5.4 Experimental results: Search effectiveness and efficiency

For this analysis the GOV2 and CW09-B datasets were partitioned into 50 and 100 topical shards, respectively. Tables 5.2 and 5.3 report the search effectiveness and efficiency results for the two datasets. The SHiRE algorithms with dynamic shard rank cutoff estimation are compared with two previously proposed shard rankers: ReDDE and SUSHI, described in Section 2.3. An oracle setup was used for these experiments where several different parameterizations were evaluated and the setting that provided the best trade-off in terms of search effectiveness and processing cost was chosen for each algorithm. For ReDDE the parameter under consideration was the fixed shard rank cutoff (T) and for the SHiRE algorithms it was the base for the exponential decay function (B). Though not a realistic setup, this provides the upper bound on the performance of both, the baselines and the SHiRE algorithms. The chosen values for these parameters are provided in brackets in the first column of the tables. For the parameter B, we explored a range of integer values from 2 through 50. The sensitivity of SHiRE algorithms to different parameterizations is analyzed later, in Section 5.4.1. The space of fractional values larger than 1 is also valid for the parameter B. Although fractional values are not explored in this work, we realize on the hindsight, that it would have been an interesting range to study. The use of exponential decay function drives the votes computed using larger B values to near-zero very quickly, but smaller B values would slowdown this convergence. The effect of this on the search performance could provide useful insights for the SHiRE algorithms.

In our experiments the number of documents retrieved from the CSI for a query was in the range of 1700 to 2000 documents. The *transform* functions of the SHiRE algorithms organized these documents into an hierarchy.

The differences in the search accuracies of ReDDE and the proposed approaches were tested for statistical significance using the paired T-test ($p < 0.05$). As before the search efficiency is reported in terms of the two cost metrics, cost-in-documents (CiD), and cost-in-shards (CiS). In addition, each algorithm's CiD is compared with that of ReDDE.

When analyzing the search effectiveness of the SHiRE algorithms the results in Tables 5.2 and 5.3 suggest that the SHiRE algorithms have potential to provide comparable performance to that of ReDDE, especially at early ranks. Results for the Recall oriented metric, MAP, show that the Conn-S algorithm is the most successful one on this metric. In terms of efficiency the Rank-S algorithm offers the largest savings in the search cost for both the datasets. For the larger dataset the cost is cut in nearly half of that of ReDDE. The Lex-S algorithm offers the smallest savings in cost and struggles to perform as well as ReDDE. A trend that is common across the three SHiRE algorithms is that the savings in CiD over that of ReDDE's are bigger for the larger dataset. This is especially noticeable for the Lex-S and the Rank-S algorithms.

The number of top shards searched for ReDDE and Lex-S exhibit similar pattern – fewer shards are searched for the larger dataset. This explains in part the bigger reduction in CiD for

Table 5.3: Selective search performance with different shard ranking algorithms. Dataset: CW09-B.

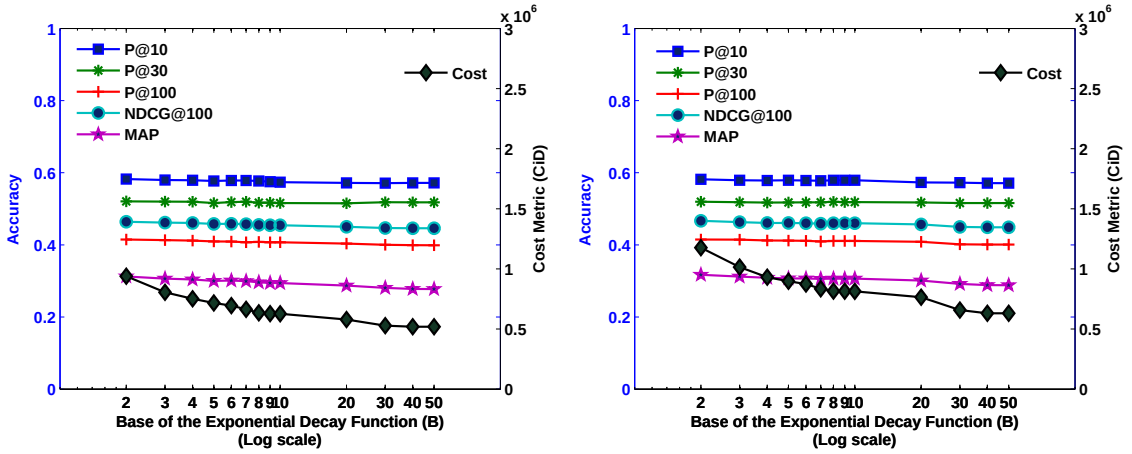
	CiS	CiD (million)	$CiD_{SHiRE-CiD_{ReDDE}}$	P@10	P@30	P@100	NDCG @100	MAP
Exhaustive	-	5.37	-	0.27	0.26	0.21	0.27	0.18
ReDDE (T=3)	3.0	0.68	-	0.29	0.28	0.20	0.27	0.17
SUSHI	5.3	0.54	-21%	0.28	0.27	0.19	0.25	0.16
Lex-S (B=20)	5.6	0.57	-16%	0.29	0.27	0.19	0.25	0.16
Rank-S (B=5)	3.6	0.36	-47%	0.29	0.28	0.20	0.27	0.17
Conn-S (B=5)	4.6	0.52	-24%	0.29	0.28	0.20	0.27	0.17

CW09-B. The predictions for the shard rank cutoff with Rank-S and the Conn-S are comparable across the two datasets in spite of the differences in dataset sizes and the total number of shards. Like CiD, the trends in CiS across the two datasets also indicate sub-linear scaling of selective search cost with dataset size.

For the baselines and all the SHiRE algorithms, the reductions in search cost over that of exhaustive search are bigger for the larger dataset. However, the reductions are much bigger for some of the SHiRE algorithms. For the Rank-S algorithm the savings are 78% and 93% for the GOV2 and CW09-B datasets, respectively. For the Conn-S algorithm the reductions are 72% and 90% for GOV2 and CW09-B, respectively.

When comparing the three SHiRE algorithms with each other we see that Lex-S is biased toward larger shard rank cutoff predictions (higher CiS). This is not surprising since the document hierarchies created by Lex-S are more flat than deep. As a result, during the bottom-up tree traversal, the votes accumulated at each level in the tree decay at a much slower rate. Consequentially many more shards are scored, and the predicted cutoff is higher. The search effectiveness results for the Lex-S algorithm in Tables demonstrate that in spite of its bias for higher CiS, it does not support a more effective selective search. This indicates that Lex-S performs poorly on the task of shard ranking. The Rank-S hierarchies are much more deeper than the other two algorithms. As a result, Rank-s exhibits a bias toward smaller predictions. The Conn-S trees are shorter than Rank-S but deeper than Lex-S. Thus Conn-S exhibits the least amount of bias of the three algorithms.

The shard rank cutoff estimator used by SUSHI optimizes its prediction for a particular precision metric (Section 2.3). The P@10 and P@30 values reported for SUSHI are thus from separate runs and the remaining values are an average over those two runs. For the GOV2 dataset SUSHI is unable to provide a good balance between search accuracy and cost for the smaller dataset. It is overly biased toward smaller shard rank cutoff which leads to lower search effectiveness. For the CW09-B dataset, although the search effectiveness results for SUSHI are



(a) Rank-S algorithm.

(b) Conn-S algorithm.

Figure 5.8: Sensitivity of Rank-S and Conn-S algorithms to parameter B (Base of the exponential decay function.). Dataset: GOV2.

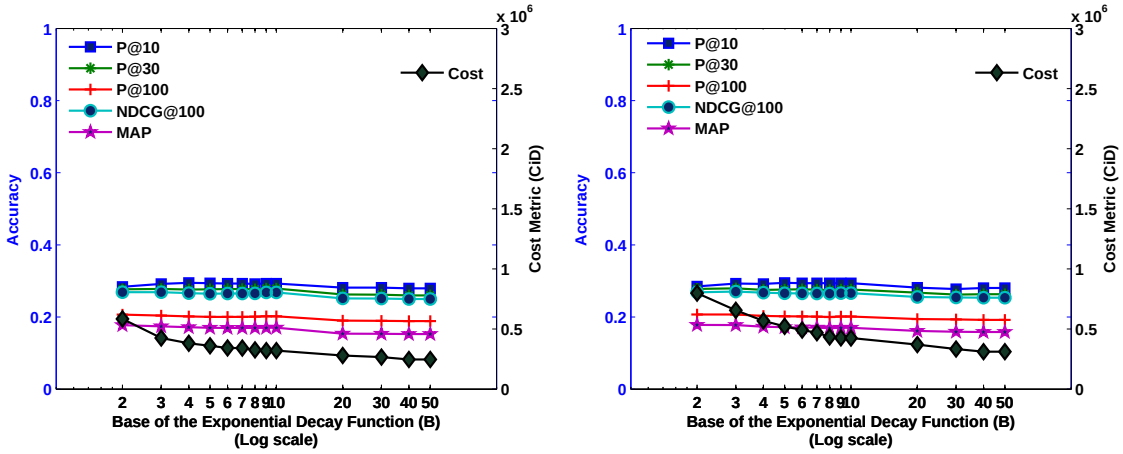
comparable to other approaches, its corresponding efficiency is lower than Rank-S and Conn-S. Overall SUSHI does not offer a stable performance across the datasets and as such is a weaker baseline than ReDDE. In the following sections we compare the SHiRE algorithms only with the ReDDE algorithm.

Overall, these experiments suggest that if competitive precision at early ranks is of interest then the Rank-S algorithm can provide the most cost effective solution. For search tasks where precision at deeper ranks is important the Conn-S algorithm can offer the best solution. Recall, that the computational cost of Rank-S and Conn-S is comparable to that of ReDDE. This further recommends the two shard ranking and cutoff estimation algorithms. Due to their better performance we only study the Rank-S and the Conn-S algorithms in the following sections.

5.4.1 Sensitivity of SHiRE algorithms to parameter B

We experimented with a range of base values (B) for the exponential decay function (Equation 5.2) in order to analyze its influence on the performance of the SHiRE algorithms. Figures 5.8 and 5.9 present the results for the Rank-S and Conn-S algorithms for the GOV2 and CW09-B dataset, respectively.

The plots show that the search effectiveness, as quantified by various metrics, is fairly stable over a range of B values for both the datasets. This trend is exhibited by both Rank-S and Conn-S algorithms. We see more variation in the values for the cost metric. A small base value leads to a slower decay which allows for higher search budget (in terms of shard rank cutoff). As a result, the corresponding search cost increases as the base value decreases.



(a) Rank-S algorithm.

(b) Conn-S algorithm.

Figure 5.9: Sensitivity of Rank-S and Conn-S algorithms to parameter B (Base of the exponential decay function.). Dataset: CW09-B.

Overall, we can conclude from these results that the SHiRE algorithms are not highly sensitive to the parameter B and that we see consistent trends that are intuitive.

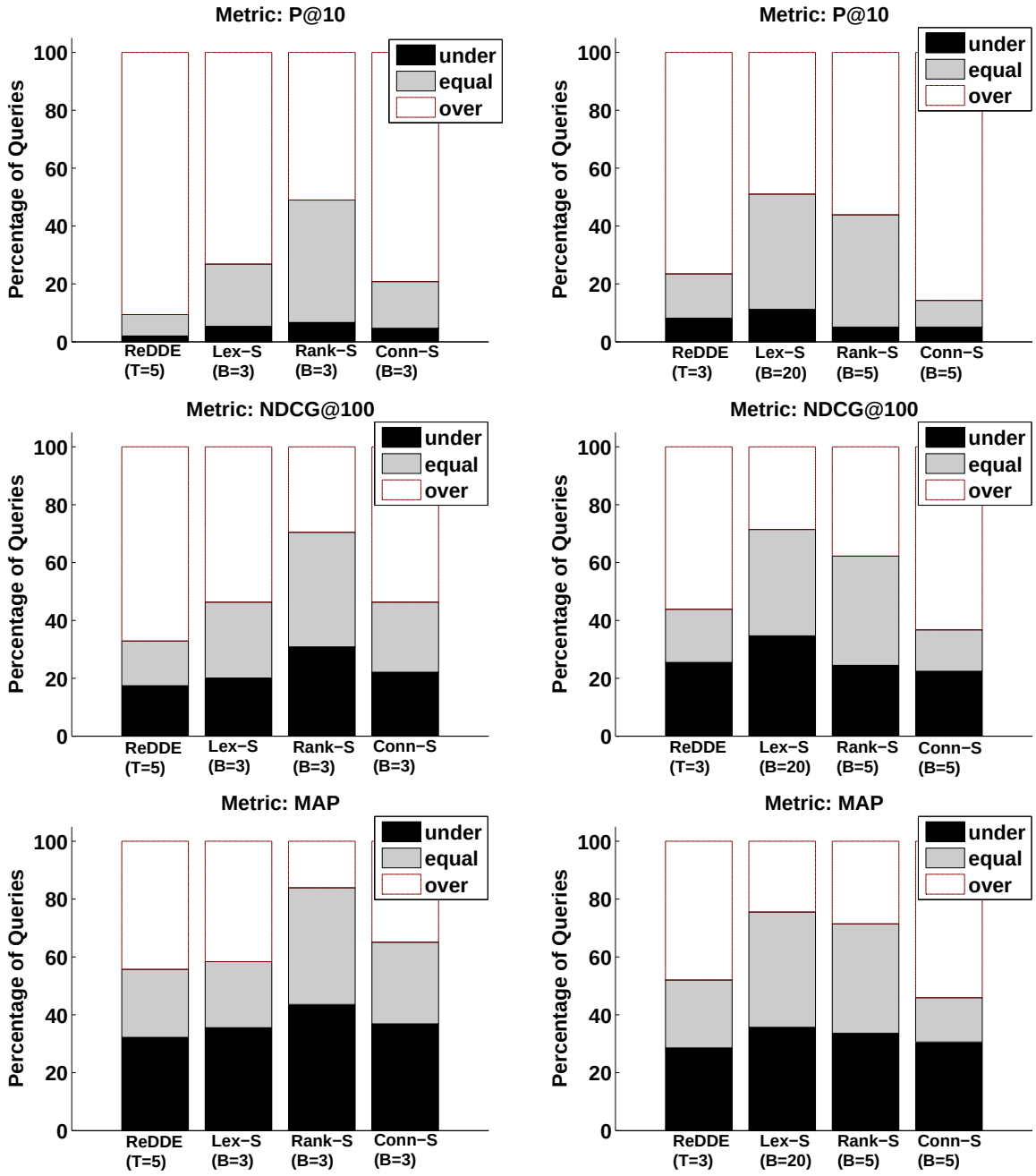
5.5 Experimental results: Shard rank cutoff estimation

In this section we analyze the effectiveness of the SHiRE algorithms at predicting the optimal shard rank cutoff for the query. The optimal cutoff is the earliest rank at which the selective search accuracy is equal or better than that of exhaustive search as measured by a particular evaluation metric. Notice that this definition of the optimal rank cutoff is metric-specific. As such, the optimal cutoff for P@10 could be different from that for MAP for the same query.

For this analysis we categorize the cutoff predictions into: under, equal and over estimates, based on comparison with the optimal rank cutoff. The equal category has a tolerance of ±1. An estimate that is off by +1 or -1 from the optimal value would be counted as an accurate estimate for this analysis. The under-estimation errors lead to ineffective search while over-estimation errors result into inefficient search.

Figure 5.10 provides the distribution of estimation errors for ReDDE and the three SHiRE algorithms for the GOV2 and CW09-B datasets, respectively. We analyze three metrics, P@10, NDCG@100, and MAP, which evaluate search effectiveness at increasingly deeper ranks. We see that the percentage of under-estimation errors increase for all the algorithms and both the datasets when search effectiveness at deeper ranks is evaluated.

When comparing the algorithms to each other, the Rank-S algorithm exhibits substantially larger percentage of *equal* predictions than the other algorithms for the GOV2 dataset. The



(a) Dataset: GOV2

(b) Dataset: CW09-B

Figure 5.10: Shard rank cutoff estimation errors.

Figure 5.11: Confusion matrix for shard rank cutoff estimation for Rank-S (B=3). Dataset: GOV2. Number of queries: 149.

(a) Metric: P@10. Cutoff averages: optimal=1.9, predicted=3.4.

		Predicted					
		1	2	3	4	5	>5
Optimal	1	14	22	24	23	9	8
	2	0	2	10	6	4	1
	3	0	0	3	7	0	1
	4	0	1	0	3	1	0
	5	0	1	0	1	0	0
	>5	1	0	1	1	5	0

(b) Metric: NDCG@100. Cutoff averages: optimal=4.9, predicted=3.4.

		Predicted					
		1	2	3	4	5	>5
Optimal	1	11	15	13	13	2	2
	2	0	5	8	5	6	0
	3	2	0	4	4	0	3
	4	0	0	0	4	1	0
	5	0	3	5	4	0	1
	>5	2	3	8	11	10	8

(c) Metric: MAP. Cutoff averages: optimal=6.7, predicted=3.4.

		Predicted					
		1	2	3	4	5	>5
Optimal	1	10	10	5	9	2	0
	2	1	4	9	6	0	0
	3	1	1	5	4	0	1
	4	0	1	3	3	1	0
	5	0	1	2	4	1	2
	>5	3	9	14	15	15	14

Table 5.4: Additional datasets.

Dataset	Size (uncompressed) (GB)	Number of Documents	Number of Words (million)	Vocabulary Size (million)	Average Document Length
TREC123-BySrc (100)	3.2	1,078,166	0.5	0.9	420
TREC4-Kmeans (100)	2.0	567,529	0.3	0.6	481

trend for the CW09-B dataset is less clear where the Lex-S and the Rank-S are on par in terms of *equal* estimates. For both the datasets and nearly all the three metrics, the percentage of over-estimation errors are larger than under-estimation errors for the Conn-S algorithm.

The above analysis provides a high-level validation of the cutoff estimator's efficacy. For a more thorough understanding we study the *magnitude* of the cutoff prediction errors for the Rank-S algorithm. The confusion matrices in Tables 5.11 and 5.12 present these values for the GOV2 and CW09-B datasets. We continue to use the three metrics from the previous analysis, P@10, NDCG@100, and MAP.

For all the metrics and both the datasets these confusion matrices are *right-heavy* around the diagonal. This reaffirms the observation from the previous analysis that over-estimation errors occur more frequently than the under-estimation errors. Another trend that is prevalent across the board is that smaller magnitude errors are more common than the larger magnitude errors – the off-diagonal cells with larger numbers are closer to the diagonal.

For the GOV2 dataset, the cutoff prediction is correct (with a tolerance of ±1) for 47%, 49%, and 56% of the queries for the P@10, NDCG@100, and MAP metrics, respectively. For the CW09-B dataset, the estimation is correct (with a tolerance of ±1) for 41%, 49%, and 53% of the queries for the P@10, NDCG@100, and MAP metrics, respectively. In contrast, the best fixed cutoff of 5 used by ReDDE for the GOV2 dataset is correct for 9%, 33%, and 56% of the queries for the P@10, NDCG@100, and MAP metrics, respectively. Similarly, the best fixed cutoff of 3 used by ReDDE for the CW09-B dataset is correct for 15%, 18%, and 23% of the queries for the P@10, NDCG@100, and MAP metrics, respectively.

Overall, the proposed query-specific rank cutoff estimators take a step in the right direction and offer improvements over the query-agnostic fixed cutoff approach. However, the above analysis also reveals the areas in which the estimator could be improved. A metric-specific estimator, in addition to query-specific, would be able to better model the distinct requirements of the different metrics.

Figure 5.12: Confusion matrix for shard rank cutoff estimation for Rank-S (B=5). Dataset: CW09-B. Number of queries: 98.

(a) Metric: P@10. Cutoff averages: optimal=2.9, predicted=3.6.

		Predicted					
		1	2	3	4	5	>5
Optimal	1	15	13	14	8	16	7
	2	0	1	2	2	2	3
	3	0	1	0	2	0	0
	4	0	0	1	1	0	2
	5	0	0	0	0	1	2
	>5	0	1	0	3	1	0

(b) Metric: NDCG@100. Cutoff averages: optimal=7.0, predicted=3.6.

		Predicted					
		1	2	3	4	5	>5
Optimal	1	15	9	9	3	6	5
	2	0	3	1	3	7	2
	3	0	1	1	0	0	0
	4	0	0	0	2	1	2
	5	0	0	2	1	1	0
	>5	0	3	4	7	5	10

(c) Metric: MAP. Cutoff averages: optimal=11.0, predicted=3.6.

		Predicted					
		1	2	3	4	5	>5
Optimal	1	14	10	7	2	5	4
	2	0	2	1	1	5	2
	3	0	1	1	1	1	0
	4	0	0	1	1	1	1
	5	0	0	2	1	1	1
	>5	1	3	5	10	7	12

Table 5.5: Query sets.

Dataset	Query Set	Average Query Length	Average Number of Relevant Documents per Query	Number of Relevance Levels
TREC123-BySrc	51-100	3.4	546 (± 375)	2
TREC4-Kmeans	201-250	9.1	130 (± 115)	2

Table 5.6: Selective search performance with different shard ranking algorithms. Dataset: TREC123-BySrc. ▼ denotes significantly worse effectiveness than ReDDE ($p < 0.05$).

	CiS	CiD (million)	$CiD_{SHiRE}-CiD_{ReDDE}$	P@10	P@30	P@100	MAP
Exhaustive	100	0.21	0.15	0.48	0.47	0.41	
ReDDE (T=25)	25	0.06	-	0.48	0.46	0.37	0.14
Lex-S (B=3)	26	0.05	-17%	0.52	0.46	0.35	0.13
Rank-S (B=2)	11	0.02	-67%	0.50	▼0.42	▼0.31	▼0.09
Conn-S (B=2)	12	0.03	-50%	0.50	0.44	▼0.32	▼0.10

5.6 Experimental results: Additional datasets

Although they are now considered small and outdated, TREC4-Kmeans and TREC123-BySrc are two of the most widely used datasets in distributed information retrieval research. Also since they were not created using the document allocation approaches proposed in this dissertation, they serve to test the sensitivity of selective search to the choice of partitioning technique. Xu and Croft [85] created the TREC4-Kmeans dataset by clustering the documents from the Text Research Collection, Volumes 2 and 3[6] into 100 clusters. French et al. [28] created the TREC123-BySrc dataset by dividing the Text Research Collection, Volumes 1, 2 and 3, into 100 partitions based on the source of the documents.

The dataset statistics are given in Table 5.4. The evaluation queries that were used with these datasets are summarized in Table 5.5. Unlike GOV2 and CW09-B the relevance judgments for the evaluation queries of TREC4-Kmeans and TREC123-BySrc contain only binary relevance scale. Thus we do not analyze NDCG@100 results for the TREC datasets.

The results for both datasets are given in Tables 5.6 and 5.7. As with the other two datasets, the differences in precision values of ReDDE and the SHiRE algorithms were tested for significance using the paired T-test ($p < 0.05$).

None of the shard ranking algorithms are able to support selective search that is competitive with exhaustive search at deeper ranks for either of the additional datasets. In contrast, selective

[6] http://trec.nist.gov/data/docs_eng.html

Table 5.7: Selective search performance with different shard ranking algorithms. Dataset: TREC4-Kmeans. ▼ denotes significantly worse effectiveness than ReDDE ($p < 0.05$).

	CiS	CiD (million)	$CiD_{SHiRE} - CiD_{ReDDE}$	P@10	P@30	P@100	MAP
Exhaustive	100	0.21	0.19	0.46	0.35	0.25	
ReDDE (T=20)	20	0.09	-	0.45	0.34	0.23	0.19
Lex-S (B=2)	22	0.08	-11%	0.46	0.34	0.22	0.16
Rank-S (B=2)	8	0.03	-67%	▼0.41	0.32	▼0.22	0.16
Conn-S (B=2)	9	0.03	-67%	▼0.41	0.32	▼0.22	0.17

search with the SHiRE algorithms provide higher search effectiveness than exhaustive search at early ranks for the TREC123-BySrc dataset, although, the improvement are not statistically significant.

When the three SHiRE algorithms are compared to ReDDE and with each other we notice another prominent deviation in these results from the earlier trends. The Lex-S algorithm provides competitive search accuracy for both the datasets. However, the Rank-S and the Conn-S algorithms do not provide a good balance between effectiveness and efficiency. The Rank-S and Conn-S algorithms predict much smaller cutoff values, as is evidenced by the CiS values, than the Lex-S algorithm. It is thus not a surprise that Rank-S and Conn-S offer large savings in search cost. This bias toward smaller cutoff predictions is a problem particularly for the dataset that is not partitioned into topical shards, like in case of TREC123-BySrc. The Lex-S algorithm on the other hand provides modest savings in search costs but offers search effectiveness that is comparable to that of the fixed cutoff approach for both the datasets.

Overall, these results indicate that the family of SHiRE algorithms together provide a search approach that has reasonable adaptability, and is cost-effective for different search needs.

5.7 Summary

This chapter studied the online phase of distributed selective search where each incoming query is processed at selected shards. The analysis of the first component of the query processing pipeline, query transformation, demonstrated that the richer query representation provides substantial improvements in search effectiveness for both exhaustive and selective search. This is especially true for the smaller datasets where the search effectiveness improves by 10% or more for nearly all of the metrics. For the larger datasets, the richer query representation offers fewer improvements in search effectiveness and most of these are not statistically significant.

This chapter also tested the sensitivity of selective search to a particular shard ranking algorithm. We chose two widely used resource ranking algorithms, CORI and ReDDE, that are quite

dissimilar to each other in terms of their design and inherent biases. The experimental results demonstrate that the distributed selective search provides competitive search effectiveness with both the shard ranking algorithm.

In addition to experimenting with existing shard ranking algorithms we also proposed a family of three SHiRE shard ranking algorithms: Lex-S, Rank-S and Conn-S. As a basis each of these algorithms employ a commonly used data structure, the *central sample index* (CSI) [69], to represent the shard contents. Running a query against the CSI yields a *flat* document ranking that the SHiRE algorithms transforms into a tree structure in order to encode additional information about the documents and the shards. A bottom-up traversal of the constructed hierarchy is used to infer shard ranking and an cutoff estimate by each algorithm. The presented algorithms are one of the few approaches to dynamically predict query-specific shard rank cutoffs. The joint formulation of the two inter-dependent problems of shard ranking and cutoff estimation is also one of the contributions of this work.

We tested the proposed algorithms on two large datasets that were both partitioned into topical shards. The experimental results suggest that the search effectiveness of the SHiRE algorithms can be on par with a strong baseline and also with exhaustive search, while the search cost is substantially lower than the baseline as well as exhaustive search. The Rank-S and the Conn-S algorithms are the most efficient search methods for large topically partitioned collections. They reduced the search cost by at least a quarter. The Rank-S algorithm also supports query-specific cutoff estimation that is at least twice as accurate as the best fixed cutoff value for topical shards.

Experiments with two supplementary datasets demonstrated that the conservative nature of the Rank-S and Conn-S estimators, which enables very efficient search, can be a detriment for smaller and topically less focused collections. However, Lex-S provides an attractive solution that is efficient and yet effective as compared to the baseline. Overall, the family of SHiRE algorithms offers cost-effective solutions for different search requirements.

Chapter 6

Search time analyses[1]

The experiments reported in the previous chapters of this dissertation made use of a simulated search setup. The efficiency of this experimentation model allowed us to explore a wide range of research problems studied as part of this thesis. However, the downside of a simulated setup is that it only provides an approximate indicator of query runtime. In order to address this limitation we study a full software implementation of distributed selective search in this chapter. In addition, we also study other search efficiency problems that focus on query runtime. We start the chapter by describing in detail the hardware and software platform used for the experiments.

6.1 Platform

For the experiments in this chapter we use the size-bounded topical shards created using the SB[2] *K*-means approach described in Section 4.4. For each shard a separate Indri index was constructed. The GOV2 dataset was spread across 208 index shards, and CW09-Eng was partitioned into 807 index shards.

Figure 6.1 provides a block diagram of the experimental platform used in this chapter. To provide distributed search Indri supports a standard client/server architecture. As is typical in this architecture, the query is received by a client process which broadcasts it to multiple servers processes, each of which searches a separate index shard. The search results from the server processes are returned to the client where they are merged and the final results are returned to the user. For our experiments we made use of this distributed search architecture, and the server processes were instances of the Indri Daemon application[2]. The memory footprint of an idle Indri Daemon is small, thus hundreds of daemon processes can be initiated on a single

[1]This work was done in collaboration with a fellow graduate student, Yubin Kim, and a summer intern, Jeevan Shankar, from International Institute of Information Technology, Hyderabad.

[2]http://sourceforge.net/p/lemur/wiki/The%20Indri%20Daemon/

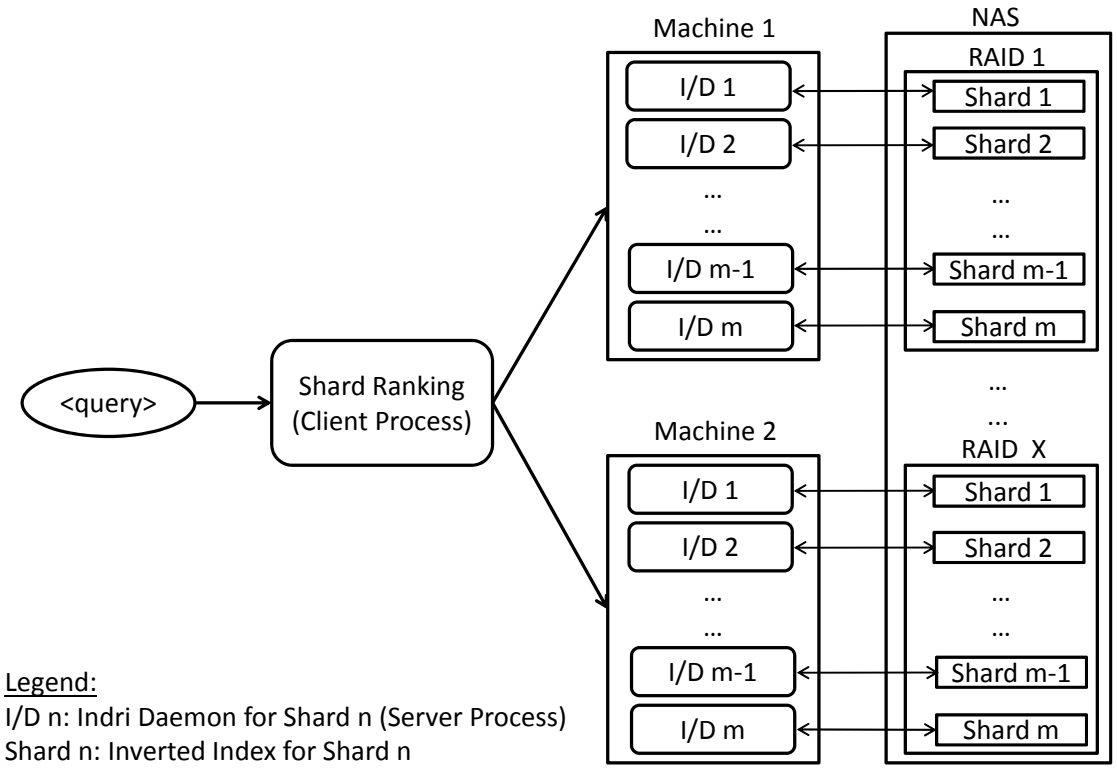


Figure 6.1: Block diagram of the platform used for the timing experiments.

machine. Since distributed selective search executes the query against only a small subset of the daemon processes the total memory requirements of this setup are not exorbitant.

The experiments were conducted on two machines, each with 8-core, 2.44 GHz CPU (Intel Xeon) and 16 GB of RAM. The total number of Indri daemon processes used for a dataset was divided evenly between the two machines. The shard indexes for both the datasets were stored on RAID partitions that are served by shared network-attached storage (NAS) units which are connected to the machines over a Gigabit network. This configuration is representative of computer hardware that one might find in an organization with modest computing resources.

For the selective search experiments we continue to use the parameter settings used in Section 4.4.5 where the results for the size-bounded shards were first reported. For the GOV2 dataset the top 20 shards are searched for each query, and the top 3 shards for CW09-Eng. As such, at any given time only 20 out of the 208 shard daemons are active for GOV2, and 3 out of the 807 daemons are active for CW09-Eng. In the best case, 10 daemon processes out of the top 20 selected for a query could be running on one machine and the remaining 10 would be on

the other machine in our setup. In the worst case, however, all the 20 daemon processes may reside on the same machine. In this case the cores in one machine would be over-utilized while the cores on the other machine idle. We do not address this problem in this thesis but identify *shard-machine assignment* as an important future direction for this work.

Since we are interested in comparing selective search with exhaustive search based on their query runtimes, it is important that both the systems are allocated same amount of computing resources. We use this requirement to guide the number of shards that each collection is divided into for distributed exhaustive search. The GOV2 collection is partitioned into 20 shards, and CW09-Eng is divided into 3 shards. A *dataset sequence order* based partitioning approach was used for both the datasets where the first 5% of the documents in the GOV2 collection were assigned to the first shard, the next 5% were allocated to the second shard, and so on. Indri's client/server architecture was used for exhaustive search as well. Specifically, each shard was searched by a dedicated Indri daemon process, and the total number of daemons was divided evenly across the two machines. This setup for distributed exhaustive search is also similar to the approach commonly chosen when working with large collections in a low-resource environments [9, 36, 71].

Ideally one would want to store each index shard on a separate RAID partition in order to facilitate a completely distributed and parallel search process with minimum resource contentions. However, this is rarely possible, especially in low-resource environments. As such, the 208 index shards for the smaller dataset, GOV2, were all stored on a single RAID partition in our experiments. For the larger CW09-Eng collection, its 807 index shards were spread across four RAID partitions. The GOV2 setup is representative of search environments that are constrained on processing resources as well as storage resources, while the CW09-Eng setup represents environments that have modest storage resources but are more constrained in processing power.

A single-threaded query processing approach was employed for the baseline as well as the experimental approach where the queries were run in serial rather than in parallel. This is appropriate since in the earlier part of this chapter we compare the runtimes of the approaches and not throughput. Also, a single-threaded setup affords a cleaner and simpler setup for the analysis of each individual query. The consequence is that in some experimental configurations all of the available 16 CPU cores are not utilized. However, this under-utilization applies equally to selective search and exhaustive search. We also present a throughput experiment that makes better use of all the available processing cores in Section 6.7.

The previous chapter argued for a query-specific shard rank cutoff which specifies the number of top ranked shards to be searched for the query when doing selective search. However, for the analysis presented in this chapter we used the ReDDE algorithm with a query-independent fixed rank cutoff because it afforded better control of the computing resources allocated for each query. This control is especially important for the experiments presented in this chapter since

Table 6.1: Storage and computational costs for shard creation. The reported wall-clock time were estimated for 16-processor machine.

	GOV2		CW09-Eng	
	Random shards	Topic-based shards	Random shards	Topic-based shards
SB^2 *K*-means (Wall-clock time)	-	2.4 hrs (208 shards)	-	181.0 hrs (807 shards)
Shard index construction (Wall-clock time)	1.6 hrs (16 shards)	10.7 hrs (208 shards)	32.9 hrs (16 shards)	57.3 hrs (807 shards)
CSI construction (Wall-clock time)	-	0.7 hrs	-	10.4 hrs
Disk space	217 GB	212 GB	5513 GB	5310 GB
Disk space for CSI (4%)	-	4.8 GB	-	112 GB

we compare the runtimes of the baseline and the experimental approach.

6.2 Storage and computational costs of random and topic-based shards

Before presenting the runtime results in the next section, we summarize the storage and computational costs that are associated with creating index shards for exhaustive search and selective search. Table 6.1 reports the time required to partition the document collection into topic-based shards using SB^2 *K*-means algorithm, and the time to transform the shards (random, and topic-based) into inverted indexes using Indri. When analyzing these numbers it is important to note that these experiments were performed on a shared computing cluster. As a result, the runtimes of these experiments have been influenced by other computing tasks executing on the cluster that time. The timing results in Table 6.1 should thus be interpreted as only being suggestive of the true trends.

The shared computing cluster that was used to perform the experiments contains 150+ processing cores that are managed by Condor [73]. The original experiments exploited as much processing power as possible by scheduling multiple jobs concurrently. The wall-clock time for those experiments was used to estimate the wall-clock time these experiments would take on a 16 core setup. This was done to facilitate a consistent use of fixed number of processing cores (at most 16) throughout this chapter. This also provides a fair comparison between the baseline and the experimental approach since both use the same number of processing cores.

The wall-clock time needed to cluster the document collections into topic-base shards represent the overhead cost of selective search. The SB^2 *K*-means algorithm takes 2.4 hrs to cluster GOV2 (25 million document collection) into 208 shards. The wall-clock time needed to cluster

CW09-Eng (500 million documents) into 807 topical shards is about 75 times longer than GOV2. This large difference in the wall-clock times for these datasets is caused by two factors: the difference in collection size, and the difference in the number of shards created. The CW09-Eng collection is 20 times larger than GOV2, and the number of shards that CW09-Eng is divided into is about 4 times larger than that for GOV2.

The inverted indexes for all the shards as well as the central sample index (CSI) were constructed using Indri. The wall-clock time for creating inverted indexes in parallel using 16 processing cores are reported in Table 6.1. As we would expect, the larger number of topic-based shards take longer to get transformed into inverted indexes than the fewer number of random shards. The CSI construction time is also an overhead cost for selective search. It is clear from these results that the indexing time for selective search (topical shards and CSI) will be longer than the indexing time for exhaustive search. However, the exact relation between the two indexing times cannot be concluded from these numbers. This is so because the index creation jobs were not run in a controlled environment.

The storage costs reported in Table 6.1 indicate that the selective search indexes (topical shard indexes and CSI) need slightly less disk space than the exhaustive search indexes (random shards). There are two main reasons for this trend. First, the topical shards cause less duplication, and second, the topical shards compress better than the random shards. For topic-based shards, most occurrences of a given term are concentrated into a few shards while in case of random shards they are uniformly distributed across all the shards. As a result, the book-keeping information that is needed for each term (for example, a term dictionary entry) has to be maintained in nearly all the indexes for random shards. For topical shards, only a few indexes duplicate such information for many terms. The topical shards compress better because of the topical homogeneity of the shard contents. Often, the topically focused terms in a shard occur in nearly all of the shard documents. The posting lists for such terms when compressed using delta encoding offer high compression ratio since the resulting deltas are small. The documents within a random shard index, on the other hand, lack topical homogeneity and thus cannot offer small deltas and better compression.

6.3 Query runtime for Exhaustive Search and Selective Search

Armed with an actual implementation of distributed selective search we revisit the objectives about search efficiency laid out in Section 3.2. Specifically, we test whether distributed selective search offers substantial speed up in query runtime over distributed exhaustive search when provided with the same amount of computing resources. The experimental results are reported in Table 6.2. The runtime values reported throughout this chapter are cumulative over all the queries. Search efficiency as measured by cost-in-documents and cost-in-shards metrics is also reproduced in Table 6.2 for completeness.

Table 6.2: Query runtime and search cost results for distributed exhaustive search and distributed selective search.

		Runtime (secs)	Gain	CiD (million)	Gain	CiS
GOV2	Exhaustive	760	-	3.63	-	20 (/20)
(150 Queries)	Selective	603	21%	0.66	82%	20 (/208)
CW09-Eng	Exhaustive	8452	-	51.29	-	3 (/3)
(100 Queries)	Selective	1570	81%	2.71	95%	3 (/807)

The total query runtime with selective search is substantially shorter than with exhaustive search for both the datasets. We also observe the reoccurring trend that the improvement is larger for the bigger dataset. However, an important difference in the datasets, other than the difference in their sizes, is the parallelization opportunity offered to the baseline approach. For GOV2 the exhaustive search proceeds simultaneously on 20 shards, whereas for CW09-Eng it is spread out on only 3 processes.

The results in Table 6.2 also show that the improvements in query runtime do not track the improvements in CiD well, especially for GOV2. One of the main reasons for this is that the two metrics, query runtime and CiD are not comparable. The former is dominated by the single longest running shard searched for the query, while the latter measures the cumulative search effort expended for the query at each of the searched shards. It is thus not surprising that the difference in their respective improvements widens as the search becomes more distributed, like it does for GOV2.

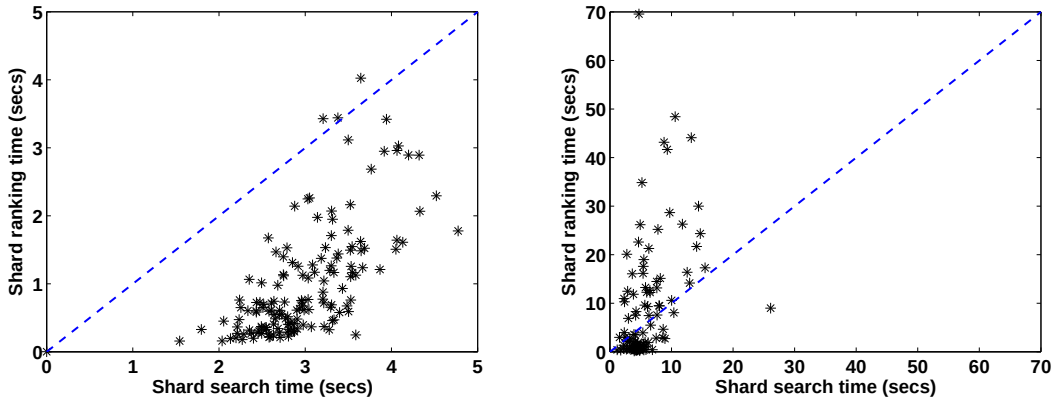
For GOV2 the average memory usage when processing a query using distributed exhaustive search was 0.33GB, while for selective search it was 3.8GB. For the larger dataset, CW09-Eng, the average memory footprint of exhaustive search less than 1GB and for selective it was about 14GB. For both datasets, the memory usage for selective search is about an order of magnitude higher than that for exhaustive search.

Overall, the runtime results ascertain that selective search meets one of its important objectives, *shorter query runtime*. The next section provides a deeper analysis of selective search's runtime.

6.3.1 Shard ranking time versus shard search time

Every query proceeds through two distinct stages during selective search: shard ranking, and shard search. The division of query runtime into these two phases is reported in the scatter plots in Figures 6.2a and 6.2b for GOV2 and CW09-Eng datasets, respectively.

These plots exhibit very different trends for the two datasets. Many queries spend more time in the shard search phase than in shard ranking for the GOV2 dataset, whereas for CW09-Eng



(a) Dataset: GOV2. Avg Shard Ranking Time: 1 secs ±0.8, Avg Shard Search Time: 3 secs ±0.8

(b) Dataset: CW09-Eng. Avg Shard Ranking Time: 10 secs ±13, Avg Shard Search Time: 6 secs ±4

Figure 6.2: Shard ranking time versus shard search time for distributed selective search with ReDDE, CSI=4%.

it is exactly the opposite. We believe that this difference is caused by a combination of factors. Recall that the shard storage configurations for the two datasets are quite different. For GOV2 all the shards are stored on a single RAID partition while the CW09-Eng shards are spread out on four RAID partitions. Furthermore, selective search processes each query against the top 20 shards for GOV2 as opposed to the top 3 shards for CW09-Eng. As a result, disk contention during the shard search phase may be much higher for GOV2 than for CW09-Eng. It is thus not surprising that for the GOV2 dataset the shard search phase takes longer than the shard ranking phase where the shard ranker process is the sole user of disk I/O.

For the CW09-Eng dataset, however, the I/O wait time due to disk contention could be minimal because only the top 3 shards are searched for each query and it is likely that the selected index shards might reside on different RAID partitions. As a result, the query runtime for this dataset is instead influenced by the sizes of the inverted indexes used by the shard ranking and the shard search phases. In our experiments the central sample index (CSI) used by the shard ranking algorithm consists of 4% of the collection documents. Since the CW09-Eng dataset is divided into 807 shards, each index shard contains an average of 0.12% of the collection documents. Given the difference in their sizes it is not surprising that running the query against the CSI index takes longer than searching the shards in parallel for the CW09-Eng dataset.

Ideally, one would want the overhead of shard ranking to be negligible and constant time for selective search. However, shard ranking accounts for about 22% of the average query runtime for GOV2, and 44% for CW09-Eng. Also, nearly half of the queries spend more time in the shard ranking phase than in shard searching for CW09-Eng. This suggests that if the cost of

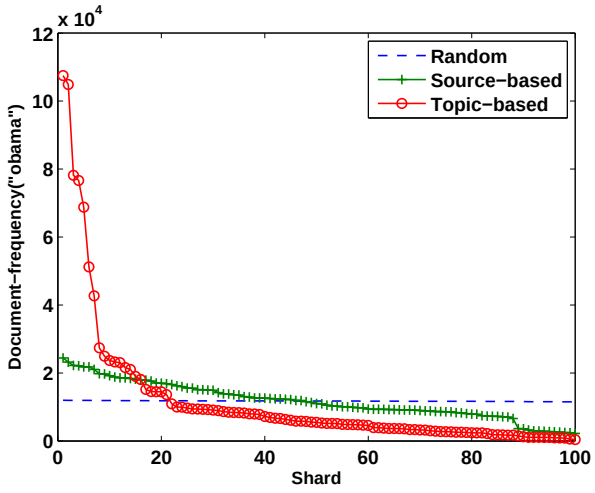


Figure 6.3: Distribution of the term *obama* across shards. Dataset: CW09-B.

the shard ranking step is reduced then the efficiency of selective search could improve further. In the following section we analyze one simple approach toward achieving this goal.

6.4 Shard ranking: CSI Size

For sample-based resource ranking algorithms such as ReDDE, the size of the centralized sample index (CSI), is an important parameter that influences the algorithm's accuracy as well as efficiency. Here the CSI size is measured in terms of number of documents sampled from each resource (shard). Prior work in federated search extensively investigated the effect of CSI size on search effectiveness [15, 66]. However, it is not clear whether the findings from this research can be directly applied to selective search for two reasons. First, the *efficiency* of the resource ranking algorithms was rarely studied. While a larger CSI might improve the accuracy of the resource ranking algorithm, it would likely degrade the efficiency of the algorithm. Since previous work primarily focused on the accuracy of the resource ranking algorithms, a tradeoff analysis that compares the *cost* and the *value* of using larger CSI is not available.

Secondly, the resources used in previous work are different from the shards used in this work in terms of their makeup. Certain statistical properties of the topical shards are markedly different from those of the resources used in the past research. For topical shards, the distribution of content terms across shards is often highly skewed, while for most of the types of resources used in previous work this distribution would be uniform or nearly-uniform. Figure 6.3 (reproduced from Chapter 5) illustrates this point using the term *obama*. The skewed distribution of terms in case of topical shards is caused by the lexical similarity based partitioning approach used to create the topical shards. We conjecture that the homogeneous nature of topical shards can be exploited to reduce the CSI size without hurting the accuracy of the resource ranking

Table 6.3: Effect of CSI size on selective search effectiveness. Dataset: GOV2. ▼ denotes significantly worse effectiveness than with 4% CSI ($p < 0.05$).

CSI size (%)	P@10	P@30	P@100	NDCG@100	MAP
4.00	0.53	0.48	0.37	0.41	0.27
2.00	0.52	0.48	0.37	0.41	▼0.26
1.00	0.53	0.47	0.36	▼0.40	▼0.25
0.50	0.53	0.47	▼0.36	▼0.40	▼0.25
0.25	0.52	▼0.44	▼0.34	▼0.38	▼0.23
0.10	▼0.51	▼0.43	▼0.33	▼0.36	▼0.21
0.05	▼0.49	▼0.41	▼0.31	▼0.34	▼0.19

Table 6.4: Effect of CSI size on selective search effectiveness. Dataset: CW09-Eng. ▼ denotes significantly worse effectiveness than with 4% CSI ($p < 0.05$).

CSI size (%)	P@10	P@30	P@100	NDCG@100	MAP
4.00	0.15	0.13	0.12	0.12	0.06
2.00	0.13	0.13	0.12	0.11	0.06
1.00	0.13	0.13	0.12	0.11	0.06
0.50	0.13	0.12	0.12	0.11	0.06
0.25	0.13	0.12	0.12	0.11	0.05
0.10	0.14	0.13	▼0.11	▼0.10	▼0.05
0.05	▼0.10	0.11	▼0.09	▼0.08	▼0.04

algorithm. We test this hypothesis in this section.

We experimented with a range of CSI sizes and report the corresponding search effectiveness results in Tables 6.3 and 6.4 for GOV2 and CW90-Eng, respectively. For both datasets, the CSI size does impact the final search effectiveness. However, the rate at which the search effectiveness degrades is slow, especially for the larger dataset and at early ranks (P@10, P@30 and P@100). These results indicate that fewer documents are sufficient to provide a reliable representation of the shard contents in case of topical shards.

We chose the CSI size of 0.5% for the runtime experiments presented next because we expect it to provide a good balance between efficiency and effectiveness. For both datasets it supports search effectiveness that is comparable to that of the 4% CSI for most metrics. The results for the search runtime experiments are presented in Table 6.5. As we would expect the smaller CSI provides shorter query runtime than exhaustive search as well as selective search with 4% CSI. The 0.5% CSI is about 88% smaller in size than the 4% CSI, however, the runtime improvements over the 4% CSI are much smaller in these results. In order to understand this discrepancy it is necessary to analyze the shard ranking time and shard search time separately.

Table 6.5: Query runtime for ReDDE-based selective search with different CSI sizes.

Dataset	Search Method	CSI size	Runtime (secs)	Gain over Exh	Gain over 4.0% CSI
GOV2 (150 Queries)	Exh (S=20)	-	760	-	-
	Sel (T=20 /208)	4.0%	603	21%	-
	Sel (T=20 /208)	0.5%	512	33%	15%
CW09-Eng (100 Queries)	Exh (S=3)	-	8,452	-	-
	Sel (T=3 /807)	4.0%	1,570	81%	-
	Sel (T=3 /807)	0.5%	856	90%	45%

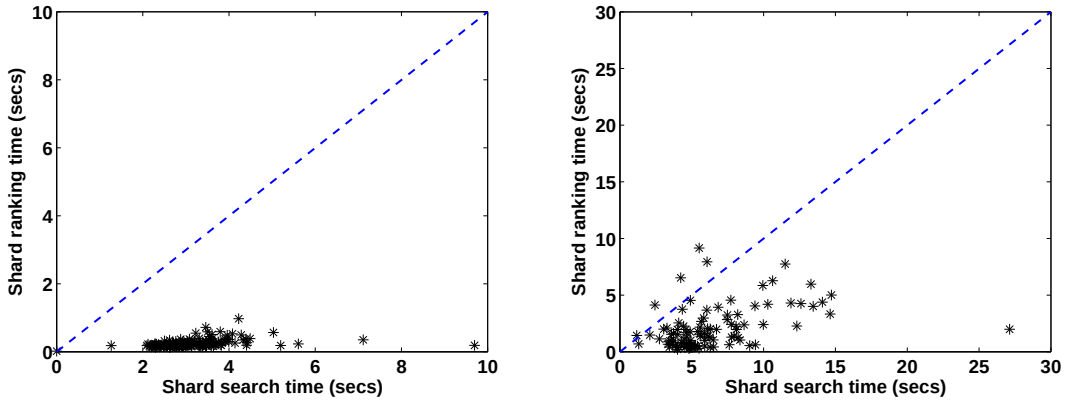
The scatter plots in Figure 6.4 provide these details. Notice that the average shard ranking time with the 0.5% CSI (specified in the label of scatter plots) as compared with that of the 4% CSI (1 sec for GOV2 and 10 secs for CW09-Eng) is 75% shorter for GOV2, and 78% for CW09-Eng. These improvements in shard ranking time are comparable to the reduction in CSI size (88%). It is not surprising that the efficiency gains due to smaller CSI are localized to the shard ranking phase.

In these scatter plots nearly all of the queries spend less time in the shard ranking phase than in the shard search phase. For GOV2, the shard ranking time is about 8% of the total query runtime, and for CW09-Eng it is 23%. If we contrast these numbers with those for 4% CSI, where shard ranking time is 22% of the total query runtime for GOV2 and 44% for CW09-Eng, then the reduction in the shard ranking overhead becomes apparent. Overall, these results validate our belief that smaller CSI can support efficient and accurate search when working with topic-based shards.

6.5 Effect of query optimization

The analysis undertaken in this section is motivated by two observations. First, large-scale search systems often employ query optimization techniques to speed up the processing of queries (also referred to as *top k retrieval*, and *dynamic index pruning*). Since one of the objectives of the search approach proposed in this thesis is to improve query processing efficiency, it behooves us to compare selective search with an optimized exhaustive search.

Secondly, we conjecture that selective search and query optimization techniques are complementary, and can be applied together to improve search efficiency further. We are interested in testing this hypothesis because selective search and query optimization methods leverage different properties of large-scale search in order to improve efficiency. Selective search uses the Cluster Hypothesis to reduce the number of evaluated documents, while query optimization identifies the documents that would not be ranked in the top k positions and eliminates them



(a) Dataset: GOV2. Avg Shard Ranking Time: 0.25 secs ±0.12, Avg Shard Search Time: 3 secs ±0.9.

(b) Dataset: CW09-Eng. Avg Shard Ranking Time: 2 secs ±2, Avg Shard Search Time: 6.5 secs ±4.

Figure 6.4: Shard ranking time versus shard search time for distributed selective search with ReDDE, CSI=0.5%.

Table 6.6: Query runtime for exhaustive search and selective search with and without query optimization.

		Optimization=Off		Optimization=On		
		Runtime (secs)	Gain over Exh	Runtime (secs)	Gain over Exhaustive	Gain over Opt=Off
GOV2	Exh (S=20)	760	-	698	-	8%
(150 Queries)	Sel (T=20 /208)	512	33%	488	30%	5%
CW09-Eng	Exh (S=3)	8,452	-	5,194	-	39%
(150 Queries)	Sel (T=3 /807)	856	90%	728	86%	15%

from the candidate set.

Also, the topic-based shards create longer posting lists for the *on-topic* terms in each shard (for example, refer to Figure 6.3). Many query optimization techniques are known to be more effective when the posting lists for the query terms are long. We study if one such query optimization technique is able to exploit this property of topical shards to its advantage. We use the term-bounded max-score (TBMS) query optimization technique which is implemented in Indri for our experiments [72]. The TBMS technique (described in detail in Section 2.1.2) is among the state-of-the-art approaches that prioritize the documents evaluated for a query and then skip over documents that are unlikely to rank high enough in the final results list for the query.

The results for exhaustive and selective search, with and without optimization, are sum-

marized in Table 6.6. If we compare selective search and TBMS as two different techniques for improving search efficiency then these results suggest two trends that are consistent across the datasets. First, TBMS improves efficiency of exhaustive search (8% for GOV2 and 39% for CW09-Eng), but these gains are substantially smaller than those with selective search (33% for GOV2 and 90% for CW09-Eng). Second, both the techniques, TBMS and selective search, are more effective for the larger dataset.

The difference in the performance of TBMS and selective search is not surprising. Most optimization techniques primarily reduce only the computations needed for query processing, while selective search reduces both I/O and computations. The ability to reduce the amount of data transferred from the disk reduces latency substantially because often that is the slowest component of query processing.

The results in Table 6.6 support the hypothesis that selective search and query optimization are complementary techniques for improving search efficiency. TBMS is able to offer substantial speed up in selective search runtime for both datasets (5% for GOV2 and 15% for CW09-Eng). This suggests that TBMS is able to identify documents in the selected topical shards that are unlikely to be ranked highly for the query.

These results also compare exhaustive search and selective search in two different settings, with and without query optimization. The improvements over exhaustive search offered by selective search when query optimization is not employed (33% for GOV2 and 90% for CW09-Eng) are only slightly larger than those when TBMS is applied for both search approaches (30% for GOV2 and 86% for CW09-Eng). The consistent performance of selective search across the two configurations indicates that, one, it is a stable search approach, and two, that TBMS and selective search are leveraging different properties of large-scale search.

Finally, these results show that even an *unoptimized* selective search is faster than an *optimized* exhaustive search. For GOV2 and CW09-Eng, the runtime for unoptimized selective search (512 secs for GOV2 and 856 secs for CW09-Eng) is 27% and 84% shorter than the runtime for optimized exhaustive search (698 secs for GOV2 and 5194 secs for CW09-Eng).

In summary, this analysis indicates that like exhaustive search, the selective search approach also benefits from query optimization techniques. However, the improvements in query runtime achieved by using selective search are substantially larger than those offered by query optimization alone.

6.6 Effect of query length

Among other factors, *query length* strongly influences the runtime of the query. Techniques used to improve search efficiency are also known to be impacted by query length. For instance, Broder et al., 2003, report higher efficiency gains for longer queries (7-word) than shorter queries (2.5-word) with their proposed query optimization approach [10]. The strong correlation between

Table 6.7: Query run-time for exhaustive search and selective search on sets of queries with different lengths.

		1 word		3 words		5 words		10 words	
		Time (secs)	Gain	Time (secs)	Gain	Time (secs)	Gain	Time (secs)	Gain
GOV2	Exh (S=20)	111	-	270	-	343	-	364	-
	Sel (T=20 /208)	70	37%	176	35%	193	44%	204	44%
CW09-Eng	Exh (S=3)	747	-	6,639	-	12,013	-	15,618	-
	Sel (T=3 /807)	231	69%	544	92%	705	94%	892	94%

query length and runtime motivates us to investigate the impact of query length on the efficiency of distributed selective search.

We experiment with a range of query lengths, specifically, 1, 3, 5, and 10 word. For each length a set of 50 queries was selected for the CW09-Eng dataset from the TREC Million Query Track query logs [20]. However, 10 word queries were rare in this query log, thus some queries had to be supplemented from the AOL query log. For GOV2 all the queries were sampled from the AOL query log. For these experiments, CSI of size 0.5% was used, and the TBMS optimization was not employed.

The runtime results for the query sets are provided in Table 6.7. We see substantial improvements in query runtime over the baseline for both the datasets and across all the query lengths. The speed up is larger for the longer queries. This is consistent with prior work. We see a plateauing effect among the two sets of long queries (5 and 10 word) for both the datasets.

Recall that the queries in the evaluation sets are on average 3.1 and 2.1 word long for GOV2 and CW09-Eng, respectively (Table 3.2). If we compare the evaluation queries with the 3-word queries in this section then we see comparable improvements in runtime with selective search for both datasets. Specifically, selective search with 0.5% CSI is 33% and 90% faster than exhaustive search for the evaluation queries in Table 6.5. The result table for this section (Table 6.7) shows 35% and 92% gain in runtime with selective search of 3-word queries.

Overall, these results demonstrate that selective search is more efficient for longer queries but provides substantial speed up for shorter queries as well.

6.7 Throughput analysis

The goal of this section to compare exhaustive and selective search based on their throughput performance. We are interested in analyzing the amount of time that each search approach needs to process a given number of queries under different degrees of parallelism. Specific details about the experimental setup are described next.

6.7.1 Experimental methodology

For this analysis we sampled 1000 queries from the AOL .gov query log for the GOV2 dataset, and 999 queries[3] from the TREC Million Query Track query logs [20] for the experiments with CW09-Eng dataset. The stopwords were filtered out from all the queries. The average query length after stopword removal was 3.4 terms for GOV2 queries and 2.3 terms for CW09-Eng. For the exhaustive search experiments both the datasets were partitioned into 16 random shards each. The Indri indexes for the 16 shards were uniformly spread across 4 RAID partitions. The topical shard indexes used for the selective search experiments were likewise spread uniformly across the same 4 RAID partitions. There were 208 topical shards for GOV2 and 807 for CW09-Eng.

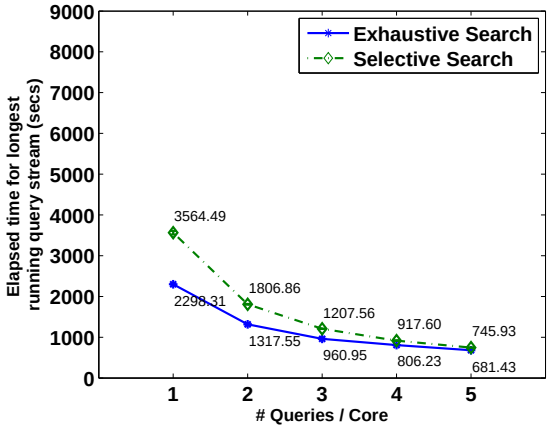
The same two machines used previously were also used for the experiments reported in this section. Recall that each of these two machine is a 8-core, 2.44 GHz CPU (Intel Xeon) and contains 16 GB RAM. The total number of cores (16) motivates the 16-way partitioning of the collection for the exhaustive search approach. The central sample index (CSI) needed for selective search was stored on the local disks of the two machine. One copy of the CSI index was used at most by two query streams on a machine. In an experimental setup, for example, that used 3 query streams for selective search, a total of 2 copies of CSI were used, while for 10 query stream configuration 5 copies of the CSI were used. The CSI size of 0.5% of the collection documents was used for both the datasets.

We used as few shared computing resources as possible in order to control the environment in which these experiments were conducted. However, some amount of sharing could not be avoided. The NAS controller and the network were used by other users too. In order to account for this partially-controlled environment we performed 5 runs of each configuration. The results reported in this section are averages (and standard deviations) over the 5 runs. Before starting each run the local and server caches were cleared to provide a cold start for each experiment.

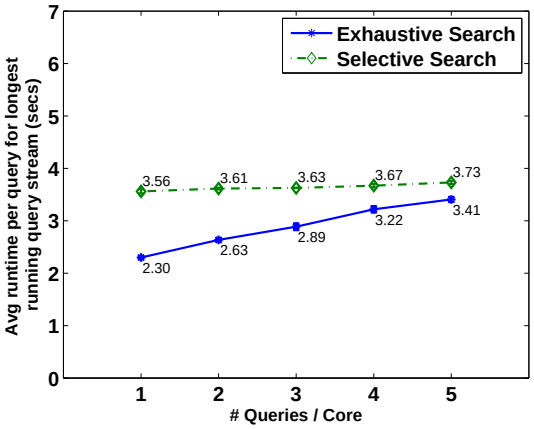
Exhaustive search executes each query against all the 16 random shards. Like in previous experiments, selective search executes each query against the top 20 topical shards for GOV2, and against top 3 shards for CW09-Eng. As a result, a selective search experiment that processes a single query stream, is able to make use of all the available cores for GOV2, but for CW09-Eng only 3 out of the 16 cores are utilized. We thus initiate 5 query streams at once for selective search with CW09-Eng which then uses 15 out of the 16 cores. This provides a setup where the baseline and the experimental method, both, utilize most of the available computing power.

In order to achieve the maximum possible utilization of the processing cores and to test its effect on the search approaches, we experimented with different degrees of parallelism. Specifically, we experiment with different number of parallel query streams where the total number of

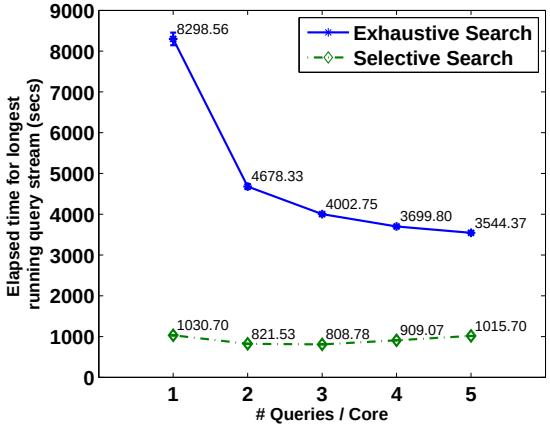
[3]We started with 1000 queries, however, after removing the stopwords one of the queries reduced to an empty set. Thus the unorthodox number of 999 queries.



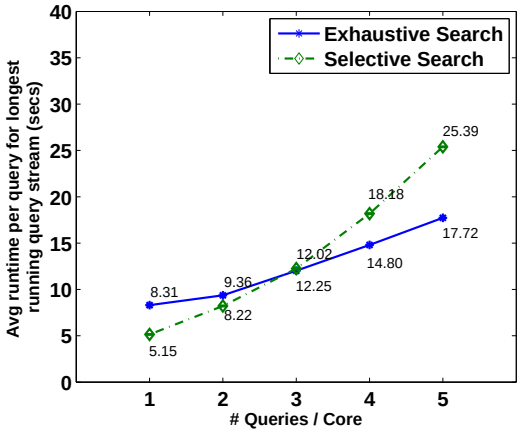
(a) Elapsed time results. Dataset: GOV2.



(b) Average query runtime results. Dataset: GOV2.



(c) Elapsed time results. Dataset: CW09-Eng.



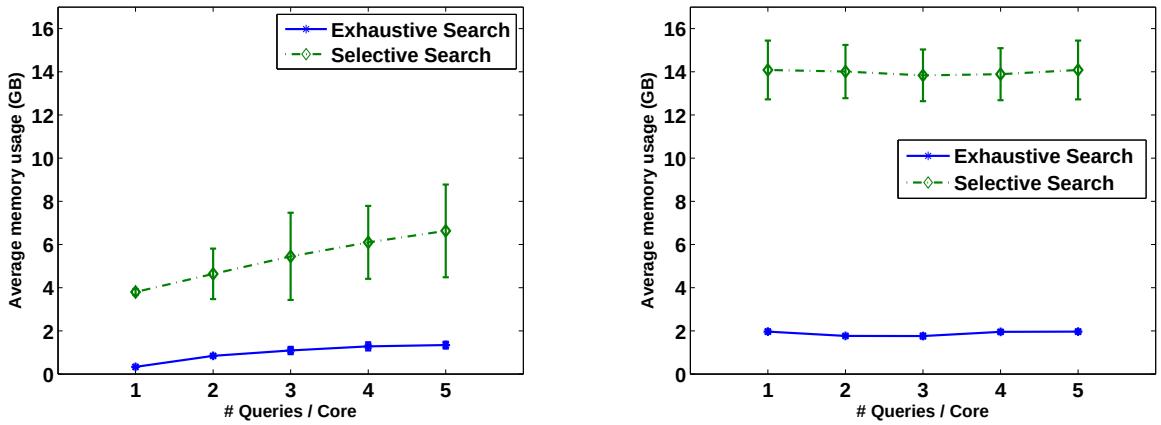
(d) Average query runtime results. Dataset: CW09-Eng.

Figure 6.5: Throughput and Query latency timing results for different degrees of parallelism.

queries is held fixed. The goal is to vary the average number of queries executed per processing core, while holding the total query load fixed. For exhaustive search we experimented with 1 through 5 parallel query streams. The corresponding selective search configurations employed 1 through 5 parallel query streams for GOV2, and 5 through 25 parallel query streams for CW09-Eng.

6.7.2 Experimental results

The results for the GOV2 and the CW09-Eng datasets are summarized in the Figure 6.5. Since the time to run a batch of queries is dominated by the longest running query thread, the Y-axis reports the timing numbers for the query stream that ran longest. The aggregate runtime for



(a) Dataset: GOV2.

(b) Dataset: CW09-Eng.

Figure 6.6: Average memory usage for exhaustive search and selective search.

selective search consists of the time it takes to rank the shards and the time it takes to search the selected shards for a query. Those two runtimes are also reported in the plots for average query runtime.

The trends for the two datasets are markedly different on both the metrics, elapsed time, and query runtime. For GOV2, exhaustive search does consistently better than selective search, but as parallelism increases the two search approaches start to converge. On the other hand, for CW09-Eng, the total elapsed time with selective search is substantially lower than that with exhaustive search. The difference in the results for the two datasets is explained by two main differences. First, each GOV2 shard is a larger fraction of the collection (0.5%) than a CW09-Eng shard (0.1%). Second, nearly 7 times as many shards are searched per query for GOV2 than for CW09-Eng. The top 20 shards out of the total of 208 shards (10% of all shards) are searched per query for GOV2. While for CW09-Eng only the top 3 shards out of the total of 807 shards (0.4% of all shards) are searched per query. The total work required for selective search, relative to the exhaustive search, is smaller for CW09-Eng than for GOV2.

For every x query per core configuration, a total of $20x$ shards need to be searched for selective search, and $16x$ for exhaustive search, in case of the GOV2 dataset. There is no guarantee that the $20x$ shards processed by selective search would be spread evenly across the two machines. In the worst case, all the $20x$ shards could be on a single machine causing higher disk and CPU contention. The possibility of this scenario occurring is higher for GOV2 than CW09-Eng because a larger fraction of the total shards is searched per query for GOV2.

When analyzing the query latency results we see that selective search provides improvement only for the CW09-Eng dataset. For 3 or fewer queries per core configurations selective search provides comparable or faster query response than exhaustive search for CW09-Eng. However,

this trend reverses after this point. Recall that for each x queries per core configuration, the total queries are divided into x query streams for exhaustive search and $5x$ query streams for selective search. As a result the individual query streams are much smaller for selective search which reduces the caching benefits. In general, we expect the caching benefits to be smaller for selective search than exhaustive search. Certain indexing data structures, such as, the term dictionary, are common to all queries in case of exhaustive search because same the set of shards is searched for every query. However, for selective search the shards searched for a query could very well be completely different from those searched for previous queries and thus would not lead to any cache hits.

The differences in the selective search performance across the two datasets suggest that the benefits of selective search are dependent on how many shards need to be searched for each query, which in turn is dependent on the distribution of relevant documents across shards. This motivates an important future direction for this work: improving topic-based allocation technique to reduce the spread of the relevant documents, which in turn would reduce the number of shards searched per query. We expect this research effort to help datasets like GOV2 where relatively large number of shards need to searched in the current setup. The other direction that these results suggest is: proposing ways to offer better load-balancing with selective search which would improve its performance in multi-query stream configurations. We expect this direction to provided a sustained improvement in query latency for datasets like CW09-Eng.

The memory usage data for these experiments is reported in Figure 6.6. For both the datasets the memory usage for selective search is higher than that for exhaustive search. This is primarily due to the difference in the number of index daemons processes running for exhaustive search and selective search. Each index shard is assign a dedicated Indri daemon process. As a result, there are 16 daemons running for exhaustive search for both the datasets. But for selective search there are 208 daemons for GOV2, and 807 for CW09-Eng. Although the number of daemons for selective are 13x and 50x larger than those for exhaustive search for GOV2 and CW09-Eng, respectively, the difference in average memory usage of exhaustive and selective search is not proportionately large. This is so because only a small fraction of all the daemons are active at a time for selective search, and idle Indri daemons have a small memory footprint.

The main goal of this section was to compare the exhaustive and selective search approaches based on their throughput performance. The results suggest that if the document collection can be partitioned into a large number of small shards such that very few shards need to be searched per query then selective search can substantially reduce the overall search effort, and also improve query latency. If such a partitioning cannot be achieved then the benefits of selective search are limited.

6.8 Cost-in-Documents metric revisited

The previous chapters have used the cost-in-documents (CiD) metric to quantify the search cost of a given approach. One of the main advantages of the CiD metric is that it allowed us to measure search effort in a simulated experimental setup. However, it is an approximate metric that does not model certain query processing costs, such as, the overheads associated with evaluating structured queries. In this section we thus revisit the CiD metric to compare it with the more direct measure of search effort: the query runtime. We analyze the sensitivity of the CiD metric to two factors in this section: query length, and query representation (unstructured versus structured queries).

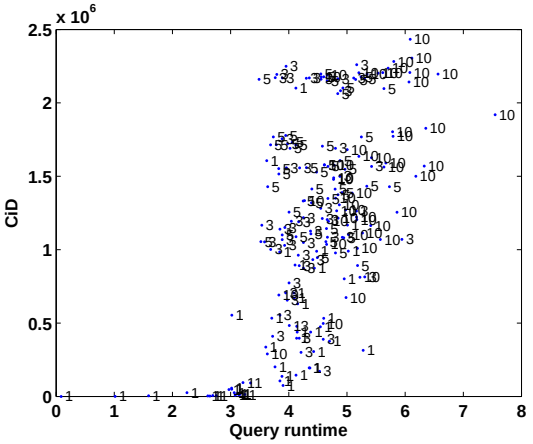
6.8.1 Query length and cost metric

For this analysis we experimented with 4 query lengths: 1, 3, 5, and 10. we sampled 50 queries for each length, from the AOL .gov query log for the GOV2 dataset. The query set for CW09-Eng also contained 50 queries for lengths 1, 3, and 5. We could only experiment with 10 queries of length 10 because our current implementation for computing CiD takes a long time for CW09-Eng. For both the datasets the query length requirement was enforced after filtering out stopwords. Each query set was evaluated using the selective search approach. The experimental setup was similar to the one used in Section 6.7 for selective search, with the exception of number of parallel query streams. For all the experiments in this section, a single-threaded query processing was employed.

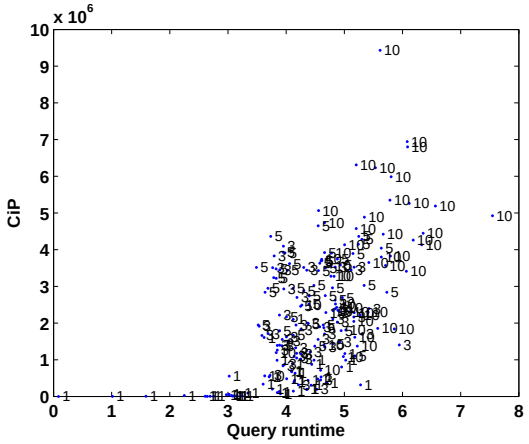
In this sub-section we also study a new but related metric: cost-in-postings (CiP). The CiP metric reports the sum of all the posting list sizes for the query terms. The volume of data transferred from the disk for a query is likely to be proportional to the total posting list sizes. It could thus be hypothesized that CiP is a better metric that models the I/O costs more accurately, while the CiD is a better metric for modeling the computational costs. Often query processing on large document collections is I/O bound rather than CPU bound, and thus we are interested in comparing both, CiD and CiP, with the query runtime metric.

The scatter plots that compare the two cost metrics with query runtime are presented in Figure 6.7. The data-point labels report the query length. The correlation between the cost metrics and query runtime, which is reported in the figure captions, is high for both the datasets. The correlation between CiP and query runtime is higher than the correlation between CiD and query runtime for both the datasets, and the difference is larger for the bigger dataset. This suggests that metrics that model the I/O costs associated with query processing are better estimators of query runtime.

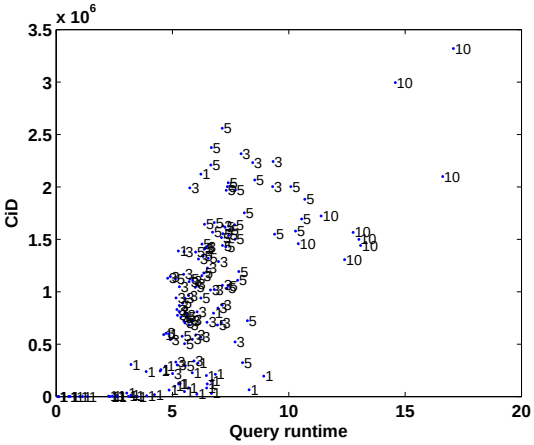
The correlation between query length and query runtime, which is also reported in the captions of Figure 6.7 indicates that there exists a strong dependence between query length and query runtime. This is not surprising, longer queries require more work and thus have



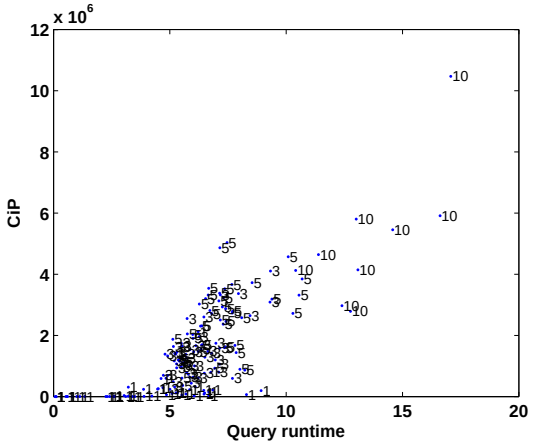
(a) CiD versus Query runtime. Pearson's correlation coefficient (ρ): 0.58. Dataset: GOV2. (Data-point labels are query lengths. ρ(Query length,Query runtime)=0.63.)



(b) CiP versus Query runtime. Pearson's correlation coefficient (ρ): 0.60. Dataset: GOV2. (Data-point labels are query lengths. ρ(Query length,Query runtime)=0.63.)



(c) CiD versus Query runtime. Pearson's correlation coefficient (ρ): 0.61. Dataset: CW09-Eng. (Data-point labels are query lengths. ρ(Query length,Query runtime)=0.74.)



(d) CiP versus Query runtime. Pearson's correlation coefficient (ρ): 0.68. Dataset: CW09-Eng. (Data-point labels are query lengths. ρ(Query length,Query runtime)=0.74.)

Figure 6.7: Correlation between cost metric (CiD or CiP) and query runtime.

Table 6.8: Influence of query length on the Pearson's correlation (ρ) between cost metrics and query runtime.

Dataset	Query length	Pearson's correlation (ρ)	
		(CiD,Query runtime)	(CiP,Query runtime)
GOV2	1	0.51	0.51
	3	0.15	0.09
	5	0.12	0.07
	10	0.57	0.52
CW09-Eng	1	0.35	0.35
	3	0.62	0.63
	5	0.44	0.51
	10	0.12	0.03

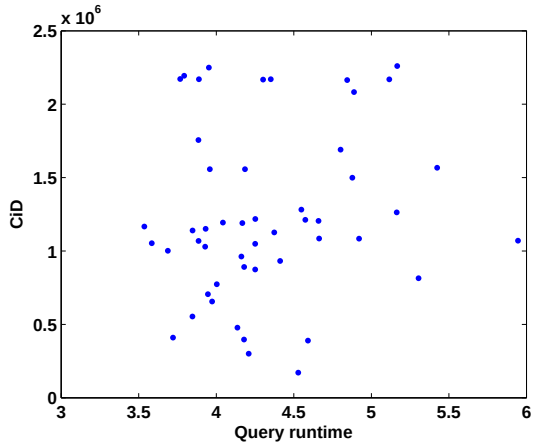
longer runtimes than shorter queries. In addition to this high-level validation, we are also interested in analyzing if the query length influences the cost metric's ability to model query runtime. For instance, is it the case that CiD is less correlated with query runtime for longer queries. The data for this analysis is provided in Table 6.8. Here the Pearson's correlation coefficient between the cost metric and query runtime were computed for each query length separately. Since longer queries require larger I/O we expected the correlation between CiP and query runtime to be higher than the correlation between CiD and query runtime for the longer queries. However, no such trends emerge from these results. This suggests that query length is not much of a factor in terms of influencing the cost metric's ability to model the query runtime. This does not imply that query length does not affect the query runtime. As was observed earlier, the Pearson's correlation coefficient between query length and query runtime for GOV2 and CW09-Eng demonstrate high strengths of 0.63 and 0.74, respectively.

The low correlation coefficients for queries of length 3 and 5 for GOV2, and for queries of length 1 and 10 for CW09-Eng also reveal that the cost metrics are not always reliable predictors of search runtime. Unfortunately, these results do not suggest any systematic trends as to when the cost metrics fail.

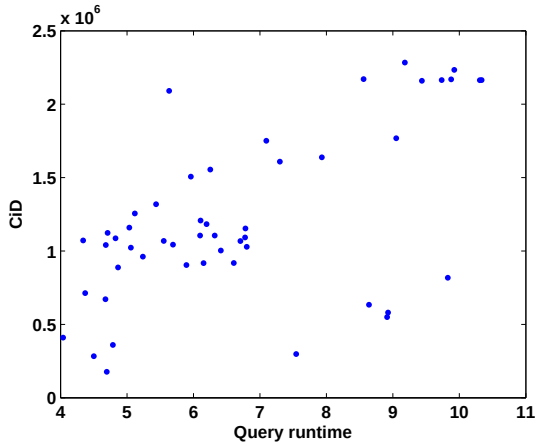
6.8.2 Query representation and cost metric

In this section our goal is to investigate if query representation influences the relation between CiD and query runtime. The bag-of-word (BOW) queries of lengths 3 and 5 from the previous section were reused for this analysis for both the datasets. Each query set contains 50 queries. These BOW queries were transformed into the full-dependence model representation using the utility made available by Metzler[4].

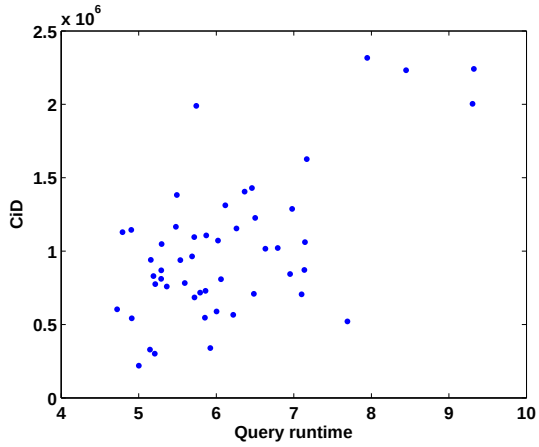
[4]http://ciir.cs.umass.edu/ metzler/dm.pl



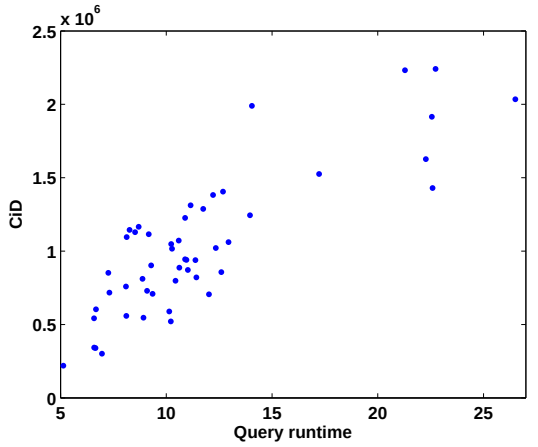
(a) CiD versus Query runtime for BOW query representation. Pearson's correlation coefficient (ρ): 0.15. Dataset: GOV2.



(b) CiD versus Query runtime for Full-dependence model query representation. Pearson's correlation coefficient (ρ): 0.59. Dataset: GOV2.

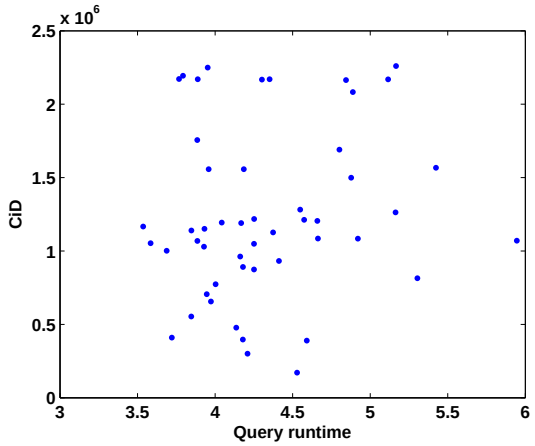


(c) CiD versus Query runtime for BOW query representation. Pearson's correlation coefficient (ρ): 0.62. Dataset: CW09-Eng.

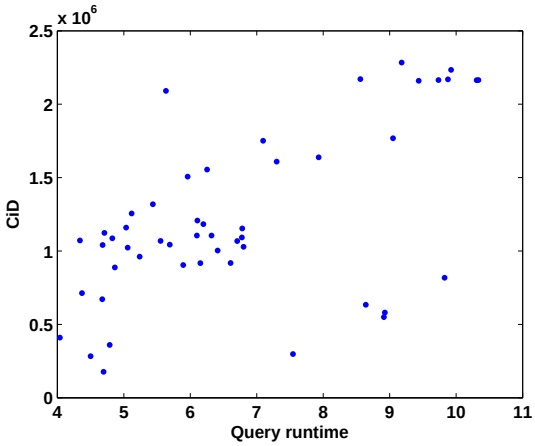


(d) CiD versus Query runtime for Full-dependence model query representation. Pearson's correlation coefficient (ρ): 0.84. Dataset: CW09-Eng.

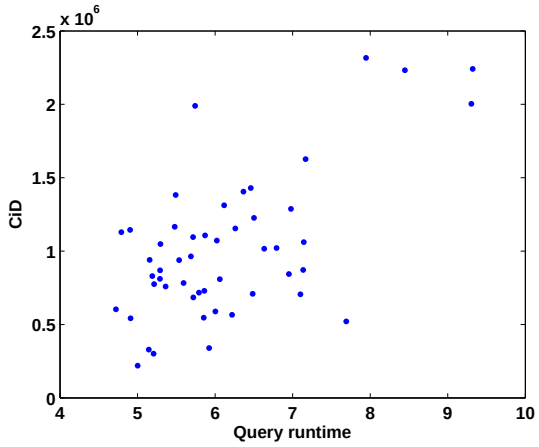
Figure 6.8: Correlation between CiD and query runtime for different query representations. BOW query length: 3



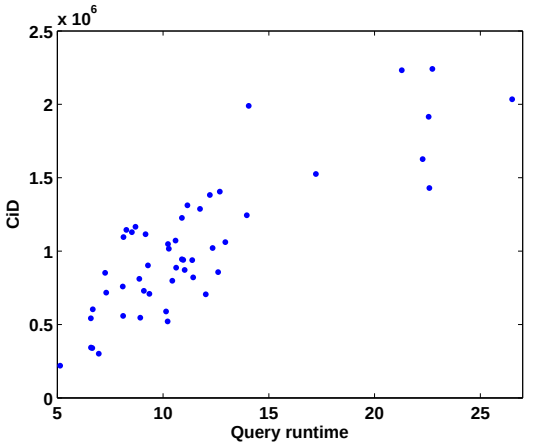
(a) CiD versus Query runtime for BOW query representation. Pearson's correlation coefficient (ρ): 0.12. Dataset: GOV2.



(b) CiD versus Query runtime for Full-dependence model query representation. Pearson's correlation coefficient (ρ): 0.76. Dataset: GOV2.



(c) CiD versus Query runtime for BOW query representation. Pearson's correlation coefficient (ρ): 0.44. Dataset: CW09-Eng.



(d) CiD versus Query runtime for Full-dependence model query representation. Pearson's correlation coefficient (ρ): 0.55. Dataset: CW09-Eng.

Figure 6.9: Correlation between CiD and query runtime for different query representations. BOW query length: 5

The scatter plots that illustrate the influence of query representation on query runtime and on the cost-in-documents (CiD) metric are reported in Figure 6.8 for queries of length 3, and in Figure 6.9 for queries of length 5. The Pearson's correlation coefficients for the different variable pairs are reported in the figure captions. Contrary to our expectations, these plots indicate higher positive correlation between CiD and query runtime for the structured queries than with unstructured queries. Since CiD does not explicitly model the overhead costs associated with structured query processing, we had expected the trends to be reverse. However, this trend is consistent across the datasets and across the query lengths as well. The cause of this trend is not clear to us.

To summarize, the high-level correlation analysis presented in this section suggests that in absence of more accurate metrics such as query runtime, the cost-in-documents and cost-in-postings can be used as proxy metrics for search effort. However, the fine-grained analysis suggests that this conclusion is not true in some situations. Further investigation is necessary to resolve this inconsistency and to identify when CiD or CiP are appropriate metrics and when they are not.

6.9 Summary

In this chapter we tested the two remaining objectives laid out in Section 3.2 for distributed selective search: first, *shorter query runtime* than distributed exhaustive search, and second, the ability of the proposed approach to function in a modestly equipped search environment (*low computing resources* requirement). The query runtime results presented in this chapter establish that selective search can be faster than exhaustive search if the document collection can be partitioned into a large number of small shards such that very few of those shards need to be searched per query. The hardware platform used for the experiments exemplifies a typical low-resource environment and thus demonstrates selective search's ability to operate in low computing environments.

This chapter also brought to light the importance of analyzing and improving the efficiency of the shard (resource) ranking algorithms. We conjectured that the topical homogeneity of the shards can be exploited to reduce the number of documents used by the resource ranking algorithm without degrading its effectiveness. The experimental results demonstrated that substantially fewer documents can support effective search while reducing the shard ranking overhead when working with topic-based shards.

In Section 6.5 we studied another approach for improving efficiency of selective search. We showed that a widely used query optimization technique could be applied to selective search to improve query runtime further. This analysis also established that selective search, with or without query optimization, is faster than exhaustive search with query optimization. Finally, a study of the impact of query length on selective search's efficiency demonstrates that the

runtime is sub-linear in query length. As such, selective search provides larger speed ups for longer queries but improvements for smaller queries are also substantial.

Chapter 7

Conclusions

This thesis recognizes the growing need to search large collections using modest computational resources and responds to the need with the tool of distributed selective search. We conclude the dissertation in this chapter by summarizing the proposed search architecture and the key ideas developed for it. The chapter also summarizes the main results presented in this dissertation along with our interpretations of the results. The broader significance of this work along with its potential impact on other work are presented in the second part of this chapter. Finally, new research challenges that this work inspires are outlined in the last section.

7.1 Thesis summary and main results

This dissertation questions the common practice of searching the complete collection (or a large portion of the collection) for each query. Instead we propose a search approach that processes each query against only a small subset of the collection, thus reducing the required search effort, and by extension, the required computational resources. This approach, *distributed selective search*, maintains competitive search effectiveness by making use of the *Cluster Hypothesis* as per which *closely associated documents tend to be relevant to the same requests*. We develop a *document allocation policy* to organize the dataset into smaller partitions (*shards*) based on their lexical similarity. Empirical evaluation demonstrates that the resulting *topical* shards concentrate the relevant documents for a particular query into a few shards. This result validates the Cluster Hypothesis, and provides the necessary basis for selective search. The skewed distribution of relevant documents across shards can be exploited to restrict the search to a few shards without degrading retrieval effectiveness.

The other key component of the distributed selective search approach is *relevance based shard selection*. We observe that the set of *relevant shards* for each query may be different in the topic-based organization of the collection. For instance, the topical shards that contain relevant documents for the query *bob marley* could very well be different from those for the

query *solomon islands*. The task of selecting relevant shards for the query consists of two subproblems: *shard ranking*, and *shard rank cutoff estimation*. The former ranks the shards based on their estimated relevance to the query. The latter predicts the number of top shards in the proposed ranking that ought be searched for the query. We develop a family of algorithms that simultaneously solves both, shard ranking and rank cutoff estimation problems. The ability to identify the smallest set of relevant shards for the query allows distributed selective search to balance search effectiveness and efficiency. Below we summarize the main findings and results from this dissertation.

Although the Cluster Hypothesis has been used in prior work to partition document collections, the topic-based allocation policy developed in this dissertation is different on three accounts. First, it is efficient. It uses approximate clustering that processes only a small sample of the dataset. Second, it can be parallelized. The task of assigning a document to one of the pre-learned topical shards can proceed in parallel for multiple documents. Third, it is widely applicable. Since it does not make use of any external resources such as query logs or categorization taxonomies it can be used to partition any given dataset. The experiments indicate that topical shards learned from samples as small as 0.1% of the dataset can support accurate selective search. In addition to topical shards we also experiment with random and source-based shards which have been used in prior work. Experiments indicate that the ability to concentrate relevant documents for a query into few shards is however unique to topical shards. We see that selective search with topic-based shards is markedly more accurate than with random or source-based. This is especially true for metrics that evaluate at deeper ranks.

Experiments with three of the largest datasets available for research demonstrate that selective search with topical shards expends 77–94% less effort than exhaustive search while being just as accurate. Previous work in large-scale search has typically evaluated search effectiveness only at early ranks. In contrast, our results show that selective search with topical shards is accurate at much deeper ranks and on graded relevance using metrics such as P@100, MAP, and NDCG@100. Another consistent trend across all the experiments in this dissertation is that the savings in search effort with selective search are bigger for the larger datasets. This is an important and useful result that makes selective search even more attractive for large-scale search.

Operational search systems typically prefer equal sized shards because they support better load balancing and provide low variance in query run times. The shards created by the topic-based algorithm, however, exhibit a high variance in their sizes. We thus develop an extension to the algorithm that bounds the sizes of the created shards. The experimental results show that the majority of the topical shards created by the new approach are of comparable sizes, for each of the datasets. Furthermore, selective search using the equal sized topical shards evaluates fewer documents on average since there are fewer large shards. Compared to the baseline, selective search is improved to 82%–95% more efficient while still being just as accurate.

The problem of ranking collections of documents (resources) based on their estimated relevance to the query has been studied extensively in the federated search field. One of the consistent findings from the resource ranking research has been that the *sample-based* algorithms perform better than the *model-based* algorithms. Contrary to this trend, however, our experiments demonstrate that a model-based algorithm (CORI) supports selective search that is as accurate as selective search with a sample-based algorithm (ReDDE). This result suggests two interesting findings. First, that selective search is not overly sensitive to the choice of shard ranking algorithm. Second, that the model-based algorithms are more effective when the resources are topically focused and distinct, like the topic-based shards used with selective search.

Although off-the-shelf resource ranking algorithms such as CORI and ReDDE can be used for shard ranking, the problem of predicting the number of top shards to search for a query has not been investigated in prior work. We show that the minimal shard rank cutoff varies widely across queries. As a result, using a preset query independent rank cutoff, as was done in most prior work, provides suboptimal search efficiency. We mend this by proposing a family of three algorithms, SHiRE, that jointly formulate the two inter-dependent problems of shard ranking and cutoff estimation. Experiments comparing the SHiRE algorithms with a state-of-the-art resource ranking algorithm and best fixed rank cutoff demonstrate that the proposed algorithms reduce the search cost by at least a quarter and these improvements are higher for larger collections.

It is commonly believed that a complete search yields the best possible search results, and by extension, defines the upper-bound on search effectiveness. The results in this dissertation challenge this belief. For both the ClueWeb09 datasets selective search with topic-based shards provides more accurate results than those with exhaustive search. Although these improvements are not statistically significant, the trend suggests that for some queries a partial search of the collection might be better than a complete search.

Another important result from this work is the quantification of the efficiency of selective search in terms of query runtime. Experiments where exhaustive search and selective search were provided the same number of computing resources show that selective search is 81% faster than exhaustive search for the larger dataset, and 21% faster for the smaller collection. Further we see that the lexical homogeneity of topic-based shards can be exploited to make selective search even more efficient. A smaller *shard ranking index* is able to support accurate selective search that is 90% faster than the baseline.

Many large-scale search systems employ *query optimization techniques* to improve retrieval efficiency. A study of the dynamics between selective search and a well established optimization approach provides several important insights. The first is that selective search supports substantially faster query processing than even an optimized exhaustive search (27–84%). Secondly, we see that selective search and query optimization are complementary approaches to

improving search efficiency. The optimized selective search provides speed up of 5–15% over unoptimized selective search. Lastly, all the gains in runtime are bigger for the larger dataset.

7.2 Thesis contributions

The primary goal of this thesis is to provide a solution to the problem of accurate and efficient large-scale search. The significance of the solution proposed in this dissertation, distributed selective search, and its potential impact are discussed in this section.

Selective search has the potential to transform large-scale search into a more *greener* process. The efficiency improvements offered by selective search can translate to financial and environmental incentives. These are especially visible for larger computing facilities. A lower search effort translates to lower energy requirements in multiple ways. The energy usage of the computing nodes, and the cooling systems used to keep the computing nodes functional, both, decrease when the search effort needed per query reduces. This advocates the use of selective search approach even in environment that are not necessarily constrained in terms of available computing resources.

Over the years the gap between the collection sizes used by the IR research community and the commercial search engines has steadily widened. Most research groups are not equipped to work with Web scale collections. As a result, it is not always clear if the findings from the research conducted by the IR community are applicable to Web search which is unfortunate because Web search is one of the biggest applications of IR by volume. In order to bring more credibility to the research findings of smaller research groups search approaches that can process large-scale collections on modest computing resources need to be developed. Designing and testing one such search approach is the focus of this dissertation.

The other equally important goal for the proposed approach is to not compromise search effectiveness in order to improve search efficiency. This is important for user-facing IR systems, and also for applications of IR, such as, question-answering systems, information extraction techniques, summarization approaches, and contextual ad placement algorithms, that build on the results provided by the search system. Although Web search is often categorized as precision-oriented, many other search applications are recall-oriented. We thus evaluate the proposed search approach using metrics such as NDCG@100 and MAP, in addition to the commonly used early rank metrics.

The reliability of selective search performance demonstrated in this dissertation helps revive the cluster-based retrieval, a line of research from early 1980s that vanished from main stream IR research primarily due to lack of consistent results' trends. Through this work we renew the research interest in cluster-based retrieval by making its connections to two active areas of research more evident. Specifically, going ahead we expect more exchange of ideas between large-scale search, federated search, and cluster-based retrieval.

The other reason for the disappearance of the cluster-based retrieval approaches was their inability to scale to larger datasets. In fact, none of the techniques described in related work would be able to efficiently partition the datasets used in this dissertation. The shard creation approach developed as part of this thesis is thus one of key contributions. The sub-linear computational complexity of the proposed collection partitioning technique makes it highly efficient.

The family of shard ranking and cutoff estimation algorithms presented in this dissertation is instrumental in two ways. It introduces a new problem of shard rank cutoff estimation to the research community, and demonstrates the interdependence between the new problem and the old problem of shard (resource) ranking that has been widely studied.

Overall, this dissertation provides one of the first end-to-end evaluations of a distributed IR technique on some of the largest available dataset, and demonstrates consistent and substantial improvements in search efficiency. speed up in query response time.

7.3 Future directions

In this section we identify three different directions along which the research pioneered by distributed selective search can be extended. One of the directions would leverage the unique characteristics of selective search, such as the shard and document level modularity, to improve its efficiency and effectiveness. The second direction would explore the challenges faced by high-traffic search systems, such as load-balancing and query response time, in the context of distributed selective search. The third part would test the ability of selective search to support a wide spectrum of research problems from language technologies.

7.3.1 Effectiveness and efficiency of selective search

A document collection that is partitioned into topical shards provides a modular organization of data. In this search environment the retrieval model for each shard can be customized to take advantage of the unique characteristics of its documents and the queries that are directed to the shard. Query categorization algorithms can be employed to identify and model the dynamics between different topics and the different types of queries that they serve. For instance, if a large fraction of the queries processed by a shard are of informational type then the retrieval algorithm could be adapted to exhibit a similar bias. Likewise if a shard tends to serve time-sensitive queries then adapting the shard's inverted index for efficient access to temporal data, employing sophisticated query transformation functions, and tailoring the retrieval algorithm appropriately would be useful. The existing research in vertical search which has some similarities with the proposed future direction, can be used to inform the next steps.

The estimator proposed in this work for the task of query-specific shard rank cutoff prediction takes a step in the right direction but more work is needed. A prediction model that integrates additional information sources, such as query difficulty predictors and functions that capture the topical affinity between the highly ranked shards for the query, could be more effective at shard rank cutoff estimation. For example, multiple shards with high affinity could indicate a flat distribution of relevant documents for the query, which in turn would suggest a larger shard rank cutoff value. Also, a larger cutoff estimate might be useful for difficult queries. This model could be especially useful for identifying and responding to faceted information needs, and ambiguous information needs. The efficiency and effectiveness of selective search, both, could be improved using these enhancements.

7.3.2 Load-balancing for selective search

For many organizations we expect the number of available computing resources (m) to be much smaller than the number of topical shards (n) created from a collection. As a result, multiple shards need to be assigned to each resource. An important future direction is to investigate the *shard-machine assignment* problem. It is likely that a shard allocation policy that distributes similar shards across different resources would be most successful at exploiting the inherent parallelism in the search process. For example, if the two shards selected for search for a query are assigned to different resources then both shards can be searched simultaneously thus improving the query response time. Like the topic-based document allocation policy, the shard allocation policy would also leverage the cluster hypothesis to identify the shards that would be relevant to the same query but instead of grouping them together it would spread them across resources.

The topic-based partitioning of the collection is central to the success of distributed selective search approach. However, this topically skewed arrangement of documents can make the task of balancing the computing load on the cluster difficult. This might be especially challenging in search environments where the query stream exhibits high fluctuations in the popularity of topics, as is often observed in Web search. In such scenarios the resources assigned to a few of the currently popular shards could be over-utilized while the other shards' resources may idle. One solution to this problem of imbalanced resource utilization is to distribute each topical shard across several of the available resources in order to spread out the needed computations. Although each resource may hold a part of many topical shards, the search can still be restricted to only a subset of the shards by maintaining the necessary mapping information. In this architecture many of the resources would be active for every query.

In order to meet the operational requirements of fast query response time and high query throughput, search environments with heavy query traffic often employ data replication as the solution. When the documents are partitioned into random shards, all the shards are typically replicated in order to reduce data contention. The topic-based partitioning of the shards can

support an adaptive replication approach that replicates only a few shards based on usage patterns, and thus lowers the overall replication factor. The past usage of the shards and the current query traffic patterns can be used to design a selective and dynamic replication strategy that makes use of the disk resources judiciously.

7.3.3 Applications of selective search

Increasing number of applications make use of search technology as an underlying tier that supports a specialized task. For example, question answering systems, such as IBM Watson[1], often use document search as one of the first steps in their pipeline architecture. Intelligent vocabulary tutors, such as REAP[2], employ document retrieval to provide the students with more engaging reading material. Information extraction projects, such as 'Read the Web' (http://rtw.ml.cmu.edu/rtw/), make extensive use of search technology to extract and organize facts from a large collection (or Web).

Selective search can be used to power the underlying *search tier* in each of the above instances. Extending and adapting selective search to apply it to a diverse set of specialized tasks is an exciting future direction.

[1] http://www-03.ibm.com/innovation/us/watson/
[2] http://reap.cs.cmu.edu/

Bibliography

[1] Vo Ngoc Anh, Owen de Kretser, and Alistair Moffat. Vector-space ranking with effective early termination. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 35–42, 2001. 2.1.2

[2] Jaime Arguello, Jamie Callan, and Fernando Diaz. Classification-based resource selection. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 1277–1286, 2009. 1.3, 2.3.3, 5.2, 5.2.3, 5.3

[3] David Arthur and Sergei Vassilvitskii. `k-means++`: The advantages of careful seeding. In *Proceedings of the ACM SIAM Symposium on Discrete Algorithms*, pages 1027–1035, 2007. 4.1.3, 4.3, 4.3.4

[4] R. Baeza-Yates, C. Castillo, F. Junqueira, V. Plachouras, and F. Silvestri. Challenges on distributed Web retrieval. In *Proceedings of the International Conference on Data Engineering*, pages 6–20, 2007. 1.3, 2.1

[5] Ricardo Baeza-Yates, Vanessa Murdock, and Claudia Hauff. Efficiency trade-offs in two-tier Web search systems. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 163–170, 2009. 1.2.1, 1.3, 2.1, 3

[6] Mark Baillie, Leif Azzopardi, and Fabio Crestani. Adaptive query-based sampling of distributed collections. In *Proceedings of the International Conference on String Processing and Information Retrieval*, pages 316–328, 2006. 2.3

[7] Luiz Andró Barroso, Jeffrey Dean, and Urs Hölzle. Web Search for a Planet: The Google cluster architecture. *IEEE Micro*, 23(2):22–28, 2003. 1.3, 2.1, 3, 4.1.1

[8] David Blei, Andrew Ng, and Michael Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003. 4.1.3

[9] Leonid Boytsov and Anna Belova. Evaluating learning-to-rank methods in the Web track Adhoc task. In *The Twentieth Text REtrieval Conference (TREC 2011) Proceedings*, 2012. 6.1

[10] Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. Efficient query evaluation using a two-level retrieval process. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 426–434, 2003. 3.4.4, 6.6

[11] Eric W. Brown. Fast evaluation of structured queries for information retrieval. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 30–38, 1995. 2.1.2

[12] Stefan Büttcher and Charles L. A. Clarke. A document-centric approach to static index pruning in text retrieval systems. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 182–189, 2006. 2.1.2

[13] James Callan, W. Bruce Croft, and Stephen M. Harding. The INQUERY retrieval system. In *Proceedings of the International Conference on Database and Expert Systems Applications*, pages 78–83, 1992. 2.3.1

[14] James P. Callan, Zhihong Lu, and W. Bruce Croft. Searching distributed collections with inference networks. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 21–28, 1995. 1.3, 2.3.1, 4.1.2, 5.2, 5.3

[15] Jamie Callan. Distributed information retrieval. In W. Bruce Croft, editor, *Advances in Information Retrieval*, pages 127–150. 2000. 1.3, 2.3, 2.3.1, 2.3.1, 3.6, 5.2.2, 6.4

[16] Jamie Callan and Margaret Connell. Query-based sampling of text databases. *ACM Transactions on Information Systems*, 19(2):97–130, 2001. 2.3, 5.2.1

[17] Jamie Callan, Margaret Connell, and Aiqun Du. Automatic discovery of language models for text databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 479–490, 1999. 2.3, 4.1.2

[18] B. Barla Cambazoglu, Vassilis Plachouras, and Ricardo Baeza-Yates. Quantifying performance and quality gains in distributed Web search engines. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 411–418, 2009. 1.2.1, 2.1.1

[19] David Carmel, Doron Cohen, Ronald Fagin, Eitan Farchi, Michael Herscovici, Yoölle S. Maarek, and Aya Soffer. Static index pruning for information retrieval systems. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 43–50, 2001. 2.1.2

[20] B. Carterette, V. Pavlu, H. Fang, and E. Kanoulas. Overview of the TREC 2009 Million Query track. In *Proceedings of the 2009 Text Retrieval Conference*, 2009. 6.6, 6.7.1

[21] Abdur Chowdhury and Greg Pass. Operational requirements for scalable search systems. In *Proceedings of the Conference on Information and Knowledge Management*, pages 435–442, 2003. 1.3, 2.1, 3

[22] Charles Clarke, Nick Craswell, and Ian Soboroff. Overview of the TREC 2004 Terabyte track. In *Proceedings of the 2004 Text Retrieval Conference*, 2004. 3.3

[23] Nick Craswell, Peter Bailey, and David Hawking. Server selection on the World Wide

Web. In *Proceedings of the ACM conference on Digital Libraries*, pages 37–46, 2000. 1.3, 2.3

[24] W. Bruce Croft. A model of cluster searching based on classification. In *Information Systems*, pages 189–195, 1980. 1.3, 2.2

[25] Edleno S. de Moura, Célia F. dos Santos, Daniel R. Fernandes, Altigran S. Silva, Pavel Calado, and Mario A. Nascimento. Improving Web search efficiency via a locality based static pruning method. In *Proceedings of the 14th International Conference on World Wide Web*, pages 235–244, 2005. 2.1.2

[26] Arthur P. Dempster, N M. Laird, and Donald B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977. 4.1.3

[27] A. El-Hamdouchi and P. Willett. Comparison of hierarchic agglomerative clustering methods for document retrieval. *The Computer Journal*, 32(3):220–227, 1989. 1.3, 2.2

[28] James C. French, Allison L. Powell, Jamie Callan, Charles L. Viles, Travis Emmitt, Kevin J. Prey, and Yun Mou. Comparing the performance of database selection algorithms. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 238–245, 1999. 4.1.2, 5.6

[29] Steven Garcia, Hugh E. Williams, and Adam Cannane. Access-ordered indexes. In *Proceedings of the 27th Australasian Conference on Computer science*, pages 7–14, 2004. 2.1.2

[30] Luis Gravano and Héctor García-Molina. Generalizing GlOSS to vector-space databases and broker hierarchies. In *Proceedings of the Conference on Very Large Data Bases*, pages 78–89, 1995. 2.3.1, 2.3.1

[31] Luis Gravano, Héctor García-Molina, and Anthony Tomasic. The effectiveness of GIOSS for the text database discovery problem. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 126–137, 1994. 1.3, 2.3.1

[32] Luis Gravano, Héctor García-Molina, and Anthony Tomasic. GlOSS: text-source discovery over the internet. *ACM Transactions on Database Systems*, 24:229–264, June 1999. 1.3, 2.3.1, 2.3.1, 5.2

[33] Alan Griffiths, H.Claire Luckhurst, and Peter Willett. Using inter-document similarity information in document retrieval systems. *Journal of the American Society for Information Science*, 37(1):3–11, 1986. 2.2

[34] Greg Hamerly and Charles Elkan. Learning the `k` in `k-means`. In *Proceedings of the Conference on Neural Information Processing Systems*, pages 281–288, 2003. 4.2

[35] J. A. Hartigan and M. A. Wong. Algorithm AS 136: A K-Means Clustering Algorithm. *Journal of the Royal Statistical Society. Series C*, 28(1):100–108, 1979. 4.1.3, 4.3, 4.3.3

[36] Claudia Hauff and Djoerd Hiemstra. University of Twente @ TREC 2009: Indexing half

a million Web pages. In *The Eighteenth Text REtrieval Conference Proceedings (TREC 2009)*, 2010. 6.1

[37] J. Heaps. *Information Retrieval – Computational and Theoretical Aspects*. Academic Press Inc., 1978. 4.1.3

[38] Marti A. Hearst and Jan O. Pedersen. Reexamining the cluster hypothesis: scatter/gather on retrieval results. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 76–84, 1996. 2.2

[39] Panagiotis G. Ipeirotis and Luis Gravano. Distributed search over the hidden Web: hierarchical database sampling and selection. In *Proceedings of the Conference on Very Large Data Bases*, pages 394–405, 2002. 2.3

[40] Panagiotis G. Ipeirotis and Luis Gravano. Distributed search over the hidden Web: Hierarchical database sampling and selection. In *Proceedings of Conference on Very Large Data Bases*, pages 394–405, 2002. 1.3, 2.3.2, 5.2

[41] Rocchio J. *Document retrieval systems - optimization and evaluation*. PhD thesis, 1966. 2.2

[42] N. Jardine and Cornelis Joost van Rijsbergen. The use of hierarchical clustering in information retrieval. *Information Storage and Retrieval*, 7:217–240, 1971. 2.2

[43] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems*, 20(4):422–446, 2002. 3.4.1

[44] Anagha Kulkarni and Jamie Callan. Document allocation policies for selective searching of distributed indexes. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 449–458, 2010. 1

[45] Anagha Kulkarni and Jamie Callan. Topic-based index partitions for efficient and effective selective search. In *SIGIR 2010 Workshop on Large-Scale Distributed Information Retrieval*, pages 19–24, July 2010. 1

[46] Anagha Kulkarni, Almer Tigelaar, Djoerd Hiemstra, and Jamie Callan. Shard ranking and cutoff estimation for topically partitioned collections. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 555–564, 2012. 5

[47] Leah S. Larkey, Margaret E. Connell, and Jamie Callan. Collection selection and results merging with topically organized U.S. patents and TREC data. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 282–289, 2000. 1.3, 4.1, 4.1.2, 4.1.3

[48] Xiaoyong Liu and W. Bruce Croft. Cluster-based retrieval using language models. In *Proceedings of the ACM SIGIR conference on Research and Development in Information Retrieval*, pages 186–193, 2004. 2.2

[49] J. B. MacQueen. Some methods for classification and analysis of multivariate observations.

In *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967. 4.1.3

[50] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. 4.1.3

[51] Donald Metzler and W. Bruce Croft. Combining the language model and inference network approaches to retrieval. *Information Processing and Management*, 40(5):735–750, 2004. 3.6, 4.1.4, 5

[52] Donald Metzler and W. Bruce Croft. A markov random field model for term dependencies. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 472–479, 2005. 3.6, 5.1, 5.1

[53] Glenn Milligan and Martha Cooper. An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 50(2):159–179, 1985. 4.1.3, 4.3

[54] Alistair Moffat and Justin Zobel. Self-indexing inverted files for fast text retrieval. *ACM Transactions on Information Systems*, 14(4):349–379, 1996. 2.1.2

[55] Linh Thai Nguyen. Static index pruning for information retrieval systems: A posting-based approach. In *SIGIR 2009 Workshop on Large-Scale Distributed Information Retrieval*, pages 25–32, 2009. 2.1.2

[56] Paul Ogilvie and Jamie Callan. Experiments using the lemur toolkit. In *Proceedings of the 2001 Text Retrieval Conference*, 2001. 3

[57] Dan Pelleg and Andrew Moore. `X-means`: Extending `K-means` with efficient estimation of the number of clusters. In *Proceedings of the International Conference on Machine Learning*, pages 727–734, 2000. 4.1.4, 4.2

[58] Michael Persin. Document filtering for fast ranking. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 339–348, 1994. 2.1.2

[59] Michael Persin, Justin Zobel, and Ron Sacks-davis. Filtered document retrieval with frequency-sorted indexes. *Journal of the American Society for Information Science*, 47:749–764, 1996. 2.1.2

[60] Diego Puppin, Fabrizio Silvestri, and Domenico Laforenza. Query-driven document partitioning and collection selection. In *Proceedings of the 1st International Conference on Scalable Information Systems*, page 34, 2006. 2.2, 4.1, 4.1.1, 5.3

[61] Diego Puppin, Fabrizio Silvestri, Raffaele Perego, and Ricardo Baeza-Yates. Tuning the capacity of search engines: Load-driven routing and incremental caching to reduce and balance the load. *ACM Transactions on Information Systems*, 28(2):5:1–5:36, 2010. 5.3

[62] Carl Edward Rasmussen. The infinite gaussian mixture model. In *Advances in Neural Information Processing Systems 12*, pages 554–560, 2000. 4.2

[63] Knut Magne Risvik, Yngve Aasheim, and Mathias Lidal. Multi-tier architecture for Web search engines. In *Proceedings of the First Latin American Web Congress*, pages 132–143, 2003. 1.2.1, 1.3, 2.1, 3

[64] Gerard Salton. Cluster search strategies and the optimization of retrieval effectiveness. In Gerard Salton, editor, *The SMART Retrieval System*, pages 223–242. 1971. 1.3, 2.2

[65] Milad Shokouhi. Central-rank-based collection selection in uncooperative distributed information retrieval. In *Proceedings of the 29th European Conference on Information Retrieval*, pages 160–172, 2007. 1.3, 5.3, 5.3.3

[66] Milad Shokouhi and Luo Si. Federated search. *Foundations and Trends in Information Retrieval*, 5(1):1–102, 2011. 1.3, 2.3, 6.4

[67] Milad Shokouhi, Falk Scholer, and Justin Zobel. Sample sizes for query probing in uncooperative distributed information retrieval. In Xiaofang Zhou, Jianzhong Li, HengTao Shen, Masaru Kitsuregawa, and Yanchun Zhang, editors, *Frontiers of WWW Research and Development - APWeb 2006*, pages 63–75. 2006. 2.3

[68] Milad Shokouhi, Justin Zobel, Saied Tahaghoghi, and Falk Scholer. Using query logs to establish vocabularies in distributed information retrieval. *Information Processing and Management*, 43(1):169–180, 2007. 2.3

[69] Luo Si and Jamie Callan. Relevant document distribution estimation method for resource selection. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 298–305, 2003. 1.3, 2.3.2, 5.1, 5.2, 5.2.1, 5.2.3, 5.3, 5.3, 5.7

[70] Alan F. Smeaton and Cornelis Joost van Rijsbergen. The nearest neighbour problem in information retrieval: an algorithm using upperbounds. In *Proceedings of the 4th Annual International ACM SIGIR Conference on Information Storage and Retrieval*, pages 83–87, 1981. 2.1.2

[71] Mark D. Smucker, Charles L. A. Clarke, and Gordon V. Cormack. Experiments with ClueWeb09: Relevance Feedback and Web tracks. In *The Eighteenth Text REtrieval Conference Proceedings (TREC 2009)*, 2010. 6.1

[72] Trevor Strohman, Howard Turtle, and W. Bruce Croft. Optimization strategies for complex queries. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 219–225, 2005. 2.1.2, 3.4.4, 6.5

[73] Todd Tannenbaum, Derek Wright, Karen Miller, and Miron Livny. Condor – a distributed job scheduler. In Thomas Sterling, editor, *Beowulf Cluster Computing with Linux*. MIT Press, October 2001. 6.2

[74] Paul Thomas and Milad Shokouhi. Sushi: Scoring scaled samples for server selection. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*,

pages 419–426, 2009. 1.3, 2.3.2, 5.2, 5.2.3, 5.3

[75] Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal Of The Royal Statistical Society Series B*, 63(2):411–423, 2001. 4.1.3, 4.2, 4.3

[76] Howard Turtle and James Flood. Query evaluation: strategies and optimizations. *Information Processing and Management*, 31(6):831–850, 1995. 2.1.2

[77] Cornelis Joost van Rijsbergen. *Information Retrieval*. Butterworths, 1979. 1.2.2, 4.1.2, 4.1.3, 5.3.1

[78] Cornelis Joost van Rijsbergen and W. Bruce Croft. Document clustering: An evaluation of some experiments with the cranfield 1400 collection. *Information Processing and Management*, 11(5-7):171–182, 1975. 2.2

[79] Ellen M. Voorhees. The cluster hypothesis revisited. In *Proceedings of the 8th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 188–196, 1985. 1.3, 2.2

[80] Ellen M. Voorhees, Narendra K. Gupta, and Ben Johnson-Laird. Learning collection fusion strategies. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 172–179, 1995. 4.1.2

[81] Jr. Ward, Joe H. Hierarchical Grouping to Optimize an Objective Function. *Journal of the American Statistical Association*, 58(301):236–244, 1963. 5.3.2

[82] Peter Willett. Recent trends in hierarchic document clustering: a critical review. *Information Processing and Management*, 24(5):577–597, 1988. 1.3, 2.2

[83] Wai Yee Peter Wong and Dik Lun Lee. Implementations of partial document ranking using inverted files. *Information Processing and Management*, 29(5):647–669, 1993. 2.1.2

[84] Jinxi Xu and Jamie Callan. Effective retrieval with distributed collections. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 112–120, New York, NY, USA, 1998. 1.3

[85] Jinxi Xu and W. Bruce Croft. Cluster-based language models for distributed retrieval. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 254–261, 1999. 1.3, 2.2, 2.2, 4.1.2, 4.1.3, 5.6

[86] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Transactions on Information Systems*, 22(2):179–214, 2004. 4.1.3, 4.1.3