ROBUST SELECTIVE SEARCH

YUBIN KIM

Language Technologies Institute
School of Computer Science
Carnegie Mellon University

Thesis Committee:
Jamie Callan, Chair
Jaime Carbonell
Ralf Brown
Alistair Moffat (The University of Melbourne)
ABSTRACT

Selective search is a modern distributed search architecture designed to reduce the computational cost of large-scale search. Selective search creates topical shards that are deliberately content-skewed, placing highly similar documents together in the same shard. During query time, rather than searching the entire corpus, a resource selection algorithm selects a subset of the topic shards likely to contain documents relevant to the query and search is only performed on these shards. This substantially reduces total computational costs of search while maintaining accuracy comparable to exhaustive distributed search.

Prior work has shown selective search to be effective in smaller scale, single query-at-a-time environments. However, modern practical, large-scale search and text analysis systems are often multi-stage pipeline systems where an initial, first-stage fast candidate retrieval forwards results onto downstream complex analysis components. These systems often contain other optimization components and are run in a parallel setting over multiple machines. This dissertation aims to bring selective search to wider adoption by addressing the questions related to efficiency and effectiveness in a practical implementation such as: do different instantiations of selective search have stable performance; does selective search combine well with other optimization components; can selective search deliver the high recall necessary to serve as a first-stage retrieval system? In addition, this dissertation provides tools to empower system administrators so that they can easily design and test selective search systems without full implementations.

First, this dissertation research investigates the effects of non-deterministic steps that exist in selective search on its accuracy and found the variance across system instances is acceptable. Then, selective search was combined with WAND, a common dynamic pruning algorithm and it was shown that selective search and WAND has better-than-additive gains due to the long posting lists of the topically focused shards. This dissertation also presents new resource selection algorithms to achieve high recall, which has been an elusive goal in prior work. A learning-to-rank based approach to resource selection can be trained without human-judged relevance data to be highly accurate and statistically equivalent to exhaustive search at deeper metrics such as MAP@1000. This result enables selective search to be used as a first-stage retrieval component in realistic multi-stage text analysis systems.

When placed in a parallel query processing environment, it was found that with judicious load-balancing to manage unequal popularity of shards, the efficiency claims of prior work remain relevant in a fully parallel processing setting, and are applicable to larger computational environments as well. A detailed simulator was built to investigate this research question and serve as a powerful tool for administrators to test out different selective search configurations.

Finally, new evaluation metric, AUReC, is presented which can be used to easily evaluate a mapping of documents to shards without implementing a full selective search system. This allows system designers to quickly sift through many potential different document allocations to easily identify the best system configuration.

Ultimately, the dissertation aims to enable cost and energy-efficient use of large-scale data collections in not only information retrieval research, but also in other fields such as text mining and question answering, in academia and industry alike, fueling future innovation.
# Contents

6.1 Related Work 45  
6.2 Model 46  
6.3 Features 46  
6.3.1 Query-independent information 47  
6.3.2 Term-based statistics 47  
6.3.3 Sample-document (CSI-based) features 47  
6.4 Experimental Methodology 48  
6.5 Experiments 49  
6.5.1 Overall comparison 49  
6.5.2 Effect of training data 49  
6.5.3 Query length 49  
6.5.4 Feature analysis 51  
6.6 Summary 52  

7 Simulator 55  
7.1 Related Work 56  
7.2 Simulation Model 56  
7.3 Experimental Methodology 60  
7.3.1 Document collections 60  
7.3.2 Resource selection parameter settings 61  
7.3.3 Query streams 63  
7.4 Experimental Results 64  
7.4.1 Selective search efficiency 65  
7.4.2 Resource allocation 67  
7.4.3 Shard assignment policies 70  
7.4.4 Scalability 74  
7.5 Summary 75  

8 AUREC 81  
8.1 Related Work and Motivation 82  
8.2 AUREC 83  
8.3 Experimental Setup 86  
8.3.1 Data and query sets 86  
8.3.2 Document retrieval 86  
8.3.3 Resource selection and baselines 87  
8.3.4 Metrics 88  
8.4 Experimental Results 89  
8.4.1 Comparison with end-to-end systems 90  
8.4.2 Different metrics and evaluation depths 93  
8.4.3 Weaker retrieval engine 95  
8.4.4 Different queries 96  
8.5 Conclusion and Recommendations 97  

9 Conclusion 99  
9.1 Summary of Findings 99  
9.2 Contributions and Wider Impact 101  
9.3 Future Work 102  
9.3.1 Selective search in pipelines 102  
9.3.2 Selective search in hierarchical corpora 103
9.3.3 Selective search in streaming datasets 104

BIBLIOGRAPHY 107
LIST OF FIGURES

Figure 1 Illustration of two different search architectures. Each box represents a machine and the cylinders represent parts of the search engine index ("shards"). The dark cylinders are shards that are searched; traditional distributed search always searches all shards while selective search searches only a few for a given query. Selective search also requires an additional resource selection database (grey hashing) in order to select the shards to search. Both architectures require a merge step (triangle) that merges the partial result lists from the shards.

Figure 2 Accuracy distribution of Rank-S and RBR system instances for the Web Track queries. Mid line in boxes represents median. Outer edges of the boxes represent lower and upper quartiles, while whiskers indicate 1.5 interquartile range. + indicates outliers. Blue dot is the performance of the typical ‘search all shards’ baseline.

Figure 3 Query Average Precision of RBR instances for Web Track queries. Each data point is a box and whisker plot with the edges of the boxes representing the upper and lower quantiles, the mid line the median, and the whiskers the 1.5 interquartile range. + are outliers.

Figure 4 Distribution of shard sizes, with a total of 100 shards.

Figure 5 Correlation between the number of postings processed for a query and the time taken for query evaluation. Data points are generated from MQT queries using both WAND and full evaluation, applied independently to all 100 topical shards and all 100 random shards. In total, 756 × 200 × 2 ≈ 300,000 points are plotted.

Figure 6 Ratio of savings achieved by WAND as a function of the total postings length of each query in the AOL set, measured on a per shard basis. A total of 100 × 713 ≈ 71,000 points are plotted. Queries containing only rare terms derive little benefit from WAND.

Figure 7 Distribution of query response times for MQT queries on shards: (a) as a box plot distribution, with a data point plotted for each query-shard pair; (b) as a table of corresponding means and medians. In (a), the center line of the box indicates the median, the outer edges of the box the first and third quartiles, and the blue circle the mean. The whiskers extend to include all points within 1.5 times the inter-quartile range of the box. The graph was truncated to omit a small number of extreme points for both Rank-S Full and Taily-Full. The maximum time for both these two runs was 6,611 ms.

Figure 8 Normalized 1,000th document scores from shards, averaged over queries and then shard ranks, and expressed as a fraction of the collection-wide maximum document score for each corresponding query. The score falls with rank, as fewer high-scoring documents appear in lower-ranked shards.
Figure 9: The micro-average w/b ratio for WAND postings evaluations, as a function of the per query shard rankings assigned by Taily. Early shards generate greater savings.

Figure 10: Normalized 1,000th document scores from shards relative to the highest score attained by any document for the corresponding query, micro-averaged over queries, assuming that shards are processed sequentially rather than in parallel, using the Taily-based ordering of topical shards and a random ordering of the same shards.

Figure 11: Ratio of postings evaluated by WAND for independent shard search versus sequential shard search, AOL queries with micro-averaging. Shard ranking was determined by Taily.

Figure 12: Comparison of impact ordered posting list and source ordered posting list block max score distributions of the term obama in a top shard for the query obama family tree (shard 57). Block size is n = 128.

Figure 13: An example of the multi-size second scores method. If k = 2, b_{1,1} is the first block selected (k̂ = 1), then b_{1,2} (k̂ = 2) resulting in shard 1 being selected for search, as ideal. However, if k = 3, b_{2,1} (k̂ = 3) is also selected, resulting in both shards being selected for search even though shard 2 is unnecessary.

Figure 14: MAP@1000 for queries on CW09-B, grouped by query length. Parentheses on the X axis present the number of queries in each group. T is the shard rank cutoff.

Figure 15: Architecture of the selective distributed search system. The i'th of the M machines has c_i cores, each of which can be used for resource selection and result merging, or for shard search across any of the p_i shards allocated to this machine. Only a defined subset of the machines are able to perform resource selection.

Figure 16: Sizes of shards used in experiments reported by [Kulkarni 2013].

Figure 17: Exhaustive search and selective search using CW09-A and Gov2 and random shard assignment for the AOL_W Test and MQT_W Test, AOL_G Test and MQT_G Test query sets (shown as AOL and MQT in the labels for brevity). In these experiments, M = 2, B = 1, and S = 2.

Figure 18: Average utilization of machines using Rank-S, the CW09-A dataset, the AOL_W Test queries, M = 2, B = 1 and S = 2. Each point is the mean over 10 sequences of 1,000 queries; error bars represent 95% confidence intervals.

Figure 19: Varying the resources assigned to broker processes using Rank-S, with M = 2, and random shard assignment.

Figure 20: Popularity of shards in the CW09-A training query sets for the top 10% of most frequently accessed shards as selected by Rank-S. The vertical axis indicates the total number of times a shard is selected for all queries.

Figure 21: Utilization of machines for selective search on Gov2, using Taily, the MQT_G Test queries, M = 4, B = 1, S = 4 and a uniform shard distribution. Each point is the mean over 10 sequences of 1,000 queries; error bars represent 95% confidence intervals. The broker resides in m_1 for both configurations. The two frames share a common legend.
Figure 22 The variance of the load spread \((\text{max load}_i - \text{min load}_i)\) of ten Random shard allocations, with \(M = 4, B = 1\). The midline of the box indicates the median, the outer edges the 25th and 75th percentile values, and the whiskers the most outlying values. The load spread for Log-based shard allocation is also plotted as a reference point. Note the different horizontal and vertical scales in the two panes.

Figure 23 The variance of the load spread \((\text{max load}_i - \text{min load}_i)\) of ten Random shard allocations, now with \(M = 8, B = 8\). All other parameters are as for Figure 22. Note the different horizontal scales in the two panes.

Figure 24 The effect of shard assignment policies on throughput of selective search, using Taily with \(M = 4, B = 1\) (left column) and \(M = 8, B = 8\) (right column). Note the differing horizontal scales in each pair.

Figure 25 Selective search compared to exhaustive search with \(M = 64, B = 64\) using CW09-A and log-based shard assignment for the AOL_W Test and MQT_W Test query sets (shown as AOL and MQT in the labels for brevity).

Figure 26 The total time required by a selective search system to process 10,000 queries as a function \(M\), the number of machines. Assumes an infinite query arrival rate and \(B = M\) in all configurations. The two query streams used are MQT_G Test and AOL_G Test, denoted as MQT and AOL respectively. The red dash-dot line represents the ideal case, where throughput is doubled with doubled machine capacity.

Figure 27 Throughput achieved with Log-based shard assignment and \(M = 64, B = 64\) to process queries against CW09-A. The mirrored configuration is a doubled \(M = 32, B = 32\) system, including disk space. The two query streams are MQT_G Test and AOL_G Test, denoted as MQT and AOL respectively.

Figure 28 Example of a recall versus fraction of shards searched curve for three different index partitions for the same query.

Figure 29 Contingency table for pair-wise comparison of index partitions using AURSC and end-to-end system evaluation (EtE).

Figure 30 Correlation between AUREC scores and P@1000 scores of an end-to-end selective search systems using Rank-S resource selection. Different shapes indicate that the shard map was created using the KLD-Rand (★), QKLD-Rand (×), or QKLD-Qinit (+) method [Dai et al. 2016].
LIST OF TABLES

Table 1  Standard Deviation (SD) and Variance Coefficient (VC) of accuracy scores for Rank-S instances. 24
Table 2  Standard Deviation (SD) and Variance Coefficient (VC) of accuracy scores for RBR instances. 24
Table 3  Comparison of MAP@500 scores (CW09-A) for Rank-S and Taily instances. 26
Table 4  Ratio of per shard per query postings evaluated and per shard per query execution time for WAND-based search, as ratios relative to unpruned search, averaged over 100 topical shards and over 100 randomized shards, and over two groups each of 700+ queries. The differences between the Topical and Random macro-averaged ratios are significant for both query sets and both measures (paired two-tailed t-test, p < 0.01). 33
Table 5  Average number of shards searched, and micro-averaged postings ratios for those selected shards and for the complement set of shards, together with the corresponding query time cost ratios, in each case comparing WAND-based search to unpruned search. Smaller numbers indicate greater savings. 33
Table 6  As for Table 5, but showing macro-averaged ratios. All differences between selected and non-selected shards are significant (paired two-tailed t-test, p < 0.01). 34
Table 7  The results of second scores and M3S resource selection methods with different cut-off alternatives compared to exhaustive search and state of the art resource selection algorithm baselines for 58 single term queries. Avg shards shows the average number of shards selected for each query. For the M3S, k = 100 unless otherwise specified. * indicates method is within a 10% margin of exhaustive search using a non-inferiority test with p = 0.05. + indicates result is within a 5% margin. 43
Table 8  Search accuracy comparison between 3 L2R models and baselines at two rank cutoffs (T) for two datasets. ▲: statistically significant improvement compared to Jnt, the best resource selection baseline. ∗: non-inferiority to exhaustive search. 50
Table 9  Effectiveness and efficiency of FAST features. ALL uses all features. FAST does not use sample-document features. T: shard rank cutoff. ∗: non-inferiority to exhaustive. 51
Table 10 Performance of L2R-MQT using feature sets constructed with leave-one-out. ‘w/o X’ means the feature was excluded from FAST. Text in bold indicates the lowest value in the column. 52
Table 11 Simulation parameters. 58
Table 12 Datasets and shards used in the experiments, with “M” standing for million, “B” standing for billion, and document lengths measured in words. 61
Table 13 Resource selection parameter settings, based on TREC Terabyte Track and TREC Web Track queries for which relevance judgments are available, as developed by [Kulkarni et al. 2012] and [Aly et al. 2013]. Those investigations used a variety of CSI sizes for Gov2, ranging from 0.5% to 4%. We used 1%, for consistency with CW09-A. The table also presents the average number of shards (using the shard assignments listed in Table 12) selected by the parameter settings.

Table 14 Effectiveness of selective search using various resource selection algorithms on the Gov2 dataset. The “Oracle” shard ranking assumes that the most useful four shards are identified and searched for each query.

Table 15 Effectiveness of selective search using various resource selection algorithms on the CW09-A dataset. The “Oracle” shard ranking assumes that the most useful four shards are identified and searched for each query.

Table 16 Sizes of the query logs, in thousands. The “Train” column indicates training queries used to set parameters. The “Test” column indicates testing queries used for reporting results. Columns “TestW” and “TestM” indicate queries that were sampled starting one week and one month after the Train queries, respectively. Those two query streams are used in Section 7.4.3.

Table 17 Average number of shards selected by Taily and Rank-S for the Test logs, measured using the configurations depicted in Figure 17.

Table 18 Dispersion of per-query processing times for selected configurations of exhaustive, Rank-S and Taily methods, at the last point on each of the corresponding curves in Figure 17. Columns display the 50th, 75th, 99th percentiles, the mean, and the maximum value of individual query processing times, where time is measured in simulation milliseconds. In these experiments, M = 2. Recall that the simulation was tuned to model the costs associated with Indri, a research search engine; a commercial system would likely have lower processing times.

Table 19 Breakdown of the average time spent processing queries, at the last point on each of the corresponding curves in Figure 17. Central queue indicates the percentage of time spent in the central query queue; network indicates the delays caused by network congestion; internal queues indicates time spent in system queues such as a machine’s shard search queue or merge queue; resource selection indicates time spent performing resource selection; and merge is time spent performing the final merge operation. The remaining time not included in these categories was spent in shard search.

Table 20 Average load$_i$ and range (max load$_i$ – min load$_i$), where load$_i$ is the time-averaged CPU load of m$_i$ during the simulation, for two shard allocation policies and a range of query arrival rates, using Taily resource selection, M = 4, B = 1, and S = 4. Results for Gov2 using MQTW Test queries are also shown in Figure 21.
Table 21: Average load \(_i\) and range (max load \(_i\) − min load \(_i\)) of shard search machine utilization as the training data ages, using the Log-based shard allocation policy. Other simulation settings are as described in Table 20. Test queries begin immediately after the training queries. Test\(_W\) and Test\(_M\) queries begin from 1 week and 1 month after the training queries, respectively. The MQT queries do not have timestamps and so were not used in this experiment.

Table 22: Effectiveness of the top submitted TREC runs compared to the runs created from fusing the top 10 submitted runs for each query set with SlideFuseMAP [Anava et al. 2016].

Table 23: Parameter settings for Rank-S and Taily used in the end-to-end selective search system evaluations.

Table 24: Comparison of AURSC against the evaluation of end-to-end systems with various resource selection algorithms. The end-to-end systems used P@1000 to measure the effectiveness of the partitions. Also presented an end-to-end system evaluation using RBR \(k = 1\), a commonly used heuristic in prior work.

Table 25: Comparison of AURSC against a Rank-S end-to-end system at different rank depths.

Table 26: Comparison of AURSC scores and Rank-S end-to-end system evaluation using P@1000, when the pool of desired documents were generated from weaker rankers.

Table 27: Comparison of AURSC and Rank-S end-to-end system evaluation using P@1000, when using varying number of MQT queries. The end-to-end system was evaluated with TREC queries, as usual.
INTRODUCTION

Modern proliferation of data has increased the importance of good information retrieval. Cheaper data storage means that the data collections that must be searched are often large. For example, ClueWeb09, a dataset used for information retrieval research, is composed of a billion web pages and is 5 terabytes in compressed size. Searching large collections requires substantial computational resources. Compounding the problem is the increasing expectation from users that searches return results in seconds or less. Large web search engines typically have sub-second response times.

Producing search results in a timely fashion from large data collections is a challenging task. While a single large, very powerful machine may be able to handle searches of a moderate data collection, these machines are expensive. In addition, it is difficult to scale up this architecture to handle larger collections. Such a hardware setup also has a single point of failure; if the large machine goes offline, the entire search system becomes non-functional.

A better solution to the problem of large-scale search is a distributed search architecture, which is commonly used by commercial search engines such as Google or Microsoft’s Bing. Distributed search is a divide and conquer approach which pools the resources of multiple machines and is set up as follows.

First, the large data collection is divided into many partitions, usually referred to as shards. Shards are typically equal in size. The shards are then distributed equally to the commodity computers so that each computer is only responsible for searching its assigned shard. When a user issues a query to the system, all shards of the data collection are searched in parallel by the many machines. The results of each individual shard are then merged and returned to the user. Figure 1a illustrates this process.

Distributed search has several advantages over a monolithic large computer set-up. By splitting the large collection into small shards, cheap computers are able complete their part quickly and greatly reduce the overall time it takes for the system to respond to the user. This enables the search system to achieve the low latency expected by users. In addition, when a single machine fails, most of the system still remains active and the failed component is easier to replace.

Distributed search is an effective architecture for large organizations that have access to many machines. However, because it searches the entire collection for every query, it has a high total computational cost.

Recently, researchers have begun exploring a new search architecture that significantly reduces the computational cost of large-scale search: selective search [Kulkarni 2013]. Selective search is similar to a traditional distributed search architecture in that a large collection is divided into small shards and spread across multiple machines. However, it is distinguished from traditional distributed search in a few major ways.
The Cluster Hypothesis [van Rijsbergen 1979] states that documents that are similar in content tend to be relevant to the same kind of requests. Selective search uses this idea and partitions the data collection such that documents about similar topics are grouped together. For example, newspaper articles covering elections, forum pages dedicated to discussing politics, and web pages of various politicians may be placed together in a single shard that represents the ‘politics’ topic. Similarly, topical shards may be formed for sports, fashion, etc. In prior research, this topical grouping of the collection was achieved by the K-means clustering algorithm. Once created, these shards are distributed to the available computers.

According to the Cluster Hypothesis, with the topical shards of selective search, it is likely that the most relevant documents to a user’s request reside in a select few shards. Therefore, in contrast to randomly-partitioned distributed search, not all shards are searched when a user issues a query to a selective search system.

A selective search system uses a resource selection algorithm to identify these useful shards. A resource selection algorithm takes a user query and creates a ranked list of the shards it believes are most useful. The selective search system then only searches the shards indicated by the resource selection algorithm. The results of the shards are merged and returned to the user as traditional. The entire pipeline is illustrated in Figure 1b.

A concern of searching only a few shards is the possibility that selective search may miss documents that are relevant to the request and thus become less accurate than searching the entire collection in an exhaustive fashion. However, it was found by Kulkarni [2013] that only a very small percentage of shards was typically needed to achieve similar accuracy to searching the entire collection. This resulted in reducing the computational cost of large-scale search by more than 90%. Compared to a traditional distributed search architecture, selective search is able to return results that are as accurate and as timely, with a fraction of the resources.
Despite the advantages of selective search, the architecture has not yet seen wide acceptance in the academic community or in industry. As a newly introduced architecture, there are important questions about its performance characteristics in a practical setting.

One major concern is whether selective search would be accurate and efficient in various conditions common in the real world. First, trust in the prior results of selective search must be strengthened. Prior work used only one partition per dataset under one specific experimental methodology and it is unknown what the variance of different selective search instances is over different possible collection partitions and experimental conditions. In addition, practical large-scale systems are commonly multi-stage pipelines including other optimizations and re-ranking components with parallel query processing. In such an environment, selective search would likely be deployed as the fast, first-stage retrieval step which can quickly return a list of candidate results. However, the behaviour of selective search in this setting in combination with other components has not been studied in depth.

Another concern is that there is currently a lack of tools to test the efficacy of selective search without implementing an end-to-end system. There is a need for simple, low-cost tools that can help system administrators design and test the best selective search implementation for their needs.

1.1 CONTRIBUTIONS

The purpose of this thesis is to answer outstanding concerns about selective search through thorough analyses and to create novel solutions where the initial approach falls short. It aims to bring selective search into wider acceptance by tackling two goals. First, it provides stronger assurances about the efficiency and effectiveness of selective search in real world settings and solutions to problems that may exist when implementing selective search as a first-stage retrieval system in a practical system. Secondly, this dissertation research provides convenient tools for system administrators to evaluate a selective search system without full implementation, in order to lower the barrier of entry.

1.1.1 Variance of random processes

Selective search contains non-deterministic processes. In the implementation used by prior work, there are up to three random decisions, two during shard formation and another during resource selection. Prior work uses a sampled K-means algorithm to form the topic shards and randomly samples the collection to form the smaller sub-collection and then randomly assigns the initial cluster seeds. An additional source of non-determinism may be present in the resource selection step; sample-based algorithms such as Rank-S search a small, random sample of the total collection.

In most prior work, accuracy and efficiency figures were reported for one instance of a collection partition and one instance of a resource selection database. Due to the randomness involved, it is possible that the performance of selective search was an outlier and a different instantiation of selective search may perform differently. An in-depth exploration of the effects of the random processes is necessary as it is unclear how each individual random decision affects the final results.

This work investigates two major sources of variance in selective search: shard formation and resource selection. One cause of variance was due to poor placement of relevant documents dur-
ing the partitioning process. These critical documents were sometimes placed in shards that contained dissimilar documents. This suggests that better partitioning may reduce variance. When comparing resource selection methods, Taily and Rank-S produced nearly equal variance despite the fact that Rank-S contains an additional random step. When the variance in performances were broken down in a query-by-query fashion, it was discovered that most of the variance of the overall accuracy was the product of a few queries, typically containing rare terms, suggesting that query type may have an impact on variance.

Overall, selective search was stable for most queries on precision-oriented metrics. However, there was higher variance for a small number of rare queries and for recall-oriented metrics, and improvements could be made, especially in the partitioning process.

1.1.2 Performance with dynamic pruning

A criticism of prior work was that selective search may not retain its efficiency advantage over exhaustive search when used with common dynamic postings pruning algorithms. Kulkarni [2013] briefly explored selective search in combination with the term-bounded max-score algorithm, but a deeper investigation with other algorithms had not been accomplished.

Weighted AND (WAND) [Broder et al. 2003] is a low-level search optimization algorithm that enables the system to avoid unnecessary evaluation of documents that would not be highly-ranked in the final result list. By skipping reading and computing scores for these documents, the cost of search is significantly reduced.

Because selective search skews the topical distribution of documents deliberately, it is unknown whether techniques such as WAND will have increased or decreased effects when combined with selective search. Selective search and WAND create efficiency gains by skipping the evaluation of documents. It is possible that the documents which are not normally evaluated by selective search are the very same documents that WAND would skip. In this scenario, there would be no additive efficiency gain by using selective search if one already uses WAND in their system.

This work alleviates this concern by experimenting with a combination of selective search and WAND. It was found that the effectiveness of WAND is enhanced on the top-ranking topic shards, suggesting that grouping similar documents is helpful for WAND. Overall, selective search and WAND were shown to be orthogonal optimization techniques creating independent gains, due to the long posting lists generated by topic clustering; together they create better-than-additive efficiency gains.

This work also experimented with sequential shard evaluation with WAND, where the selected shards were searched one-at-a-time in sequence rather than searching them simultaneously in parallel. By keep track of and passing on the internal evaluation state of WAND from one shard to the next, the total documents evaluated were significantly decreased. This demonstrated the possibility of using tiered retrieval to further decrease the computational cost of search.

1.1.3 Resource selection

In a practical large-scale system that uses a multi-stage retrieval architecture, selective search is most likely to be deployed as the fast first-stage step. In order for this to be feasible, selective search must be able to supply high recall results such that downstream re-rankers are able to see the relevant documents.
One of the most critical components in selective search accuracy is the resource selection algorithm. While existing methods perform well on precision-oriented metrics such as Precision at 10, they fall short of exhaustive search in recall-oriented metrics such as Mean Average Precision. This dissertation presents two new solutions for resource selection: a light-weight method that uses pre-existing statistics, and a more sophisticated method that has higher accuracy.

### 1.1.3.1 Light-weight resource selection

Most resource selection algorithms require building an external database. Term-based algorithms such as Taily collect corpus statistics and build a database containing model parameters for each term. Sample-based algorithms like Rank-S require building an index containing sampled documents from the full collection. This process can be laborious for large collections. Many search systems come with optimization methods which sometimes require similar analyses of the corpus and information for their optimization algorithm. For example, WAND stores the maximum score of each term. It was hypothesized that the information collected could be re-used to produce an effective, light-weight resource selection method that eliminates the need for additional corpus analyses and separate external databases.

Block-Max WAND [Dimopoulos et al. 2013], a variant of the WAND algorithm, collects statistics in finer granularity than WAND to perform dynamic postings pruning more efficiently. In this work, the block max scores collected by Block-Max WAND were utilized to perform resource selection.

When used for single-term queries and searching an equivalent number of shards to two state-of-the-art resource selection algorithms, it was found that the methods based on the block max scores were comparable to exhaustive search with statistical significance, whereas prior state-of-the-art methods were not. Furthermore, it was possible to produce a set of shards that would produce a result set identical to exhaustive search that was close to the size of an oracle-derived set. This work suggests that it is feasible to produce a good shard ranking from the statistics collected by Block-Max WAND and furthermore creates the possibility of creating a stable variant of selective search which closely reproduces the results of exhaustive search.

### 1.1.3.2 Accurate resource selection

While prior work showed that selective search has comparable accuracy to exhaustive search, it was based on a single collection partition of two data sets and the results were reported as a single aggregated mean over four query sets. When a different partitioning and experimental methodology was used, a more complete picture selective search emerged. By presenting the results of the four query sets separately it was found that selective search performed well in some query sets and poorly in others. Averaging the results across all query sets masked this issue. In the poorly performing query sets, resource selection was a major source of error in the system.

It was also found by Dai et al. [2016] that selective search performs worse for recall-oriented metrics than for precision-oriented metrics. This finding motivated additional research into resource selection that is more accurate, especially for deeper metrics such as MAP@1000.

This work presents a learning to rank based approach to resource selection. The advantage of using a learning to rank framework as opposed to multiple binary classifiers [Arguello et al. 2009b; Cetintas et al. 2009] is that it is better suited to selective search, where the number of
resources to search may number in hundreds. Training hundreds of classifiers can take significant computational resources.

This work describes new features suitable for resource selection. We also show that human-judged relevance data is not necessary for training the model. By assuming that selective search wants to mirror exhaustive search (at a lower computational cost), we can generate training data by searching the entire corpus and using the exhaustive search results as gold standard.

The new resource selection algorithm is highly accurate and is statistically non-inferior to exhaustive search even at MAP@1000. In addition, even when the expensive features that require a retrieval from a sample index are removed, the remaining fast features are sufficient to achieve good MAP scores. Learning to rank resources is an efficient and effective method and advances the state-of-the-art in selective search resource selection.

1.1.4 Performance in parallel processing

Initial work on selective search demonstrated the efficiency of the system in an environment where search requests were processed one at a time, i.e. a new query would be processed by the system only when the old is completed and the results returned to the user. However, in a practical, large-scale selective search system with multiple available machines, it is desirable to process search requests in parallel in order to maximize the usage of available computational resources. Therefore, selective search efficiency in parallel environments must also be considered.

A concern that arises from a system which handles parallel requests is a potential load imbalance caused by the unequal popularity of shards. Because selective search shards are topically focused, it is likely that some shards may be more popular than others. For example, a shard with a sports focus may be selected more frequently than a shard about information retrieval research. Difference in topic popularity can lead to an uneven computational load amongst the machines hosting the shards and reduce system throughput. Note that randomly-partitioned distributed search does not have this potential load balance problem because all shards are searched for every query and due to the random distribution of documents to shards, each shard is expected to do the same amount of work.

One contribution of this work is a software simulator which was used to conduct analyses on different configurations of selective search, including the parallel processing environment described above. The simulator was tuned to model the behavior of selective search under many different hardware configurations. The simulator enabled the exploration of a wide range of system configurations that would have otherwise been infeasible due to time and resource constraints. In particular, it is flexible enough to model selective search architectures as well such as traditional distributed search.

The simulator is a valuable tool for those seeking to evaluate or implement a selective search system as their search architecture. Due to the parameterization of the simulation, a few quick measurements of a computing environment can produce an immediate comparison between an existing system and a selective search implementation. It also allows the implementor to prototype and evaluate variations of selective search configurations without the expense of creating multiple real implementations.

This dissertation presents the results of the detailed analyses of the performance characteristic of selective search under various configurations. Using the simulator and an improved experimental methodology, the efficiency improvements reported by prior work were validated under more realistic conditions with some caveats. We found that selective search may need to search
more shards than previously reported in order to achieve accuracy equivalent to searching the entire collection. Also, the experiments revealed that load imbalances are present in a naive implementation of selective search due to unequal shard popularity and the cost of resource selection, particularly for sample-based algorithms. However, these can be corrected by using training queries to guide the layout of shards.

Further experiments showed that the result of the training queries are durable; the configuration performed efficiently even a month later. Additional experiments with different hardware configurations indicated that selective search can be a competitive architecture even at a larger scale with many machines. The results of this work asserted the efficiency of selective search over traditional distributed search at both small and larger scales.

1.1.5 Partition evaluation

There are many ways to partition a document collection for selective search to create shard maps, a mapping of documents to shards. Prior work has used K-means, source-based clusters [Kulkarni and Callan 2010a], and query-biased partition schemes [Dai et al. 2016]. The effectiveness of the shard map was then evaluated by running queries with relevance judgements on an end-to-end selective search system using the shard map. The accuracy of the final results of the end-to-end system using traditional IR metrics was used as proxy for the effectiveness of a shard map.

This method has a few issues. To evaluate an end-to-end selective search system, a resource selection algorithm must be used. Given that resource selection has a large impact on the accuracy of selective search, different choices of resource selection algorithms and their parameters may impact the accuracy results more so than the shard map. In addition, resource selection requires additional corpus analyses and the creation of resource selection databases (term-based methods) or centralized sample indexes (sample-based methods). To do so for many different shard maps is slow and cumbersome and makes evaluating many shard maps difficult.

This work presents Area Under Recall Curve (AUReC), a new metric for evaluating shard maps. For a given query, based on a desired set of documents that should be retrieved for that query, AUReC builds a curve describing the relationship between the amount of shards searched to the maximum possible recall of the desired set of documents from the searched shards. The area under this curve is the effectiveness of a shard map. The set of desired documents is built from exhaustive retrieval, thus AUReC does not require an end-to-end evaluation of selective search or manual gold-standard judgements. Because the shards are ranked based on the overlap with the desired documents and the area under the curve considers the full set of shards, AUReC does not require choosing and implementing a resource selection algorithm.

Experiments show that AUReC is highly correlated with the end-to-end evaluation of a full selective search system using a variety of resource selection algorithms, while being easier to implement and computationally inexpensive. AUReC is also a robust, stable method, and better able to distinguish finer differences between shard maps than end-to-end evaluation due to having more data points; relevance assessments can be sparse. Experimentation show that AUReC is high correlated with several traditional IR metrics including precision, MAP and NDCG at different depths. Finally, when the sample retrieval was generated from different queries than the relevance assessed evaluation queries for end-to-end systems, the resulting AUReC scores remained highly correlated. More queries produced better correlated results, with diminishing returns.
AUREC is a robust, stable metric that enables quick, easy comparisons of a large number of shard maps. This allows system administrators to quickly identify the best set up for their selective search systems. In addition, it supplies researchers with an optimization function for generating shard maps, empowering research in more efficient and accurate topic-based partitioning of large corpora.

1.2 WIDER IMPACT

This dissertation demonstrates that selective search works well in realistic, large-scale production environments as well as in smaller lab settings. In addition, the dissertation provides tools to prototype and evaluate selective search without implementing the full system, paving the way for easier adoption of the system. The adoption of selective search benefits both industry and academia alike. In industry, selective search is a cost and energy saving alternative to traditional distributed search. In academia, the low computational cost of large-scale search delivered by selective search promotes not only information retrieval research, but also other research fields that often use search as a basic step in a more complex task, such as text mining and question answering tasks.

1.3 FUTURE DIRECTIONS

For wider acceptance of selective search, two research directions are necessary. First, selective search must confirm that it is able to easily and successfully fit into existing search and text analysis systems. Second, selective search must be expanded to handle other important domains beyond web documents. These directions are suggested in brief below.

1.3.1 Selective search in pipelines

The work in this dissertation has given confidence that selective search has sufficient recall to be used as a first-stage retrieval system in more complex text analysis pipelines. The logical next step is continued research into how selective search can fit into and benefit existing text pipeline systems. First, a detailed investigation must be conducted on how selective search changes the efficiency and accuracy of a complex pipeline which may contain other components such as caching, tiering, and machine learning re-rankers. Then, by studying the interactions between these components, we may be able to better tailor selective search to improve the entire pipeline. For example, being able to control the recall of selective search with strict guarantees would be useful, as is being able to push certain filtering tasks up to the selective search level for less work to do downstream in expensive, complex components.

1.3.2 Selective search in hierarchical corpora

While selective search research has been conducted with web corpora, structured, hierarchical data is common, especially in business-critical applications such as product search in e-commerce. Extending selective search beyond the limited bag-of-words model and using the additional information to inform its clustering, resource selection, and result merging would better adapt the architecture to these domains and ideally increase efficiency and effectiveness.
1.3.3 Selective search in streaming datasets

Rapidly changing, online data sources such as Twitter and Instagram are becoming an increasingly important source of information on the web. The high volumes and the varied topics seen in such streams make it ideal for selective search. These domains have unique challenges that must be addressed for selective search to be effective. The system must be able to handle constant updates, including assigning them to the correct shard. It also needs detect and respond to fast changes in topic and queries, dynamically load balancing as necessary. Finally, selective search must be extended to be aware of the temporal nature of this domain, which has a large impact on its usage and accuracy in search.

1.4 Organization

This dissertation presents the results of research done to answer questions on selective search effectiveness and efficiency and outlines further research to be done on selective search. Chapter 2 outlines prior related work. The dissertation begins with stronger assurances of selective search effectiveness and efficiency under different conditions. Chapter 3 presents an exploration of the random decisions in selective search and their effect. Chapter 4 investigates the interactions between selective search and WAND optimization techniques. Then, the shortcoming of selective search recall is addressed through developing stronger resource selection algorithms, allowing selective search to be feasibly used as a first-stage retrieval system. Chapter 5 explores the possibility of using statistics collected by Block-Max WAND to perform resource selection. Chapter 6 presents learning to rank resources, a new resource selection method. Finally, two powerful tools to evaluate selective search are presented. Chapter 7 presents the research results on load balancing issues of selective search and provides a simulator tool for modelling selective search. Chapter 8 presents AUReC, a new metric for quickly evaluating shard maps. Chapter 9 concludes, summarizes the contributions and impact of the dissertation, and outlines future directions of research.
SELECTIVE SEARCH

Introduction

1. INTRODUCTION

Selecting the words from prior work in a broad range of research areas including traditional
distributed search, cluster-based retrieval, and federated search research.

1.1 TRADITIONAL DISTRIBUTED SEARCH

Modern large-scale retrieval systems often search corpora that are much larger than what one ma-
chine can fit. Therefore, by necessity, large-scale search engines have a distributed architecture in
which the document collection is partitioned, and each partition is assigned to a distinct machine.
A collection may be partitioned by terms or by documents.

In term-based index partitioning, each partial index is responsible for a non-overlapping subset
of the terms in the vocabulary [Moffat et al. 2006; Lucchese et al. 2007; Zhang and Suel 2007; Cam-
bazoglu et al. 2013]. When the collection is searched, only indexes that contain the query terms
are searched. Because queries are typically short, only a few of the term indexes are required for
each query, allowing multiple queries to be evaluated in parallel. This style of index has largely
fallen out of favor because it is prone to load imbalances [Moffat et al. 2007]. Cambazoglu et al.
[2013] provide more details of term-based partitioning approaches.

In document-based partitioning, each partial index is responsible for a non-overlapping subset
of the documents in the collection. There are two major approaches to creating the document
subsets: tiering and sharding.

Tiering creates partitions that have different priorities in order to reduce the total computational
cost of search. Initially, only the top priority partition, or the top tier, is searched. If the results
are deemed insufficient, lower tiers are searched as necessary. Many different strategies exist for
deciding when to cascade to lower tiers [Risvik et al. 2003; Baeza-Yates et al. 2009b; Cambazoglu
et al. 2010; Brefeld et al. 2011]. Tiers can be defined based on various document characteristics.
Geographically defined tiers place documents that are local to the area in physically closer ma-
chines [Cambazoglu et al. 2010; Brefeld et al. 2011]. One can also create tiers using estimates of
the quality of documents, which are calculated a priori using query independent features such as
spam scores or term statistics [Risvik et al. 2003]. Historical query logs can be used to create tiers
of the most popularly retrieved documents [Risvik et al. 2003]. In tiering, the order in which the
tiers are searched is the same across all queries.

The alternative approach, sharding, creates partitions or shards that all have the same priority.
Sharding is used to pool the resources of many machines to divide-and-conquer the costs of
search in order to minimize latency. For each query, all shards are searched in parallel [Puppin
et al. 2006; Badue et al. 2007]. Documents are usually assigned to shards by source, randomly, or
in a round-robin approach. Typically, it is desirable for all shards to have similar characteristics
in order to minimize load imbalances.
Tiering improves throughput and sharding reduces latency; thus, these complementary methods can be combined to improve in both areas. For example, the corpus might be divided into tiers based on document popularity or authority, and then each tier divided into shards, with the shards distributed across a cluster of computers for lower latency search within that tier [Orlando et al. 2001; Barroso et al. 2003; Baeza-Yates et al. 2007, 2009a; Brefeld et al. 2011; Francès et al. 2014].

A range of work has explored the efficiency of sharded search systems, covering topics including reducing the communications and merging costs when large numbers of shards are searched [Cacheda et al. 2007b]; load balancing in mirrored systems [Macdonald et al. 2012; Freire et al. 2013]; query shedding under high load to improve overall throughput [Broccolo et al. 2013]; and query pruning to improve efficiency [Tonellotto et al. 2013]. Other work focuses on addressing the load imbalances that arise when term-based indexes are used and selective replication of frequently-accessed elements [Moffat et al. 2006, 2007]. A common theme is that when a tier is searched, all of its shards are searched, that is, an exhaustive search strategy is employed.

2.2 CLUSTER-BASED RETRIEVAL

Selective search is a type of cluster-based retrieval architecture [Croft 1980; Voorhees 1985; Griffiths et al. 1986; Willett 1988]. Cluster-based retrieval systems organize the corpus into hierarchical or flat clusters during indexing. When a query is received, clusters are selected by traversing the hierarchy or comparing the query to cluster centroids to identify the most similar cluster. Small-scale systems may return all of the documents in selected clusters; larger-scale systems rank documents in the selected clusters and return just the highest-scoring documents.

Classic cluster-based retrieval systems produce many small clusters. For example, Can et al. [2004] used 1,640 clusters that each contained an average of 128 documents. When clusters are small, many must be selected to maintain acceptable accuracy. Can et al. [2004] searched 10% of the best-matching clusters (164 clusters), a heuristic threshold that was also used in earlier investigations. Similarly, Croft [1980] used over 400 clusters for 1,400 documents.

Selective search systems produce and search a smaller number of large clusters. For example, Kulkarni [2013] and Aly et al. [2013] used clusters that contained approximately 500,000 documents, and queries typically searched 3-5 clusters.

Classic cluster-based retrieval systems also used a single index to store the entire corpus. Efficiency improvements are obtained by storing cluster membership information in inverted list data structures and grouping postings by cluster membership so that large portions of an inverted list may be skipped during query processing [Can et al. 2004; Altingovde et al. 2008]. This architecture must bear the I/O costs of reading complete inverted lists, and the computational costs of processing them (albeit, efficiently). Can et al. [2004] note that storing each cluster in its own index would reduce computational costs, but incur prohibitive I/O costs (primarily disk seeks) due to the large number of clusters selected. The selective search architecture stores each cluster in its own index, but avoids the I/O costs by selecting a very small number of clusters for each query.

There are other differences between classic cluster-based retrieval and selective search. For example, selective search uses more sophisticated methods of determining which clusters match each query. Kulkarni and Callan [2015] provide a detailed description of cluster-based retrieval and its relationship to selective search.
2.3 FEDERATED SEARCH

Sharded indexes and their search engines are a special case of federated search systems. Typically, federated search systems integrate multiple collections or search systems. Common types of federated search systems include aggregated or vertical search engines, peer-to-peer search networks, and meta-search engines [Shokouhi and Si 2011]. Vertical search combines the results of specialized search engines for different domains or media (e.g., images, maps, news) to provide a single unified search portal [Arguello et al. 2009b]. In peer-to-peer search networks, each peer node contains some subset of documents and search is conducted by forwarding queries to appropriate nodes [Lu and Callan 2006]. Finally, meta search engines do not index any documents, but combine the results of multiple search engines controlled by different organizations by forwarding a query to the various systems and merging the results appropriately [Selberg and Etzioni 1997].

A common task shared by federated search systems is determining to which of the underlying search services should a given query be forwarded. Normally the goal of federation is to send queries to as few of the underlying search services as possible, so a resource selection algorithm is used. Three types of resource selection have been proposed: term-based, sample-based, and classification-based algorithms.

Term-based algorithms treat each search service as a bag of words. Common document ranking algorithms can be adapted to the task of ranking resources or services; GLOSS [Gravano et al. 1999], CORI [Callan 2000], and the query likelihood model [Si and Callan 2004a] are examples of this approach. Algorithms developed specifically for resource ranking usually model the distribution of vocabulary across search services [Yuwono and Lee 1997; Hawking and Thistlewaite 1999; Aly et al. 2013]. Term-based algorithms typically only support bag-of-words queries, but a few also support corpus-level or cluster-level preferences, or Boolean constraints [Gravano et al. 1999; Xu and Croft 1999; Callan 2000; Liu and Croft 2004].

Sample-based algorithms represent each search service using a sample of its contents. Samples from all services are combined into a centralized sample index, or CSI. When a query is received, the CSI is searched, and each top-ranked document found in the CSI is treated as a vote for the resource from which it was sampled. Many different methods for weighting votes from different resources have been described [Si and Callan 2003, 2004b, 2005; Shokouhi 2007; Paltoglou et al. 2008; Thomas and Shokouhi 2009; Kulkarni et al. 2012].

Classification-based algorithms represent each search service using a model learned from training data. The features used might include the presence of specific words, the scores of term-based and sample-based algorithms, and the similarity of the query to a resource-specific query log [Arguello et al. 2009a; Kang et al. 2012].

During most of the last decade, sample-based algorithms have been regarded as being a little more effective than term-based algorithms [Shokouhi and Si 2011]; however, recently Aly et al. [2013] argued that Taily, a new term-based algorithm, is more effective than the best sample-based algorithms. Term-based and sample-based algorithms are effective when the search engines are mostly homogeneous. Both types of algorithm are unsupervised, meaning that training data is not required. Supervised classification-based algorithms can be more effective than unsupervised methods; however, their main advantage is their ability to select among heterogeneous resources (for example, “vertical” search engines), and exploit a wide range of evidence. Resource selection algorithms have also been applied to a variety of other tasks, including blog search and desktop search [Seo and Croft 2008; Elsas et al. 2008; Kim and Croft 2010].
The three resource selection algorithms commonly used with selective search are ReDDE, Taily, and Rank-S.

ReDDE, a sample-based algorithm, uses a centralized sample index (CSI) [Si and Callan 2003]. The query is run against the CSI and the documents retrieved from it act as votes towards the shards that they were sampled from. Let \( n \) be the set of documents retrieved from the CSI, and \( n_R \) represent the subset of \( n \) which were sampled from shard \( R \). Then, the score \( s_R \) for shard \( R \) is:

\[
s_R = n_R \times w_R
\]

where \( w_R \) is a shard weight calculated from the size of the shard and the size of its sample.

Rank-S is another sample-based algorithm from the SHiRE family of algorithms [Kulkarni et al. 2012]. In Rank-S, the scores of documents retrieved from the CSI are decayed exponentially and then treated as votes for the shards the documents were sampled from. The exponentially-decayed vote of a document for its parent resource is computed as:

\[
\text{Vote}(d) = \text{Score}_{CSI}(d) \times \text{base}^{-\text{Rank}_{CSI}(d)}
\]

where \( \text{Score}_{CSI}(d) \) and \( \text{Rank}_{CSI}(d) \) are the document score and rank obtained by searching the CSI; and \( \text{base} \) is a configurable parameter. The scores of the documents naturally converge to zero due to the decay. Resources with a total score above 0.0001 are then selected, as originally described by Kulkarni et al. [2012].

Taily assumes that the distribution of document scores for a single query term is approximated by a Gamma distribution. The allocation algorithm uses two parameters, \( n_c \) and \( v \), where \( n_c \) is roughly the depth of the final ranked list desired, and \( v \) is the number of documents in the top \( n_c \) that a resource must be estimated as contributing in order to be selected. Term scores are calculated from simple corpus statistics and fitted to a Gamma distribution for each shard-term pair. Taily’s resource selection database records these fitted Gamma distributions for each term \( t \) in resource \( i \), describing the term’s score distribution in that shard. Gamma distributions are represented by two parameters, the shape parameter \( k_{ti} \) and the scale parameter \( \theta_{ti} \). At query time, the cumulative distribution function of the Gamma distribution is used to estimate the number of documents from each resource that will have a score above a threshold derived from \( n_c \). Each resource that provides \( v \) or more documents is selected [Aly et al. 2013].

When a query is received, Taily looks up two floating point numbers for each index shard per query term, whereas Rank-S must retrieve an inverted list for each query term. Taily’s computational costs are linear in the number of index shards \( |S| \), and nearly identical for each query of length \( |q| \). The computational costs for Rank-S vary depending on the size of the CSI and the document frequency (df) of each query term in the CSI. The Rank-S approach is more efficient only when \( df_t < |S| \) for query term \( t \). For most applications Taily is substantially more economical than Rank-S [Aly et al. 2013].

2.4 Selective Search

Selective search is a cluster-based retrieval system that combines ideas from conventional distributed search and federated search architectures [Kulkarni and Callan 2010a,b; Kulkarni 2013; Kulkarni and Callan 2015]. The corpus is divided into topic-based shards and these are distributed across the processing resources. When a query arrives, a resource selection algorithm identifies a subset of shards that are judged likely to contain most of the relevant documents. The selected shards are then searched in parallel, and their answer lists merged to form an overall answer. The
The key argument in favor of this approach is that if only a few shards are searched for each query, each machine can service query traffic for multiple shards, with a direct consequential reduction in processing cost compared to searching all of the shards for every query.

Prior research established selective search as a viable architecture and demonstrated its advantages in small laboratory settings. Kulkarni and Callan [2010a] compared partitioning a collection into shards by topic against partitioning by source or partitioning randomly. They showed when documents are topically allocated into shards, queries were able to be accurately answered by searching only a few shards rather than searching every shard, introducing selective search as an architecture for the first time.

As a critical component of selective search, the shard selection process in selective search has received much research attention in past work. Kulkarni et al. [2012] noted that in the selective search system presented by Kulkarni and Callan [2010a], the ReDDE resource selection algorithm was used to rank the shards then the same shard cut-off (i.e. the number of shards to search) was applied for all queries. Kulkarni et al. presented the SHiRE family of sample-based resource selection algorithms which create query dependent, dynamic shard cut-offs. Rank-S, which was previously introduced, is a member of the SHiRE family. The authors demonstrated that accuracy and efficiency improvements can be realized by jointly formulating the shard ranking and cut-off estimation problem. The authors compared SHiRE algorithms against existing resource selection baselines and demonstrated that they are more effective and efficient due to the dynamic shard cut-off produced by SHiRE, which creates significant efficiency savings over static cut-offs by searching less shards when possible.

Aly et al. [2013] presented a new term-based resource selection for selective search, Taily. The authors showed that sample-based resource algorithms such as ReDDE and the SHiRE family of algorithms can have significant efficiency costs because it requires a retrieval from a centralized sample index (CSI), the cost of which is linear to the number of documents in the CSI that contain the query terms. Taily is term-based and thus the cost is bounded by the number of shards in the selective search system, which is typically much smaller than the cost incurred by a retrieval from the CSI. Aly et al. compared Taily against sample-based and term-based baselines and demonstrated that it is as effective as prior algorithms while being more efficient, especially in latency.

Kulkarni [2015] focused on the shard cut-off estimation aspect of resource selection and developed ShRkC, a query and metric dependent cut-off estimator that reduces the search costs of a selective search system without degrading efficiency. The paper formulates the cut-off estimation as a supervised regression problem, using training queries to train a different regression model for each metric of interest. The trained models were used to estimate cut-offs for a ReDDE shard ranking and were able to search less shards on average while maintaining similar effectiveness as baseline methods.

Dai et al. [2016] studied shard creation, another component of selective search important for accuracy, and presented an improved partitioning method for creating topical shards. The QInit step clusters terms from training queries to initialize cluster seeds. During the clustering step, the QKLD similarity function places more weight on terms that are important in the training query logs when calculating document to centroid similarity. In combination, the QInit-QKLD method creates more evenly sized shards that are better aligned with the topics in the query traffic and deliver higher accuracy.

The efficiency of selective search system as a whole has seen comparatively less attention than the problems of resource selection and accuracy. Kulkarni and Callan [2015] extended the work of Kulkarni and Callan [2010a] and conducted a more detailed examination of selective search as
an architecture with stronger exhaustive search baselines that use a distributed search model. The paper presented a new variation of K-means, size-bounded sample-based K-means, that creates more evenly sized shards and studied its effects on selective search efficiency. The paper also studied how selective search behaves in conjunction with term-bounded max score, a dynamic optimization method; how it behaves with queries of different lengths; and its effectiveness when different sizes of CSI are used.

Chapter 6 of Kulkarni [2013] implemented a proof-of-concept selective search system over two machines and conducted experiments which confirmed the efficiency advantages of selective search over exhaustive search.

Overall, past work in selective search was largely motivated by increasing selective search accuracy and studying its efficiency in resource-scarce environments. Prior work successfully established selective search as a viable distributed search architecture and demonstrated that it can deliver exhaustive-equivalent accuracy in shallow metrics and produce significant efficiency gains over exhaustive search under small-scale laboratory settings.

In the following subsections, the evolution of the experimental methodologies of extant selective search research are elaborated in more detail to serve as a starting point for future research.

### 2.4.1 Datasets

The most common datasets used with selective search are the Gov2, ClueWeb09 Category A English (CW09-A) and ClueWeb09 Category B (CW09-B) datasets. Gov2 is a web collection containing 25 million web pages crawled in early 2004 from .gov domains\(^1\). ClueWeb09 Category A English dataset is the English document subset of ClueWeb09 and is a web collection of 500 million web pages crawled in January and February of 2009. CW09-B is the first 50 million English documents of Category A\(^2\).

The Gov2 dataset was used consistently by all prior selective search research and past research used queries provided by the TREC Terabyte Track 2004–2006, 150 topics with graded relevance judgements.

The CW09-B dataset was also used in most papers, excluding Kulkarni [2015]. The larger CW09-A dataset was used by about half of past papers [Kulkarni and Callan 2010a, 2015; Kulkarni 2015; Aly et al. 2013]. While most papers used the datasets as provided, Aly et al. [2013] used the Waterloo Fusion spam scores [Cormack et al. 2011b] to filter out documents whose spam scores were 50% or higher in CW09-A, stating that the exhaustive search accuracy of the unfiltered dataset was too low to provide good signals about the efficacy of selective search. This assertion is supported by other papers which used the dataset, which report MAP values low as 0.06 [Kulkarni and Callan 2015; Kulkarni 2015]. CW09-B does not share these problems as the first 50 million English documents contain many high quality documents from Wikipedia and spam filtering for higher effectiveness is not necessary.

The ClueWeb09 datasets were used with queries provided by the TREC Web Track. The number of queries used vary. Earlier papers used less of the query set, with Kulkarni and Callan [2010a] using 50 topics from the 2009 iteration of the Web Track. Kulkarni et al. [2012], Kulkarni et al. [2012] and Aly et al. [2013] used 100 topics from Web Track 2009–2010. Others [Kulkarni and Callan 2015; Kulkarni 2015; Dai et al. 2016] used 200 topics from 2009–2012. The queries used with ClueWeb09 are particularly important when interpreting the accuracy results because selective

---

\(^1\) [http://ir.dcs.gla.ac.uk/test_collections/gov2-summary.htm](http://ir.dcs.gla.ac.uk/test_collections/gov2-summary.htm)

\(^2\) [https://www.lemurproject.org/clueweb09.php](https://www.lemurproject.org/clueweb09.php)
search does particularly well with the 2010 queries compared to exhaustive search, but does particularly poorly with the 2011 queries [Kim et al. 2017; Dai et al. 2016]. Thus, if a paper’s query set contains the 2010 queries but not the queries from later years, the accuracy of selective search compared to exhaustive search may be misleading. In summary, the most modern practices with the ClueWeb09 dataset are to filter spam documents if using CW09-A and to use the full 200 topics from the 2009–2012 years of the TREC Web Track.

2.4.2 Shards

Prior selective search work used a wide variety of shard maps, i.e. a mapping of documents to shards. Kulkarni and Callan [2010a] created the first set of shard maps created for selective search using a sampling-based variation of K-means. A small percentage of the dataset is randomly sampled then K-means is run on the sample using the symmetric version of negative Kullback-Liebler divergence as the similarity function. Afterwards, the rest of the dataset is projected to the nearest cluster centroid. The paper used a 0.1% sample for CW09-A and a 1% sample for Gov2 and CW09-B. K was set such that the shards contain about 500,000 documents on average, equating to 50 shards for Gov2, 100 shards for CW09-B and 1,000 shards for CW09-A. This shard map was used by several papers [Kulkarni et al. 2012; Aly et al. 2013; Kulkarni 2015].

Kulkarni [2013] noted a flaw of the above shard maps, which is that most of the shards were of very skewed sizes, with most shards being much larger or smaller than the target average shard size of 500,000 documents. To create more evenly-sized shards, she presented size bounded sampling-based K-means (SB^2 K-means). In SB^2 K-means, a split phase splits up large shards and a merge step combines small shards together. The process created 208 shards for Gov2, 92 shards for CW09-B, and 807 shards for CW09-A. Kulkarni and Callan [2015] presents the same process in a journal publication, but define 50 shards for Gov2 instead.

The most recent advance in selective search shard creation was presented in Dai et al. [2016] where historical query logs are used to inform the topics created in the clustering step. A note of interest is that the shard maps presented in this paper use smaller shards for Gov2 than prior work as they resulted in more accurate results.

The research in this dissertation was developed in parallel to the development of improved selective search shard maps and uses different shard maps depending on the state-of-the-art at the time of the work. This process of deciding between many available shard maps partially inspired the research in Chapter 8.

2.4.3 Resource selection

Resource selection has been a major focus in prior research, with many papers being devoted to more accurate or efficient resource selection [Kulkarni et al. 2012; Aly et al. 2013; Kulkarni 2015]. The two most commonly used resource selection algorithms in prior selective search research are ReDDE [Kulkarni and Callan 2010a, 2015; Kulkarni 2015; Kulkarni et al. 2012] and Rank-S [Aly et al. 2013; Dai et al. 2016]. Both methods share an important parameter, the size of the centralized sample index (CSI). Earlier papers used a larger CSI size, of up to 4% of the entire collection [Kulkarni et al. 2012], which delivers more accurate results. However, Aly et al. [2013] showed that the resource selection using such large CSIs can be costly and have argued for smaller CSIs. Later work [Kulkarni and Callan 2015; Kulkarni 2015; Dai et al. 2016] used a 0.5% sample...
for their CSI and this setting is followed in this dissertation. Of the two common methods, Rank-S is more modern and was shown to be more accurate and is used in this dissertation.

Aly et al. [2013] presented Taily and argued that it is more accurate and efficient than Rank-S. As Taily is term-based and represents a different family of resource selection algorithms than Rank-S, this dissertation uses both Taily and Rank-S in conjunction so that the results are broadly applicable.

2.4.4 Evaluation of accuracy

Selective search accuracy has been measured by scoring its result list using queries with relevance judgements and comparing these scores against exhaustive search baseline scores. Standard information retrieval metrics are used, such as Precision, Normalized Discounted Cumulative Gain (NDCG), and Mean Average Precision (MAP). Measured at lower ranks such as 10 or 30 (typically abbreviated as P@10, NDCG@30 etc.) the metrics measure precision. At deeper ranks (i.e. NDCG@100, MAP@1000 etc.), the metrics are recall-oriented. Because selective search only searches and returns a small portion of the index, it is expected to perform better in precision-based metrics than recall-oriented metrics.

Selective search papers have demonstrated that it is as accurate as exhaustive search baselines, but are a few caveats to this claim. The first selective search paper [Kulkarni and Callan 2010a] used precision-oriented metrics such as P@10, P@20, and P@30 to evaluate selective search, where it successfully met exhaustive search baselines. However, its performance in recall-oriented metrics was not measured. Later papers reported recall-oriented metrics such as MAP@1000 but often had difficulty meeting exhaustive search results, especially in Gov2 [Kulkarni and Callan 2015; Dai et al. 2016; Aly et al. 2013]. Finally, when comparing an exhaustive search system (that has a mean of $\mu_A$) to a selective search system (that has a mean of $\mu_B$), prior works use $H_0 : \mu_A = \mu_B$ (i.e. “these two systems have the same mean”) as the null hypothesis. This is mathematically problematic; failing to reject the null hypothesis does not mean that the null hypothesis is true. That is, one cannot conclude that by failing a typical paired t-test or permutation test that selective search is equivalent to exhaustive search. A more theoretically sound alternative which uses the null hypothesis of $H_0' : \mu_A - \mu_B \geq \delta$ is presented in Chapter 5.

2.4.5 Evaluation of efficiency

Studies of selective search efficiency determined computation cost in several different ways. Initially, Kulkarni and Callan [2010a] used the proportion of documents in the collection that were present in the selected shards as the cost, using this as an estimate of the total amount of work done by selective search. This idea was improved in Kulkarni et al. [2012], where rather than counting all the documents in a selected shards, only documents that contained one of the query terms contributed to the cost.

Aly et al. [2013] observed that resource selection significantly added to the computation cost of selective search and added a term representing resource selection term to the cost equation. In addition, to the total cost metric, Aly et al. proposed an alternate metric that combined the cost of resource selection and cost of the most expensive shard. This loosely represents the lowest possible latency of selective search that can be achieved in a parallel setting if there was one machine for every selected shard and resource selection database. These metrics are used by most recent papers [Kulkarni and Callan 2015; Dai et al. 2016; Kulkarni 2015, 2013].
Kulkarni [2013] implemented a proof-of-concept selective search system that was distributed over two machines and measured the total execution time as the cost. This system was used to conduct the first preliminary explorations into the throughput of selective search under a parallel query processing environment. In addition, Kulkarni presented a further refinement to the cost metric by counting the total number of postings retrieved from disk rather than counting the number of documents. Kulkarni showed that this Cost-in-Postings metric is better correlated with actual execution times than the Cost-in-Documents used by most prior work, especially for larger datasets. This dissertation uses the Cost-in-Postings when measuring total computation cost. The details of measuring selective search efficiency under parallel environments are given in Chapter 7.
VARIANCE

As defined by prior research, selective search has several non-deterministic steps. Kulkarni [2013] used sample-based K-means clustering to create topical shards, which is non-deterministic due to random sampling and cluster seeds. Some resource selection algorithms, such as Rank-S [Kulkarni et al. 2012], use a random sample of the corpus to decide which shards to search for each query, which introduces additional non-determinism.

Prior research showed that selective search can be as accurate as a typical ‘search all shards’ distributed architecture but at a substantially reduced computational cost [Aly et al. 2013; Kulkarni 2013; Kulkarni et al. 2012]. However, it was based on a single partitioning of the corpus. A different partitioning might yield different results.

We are not the first to notice this problem. Jayasinghe, et al. [Jayasinghe et al. 2014] proposed a linear model to statistically compare non-deterministic retrieval systems, using selective search as an example. Their model took into consideration the multi-dimensional variance in selective search. They found that there can be significant differences between selective search and exhaustive search based on the collection sample used to build the CSI, especially at low sample rates. However, their work focused on testing whether selective search has equivalent mean effectiveness as a baseline search architecture. In our work, we focus on the variance itself and seek to answer the following research question.

RQ 1 What is the effect of random decisions on selective search accuracy?

We compare results obtained with different partitionings of two ClueWeb datasets and examine the variance of a system across four query sets (Section 3.2.1), and for individual queries (Section 3.2.2).

3.1 EXPERIMENTAL METHOD

We define a system instance to be a partition of a corpus index and its corresponding resource selection database for algorithms such as Rank-S [Kulkarni et al. 2012] or Taily [Aly et al. 2013]. Our system instances were defined by a slight adaptation of a process defined by Kulkarni and Callan [2010a] also used by Aly et al. [2013]. First, 500K documents were randomly sampled from the corpus, K of those documents were selected to be seeds, and K-means clustering was used to form clusters. Second, the remaining documents were projected onto the most similar

This is a lightly-revised version of Zhuyun Dai, Yubin Kim, and Jamie Callan. How random decisions affect selective distributed search. In Proceedings of the 38th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 771–774, 2015. The initial investigation was started by Callan and the experiments were conducted by Dai. I joined the project soon after and guided the direction of the experiments and provided analytical insight. The paper writing was equally shared by the authors.
clusters to form index shards. Third, documents scoring higher than 50\% in the Waterloo Fusion spam scores [Cormack et al. 2011a] were removed and a resource selection index was constructed. The Rank-S resource selection index was formed from a 1\% random sample of each cluster. The Taily resource selection index is created deterministically. Thus, each system instance involved 2\--3 random processes.

The parameters used for Taily and Rank-S were suggested by Aly et al. [2013] and Kulkarni et al. [2012]. They were \( n = 400, v = 50 \) for Taily and \( B = 5, CSI = 1\% \) for Rank-S.

The experiments also tested Relevance-Based Ranking (RBR), an oracle resource selection algorithm that ranks shards by the number of relevant documents that they contain. RBR makes it possible to distinguish between different types of variance. For query \( q \), RBR searched the average number of shards that Rank-S and Taily searched for \( q \).

Experiments were conducted on the ClueWeb09 Category A (CW09-A) corpus, which contains 500 million English web pages, and ClueWeb09 Category B (CW09-B), which contains 50 million English web pages. Queries were from the TREC 2009-2012 Web Tracks: 4 sets of 50 queries.

Parameters were set to produce shards of 500K documents on average. Each CW09-A system instance had 1000 shards; CW09-B system instances had 100 shards.

Figure 2: Accuracy distribution of Rank-S and RBR system instances for the Web Track queries. Mid line in boxes represents median. Outer edges of the boxes represent lower and upper quartiles, while whiskers indicate 1.5 interquartile range. + indicates outliers. Blue dot is the performance of the typical ‘search all shards’ baseline.
3.2 EXPERIMENT RESULTS

The effects of random decisions during partitioning and indexing can be measured across query sets or individual queries. Measuring across query sets provides information about the reliability of metrics such as MAP and NDCG that are typically reported in research papers. Measuring across individual queries provides information about the behavior that people will observe when they use the system. The experiments below provide both types of measurement.

3.2.1 Variability for query sets

The first experiment examined how random decisions affect the ‘average case’ metrics typically reported in research papers. It used three resource selection algorithms: Rank-S, which has a random component; Taily, which is deterministic; and RBR, the oracle algorithm. Ten random partitions were created for each dataset. Thus there were 2 datasets × 10 random partitions × 3 resource selection algorithms = 60 system instances. On average, Rank-S searched 3.91 CW09-A shards and 3.75 CW09-B shards; Taily searched 2.58 CW09-A shards and 2.53 CW09-B shards; and RBR searched 3.13 CW09-A shards and 2.80 CW09-B shards.

Retrieval accuracy was evaluated using three metrics: Precision at rank 10 (P@10), Normalized Discounted Cumulative Gain at rank 100 (NDCG@100), and Mean Average Precision at rank 500 (MAP@500) for compatibility with prior research [Aly et al. 2013]. These metrics help capture the performance of selective search at precision-oriented (P@10) tasks and more recall-oriented tasks (NDCG@100 and MAP@500).

The distributions of measurements over each type of system instance and the ‘search all shards’ baseline are shown in Figure 2. Table 1 and Table 2 report the standard deviation and variance coefficient of Rank-S and RBR system accuracy scores. Results for Taily and NDCG@100 were similar to results for Rank-S and other metrics. The standard deviation is a common measure, however it has two flaws: i) values produced by different metrics (e.g., P@10, MAP@500) cannot be compared because it is not dimensionless; and ii) the importance of a given level of deviation depends upon the value of the mean. The variance coefficient (VC) normalizes the standard deviation by the mean, thus eliminating these flaws.

The Relevance-Based Ranking (RBR) was generally more accurate and had lower variance across system instances than Rank-S and Taily, as expected. The one exception was the 2011 query set and the MAP@500 metric; Rank-S had lower variance than RBR under this condition. This exception was caused by a small set of queries. Rank-S had consistently poor accuracy on these queries, whereas RBR had greater accuracy but also greater variance.

The experimental results suggest that there are two sources of variance in selective search: Partitioning, and resource selection. RBR always selects the best shards, thus variance in RBR results is due to differences in partitioning effectiveness. Rank-S and Taily use models (resource selection indices) to make decisions about which shards to select for each query, however those models are necessarily incomplete. The increases in variance for Rank-S and Taily as compared to RBR are due to resource selection errors caused by weaknesses in the models of shard contents.

Selective search instances displayed differing levels of variability across different metrics. P@10 overall had a lower variance coefficient than MAP@500 (Table 1 and Table 2). Similar behavior is observed for Taily instances. This behavior indicates that selective search is more stable at the top of the ranking.
Table 1: Standard Deviation (SD) and Variance Coefficient (VC) of accuracy scores for Rank-S instances.

<table>
<thead>
<tr>
<th>Query Set</th>
<th>CW09-B</th>
<th>CW09-A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SD</td>
<td>VC</td>
</tr>
<tr>
<td>Web Track 2009</td>
<td>$2.22 \times 10^{-2}$</td>
<td>5.06%</td>
</tr>
<tr>
<td>Web Track 2010</td>
<td>$2.11 \times 10^{-2}$</td>
<td>5.57%</td>
</tr>
<tr>
<td>Web Track 2011</td>
<td>$1.52 \times 10^{-2}$</td>
<td>4.84%</td>
</tr>
<tr>
<td>Web Track 2012</td>
<td>$1.79 \times 10^{-2}$</td>
<td>5.44%</td>
</tr>
</tbody>
</table>

(a) P@10

| | | | | |
| Web Track 2009 | $7.77 \times 10^{-3}$ | 10.04% | $6.65 \times 10^{-3}$ | 10.65% |
| Web Track 2010 | $7.56 \times 10^{-3}$ | 7.16% | $3.31 \times 10^{-3}$ | 3.66% |
| Web Track 2011 | $8.52 \times 10^{-3}$ | 8.89% | $7.67 \times 10^{-3}$ | 7.62% |
| Web Track 2012 | $7.17 \times 10^{-3}$ | 6.37% | $7.96 \times 10^{-3}$ | 8.23% |

(b) MAP@500

Table 2: Standard Deviation (SD) and Variance Coefficient (VC) of accuracy scores for RBR instances.

<table>
<thead>
<tr>
<th>Query Set</th>
<th>CW09-B</th>
<th>CW09-A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SD</td>
<td>VC</td>
</tr>
<tr>
<td>Web Track 2009</td>
<td>$1.38 \times 10^{-2}$</td>
<td>2.71%</td>
</tr>
<tr>
<td>Web Track 2010</td>
<td>$1.16 \times 10^{-2}$</td>
<td>2.15%</td>
</tr>
<tr>
<td>Web Track 2011</td>
<td>$1.36 \times 10^{-2}$</td>
<td>3.53%</td>
</tr>
<tr>
<td>Web Track 2012</td>
<td>$5.39 \times 10^{-3}$</td>
<td>1.39%</td>
</tr>
</tbody>
</table>

(a) P@10

| | | | | |
| Web Track 2009 | $3.56 \times 10^{-3}$ | 3.25% | $3.54 \times 10^{-3}$ | 2.97% |
| Web Track 2010 | $3.87 \times 10^{-3}$ | 2.24% | $2.46 \times 10^{-3}$ | 1.35% |
| Web Track 2011 | $1.10 \times 10^{-2}$ | 7.21% | $9.47 \times 10^{-3}$ | 4.40% |
| Web Track 2012 | $3.53 \times 10^{-3}$ | 2.41% | $4.97 \times 10^{-3}$ | 2.91% |

(b) MAP@500
Rank-S is affected by one more random component than Taily, thus it might be expected to have greater variability across system instances. Table 3 shows the variance coefficient of MAP@500 for Rank-S and Taily across ten CW09-A system instances; similar behavior was observed across all metrics for both collections. Taily only had lower variance than Rank-S on 1 of the 4 query sets. These results might be considered surprising. The additional random component does not appear to cause greater instability in Rank-S results. As far as we know, prior research has not investigated the variability of results produced by sample-based and term-based algorithms such as Rank-S and Taily.

3.2.2 Variability for queries

Ideally selective search would provide similar results for a given query regardless of which system instance is used. The second experiment examined variability on a query-by-query basis. We examined the variance of Average Precision of each query across Rank-S and Taily instances on CW09-A and observed some highly variant queries in both Rank-S and Taily. Rank-S had 14 queries with AP standard deviation higher than 0.1; Taily had 17 such high variance queries.

High variance could be due to errors from the partitioning process or errors from resource selection. To discover the source of the variance, a query-by-query experiment was conducted with RBR on the same partitions. RBR does not make resource selection errors but is affected by partitioning errors. Thus, if Rank-S, Taily, and RBR all have trouble with a query, the problem is likely to be partitioning.

Figure 3: Query Average Precision of RBR instances for Web Track queries. Each data point is a box and whisker plot with the edges of the boxes representing the upper and lower quantiles, the mid line the median, and the whiskers the 1.5 interquartile range. + are outliers.
Table 3: Comparison of MAP@500 scores (CW09-A) for Rank-S and Taily instances.

<table>
<thead>
<tr>
<th>Query Set</th>
<th>Mean Rank-S</th>
<th>Mean Taily</th>
<th>Variance Coefficient Rank-S</th>
<th>Variance Coefficient Taily</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Track 2009</td>
<td>0.062</td>
<td>0.065</td>
<td>10.65%</td>
<td>10.98%</td>
</tr>
<tr>
<td>Web Track 2010</td>
<td>0.090</td>
<td>0.087</td>
<td>3.66%</td>
<td>6.89%</td>
</tr>
<tr>
<td>Web Track 2011</td>
<td>0.101</td>
<td>0.084</td>
<td>7.62%</td>
<td>15.67%</td>
</tr>
<tr>
<td>Web Track 2012</td>
<td>0.097</td>
<td>0.085</td>
<td>8.23%</td>
<td>5.36%</td>
</tr>
</tbody>
</table>

Figure 3 shows the average precision (AP) of each query for ten RBR system instances. A notable observation is that most of the queries have stable AP; 177 of the 200 queries had a standard deviation of less than 0.05. However, there were a few queries with very high variance that contributed most of the average variance.

Whether the query is variable or stable remained mostly consistent across RBR, Rank-S, and Taily. While there were some highly variable queries in RBR that were not so in Taily and Rank-S, these were difficult queries (low ‘search all shards’ search accuracy), and had nearly zero MAP@500 in Rank-S and Taily. All easy queries with high variance in RBR also had high variance in Rank-S and Taily. The consistency across RBR, Rank-S and Taily suggests that errors from the partitioning stage are a major source of variance of selective search accuracy; however, most of that variation comes from a small number of queries.

We investigated why some partitions performed much better than others on the high variance queries. One might expect that ‘good’ partitions group relevant documents into fewer shards. However, in all of our partitions, relevant documents were distributed across a small number of shards. Typically the 3 most relevant shards contained more than 60% of the relevant documents, which is consistent with the standards of most prior federated search research. Furthermore, in this experiment, all ten instances retrieved 100% of the relevant documents for 4 of the 5 most variant queries. However, these queries had AP values ranging from values similar to exhaustive search to as much as ten times better than exhaustive search.

An examination of the instances with unusually high AP in the RBR experiment revealed that the relevant documents and the most likely false positives were in different shards. Thus, the combination of partitioning and resource selection ‘filtered out’ non-relevant documents that otherwise would have been ranked highly. Although this behavior might seem desirable, it occurred because the partitioning process incorrectly assigned relevant documents to shards that contained documents that were largely dissimilar to the relevant documents. This poor clustering made it nearly impossible for Rank-S and Taily to identify which shards contained relevant documents due to the overwhelming number of dissimilar documents in the shard. RBR, which knows the number of relevant documents in each shard, had no such problem.

Figure 3 also indicates that different query sets produced different levels of variability. The 2011 query set was notably more unstable than the others. Among the 24 queries with AP standard deviation higher than 0.05, 14 were from TREC 2011. We believe that this difference is due to the other three query sets using topics of medium-to-high frequency, while TREC 2011 used more obscure topics [Clarke et al. 2011]. Our results indicate that the type of query has an impact on not only the average accuracy of a system, but also the variance of a system with random
components. Topic-based partitioning may produce stable results for common topics, but may need improvement for queries about rare topics.

3.3 SUMMARY

Understanding the variance of selective search effectiveness is critical for evaluating and improving selective search. This chapter explored selective search variance and the effects of document collections, resource selection algorithms, and query characteristics. Partitioning and resource selection processes both introduced variance into search behavior, however in our experiments the effect on most queries was not large. Results were more stable at the top of the ranking, but variance at rank 500 was not unreasonable, especially for an architecture that avoids searching most of the index.

Rank-S and Taily produced nearly equal variance, which might be considered surprising given that Rank-S has one more random component than Taily.

Most of the variance observed in our per-query experiments was caused by a small number of queries — typically, rare queries. In some instances the partitioning process correctly grouped relevant documents together, but placed them in unrepresentative shards, which caused poor resource selection. This behavior may indicate the need for partitioning processes more sophisticated than simple k-means clustering.
DYNAMIC OPTIMIZATION

Dynamic pruning algorithms such as Weighted AND (WAND) [Broder et al. 2003] and term-bounded max score (TBMS) [Strohman et al. 2005] improve the computational efficiency of retrieval systems by eliminating or early-terminating score calculations for documents which cannot appear in the top-k of the final ranked list.

But topic-based partitioning and resource selection change the environment in which dynamic pruning is performed, and query term posting lists are likely to be longer in shards selected by the resource selection algorithm than in shards that are not selected. As well, each topic-based shard should contain similar documents, meaning that it might be difficult for dynamic pruning to distinguish amongst them using only partial score calculations. Conversely, the documents in the shards that were not selected for search might be the ones that a dynamic pruning algorithm would have bypassed if it had encountered them. That is, while the behavior of dynamic pruning algorithms on randomly-organized shards is well-understood, the interaction between dynamic pruning and the topic-based shards used by selective search is not. As an extreme position, it might be argued that selective search is simply achieving the same computational savings that dynamic pruning would have produced, but incurs the additional overhead of clustering the collection and creating the shards.

To address these concerns, we investigate the behavior of the well-known Weighted AND (WAND) dynamic pruning algorithm in the context of selective search, considering two research questions:

RQ 2 Does dynamic pruning improve selective search, and if so, why?

RQ 3 Can the efficiency of selective search be improved further using a cascaded pruning threshold during shard search?

4.1 RELATED WORK

Weighted AND (WAND) is a dynamic pruning algorithm that only scores documents that may become one of the current top k based on a preliminary estimate [Broder et al. 2003]. Dimopoulos et al. [2013] developed Block-Max WAND (BM-WAND) in which continuous segments of postings data are bypassed under some circumstances by using an index where each block of postings has a local maximum score. Petri et al. [2013] explored the relationship between WAND-style pruning and document similarity formulations. They found that WAND is more sensitive than

---

1 This is a lightly-revised version Yubin Kim, Jamie Callan, J. Shane Culpepper, and Alistair Moffat. Does selective search benefit from WAND optimization? In Advances in Information Retrieval: 38th European Conference on IR Research, pages 145–158, 2016b. 

[June 14, 2018 at 4:19 – classicthesis version 2.0 ]
BM-WAND to the document ranking algorithm. If the distribution of scores is skewed, as is common with BM25, then WAND alone is sufficient. However, if the scoring regime is derived from a language model, then the distribution of scores is top-heavy, and BM-WAND should be used. Rojas et al. [2013] presented a method to improve performance of systems combining WAND and a distributed architecture with random shards.

*Term-Bounded Max Score* (TBMS) [Strohman et al. 2005] is an alternative document-at-a-time dynamic pruning algorithm that is currently used in the Indri Search Engine. The key idea of TBMS is to precompute a “topdoc” list for each term, ordered by the frequency of the term in the document, and divided by the document length. The algorithm uses the union of the topdoc lists for the terms to determine a candidate list of documents to be scored. The number of documents in the topdoc list for each term is experimentally determined, a choice that can have an impact on overall performance. Kulkarni and Callan [2015] explored the effects of TBMS on selective search and traditional distributed search architectures. Based on a small set of queries they measured efficiency improvements of 23-40% for a traditional distributed search architecture, and 19-32% for selective search, indicating that pruning can improve the efficiency of both approaches.

### 4.2 Experiments

The observations of Kulkarni and Callan [2015] provide evidence that dynamic pruning and selective search can be complementary. Our work extends that exploration in several important directions. First, we investigate whether there is a correlation between the rank of a shard and dynamic pruning effectiveness for that shard. A correlation could imply that dynamic pruning effectiveness depends on the number of shards searched. We focus on the widely-used WAND pruning algorithm, chosen because it is both efficient and versatile, particularly when combined with a scoring function such as BM25 that gives rise to skewed score distributions [Dimopoulos et al. 2013; Petri et al. 2013].

Experiments were conducted using the ClueWeb09 Category B dataset, containing 50 million web documents. The dataset was partitioned into 100 topical shards using K-means clustering and a KL-divergence similarity metric, as described by Kulkarni and Callan [2010a], and stopped using the default Indri stoplist and stemmed using the Krovetz stemmer. On average, the topical shards contain around 500k documents, with considerable variation, see Figure 4. A second par-
4.2 Experiments

Figure 5: Correlation between the number of postings processed for a query and the time taken for query evaluation. Data points are generated from MQT queries using both WAND and full evaluation, applied independently to all 100 topical shards and all 100 random shards. In total, $756 \times 200 \times 2 \approx 300,000$ points are plotted.

A system in which exhaustive “all shards” search is the only way of obtaining effective retrieval. Each shard in the two systems was searched using BM$_{25}$, with $k_1 = 0.9$, $b = 0.4$, and global corpus statistics for $idf$ and average document length. BM$_{25}$ was chosen over the language model scores typically used by Indri because WAND has reduced effectiveness on systems using language model scoring [Petri et al. 2013].

Each selected shard returned its top 1,000 documents, which were merged by score to produce a final list of $k = 1,000$ documents. In selective search, deeper ranks are necessary because most of the good documents may be in one or two shards due to the term skew. Also, deeper $k$ supports learning-to-rank algorithms. Postings lists were compressed and stored in blocks of 128 entries using the FastPFOR library [Lemire and Boytsov 2015], supporting fast block-based skipping during the WAND traversal.

Two resource selection algorithms were used: Taily [Aly et al. 2013] and Rank-S [Kulkarni et al. 2012]. The Taily parameters were taken from Aly et al. [2013]: $n = 400$ and $v = 50$, where $v$ is the cut-off score and $n$ represents the theoretical depth of the ranked list. The Rank-S parameters used are consistent with the values reported by Kulkarni et al. [2012]. A decay base of $B = 5$ with a centralized sample index (CSI) containing 1% of the documents was used – approximately the same size as the average shard. We were unable to find parameters that consistently yielded better results than the original published values.

We conducted evaluations using the first 1,000 unique queries from each of the AOL query log and the TREC 2009 Million Query Track. We removed single-term queries, which do not benefit from WAND, and queries where the resource selection process did not select any shards. Removing single-term queries is a common procedure for research with WAND [Broder et al. 2003] and allows our results to be compared with prior work. That left 713 queries from the AOL log, and 756 queries from MQT, a total of 1,469 queries.

---

2 The values for $b$ and $k_1$ are based on the parameter choices reported for Atire and Lucene in the 2015 IR-Reproducibility Challenge, see github.com/lintool/IR-Reproducibility.
Our focus in this chapter is on the efficiency of shard search, rather than resource selection. To compare the efficiency of different shard search methods, we count the number of postings scored, a metric that is strongly correlated with total processing time [Broder et al. 2003], and is less sensitive to system-specific tuning and precise hardware configuration than is measured execution time. As a verification of this relationship, Figure 5 shows the correlation between processing time per query, per shard, and the number of postings evaluated. There is a strong linear relationship; note also that more than 99.9% of queries completed in under 1 second with only a few extreme outliers requiring longer.

4.2.1 Pruning effectiveness of WAND on topical shards

The first experiment investigated how WAND performs on the topical shards constructed by selective search. Each shard was searched independently, as is typical in distributed settings – parallelism is crucial to low response latency. \( w \), the number of posting evaluations required in each shard by WAND-based query evaluation was recorded. The total length of the postings for the query terms in the selected shards was also recorded, and is denoted as \( b \), representing the number of postings processed by an unpruned search in the same shard. The ratio \( w/b \) then measures the fraction of the work WAND carried out compared to an unpruned search. The lower the ratio, the greater the savings. Values of \( w/b \) can then be combined across queries in two different ways: micro- and macro-averaging. In micro-averaging, \( w \) and \( b \) are summed over the queries and a single value of \( w/b \) is calculated from the two sums. In macro-averaging, \( w/b \) is calculated for each query, and averaged across queries. The variance inherent in queries means that the two averaging methods can produce different values, although broad trends are typically consistent.

Figure 6 and Table 4 provide insights into the behavior of macro- and micro-averaging. Figure 6 uses the AOL queries and all 100 topical shards, plotting \( w/b \) values on a per query per shard basis as a function of the total length of the postings lists for that query in that shard. Queries involving only rare terms benefit much less from WAND than queries with common terms. Thus, the macro-average of \( w/b \) is higher than the micro-average. Micro-averaging more accurately
represents the total system savings, whereas macro-averaging allows paired significance testing. We report both metrics in Table 4. The second pair of columns gives millisecond equivalents of \( \frac{w}{b} \), to further validate the postings-cost metric. These values are micro- and macro-averaged \( \frac{w_t}{b_t} \) ratios, where \( w_t \) is the time in milliseconds taken to process one of the queries on one of the 100 shards using WAND, and \( b_t \) is the time taken to process the same query with a full, unpruned search. A key result of Table 4 is that WAND is just as effective across the full set of topical shards as it is on the full set of randomly formed shards. Moreover, the broad trend of the postings cost ratios – that WAND avoids nearly half of the postings – is supported by the execution time measurements.

### 4.2.2 WAND and resource ranking interactions

The second experiment compares the effectiveness of the WAND algorithm on the shards that the resource ranking algorithm would, and would not, select in connection with each query. The Taily and Rank-S resource selection algorithms were used to determine which shards to search.

---

**Table 4:** Ratio of per shard per query postings evaluated and per shard per query execution time for WAND-based search, as ratios relative to unpruned search, averaged over 100 topical shards and over 100 randomized shards, and over two groups each of 700+ queries. The differences between the Topical and Random macro-averaged ratios are significant for both query sets and both measures (paired two-tailed t-test, \( p < 0.01 \)).

<table>
<thead>
<tr>
<th>Shards searched</th>
<th>WAND postings cost ratio</th>
<th>WAND runtime cost ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Selected</td>
<td>Non-selected</td>
</tr>
<tr>
<td>Taily AOL</td>
<td>3.1</td>
<td>0.32</td>
</tr>
<tr>
<td>Taily MQT</td>
<td>2.7</td>
<td>0.23</td>
</tr>
<tr>
<td>Rank-S AOL</td>
<td>3.8</td>
<td>0.27</td>
</tr>
<tr>
<td>Rank-S MQT</td>
<td>3.9</td>
<td>0.24</td>
</tr>
</tbody>
</table>

Table 5: Average number of shards searched, and micro-averaged postings ratios for those selected shards and for the complement set of shards, together with the corresponding query time cost ratios, in each case comparing WAND-based search to unpruned search. Smaller numbers indicate greater savings.
Table 6: As for Table 5, but showing macro-averaged ratios. All differences between selected and non-selected shards are significant (paired two-tailed t-test, p < 0.01).

<table>
<thead>
<tr>
<th></th>
<th>WAND postings cost ratio</th>
<th>WAND runtime cost ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Selected</td>
<td>Non-selected</td>
</tr>
<tr>
<td>Taily AOL</td>
<td>0.42</td>
<td>0.52</td>
</tr>
<tr>
<td>Taily MQT</td>
<td>0.52</td>
<td>0.61</td>
</tr>
<tr>
<td>Rank-S AOL</td>
<td>0.42</td>
<td>0.53</td>
</tr>
<tr>
<td>Rank-S MQT</td>
<td>0.52</td>
<td>0.61</td>
</tr>
</tbody>
</table>

For each query the WAND savings were calculated for the small set of selected shards, and the much larger set of non-selected shards.

Table 5 lists micro-averaged w/b ratios, and Table 6 the corresponding macro-averaged ratios. While all shards see improvements with WAND, the selected shards see a greater efficiency gain than the non-selected shards, reinforcing our contention that resource selection is an important component in search efficiency. When compared to the ratios shown in Table 4, the selected shards see substantially higher benefit than average shards; the two orthogonal optimizations generate better-than-additive savings.

Figure 7a shows the distribution of the individual per query per shard times for the MQT query set, covering only the shards chosen by the two resource selection processes, in the first four configurations. The fifth exhaustive search configuration includes data for all of the 100 randomly-generated shards making up the second system, and is provided as a reference point. Figure 7b gives numeric values for the mean and median of each of the five distributions. When WAND is combined with selective search, it both reduces the average time required to search a shard and also reduces the variance of the query costs. Note the large differences between the mean and median query processing times for the unpruned search and the reduction in that gap when WAND is used; this gain arises because query and shard combinations that have high processing times due to long postings lists are the ones that benefit most from WAND. Therefore, in typical distributed environments where shards are searched in parallel, the slowest, bottleneck shard will benefit the most from WAND and may result in additional gains in latency reduction. Furthermore, while Figure 7 shows similar per shard query costs for selective and exhaustive search, the total work associated with selective search is substantially less than exhaustive search because only 3–5 shards are searched per query, whereas exhaustive search involves all 100 shards. Taken in conjunction with the previous tables, Figure 7 provides clear evidence that WAND amplifies the savings generated by selective search, answering the first part of Research Question 2 with a “yes”. In addition, these experiments have confirmed that execution time is closely correlated with measured posting evaluations. The remaining experiments utilize postings counts as the cost metric.

We now consider the second part of Research Question 2 and seek to explain why dynamic pruning improves selective search. Part of the reason is that the postings lists of the query terms associated with the highly ranked shards are longer than they are in a typical randomized shard. With these long postings lists, there is more opportunity for WAND to achieve early termination. Figure 8 shows normalized final heap-entry thresholds, or equivalently, the similarity score of the
1,000th ranked document in each shard. The scores are expressed as a fraction of the maximum document score for that query across all shards, then plotted as a function of the resource selector’s shard ranking using Taily, averaged over queries. Shards that Taily did not score because they did not contain any query terms were ordered randomly. For the AOL log the 1,000th document in the shard ranked highest by Taily attains, on average across queries, a score that is a little over 80% of the maximum score attained by any single document for that same query. The downward trend in Figure 8 indicates that the resource ranking process is effective, with the high heap-entry thresholds in the early shards suggesting – as we would hope – that they contain more of the high-scoring documents.

To further illustrate the positive relationship between shard ranking and WAND, w/b was calculated for each shard in the per query shard orderings, and then micro-averaged at each shard rank. Figure 9 plots the average as a function of shard rank, and confirms the bias towards greater savings on the early shards – exactly the ones selected for evaluation. As a reference point, the same statistic was calculated for a random ordering of the randomized shards (random since no shard ranking is applied in traditional distributed search), with the savings ratio being a near-
Figure 8: Normalized 1,000th document scores from shards, averaged over queries and then shard ranks, and expressed as a fraction of the collection-wide maximum document score for each corresponding query. The score falls with rank, as fewer high-scoring documents appear in lower-ranked shards.

Figure 9: The micro-average w/b ratio for WAND postings evaluations, as a function of the per query shard rankings assigned by Taily. Early shards generate greater savings.

4.2.3 Improving efficiency with cascaded pruning thresholds

In the experiments reported so far, the rankings were computed on each shard independently, presuming that they would be executing in parallel and employing private top-k heaps and private heap-entry thresholds, with no ability to share information. This approach minimizes search latency when multiple machines are available, and is the typical configuration in a distributed search architecture. An alternative approach is suggested by our second research question: what happens if the shards are instead searched sequentially, passing the score threshold and top-k
heap from each shard to the next? The heap-entry score threshold is then non-decreasing across the shards, and additional savings should result. While this approach would be unlikely to be used in an interactive system that demands low latency, it provides an upper bound on the efficiency gains that are possible if a single heap was shared by all shards, and would increase throughput when limited resources are available and latency is not a concern: for example, in off-line search and text analytics applications.

Figure 10 demonstrates the threshold in the sequential WAND configuration, with shards ordered in two ways: by Taily score, and randomly. The normalized threshold rises quickly towards the maximum document score through the first few shards in the Taily ordering, which is where most of the documents related to the query are expected to reside. Figure 11 similarly plots the w/b WAND savings ratio at each shard rank, also micro-averaged over queries, and with shard ordering again determined by the Taily score. The independent and sequential configurations

Figure 10: Normalized 1,000th document scores from shards relative to the highest score attained by any document for the corresponding query, micro-averaged over queries, assuming that shards are processed sequentially rather than in parallel, using the Taily-based ordering of topical shards and a random ordering of the same shards.

Figure 11: Ratio of postings evaluated by WAND for independent shard search versus sequential shard search, AOL queries with micro-averaging. Shard ranking was determined by Taily.
DIVERGE MARKEDLY IN THEIR BEHAVIOR, WITH A DEEP SEARCH IN THE LATTER PROCESSING FAR FEWER POSTINGS THAN A DEEP SEARCH IN THE former. THE MQT QUERY SET DISPLAYED SIMILAR TRENDS. SHARING THE DYNAMIC PRUNING THRESHOLDS HAS A LARGE EFFECT ON THE EFFICIENCY OF SELECTIVE SEARCH.

OUR MEASUREMENTS SUGGEST THAT A HYBRID APPROACH BETWEEN INDEPENDENT AND SEQUENTIAL SEARCH COULD BE BENEFICIAL. A RESOURCE-RANKER MIGHT BE CONFIGURED TO UNDERESTIMATE THE NUMBER OF SHARDS THAT ARE REQUIRED, WITH THE UNDERSTANDING THAT A SECOND ROUND OF SHARD RANKING CAN BE INSTIGATED IN SITUATIONS WHERE DEEPER SEARCH IS NEEDED, IDENTIFIED THROUGH EXAMINING THE SCORES OR THE QUANTITY OF DOCUMENTS RETRIEVED. WHEN A SECOND WAVE OF SHARDS IS ACTIVATED, PASSING THE MAXIMUM HEAP-ENTRY THRESHOLD ATTAINED BY THE FIRST-WAVE PROCESS WOULD REDUCE THE COMPUTATIONAL COST. IF THE MAJORITY OF QUERIES ARE HANDLED WITHIN THE FIRST WAVE, A NEW COMBINATION OF LATENCY AND WORKLOAD WILL RESULT.

4.3 SUMMARY

TO DATE THERE HAS BEEN ONLY LIMITED CONSIDERATION OF THE INTERACTION BETWEEN DYNAMIC PRUNING AND SELECTIVE SEARCH [KULKARNI AND CALLAN 2015], AND IT HAS BEEN UNCLEAR WHETHER DYNAMIC PRUNING METHODS IMPROVE SELECTIVE SEARCH, OR WHETHER SELECTIVE SEARCH IS CAPTURING SOME OR ALL OF THE SAME UNDERLYING SAVINGS AS PRUNING DOES, JUST VIA A DIFFERENT APPROACH. IN THIS DISSERTATION WE HAVE EXPLORED WAND DYNAMIC PRUNING USING A LARGE DATASET AND TWO DIFFERENT QUERY SETS. IN CONTRAST TO KULKARNI’S FINDINGS WITH TBMS [KULKARNI AND CALLAN 2015], WE SHOW THAT WAND-BASED EVALUATION AND SELECTIVE SEARCH GENERATE WHAT ARE EFFECTIVELY INDEPENDENT SAVINGS, AND THAT THE COMBINATION IS MORE POTENT THAN EITHER TECHNIQUE IS ALONE – THAT IS, THAT THEIR INTERACTION IS A POSITIVE ONE. IN PARTICULAR, WHEN RESOURCE SELECTION IS USED TO CHOOSE QUERY-APPROPRIATE SHARDS, THE IMPROVEMENTS FROM WAND ON THE SELECTED SHARDS IS GREATER THAN THE SAVINGS ACCRUING ON RANDOM SHARDS, CONFIRMING THAT DYNAMIC PRUNING FURTHER IMPROVES SELECTIVE SEARCH – A RARE SITUATION WHERE ORTHOGONAL OPTIMIZATIONS ARE BETTER-THAN-ADDITIVE. WE ALSO DEMONSTRATED THAT THERE IS A DIRECT CORRELATION BETWEEN THE EFFICIENCY GAINS GENERATED BY WAND AND THE SHARD’S RANKING. WHILE IT IS WELL-KNOWN THAT RESOURCE SELECTION IMPROVES EFFECTIVENESS, OUR RESULTS SUGGEST THAT IT CAN ALSO IMPROVE OVERALL EFFICIENCY TOO.

FINALLY, TWO DIFFERENT METHODS OF APPLYING WAND TO SELECTIVE SEARCH WERE COMPARED AND WE FOUND THAT PASSING THE TOP-K HEAP THROUGH A SEQUENTIAL SHARD EVALUATION GREATLY REDUCED THE VOLUME OF POSTINGS EVALUATED BY WAND. THE SIGNIFICANT DIFFERENCE IN EFFICIENCY BETWEEN THIS APPROACH AND THE USUAL FULLY-PARALLEL MECHANISM SUGGESTS AVENUES FOR FUTURE DEVELOPMENT IN WHICH HYBRID MODELS ARE USED TO BALANCE LATENCY AND THROUGHPUT IN NOVEL WAYS.
Block-Max WAND (BM-WAND) [Dimopoulos et al. 2013] is an extension of WAND, a dynamic pruning algorithm which skips evaluating sections of the postings list during query time when those segments are guaranteed not to contain high scoring documents. In order to accomplish this, at index time BM-WAND segments a term posting list into blocks that span \( n \) documents each and stores the local maximum term score found in the block as the block max score. This provides information on the term score distribution of the postings which may be useful for resource selection. This chapter shows that using BM-WAND statistics for resource selection is a light-weight solution to some resource selection tasks that does not require gathering additional corpus statistics or building a resource selection database. To summarize, our research question is:

\textbf{RQ 4} Can BM-WAND information be used to design an effective resource selection algorithm?

\section*{5.1 Methods}

Typically, BM-WAND gathers the statistics it needs from a postings list from the index without any special sorting operations. Consequently, the documents in the postings list are in the order that was supplied to the search index at index time, which is usually random or approximately grouped by the source domains of the documents. The shards used in this work roughly ordered the documents by source and preliminary experiments found that source ordering was not ideal. We found that the block max scores calculated from the source ordered postings list contained very little signal for the actual term score distribution of the documents in the list.

\textbf{Figure 12} shows the block max scores of the term \textit{obama} for a topical shard that was highly ranked for the query \textit{obama family tree} where the postings list was in source order. The distribution of scores in this document order was near random, which is typical. When the max scores were calculated from each block, it formed a flat distribution with little variation, which was not representative of the full range of document scores.

Alternatively, the documents in a postings list can be sorted in \textit{impact ordering} [Anh and Moffat 2002]. In this ordering, a term’s postings list is sorted such that the highest scoring documents for that term appear first in the list. As seen in \textbf{Figure 12}, the block max scores calculated from this ordering give much better insight into the distribution of document scores for the term across the shard. Furthermore, block max scores of impact ordered lists have a clear intuitive meaning; the block max score of the first block is the score of the highest scoring document for the term in that shard, the block max score of the second block using size \( n \) is the score of the \( n+1 \)’th highest scoring document for the term, etc. Therefore, we use block max scores for impact ordered posting lists over source order posting lists in our experiments.
Initially, we experimented with a range of query lengths and attempted to handle multi-term queries in two different ways. In the first method, for a given shard, a score was generated for each query term and then the whole query score was produced by adding the individual term scores. In the second method, we sought to better capture term co-occurrence using source-ordered posting lists, where document ordering is consistent across all terms. We identified the blocks where the query terms overlapped and calculated the density of the maximum possible score for the overlapped blocks, as defined by the maximum score possible in the overlapped block divided by the number of documents in that block.

Neither of these methods produced good results. Our hypothesis was that we were mishandling merging information across terms in multi-term queries rather than an innate lack of signal in the block max scores. In order to test this hypothesis, we narrowed our focus to single term queries. It is true that for single term queries, the best shards to search can be pre-computed based on the top scoring documents for the term. However, the goal of this work is exploratory, to gauge the usefulness of block max scores and lay the foundation for future work to merge the term scores to serve multi-term queries as well.

We compared two methods of using block max scores for shard selection of single term queries. The first method, second score, orders the shards by the score of second highest block, where the block size $n = 128$. In a single term query, this is equivalent to ranking the shards by the score of the 129th highest scoring document from that shard. After the shards were ranked, a static cut-off was applied. In our experiments, we used a cut-off of 3 and 5 in order to produce comparable results to Taily and Rank-S, two state-of-the-art resource selection algorithms to which we compare our method.

The second method, multi-size second scores (M3S), extends the above idea by using block max scores from blocks of different sizes. Rather than using a single data point to represent a shard’s worth, the block max scores of the second highest block of block sizes $N = \{1, 2, 4, 8, 16, 32, 64, 128\}$ are collected. (Alternatively, this is equivalent to sampling the score of the impact ordered posting list at document rank 2, 3, 5 etc.) These block max scores are then pooled and blocks with the highest scores are selected and removed from the pool. Each selected block of size $n_i$ contributes $n_i - n_{i-1}$ additional documents to the estimated ranked list. Because of the impact ordered postings list, it is impossible that a block of a larger size would be selected before a block of a
5.1 Methods

Algorithm 1 Multi-size second scores (M3S) algorithm for shard selection.

\[
\begin{align*}
\text{k} &= \text{input threshold} & \triangleright \text{document ranking length} \\
N &= \{1, 2, 4, 8, 16, 32, 64, 128\} & \triangleright \text{block size steps used} \\
S &= \{\} & \triangleright \text{set of selected shards} \\
B &= \{b_{n_i,s} \mid \forall n_i \in N, \forall s \in \text{all shards}\} & \triangleright \text{all second score blocks from all shards} \\
\hat{k} &= 0 & \triangleright \text{estimated number of top scoring documents in S} \\
\text{while } \hat{k} < k \text{ do} \\
\quad b_{n_i,s} &= \text{block with highest max score in B} \\
\quad \text{add shard } s \text{ to } S \\
\quad \text{remove } b_{n_i,s} \text{ from } B \\
\quad \hat{k} &= \hat{k} + n_i - n_{i-1} & \triangleright \text{where } n_0 \text{ is defined as } 0 \text{ for convenience} \\
\text{return } S
\end{align*}
\]

![Figure 13](image)

Figure 13: An example of the multi-size second scores method. If \( k = 2 \), \( b_{1,1} \) is the first block selected (\( \hat{k} = 1 \)), then \( b_{1,2} \) (\( \hat{k} = 2 \)) resulting in shard 1 being selected for search, as ideal. However, if \( k = 3 \), \( b_{2,1} \) (\( \hat{k} = 3 \)) is also selected, resulting in both shards being selected for search even though shard 2 is unnecessary.

smaller size from the same shard. Therefore, a new block contributes additional documents equal to its size less the total number of documents already contributed by the shard, i.e. the size of a block of one size step lesser. This process continues until the estimated ranked list size exceeds a threshold, \( k \). Then, all shards represented by the selected blocks become the set of shards to search. This algorithm is summarized in Algorithm 1.

The M3S method uses the block max scores to attempt to find all shards represented in the top-\( k \) of a document ranking created by exhaustive search. In the example shown in Figure 13, for a desired \( k = 2 \), the algorithm first selects \( b_{1,1} \) (\( \hat{k} = 1 \)), then \( b_{1,2} \) (\( \hat{k} = 2 \)) and stops, resulting in shard 1 being selected for search.

In practice, the algorithm may select more shards than necessary. If the desired \( k = 3 \) in Figure 13, the algorithm would select an additional block \( b_{2,1} \) (\( \hat{k} = 3 \)) before halting. This results in both shard 1 and 2 being selected even though the true top 3 documents all reside in shard 1.

Unlike the second score method, M3S can select a different number of shards for each query. In practice, M3S selects a large number of shards because there are many shards that contribute only one or two documents to the final result list. Thus, we also explored a static cut-off variation, in which shards are sorted by the number of documents they are estimated to contribute to the top-\( k \) and a static cut-off of 3 or 5 is applied, similar to the setup for the second scores method. In our experiments, we set \( k = 100 \) unless otherwise specified.
5.2 EXPERIMENTAL SETUP

The experiments were carried out with the ClueWeb09 Category B dataset, which includes 50 million English documents, and single term queries from the TREC Web track query set from 2009–2012, a total of 58 queries with relevance judgments.

The dataset was clustered into 296 topical shards using the SB2 K-means clustering algorithm [Kulkarni 2013]. The method initially formed 200 shards then in a second step reclustered the largest shards to smaller pieces in order to balance the shard sizes. The larger number of shards used for the dataset compared to previous experiments was due to the fact that smaller shards produced more accurate results [Dai et al. 2016].

The shards were searched using BM25 with $k_1 = 0.9$ and $b = 0.4$ and using the global corpus statistics for idf and average document length. BM25 was chosen for its simple operator for combining term scores (i.e. addition) and because it is known to work well with WAND. Each shard returned 1,000 results and the top 1,000 documents from the final merged list were used to calculate effectiveness metrics.

The baselines are exhaustive search, i.e. the effectiveness gained by searching the entire corpus, and two state-of-the-art resource selection algorithms, Taily [Aly et al. 2013] and Rank-S [Kulkarni et al. 2012]. Exhaustive search was performed using BM25 with the same parameters as above. The parameters used for Taily and Rank-S were the best parameters as reported by Aly et al. [2013] ($n_c = 400, v = 50$) and Kulkarni [2013] ($\text{base} = 5, \text{CSI} = 1\%$).

Statistical significance testing was performed with a non-inferiority test commonly used in the medical field to test that two treatments have similar effectiveness. In this test, rather than testing to reject the null hypothesis $H_0 : \mu_A = \mu_B$, we test to reject $H'_0 : \mu_A - \mu_B \geq \delta$ for some small margin, $\delta$. Intuitively, by rejecting $H'_0$ we accept the alternative hypothesis, which is that any reduction of performance in system B compared to system A is inconsequential. Note that if $\delta = 0$, $H'_0$ tests for the statistical superiority of system B. Jayasinghe et al. [2015] presents a more thorough treatment of the subject of equivalence testing in IR systems. In our experiments, we set the margin $\delta$ to 5% and 10% of the mean of the system A.

5.3 EXPERIMENTS

Table 7 presents the results of the experiments. We first present the exhaustive search result, which is the target that the resource selection algorithms attempt to meet and is the system used as the basis of comparison in all non-inferiority testing. The following two groupings of results present our methods compared to two state-of-the-art resource selection baselines. A static cut-off was applied such that our methods search an equivalent number of shards to the baseline. This ensures that the accuracy of the results are presented for an equivalent efficiency level in the shard search step. The last group are two versions of the M3S method with different $k$ settings that were allowed select a different number of shards for each query.

Taily performs slightly better than Rank-S in single term queries across all metrics, whereas we have previously seen Rank-S outperform Taily when using the full query set (see Table 15). This result is expected, given that Rank-S has more information about term co-occurrence from its sample index, whereas Taily uses an explicit term independence assumption in its estimation. The results of Table 7 show that while Taily and Rank-S have lower accuracy in MAP@1000, they have similar or higher mean accuracies to exhaustive search when using shallow metrics. However, the results fail the non-inferiority test at both 5% and 10% margins, indicating that we cannot
Table 7: The results of second scores and M3S resource selection methods with different cut-off alternatives compared to exhaustive search and state of the art resource selection algorithm baselines for 58 single term queries. Avg shards shows the average number of shards selected for each query. For the M3S, k = 100 unless otherwise specified. * indicates method is within a 10% margin of exhaustive search using a non-inferiority test with p = 0.05. + indicates result is within a 5% margin.

<table>
<thead>
<tr>
<th></th>
<th>Avg shards</th>
<th>P@10</th>
<th>NDCG@30</th>
<th>MAP@1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exhaustive</td>
<td>-</td>
<td>0.2741</td>
<td>0.1724</td>
<td>0.0903</td>
</tr>
<tr>
<td>Taily</td>
<td>2.78</td>
<td>0.2810</td>
<td>0.1797</td>
<td>0.0756</td>
</tr>
<tr>
<td>M3S (3 shards)</td>
<td>3</td>
<td>0.2948*</td>
<td>0.1856*</td>
<td>0.0769</td>
</tr>
<tr>
<td>2nd Score (3 shards)</td>
<td>3</td>
<td>0.2983*</td>
<td>0.1711</td>
<td>0.0727</td>
</tr>
<tr>
<td>Rank-S</td>
<td>4.76</td>
<td>0.2776</td>
<td>0.1690</td>
<td>0.0666</td>
</tr>
<tr>
<td>M3S (5 shards)</td>
<td>5</td>
<td>0.2948*</td>
<td>0.1947*</td>
<td>0.0815</td>
</tr>
<tr>
<td>2nd Score (5 shards)</td>
<td>5</td>
<td>0.3207*</td>
<td>0.1927*</td>
<td>0.0855</td>
</tr>
<tr>
<td>M3S</td>
<td>29</td>
<td>0.2741*</td>
<td>0.1728*</td>
<td>0.0873*</td>
</tr>
<tr>
<td>M3S (k = 1000)</td>
<td>135</td>
<td>0.2741*</td>
<td>0.1724*</td>
<td>0.0903*</td>
</tr>
</tbody>
</table>

conclude that the Taily and Rank-S do not perform significantly worse than exhaustive search, even for shallow metrics.

When searching a similar number of shards as Taily and Rank-S, second score produced comparable results to Taily and more accurate results than Rank-S. M3S was more accurate than both Taily and Rank-S. The strong performance in precision metrics is unsurprising, as both second score and M3S explicitly focus on shards that deliver top-ranked documents. Unlike Taily and Rank-S, second score and M3S were statistically non-inferior to exhaustive search in precision-oriented metrics (P@10 and NDCG@30) in most cases, which allows us to conclude that resource selection using these methods produces results that are as accurate as exhaustive search.

In MAP@1000, a recall-oriented metric, while M3S using cut-offs performed better than Taily and Rank-S, it failed the non-inferiority test. When the full set of selected shards is searched for M3S (i.e. 29 on average for k = 100 and 135 on average for k = 1000), the results pass the non-inferiority test for MAP as well. Full M3S also passes the non-inferiority test for both P@10 and NDCG@30, even though the mean for these metrics is sometimes lower than that of Taily or Rank-S. This is due to the variance of the systems. While Taily and Rank-S sometimes perform better than exhaustive search, they perform worse in more queries. For P@10, the win-loss ratio of Taily compared to exhaustive search is 15 wins and 17 losses. For Rank-S, it is 20 wins and 22 losses. However, M3S exactly replicates the exhaustive search ranking to these depths and produces a 0 win 0 loss record, passing the non-inferiority test unlike Taily and Rank-S. While M3S does not have a higher mean performance, the results are stable and would be particularly useful in systems that have a high penalty for missing documents, such as web search systems that use re-ranking algorithms.

The amount of shards searched for M3S may seem large, but it is only necessary for tasks requiring high recall. Otherwise, searching 5 shards is sufficient for shallower metrics such as P@10 and NDCG@30. We can also compare the number of shards searched for M3S to that of
an oracle method. If one had perfect knowledge of which documents would appear in the top-
$k$ results of the final result list and which shards they came from, one can construct the ideal,
minimum set of shards required to exactly reproduce the final ranked list. Compared to this ideal
minimum, the number of shards selected by M3S is quite reasonable, especially for the $k = 100$
configuration. The oracle method selects an average of 22 shards for $k = 100$ and an average of
87 shards for $k = 1000$.

The efficiency of the resource selection methods used in this chapter are independent of the
shard search step and do not vary with the shard cut-offs. In the order of increasing latency, the
efficiency of the algorithms are as follows: second score $< \text{Taily} < M3S << \text{Rank-S}$. That is, the
second score method is the most efficient and Rank-S, the least. For every shard-term pair, the
second score method stores one value (the score of the second block), Taily stores two values (the
$n_c$ and $\nu$ parameters), and M3S stores eight values (one second score per block size in $N$). Rank-S
stores 1% of all postings in its CSI, which is typically much larger.

This result verifies our hypothesis that BM-WAND scores can be used to perform resource se-
lection in single term queries. When searching an equivalent number of shards, the new methods
out-performed existing baselines and were statistically significantly non-inferior to exhaustive
search in precision-oriented metrics. When M3S was allowed to search all shards that it selected,
it produced significantly non-inferior results in recall-oriented MAP@$1000$ and stable precision
results. Overall, M3S was more accurate and had greater stability than Taily or Rank-S while not
requiring additional corpus analysis or resource selection databases.

### 5.4 Summary

Some dynamic pruning algorithms such as BM-WAND [Dimopoulos et al. 2013] collect statistics
during indexing time to perform online optimizations. We showed that this information can be
used to perform accurate, stable resource selection for single term queries. When impact and
source ordered posting lists were compared, the block max scores for impact ordered postings
displayed more score variation and a stronger signal for resource selection.

We experimented with two variations of using block max scores for resource selection with
single term queries. When compared to two state-of-the-art resource selection algorithms, results
from searching the shards selected by M3S were found to be highly accurate in precision-oriented
conditions and statistically non-inferior to exhaustive search in high-recall conditions, a difficult
goal that was not achieved by the baselines. This indicates that a way of combining the term
scores from the block max score information may produce a good resource selection method for
all queries.

Furthermore, we saw that it is feasible to determine a set of shards that would exactly replicate
exhaustive search results that was close to the size of an oracle-based method. This presents the
possibility of performing a low-variance, stable search within a selective search paradigm, which
closely or exactly replicates exhaustive search results.
LEARNING TO RANK RESOURCES

Prior work showed that while selective search is equivalent to exhaustive search for shallow metrics (e.g. $P@10$), it performs worse for recall-oriented metrics (e.g. MAP) under commonly studied parameter settings [Kulkarni 2013; Aly et al. 2013; Dai et al. 2016]. This is a problem because modern retrieval systems apply re-ranking operations to a base retrieval, which can require deep result lists [Macdonald et al. 2013]. Poor recall prevents selective search from being used as an efficient first-stage retrieval in a multi-stage search system.

In this chapter, we present learning to rank resources, a resource selection method based on learning-to-rank [Liu 2009]. While learning-to-rank has been widely studied for ranking documents, its application to ranking resources has not been studied in depth. We take advantage of characteristics of the resource ranking problem that are distinct from document ranking; we present new features; and we propose a training approach that uses exhaustive search results as the gold standard and show that human judgments are not necessary. We seek to answer the following major research questions:

RQ 5 Is learning to rank resources an efficient algorithm that is as accurate as exhaustive search in deep recall-oriented metrics?

RQ 6 Can learning to rank resources be trained without human judgements?

6.1 RELATED WORK

Learning to rank is a supervised approach to the ranking problem of information retrieval and can be grouped into three broad approaches: point-wise, pair-wise, and list-wise [Liu 2009]. Point-wise approaches attempt to predict the exact relevance degree of each document. Because point-wise approaches take as input a single document, they cannot see the relative or positional information of the ranked list. Pair-wise approaches such as SVM$^{\text{rank}}$ [Joachims 2006] reduce the ranking problem into a binary classification problem, where a pair of documents are given as input and the classifier must decide the ordering of the documents. List-wise approaches such as ListMLE [Xia et al. 2008] take the entire set of documents for a given query as input and attempt to produce the correct permutation of documents. List-wise approaches are thought to generally perform better than pair-wise or point-wise approaches.

---

1 This is a lightly-revised version of Zhuyun Dai, Yubin Kim, and Jamie Callan. Learning to rank resources. In Proceedings of the 40th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 837–840, 2017. The learning-to-rank model framework was implemented and evaluated by Dai. Both authors designed and implemented features for the model. I created the Tally, champion list, and average distance to centroid features. The design of experiments, analysis, and paper writing were jointly shared by both authors.
Supervised resource selection algorithms use training data to learn models to evaluate shard relevance. Kulkarni [2015] used a regression classifier to learn the appropriate number of shards to search for a given query and metric, but do not use supervised information for shard ranking. Most methods that rank shards using supervised methods train a classifier per shard [Arguello et al. 2009b; Cetintas et al. 2009]. However, training a classifier for every shard is expensive in selective search, where shards number in hundreds. Thus, supervised methods have not been used for selective search. Techniques that train a single classifier would be more suitable for selective search. Balog [Balog 2014] trained a learning-to-rank algorithm for Federated Web Search TREC track and presented preliminary results. Hong et al. [Hong et al. 2010] learned a joint probabilistic classifier called Jnt, which is used as a baseline in this work.

6.2 Model

Let q denote a query, \( P(q) \) the distribution of queries, and \( \Phi(q, s_i) \) denote features extracted from the i’th shard for the query. The goal of learning-to-rank is to find a shard scoring function \( f(\Phi(q, s)) \) that can minimize the loss function defined as:

\[
L(f) = \int_{q \in Q} l(q, f)dP(q)
\]

We use:

\[
l(q, f) = \sum_{s_i > q s_j} 1\{f(\Phi(q, s_i)) < f(\Phi(q, s_j))\}
\]

where \( s_i > q s_j \) denotes shard pairs for which \( s_i \) is ranked higher than \( s_j \) in the gold standard shard ranking with respect to query q. The \( 1\{\} \) function returns 1 if the boolean statement contained in the braces is true, and 0 if false. In other words, the loss is simply the number of times that the learned function was wrong about the ordering of shard pairs.

We used SVM\textsuperscript{rank} [Joachims 2006], which optimizes pair-wise loss. List-wise algorithms such as ListMLE [Xia et al. 2008] produced similar results, thus we only report results with SVM\textsuperscript{rank}.

The training process requires a gold standard shard ranking for each training query. We propose two definitions of the ground truth, relevance-based and overlap-based. In the relevance-based approach, the optimal shard ranking is determined by the number of relevant documents a shard contains. Thus, the training data require queries with relevance judgments, which can be expensive to obtain. The overlap-based approach assumes that the goal of selective search is to reproduce the document ranking of exhaustive search. The optimal shard ranking is determined by the number of documents in a shard that were ranked highly by exhaustive search. This does not require manual relevance judgments.

6.3 Features

For a given query and a shard, we use various sources of information or features to generate a vector representation of how well the shard matches the query. We use three sources of information to define our features: query-independent information, term-based statistics, and information about sample-document rankings.
6.3.1 Query-independent information

**Shard Popularity:** Indicates how often the shard had relevant (relevance-based) or top-ranked (overlap-based) documents for training queries. It is query-independent and acts as a shard prior.

6.3.2 Term-based statistics

Term-based features can be easily precomputed, thus are efficient.

**Taily Features:** One feature is the Taily [Aly et al. 2013] score calculated for query \( q \) and shard \( s \). However, Taily scores can vary greatly across shards and queries. For robustness, we add two additional features. If shard \( s \) is ranked \( r_s \) for query \( q \), the inverse rank is \( 1/r_s \) [Guan et al. 2014], which directly describes the importance of \( s \) relative to other shards. The binned rank is \( \text{ceiling}(r_s/b) \), where \( b \) is a bin-size. We use \( b = 10 \), meaning that every 10 consecutive shards are considered equally relevant. This feature helps the model to ignore small differences between shards with similar rankings.

**Champion List Features:** For each query term, the top-\( k \) best documents were found in an offline process; for each term, the term scores for all documents containing the term are computed and the documents with the top-\( k \) highest scores for a given term are determined. The number of documents each shard contributes to the top-\( k \) was stored for each shard-term pair. For multi-term queries, the feature values of each query term were summed. We use two values of \( k = \{10, 100\} \), generating 2 features.

**Query Likelihood Features:** The log-likelihood of a query with respect to the unigram language model of each shard is:

\[
L(q|s) = \sum_{t \in q} \log p(t|s),
\]

where \( p(t|s) \) is the shard language model, the average of all document language models \( p(t|d) \) in the shard [Si et al. 2002]. Document language model \( p(t|d) \) is estimated using MLE with Jelinek-Mercer smoothing. Query likelihood, inverse query likelihood, and binned query likelihood features are created for body, title, and inlink representations, yielding a total of 9 features.

**Query Term Statistics:** The maximum and minimum shard term frequency across query terms, e.g. \( \text{stf}_{\text{max}}(q, s) = \max_{t \in q} \text{stf}(t, s) \), where \( \text{stf}(t, s) \) is the frequency of term \( t \) in shard \( s \). We include the maximum and minimum of \( \text{stf} \cdot \text{idf} \) where \( \text{idf} \) is the inverse document frequency over the collection. These 4 features are created for body, title, and inlink representations, yielding 12 features.

**Bigram Log Frequency:** The frequency of each bigram of the query in a shard is \( \text{bf}_q(s) = \sum_{b \in q} \log \text{bf}_q(b, s) \), where \( \text{bf}_q(b, s) \) is the frequency of bigram \( b \) in shard \( s \). This feature can estimate term correlation. To save storage, we only store bigrams that appear more than 50 times in the collection.

6.3.3 Sample-document (CSI-based) features

These features are based on retrieval from the centralized sample index (CSI), which may provide term co-occurrence information. CSI retrieval is expensive, and thus is slower to calculate.

**Rank-S and ReDDE Features:** Similar to Taily features, the shard scores given by Rank-S [Kulkarni et al. 2012] and ReDDE [Si and Callan 2003], as well as the inverse rank and binned rank features for a total of 6 features.

**Average Distance to Shard Centroid:** The distance between the top-\( k \) documents retrieved from the CSI to their respective shards’ centroids. Intuitively, if the retrieved documents are close
learning to rank resources

...to the centroid, the shard is more likely to contain other similar, highly-scoring documents. For multiple documents from the same shard, the distances are averaged. We use two distance metrics: KL divergence and cosine similarity. Note that because KL divergence measures distance rather than similarity, we use the inverse of the averaged KL divergence as the metric. We generated features for \( k = \{10, 100\} \) and also a feature measuring the distance between the shard’s centroid to its single highest scoring document in the top 100 of the CSI results, for a total of 6 features.

6.4 EXPERIMENTAL METHODOLOGY

Datasets: Experiments were conducted with ClueWeb09-B and Gov2. ClueWeb09-B (CW09-B) consists of 50 million pages from the ClueWeb09 dataset. Gov2 is 25 million web pages from the US government web domains. For relevance-based models, 200 queries from the TREC 09-12 Web Track topics were used for CW09-B, and 150 queries from the TREC 04-06 Terabyte Track topics were used for Gov2. Models were trained by 10-fold cross-validation. For overlap-based models, training queries were sampled from the AOL and Million Query Track query logs. For CW09-B, we sample 1000 queries from each source, giving us 2 training sets (AOL and MQT). For Gov2, the 1000 MQT queries were used as one training set. The second training set consisted of 1000 queries sampled from the AOL query log that have clicks on ‘.gov’ URLs. Models were tested with the TREC queries. Optimal shard ranking for the overlap method was defined by the number of documents each shard contains that were within the top \( N = 2K \) retrieved from exhaustive search. We found \( N \in [1K, 3K] \) produced stable results.

Proposed methods and baselines: We used three sources of training data: relevance-based training data (L2R-TREC), and overlap-based training data (L2R-AOL and L2R-MQT). We used linear SVM\textsuperscript{rank}, where C was chosen by cross validation. Our method was compared against state-of-the-art unsupervised models (Taily [Aly et al. 2013], ReDDE [Si and Callan 2003], and Rank-S [Kulkarni et al. 2012]); and a supervised model Jnt [Hong et al. 2010].

In order to ensure that the accuracy results are apples-to-apples across resource selection algorithms (searching more shards usually results in higher accuracy), we compare the methods at various imposed static shard cutoffs rather than the built-in cutoffs from the resource selection algorithms.

Jnt was trained and tested using TREC queries with 10-fold cross-validation. ReDDE parameter \( n \) was set to 1000. The Rank-S exponential decay was set to \( B = 1.1 \), which makes the score decay slower compared to prior work, so that more shards had non-zero scores to use with the static shard cutoffs. Taily scores were generated with \( n_c = 400 \).

Evaluation Metrics: Search accuracy was measured by P@10, NDCG@30 and MAP@1000. Evaluation was performed at each shard rank cutoff and the automatic cutoff of Rank-S and Taily. To test the proposed methods’ superiority to baselines, a query-level permutation test with \( p < 0.05 \) was used. To test the equivalence to exhaustive search, a non-inferiority test [Walker and Nowacki 2011] was used to assert that results of the more efficient selective search were at least as accurate as exhaustive search. The equivalence is established by rejecting the null hypothesis that selective search is at least 5% worse than exhaustive search with a 95% confidence interval.

Selective Search Setup: We used 123 shards for CW09-B and 199 shards for Gov2 created by the QKLD-Qinit method from Dai et al. [2016]. A 1% central sample index (CSI) was created for ReDDE and Rank-S baselines and CSI based features. Jnt followed the original implementation and used a 3% CSI. The larger CSI increases the cost of resource selection for Jnt but is expected to give Jnt an advantage in accuracy.
Search Engine Setup: Retrieval was performed with Indri, using default parameters. Queries were issued using the sequential dependency model (SDM) with parameters \((0.8, 0.1, 0.1)\) [Metzler and Croft 2005]. For CW09-B, documents with a Waterloo spam score [Cormack et al. 2011b] below 50 were removed.

6.5 Experiments

Four experiments investigated the effectiveness of learned resource selection models, and their sensitivity to training data, query lengths, and features of varying computational costs and characteristics.

6.5.1 Overall comparison

Our method was compared to four baselines and exhaustive search. We tested a variety of shard rank cutoffs ranging from 1 up to 10 for CW09-B and 16 for Gov2. Shard rankings by L2R enabled more accurate search than all baselines in both datasets (Table 8). The search accuracy of L2R models was higher than the baselines at nearly every shard rank cutoff.

Table 8 presents a comparison of L2R models and the baselines at two particular shard rank cutoffs. The first cutoff is the point where the shallow metrics (P@10 and NDCG@30) stabilize: 4 for CW09-B and 6 for Gov2. The second cutoff is where MAP@1000 become stable: 8 for CW09-B and 12 for Gov2. The automatic cutoffs of Rank-S and Taily performed similarly to fixed cutoffs and are not shown. L2R models improve over the baselines at both shard cutoffs. For shallow metrics, L2R reaches exhaustive search at the first cutoff. Furthermore, searching the first 8 out of 12 shards ranked by L2R is statistically non-inferior to searching all shards exhaustively, even for the recall-oriented MAP@1000. All the baselines are substantially worse than exhaustive search in MAP@1000.

6.5.2 Effect of training data

One might expect the relevance-based model (L2R-TREC) to be better than overlap-based models (L2R-AOL and AOL-MQT), because it uses manual relevance judgments. A model trained with overlap data might favor shards that contain false-positive documents. However, there is little difference between the two training methods. L2R-TREC was statistically significantly better than L2R-AOL or L2R-MQT for MAP@1000 in Gov2, but the relative gain is only 2%; in all other cases, there is no statistically significant differences among the three models. Furthermore, when the feature weights were examined, models trained with relevance and overlap data agreed on which features are important. This analysis indicates that unlike learning to rank document models, we can train a learning to rank resource selector on a new dataset before we have relevance judgments, positively answering Research Question 6.

6.5.3 Query length

We compare L2R-MQT to the baselines using MAP@1000 for queries with different lengths on CW09-B, shown in Figure 14. Gov2 and other training data produced similar results. For single-
Table 8: Search accuracy comparison between 3 L2R models and baselines at two rank cutoffs (T) for two datasets. ▲: statistically significant improvement compared to Jnt, the best resource selection baseline. •: non-inferiority to exhaustive search.

<table>
<thead>
<tr>
<th>Method</th>
<th>P@10</th>
<th>NDCG@30</th>
<th>MAP@1000</th>
<th>P@10</th>
<th>NDCG@30</th>
<th>MAP@1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redde</td>
<td>0.355</td>
<td>0.262</td>
<td>0.176</td>
<td>0.363*</td>
<td>0.275*</td>
<td>0.187</td>
</tr>
<tr>
<td>Rank-S</td>
<td>0.350</td>
<td>0.259</td>
<td>0.175</td>
<td>0.360*</td>
<td>0.268</td>
<td>0.183</td>
</tr>
<tr>
<td>Taily</td>
<td>0.346</td>
<td>0.260</td>
<td>0.172</td>
<td>0.346</td>
<td>0.260</td>
<td>0.175</td>
</tr>
<tr>
<td>Jnt</td>
<td>0.370*</td>
<td>0.269</td>
<td>0.178</td>
<td>0.367*</td>
<td>0.277*</td>
<td>0.192</td>
</tr>
<tr>
<td>L2R-TREC</td>
<td>0.374*</td>
<td>0.281*</td>
<td>0.192▲</td>
<td>0.377*</td>
<td>0.286▲*</td>
<td>0.202▲*</td>
</tr>
<tr>
<td>L2R-AOL</td>
<td>0.374*</td>
<td>0.281▲*</td>
<td>0.191▲</td>
<td>0.375*</td>
<td>0.287▲*</td>
<td>0.202▲*</td>
</tr>
<tr>
<td>L2R-MQT</td>
<td>0.382*</td>
<td>0.285▲*</td>
<td>0.193▲</td>
<td>0.375*</td>
<td>0.286▲*</td>
<td>0.202▲*</td>
</tr>
<tr>
<td>Exh</td>
<td>0.372</td>
<td>0.288</td>
<td>0.208</td>
<td>0.372</td>
<td>0.288</td>
<td>0.208</td>
</tr>
</tbody>
</table>

(a) CW09-B

<table>
<thead>
<tr>
<th>Method</th>
<th>P@10</th>
<th>NDCG@30</th>
<th>MAP@1000</th>
<th>P@10</th>
<th>NDCG@30</th>
<th>MAP@1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redde</td>
<td>0.580*</td>
<td>0.445</td>
<td>0.267</td>
<td>0.587*</td>
<td>0.460*</td>
<td>0.289</td>
</tr>
<tr>
<td>Rank-S</td>
<td>0.570</td>
<td>0.440</td>
<td>0.263</td>
<td>0.585*</td>
<td>0.461*</td>
<td>0.286</td>
</tr>
<tr>
<td>Taily</td>
<td>0.518</td>
<td>0.403</td>
<td>0.235</td>
<td>0.530</td>
<td>0.418</td>
<td>0.256</td>
</tr>
<tr>
<td>Jnt</td>
<td>0.582*</td>
<td>0.459</td>
<td>0.278</td>
<td>0.588*</td>
<td>0.465*</td>
<td>0.292</td>
</tr>
<tr>
<td>L2R-TREC</td>
<td>0.593*</td>
<td>0.469*</td>
<td>0.299▲</td>
<td>0.591*</td>
<td>0.475▲*</td>
<td>0.313▲*</td>
</tr>
<tr>
<td>L2R-AOL</td>
<td>0.593*</td>
<td>0.470▲*</td>
<td>0.291▲</td>
<td>0.587*</td>
<td>0.470*</td>
<td>0.307▲*</td>
</tr>
<tr>
<td>L2R-MQT</td>
<td>0.586*</td>
<td>0.465*</td>
<td>0.292▲</td>
<td>0.593*</td>
<td>0.474▲*</td>
<td>0.309▲*</td>
</tr>
<tr>
<td>Exh</td>
<td>0.585</td>
<td>0.479</td>
<td>0.315</td>
<td>0.585</td>
<td>0.479</td>
<td>0.315</td>
</tr>
</tbody>
</table>

(b) Gov2

term queries, existing methods are already equivalent to or better than exhaustive search, and L2R-MQT retains this good performance. The advantage of L2R-MQT comes from multi-term queries, where the best baseline Jnt still has a 10% gap from exhaustive search. For these queries, the improvement of L2R-MQT over the Taily is expected, because Taily does not model term co-occurrence. However, L2R-MQT also out-performs ReDDE and Rank-S, which account for term co-occurrence by retrieving documents from the CSI, but are limited by only having a sample view of the collection. L2R draws evidence from both the sample and the whole collection. Jnt also fuses sample- and term-based features, but most of its features are derived from ReDDE
or Taily-like methods and do not carry new information. L2R improved over Jnt by using novel features that encode new evidence.

Figure 14: MAP@1000 for queries on CW09-B, grouped by query length. Parentheses on the X axis present the number of queries in each group. T is the shard rank cutoff.

6.5.4 Feature analysis

The L2R approach uses three classes of features: query-independent, term-based, and sample-document (CSI). These three feature classes have substantially different computational costs and contributions.

Fast vs. Slow features: Sample-document (CSI-based) features have a high computational cost, because they search a sample (typically 1-2%) of the entire corpus. Term-based features have a low computational cost, because they lookup just a few statistics per query term per shard. Costs for query-independent features are lower still. The third experiment compares a slow model that uses all features (ALL) to a fast version that does not use sample-document features (FAST).
Table 10: Performance of L2R-MQT using feature sets constructed with leave-one-out. ‘w/o X’ means the feature was excluded from FAST. Text in bold indicates the lowest value in the column.

<table>
<thead>
<tr>
<th>Feature Set</th>
<th>P@10</th>
<th>NDCG@30</th>
<th>MAP@1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>CW09-B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(T=8) FAST</td>
<td>0.373</td>
<td>0.285</td>
<td>0.201</td>
</tr>
<tr>
<td>w/o Unigram</td>
<td>0.303</td>
<td>0.226</td>
<td>0.138</td>
</tr>
<tr>
<td>w/o Bigram</td>
<td>0.364</td>
<td>0.275</td>
<td>0.187</td>
</tr>
<tr>
<td>w/o Independent</td>
<td>0.368</td>
<td>0.282</td>
<td>0.199</td>
</tr>
<tr>
<td>Gov2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(T=12) FAST</td>
<td>0.592</td>
<td>0.471</td>
<td>0.310</td>
</tr>
<tr>
<td>w/o Unigram</td>
<td>0.592</td>
<td>0.468</td>
<td>0.301</td>
</tr>
<tr>
<td>w/o Bigram</td>
<td>0.582</td>
<td>0.462</td>
<td>0.296</td>
</tr>
<tr>
<td>w/o Independent</td>
<td>0.591</td>
<td>0.471</td>
<td>0.303</td>
</tr>
</tbody>
</table>

We estimate the resource selection cost by the amount of data retrieved from storage. For CSI-based features, the cost is the size of postings of every query term in the CSI [Kulkarni 2013]. For term-based features, the cost is the amount of sufficient statistics required to derive all term-based features [Aly et al. 2013]. The query-independent feature only looks up the shard popularity, so the cost is one statistic per shard.

Table 9 compares FAST with ALL and baselines by their accuracy and average resource selection cost per query. ReDDE results were similar to Rank-S and are not shown. Taily has been the state-of-the-art term-based (‘faster’) resource selection algorithm. However, FAST is substantially more accurate. FAST also outperformed Jnt with over $100 \times$ speed up. Compared to ALL, FAST is 67 times faster on CW09-B and 34 times faster on Gov2. Although FAST has slightly lower search accuracy than ALL, the gap is not large and is not statistically significant, indicating that the information from the CSI features can be covered by the more efficient features.

We conclude that a resource ranker composed of only query-independent and term-based features is as accurate as exhaustive search and a ranker that includes CSI features (Research Question 5). CSI features improve accuracy slightly, but at a significant additional computational cost.

**Importance of Feature Types:** We investigate the contribution of other types of features: query-independent features and term-based features, where the term-based features were sub-divided into unigram and bigram features. Table 10 presents the results for the leave-one-out analysis conducted on FAST. On CW09-B, removing any feature set from FAST led to lower performance. This indicates that each set of features covers different types of information, and all are necessary for accurate shard ranking. Among these features, unigram features were most important because CW09-B has many single-term queries. On Gov2, the only substantial difference is observed when bigram features are excluded. This is expected as Gov2 on average has longer queries than CW09-B.

### 6.6 Summary

This chapter investigates a learning-to-rank approach to resource selection for selective search. Much attention has been devoted to learning-to-rank documents, but there has been little study of learning-to-rank resources such as index shards. Our research shows that training data for
this task can be generated automatically using a slower system that searches all index shards for each query. This approach assumes that the goal of selective search is to mimic the accuracy of an exhaustive search system, but with lower computational cost. This assumption is not entirely true—we would like selective search to also be more accurate—but it is convenient and effective.

We show that the learned resource selection algorithm produces search accuracy comparable to exhaustive search down to rank 1,000. This work is the first that we know of to demonstrate results that are statistically significantly equivalent to exhaustive search for MAP@1000 on an index that does not have badly skewed shard sizes. Accuracy this deep in the rankings opens up the possibility of using a learned reranker on results returned by a selective search system, which was not practical in the past.

Most prior research found that sample-document algorithms such as ReDDE and Rank-S are a little more accurate than term-based algorithms such as Taily for selective search resource selection; however, sample-document resource selection algorithms have far higher computational costs that increases query latency in some configurations [Kim et al. 2016a]. This work suggests that sample-document features provide only a small gain when combined with other types of features. It may no longer be necessary to choose between accuracy and query latency when using a learned resource ranker.
Previous investigations argued that selective search is substantially more efficient than a typical distributed search architecture based on the number of postings processed when evaluating a single query stream [Kulkarni 2013; Kulkarni and Callan 2010; Kulkarni et al. 2012]. While this metric is suitable for comparing the work done between different architectures, it does not consider how work is divided across processors, or how to compare multiple query streams being evaluated in parallel. The traditional distributed search architecture using a random assignment of documents to shards tends to spread the workload evenly across processors, and is relatively immune to bottlenecks. In contrast, a selective search architecture, which deliberately concentrates similar documents into a few index shards, might be more prone to uneven workloads, and hence risk leaving processing resources idle. Selective search might also be more sensitive to tuning parameters.

This chapter uses an event-based simulator to investigate the efficiency of the selective search architecture. A simulator makes it possible to investigate a wider range of machine configurations than would be practical in a live system. Our simulator was tuned to provide realistic measurements of query waiting times, query processing costs, query latency, system throughput, and hardware utilization under a parallel query processing environment representative of a practical real-world implementation (Section 7.2). Our investigation also extends prior work by defining a more realistic experimental methodology for studying efficiency that uses more similarly sized index shards, employs long query streams extracted from web search logs, and varies query arrival rates in order to better measure the trade-off between query latency and query throughput (Section 7.3). In particular, we present a detailed study of the computational costs, load distribution, and throughput of selective search in order to address four specific research questions:

RQ 7 Is selective search more efficient than exhaustive search in a parallel query processing environment? (Section 7.4.1)

RQ 8 How does the choice of resource selection algorithm affect throughput and load distribution in selective search, and how can any imbalances originating from resource selection be overcome? (Section 7.4.2)

RQ 9 How do different methods of allocating shards to machines affect throughput and load distribution across machines? (Section 7.4.3)

RQ 10 Does selective search scale efficiently when adding more machines and/or shard replicas? (Section 7.4.4)

Our experiments address many of the questions that have arisen in connection with the efficiency and load balancing characteristics of selective search, to both broaden and deepen our understanding of this retrieval architecture.

7.1 RELATED WORK

In search and distributed systems, simulations are commonly used to evaluate the efficacy of new methods and diagnose problems because acquiring, configuring, and producing repeatable experiments with a variety of real hardware configurations and system variables is expensive and time-consuming [Webber and Moffat 2005].

Tomasic and García-Molina [1993] used a simulation to compare the performance of different physical organizations for inverted lists and evaluated the impact of various system variables on the optimal strategy. In particular they found the access time of the storage device and interprocessor communication bandwidth had a strong impact on the optimal layout.

Simulations are particularly popular in evaluating distributed systems. Cahoon et al. [2000] and Cacheda et al. [2004, 2007a,b] used simulations to evaluate distributed search system architectures and identify system bottlenecks. A range of system parameters were used and based on these experiments, they recommend an ideal architecture. Cacheda et al. [2004] placed a stronger focus on searching large text collections and Cacheda et al. [2007b] focused on a more detailed and accurate simulation of network models and outlined improvements to distributed search that can be made to reduce network bottlenecks. Our network implementation follows this detailed model. [Cacheda et al. 2007a] compared clustered and replicated distributed search architectures.

Simulated systems and workloads have been used to judge query scheduling and evaluation schemes as well [Freire et al. 2013; Moffat et al. 2006].

Badue et al. [2007] explores some of the limitations of these previous works, which assume that homogeneous servers have balanced execution times. They found that even when documents are randomly distributed, load imbalances occur due to differences in caching behavior in the shard servers that process queries, which is further exacerbated by any differences in main memory size. This work is particularly relevant to selective search, which may have an even stronger load imbalance due to the topically focused shards.

7.2 SIMULATION MODEL

A simulator was developed based on DESMO-J, a discrete event simulation modeling framework [DESMO-J]. The simulator has the benefit of allowing us to quickly investigate a range of hardware configurations, and provides precise estimates of a broad suite of performance indicators. The implementation models a selective search system that incorporates a cluster of multi-core machines, and mimics parallel query execution across those machines. Figure 15 describes the computational model embedded in the simulator and Table 11 lists the quantities that are manipulated.

The hardware is assumed to consist of M machines, with the i’th of those, machine m_i, providing c_i CPU cores (c_i = 8 throughout the chapter). Machines may be configured to act as a broker, or as a searcher, or may be configured to handle both roles.

A broker machine holds a copy of the resource selection database and performs two tasks: resource selection, and result merging. For resource selection, the machine has access to a shared central query queue, from which it extracts incoming queries, determines which shards need to be
searched, and then assigns shard search tasks to other machines. Each broker machine also has a job queue for pending result merge processes. This queue contains results returned by the searcher machines, now waiting to be merged to produce a final result list for some query.

A machine \( m_i \) is a searcher if it is allocated \( p_i > 0 \) selective search shards. A searcher also has a job queue that holds shard search requests pertinent to the shards hosted on that machine. Each of the available cores on the machine can access any of the shards assigned to the machine, and hence can respond to any request in that machine’s search queue. When a search job is finished, the result is returned to the result merge queue of the originating broker. The assignment of shards and copies of the resource selection database to machines is assumed to be fixed at indexing time, and machines cannot access shards that are not hosted locally. A key factor for success is thus the manner in which the \( P \) shards are partitioned across the \( M \) machines.

Algorithm 2 describes the sequence of actions that take place in each of the machines. First, if the machine is a broker, the local result merge queue is checked for queries for which all shard searches have been completed, and merged output lists are generated if any queries can now be finalized. Otherwise, if the machine is a searcher, the local shard search queue is checked to see if there are any shard searches pending; if so, the next one is extracted and actioned, and the results directed to the merge queue for the machine that acted as broker for that query, which might be
Table 11: Simulation parameters.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M$</td>
<td>Number of machines; $m_i$ is the $i$’th of these.</td>
</tr>
<tr>
<td>$c_i$</td>
<td>Number of cores on $m_i$; the default is $c_i = 8$.</td>
</tr>
<tr>
<td>$C$</td>
<td>Total number of cores, $\sum_{i=1}^{M} c_i$.</td>
</tr>
<tr>
<td>$p_i$</td>
<td>Number of shards assigned to $m_i$.</td>
</tr>
<tr>
<td>$P$</td>
<td>Total number of shards. When each shard is assigned to just one machine, $P = \sum_{i=1}^{M} p_i$.</td>
</tr>
<tr>
<td>$B$</td>
<td>Number of broker machines.</td>
</tr>
<tr>
<td>$S$</td>
<td>Number of searcher machines.</td>
</tr>
<tr>
<td>$T$</td>
<td>Query arrival rate described by an exponential distribution with mean $1/\lambda$, $T = \lambda$.</td>
</tr>
<tr>
<td>$t_s$</td>
<td>Seek plus latency access time, milliseconds per postings list, $t_s = 4$ throughout.</td>
</tr>
<tr>
<td>$t_p$</td>
<td>Processing cost, milliseconds per posting, $t_p = 9 \times 10^{-4}$ throughout.</td>
</tr>
<tr>
<td>$t_m$</td>
<td>Merging cost, milliseconds per item, $t_m = 5 \times 10^{-5}$ throughout.</td>
</tr>
</tbody>
</table>

the same machine. A shard search process on machine $m_i$ can search any shard assigned to that machine.

If neither of these two activities are required, and if the machine is a broker, the next query (if one exists) is taken from the central query queue and resource selection carried out. The result of resource selection is a list of shards to be searched in order to resolve the query; that list is mapped to a set of machine identifiers, and the query $q$ is added to the shard search queues for those machines. The processing order for these operations prioritizes query completion over query initiation. This minimizes the processing time for each query, and ensures that no query has an infinite wait time.

In the simulation, queries are assumed to arrive at the central queue at random intervals determined by an exponential distribution governed by a mean query arrival rate $T$. The number of machines permitted to host broker processes may be less than $M$, the total number of machines, but is always at least one. Query processing costs at the shards are computed based on the number of postings processed from the shard index, plus an overhead cost to account for initial latency for a disk seek. A postings list of $\ell$ postings is thus presumed to require $t_s + \ell \cdot t_p$ milliseconds, where $t_s = 4$ milliseconds [Shimpi], and $t_p = 9 \times 10^{-4}$ milliseconds per posting. The processing rate – around a million postings per second – is based on measurement of the cost of handling posting lists in the open source Indri search engine [The Lemur Project] on a machine with a 2.44 GHz CPU, and encompasses I/O costs as well as similarity calculation costs. In the case of in-memory execution, $t_s$ can be set to zero. The parameter $t_s$ can also be used to selectively emulate caching of postings lists. For simplicity, we assume all postings are cold cached in the experiments presented in this chapter.

Resource selection costs differ according to the approach used. For sample-based algorithms such as ReDDE or Rank-S, the cost is dominated by the need to process postings from the central sample index (CSI). For these approaches, the same computational model is used as for shard
### Algorithm 2 – Processing loop for each core on machine $m_i$.

```plaintext
while forever do
    if isBroker($m_i$) and $|\text{mergequeue}_i| > 0$ and all shard responses have been received for $q$
    then
        remove those responses from $\text{mergequeue}_i$
        finalize the output for query $q$ and construct a document ranking
    else if isSearcher($m_i$) and $|\text{searchqueue}_i| > 0$
    then
        remove a query request $(q, p, b)$ from $\text{searchqueue}_i$
        perform a shard search for query $q$ against shard $p$
        append the results of the search to $\text{mergequeue}_b$
    else if isBroker($m_i$) and $|\text{centralqueue}| > 0$
    then
        remove a query $q$ from $\text{centralqueue}$
        perform resource selection for $q$
        for each partition $p$ to be searched for $q$ do
            determine the machine $m_h$ that is host for $p$
            append $(q, p, i)$ as a search request to $\text{searchqueue}_h$
        endfor
    endwhile
```

search, and a cost of $t_s + \ell \cdot t_p$ milliseconds is assumed for a postings list of length $\ell$. On the other hand, term-based algorithms such as Taily process statistics from each shard that contains a query term. The cost for term-based approaches is thus equivalent to processing a posting list of length equal to the number of shards that contain the query term, which is always less than or equal to $P$, the number of shards.

Result merging may require network transfer if the results are returned to a broker that is not located within the same machine as the shard.

This requires transferring of up to $k$ $(\text{doc-id}, \text{score})$ results returned from each shard searched, where $k$ is either fixed on a system-wide basis, or is determined as part of the resource selection step. Network messages are also generated when brokers request searches for shards that are not stored within the same machine. To ensure that the simulation was accurate, the cost of network communication over a Gigabit switched network was modeled as described by Cacheda et al. [2007b]. The cost of merging was measured on the same $2.44$ GHz machine, and an allocation of $t_m = 5 \times 10^{-5}$ milliseconds per document was found to be appropriate.

The system is configured to return the top 1,000 documents to support applications such as learning to rank algorithms, text-mining applications, and TREC evaluations that need deeper rankings. Cacheda et al. [2007b] showed that when shards are formed randomly, only a small number of documents need to be returned from each shard for the true top-$k$ documents to be returned with a high probability. Therefore, in the exhaustive search system used as a baseline, each shard only returns the number of documents that results in a $10^{-5}$ probability of missing a result in the top 1,000. This equates to returning 102 documents per shard for a 16 shard configuration and 19 documents per shard for 512 shards. The assumption that documents are distributed randomly across shards does not apply to the topical indexes used by selective search; clustering concentrates similar documents in a small number of shards. Thus, Cacheda’s technique cannot be used with selective search and each selective search shard returns 1,000 documents. Since the exhaustive search baseline accesses all shards, whereas selective search typically accesses 3-5 shards, the total number of documents that must be merged by the two architectures is roughly compara-
ble. In our experiments, the total number of documents merged by the two architectures usually varied between 1,600 and 9,700, depending upon the number of machines (exhaustive search) and query (selective search). Generally, exhaustive search merges fewer documents than selective search in configurations with fewer machines (and random shards), and more documents than selective search in configurations with more machines (and random shards).

Overall, the simulator takes as input a list of queries, the resource selection cost for each query, the shards to be searched for the query, and the search cost for each shard. The cost is described by the total length of the posting lists retrieved for all query terms for the specified shard or resource selection method. The simulator then converts these posting list costs into “simulator milliseconds”. The simulator calculates the overall elapsed time required to process each query as the difference between the arrival time of that query in the central queue, and the moment at which all processing of that query is completed. This cost includes time spent waiting in queues, network delays, and computation time. The median end-to-end elapsed query processing time is used as the primary measure of query latency, but we also show the latency distributions in some key cases. The principal control variable in the simulator is the query arrival rate, which determines the load in the system, and hence the extent to which query response times are affected by queuing delays.

The simulator also tracks the load on each simulated machine. The load is measured by the utilization of the simulated cores, as a time-weighted fraction of available capacity that is utilized for query computation. The output of this measurement was used to evaluate the evenness of load across the simulated machines.

7.3 EXPERIMENTAL METHODOLOGY

The simulator was used for a detailed evaluation of selective search schemes, applying them to two large experimental collections. This section describes the data resources employed, including a revised partitioning scheme for the experimental collections that constructs more similarly-sized partitions, and gives details of the effectiveness baselines we compare against.

7.3.1 Document collections

Two web datasets, ClueWeb09 Category A English (abbreviated to CW09-A) and Gov2 were used in the experimentation. These are large collections, covering more than 500 million and 25 million web pages respectively. The selective search configurations applied to them are derived from the shard definitions for CW09-A and Gov2 generated as part of the investigation carried out by Kulkarni [2013]. Those investigations established the validity of the selective search approach, and explored clustering techniques and resource selection processes in order to determine how to achieve search effectiveness comparable to exhaustive search.

One issue with the CW09-A shard map produced by Kulkarni is that the shard sizes can vary. Figure 16 plots the distribution of shard sizes, and shows that the largest one is 12 times the average size. This imbalance is problematic for two reasons. Firstly, there is a significant correlation between the size of the shard and the frequency at which it is selected for querying (Pearson’s correlation of 0.5). That is, the larger the shard, the more likely it is to be selected for querying. Secondly, when they do get selected, large shards typically take longer to process queries, because of their size. In combination, these two effects produce an imbalance in computation load which is not present in the Gov2 shards.
To address the imbalance of shard sizes in CW09-A, the largest shards in the original shard partitioning of Kulkarni were divided; large shards were split via a random assignment when the resulting sub-shards would be closer to the average shard size (by number of documents) than the complete shard. A total of 51 shards were split into two or more smaller shards, resulting in an increase of 77 shards, for a total of 884. Other statistics for the two datasets are listed in Table 12.

### 7.3.2 Resource selection parameter settings

Two resource selection mechanisms are used in our experiments: Rank-S [Kulkarni et al. 2012] and Taily [Aly et al. 2013]. Both methods require that values be set for two system parameters. For Rank-S the two key parameters are the size of the sample for the centralized index (the CSI), and the quantity base used for the exponential discounting. Taily also requires two parameters to be specified: n, conceptually the depth of the result list used for estimating relevant document counts and v, the score cut-off. The parameters used in our experiments are as suggested by Aly et al. [2013] and Kulkarni [2013], and are summarized in Table 13.

Table 14 and Table 15 list the overall effectiveness of selective search using the default parameters, using queries that span multiple years of the TREC Terabyte and Web Tracks. We use Precision at rank 10 (P@10) and Normalized Discounted Cumulative Gain at rank 30 (NDCG@30) to measure the effectiveness of the systems, which makes these results comparable to prior work [Aly et al. 2013; Kulkarni et al. 2012]. These metrics are precision-oriented, that is, they place importance in the top of the ranked result list.
Table 13: Resource selection parameter settings, based on TREC Terabyte Track and TREC Web Track queries for which relevance judgments are available, as developed by Kulkarni et al. [2012] and Aly et al. [2013]. Those investigations used a variety of CSI sizes for Gov2, ranging from 0.5% to 4%. We used 1%, for consistency with CW09-A. The table also presents the average number of shards (using the shard assignments listed in Table 12) selected by the parameter settings.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Selection algorithm</th>
<th>Parameters</th>
<th>Average shards searched</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gov2</td>
<td>Taily</td>
<td>n = 400, v = 50</td>
<td>3.0</td>
</tr>
<tr>
<td></td>
<td>Rank-S</td>
<td>CSI = 1%, base = 3</td>
<td>4.7</td>
</tr>
<tr>
<td>CW09-A</td>
<td>Taily</td>
<td>n = 400, v = 50</td>
<td>3.3</td>
</tr>
<tr>
<td></td>
<td>Rank-S</td>
<td>CSI = 1%, base = 5</td>
<td>4.2</td>
</tr>
</tbody>
</table>

Table 14: Effectiveness of selective search using various resource selection algorithms on the Gov2 dataset. The “Oracle” shard ranking assumes that the most useful four shards are identified and searched for each query.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Terabyte Track 2004</th>
<th>Terabyte Track 2005</th>
<th>Terabyte Track 2006</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P@10</td>
<td>NDCG@30</td>
<td>P@10</td>
</tr>
<tr>
<td>Baseline</td>
<td>0.56</td>
<td>0.43</td>
<td>0.62</td>
</tr>
<tr>
<td>Rank-S</td>
<td>0.57</td>
<td>0.42</td>
<td>0.59</td>
</tr>
<tr>
<td>Taily</td>
<td>0.51</td>
<td>0.37</td>
<td>0.48</td>
</tr>
<tr>
<td>Oracle</td>
<td>0.64</td>
<td>0.47</td>
<td>0.64</td>
</tr>
</tbody>
</table>

Note that the effectiveness results are independent of the number of machines and how shards are assigned to machines. Once the assignment of documents to shards has been completed, and the resource selection parameters determined, selective search generates the same final ranked list regardless of the number of machines or shard allocations.

That fact means that these effectiveness results apply uniformly to all of the efficiency experiments we report. The baseline scores reported for exhaustive search were generated using structured queries and the sequential dependency model (SDM) [Metzler and Croft 2005], removing spam documents from the final result using the Waterloo Fusion spam scores [Cormack et al. 2011], at a threshold of 50%. The selective search runs use a similar set-up whenever possible; for Rank-S, the CSI was searched using SDM queries. For both Taily and Rank-S, the selected shards were searched using SDM queries and the result list was filtered for spam. These arrangements provide a more effective baseline than that used by Kulkarni [2013], and one that is comparable to the baseline used by Aly et al. [2013], producing the strongest possible baseline run.

In addition to the 2009 and 2010 Web Track queries used in previous investigations, our experiments include query sets from the 2011 and 2012 Web Tracks. Results for query sets are reported separately rather than averaging across query sets, in order to distinguish the effects of different tasks on selective search. Notable in these results is that selective search is not always as competitive as previously reported, particularly for the CW09-A dataset.

In the results reported by Kulkarni [2013] and by Aly et al. [2013], effectiveness was averaged across the 2009 and 2010 Web Track query sets. However, the two query sets have different char-
The 2010 queries tend to be more difficult, resulting in lower effectiveness from the exhaustive search baseline. The exhaustive search system had difficulty separating relevant documents from similar-looking non-relevant documents. However, selective search was less affected because clustering assigned these documents to different shards, and then resource selection ignored the shards that contained the similar-looking non-relevant documents. Thus, as observed by Powell et al. [2000], a partitioned index and good resource selection became more effective than exhaustive search. This effect is further studied in detail in Chapter 3.

When the 2010 results are averaged with the 2009 results, the weaker 2009 results are masked, and it appears as if selective and exhaustive search have comparable performance. While the accuracy results are reported with our modified shard maps, these results hold when using the original shard map with very little difference in accuracy.

The effectiveness results reported in Table 14 and Table 15 suggest that the parameters used for Taily and Rank-S might not be optimal. To address that concern, we performed a parameter sweep to check for better settings, but were unable to identify parameters that both yielded accuracy comparable to exhaustive search and also searched a moderate number of shards. We also experimented with a clairvoyant “oracle” shard ranking, in which the four shards containing the greatest number of relevant documents were always presumed to always be selected. Those hypothetical results are shown in the final row of Table 14 and Table 15. The marked gap between the attained effectiveness and the goal set by the oracle system indicates that the accuracy of selective search is a function of the resource selection process, and makes it clear that further algorithmic improvements may be possible. This subject is later explored in Chapter 3 and Chapter 6. We use the configurations reported in Table 13, and in the remainder of this chapter focus solely on efficiency.

### 7.3.3 Query streams

A key contribution of this work is the measurement of the performance of selective search under realistic query loads, and determination of the point at which each configuration saturates. For these experiments a much larger query log is needed than the Web Track and Terabyte Track query sets used in the effectiveness study. The experiments that are the main focus of this chapter make use of the AOL query log\(^2\) and the TREC Million Query Track query set [Carterette et al. 2009]. These query logs are from a different timespan than the collections, but the queries are

---

2 Queries span March through May 2006.
Table 16: Sizes of the query logs, in thousands. The “Train” column indicates training queries used to set parameters. The “Test” column indicates testing queries used for reporting results. Columns “TestW” and “TestM” indicate queries that were sampled starting one week and one month after the Train queries, respectively. Those two query streams are used in Section 7.4.3.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Log</th>
<th>Train</th>
<th>Test</th>
<th>TestW</th>
<th>TestM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gov2</td>
<td>AOLG</td>
<td>1K</td>
<td>10K</td>
<td>10K</td>
<td>10K</td>
</tr>
<tr>
<td>Gov2</td>
<td>MQTG</td>
<td>1K</td>
<td>10K</td>
<td>10K</td>
<td>10K</td>
</tr>
<tr>
<td>CW09-A</td>
<td>AOLW</td>
<td>1K</td>
<td>10K</td>
<td>10K</td>
<td>10K</td>
</tr>
<tr>
<td>CW09-A</td>
<td>MQTW</td>
<td>1K</td>
<td>10K</td>
<td>10K</td>
<td>10K</td>
</tr>
</tbody>
</table>

only used to measure efficiency, not effectiveness, and the small discrepancy in their temporal coverage is not a concern.

In order to simulate a live query stream, the AOL log was sorted by timestamp, and deduplicated to only contain unique queries. Deduplication is performed because a production system would serve repeated queries from a cache, and most queries would be seen only once by the search engine. For CW09-A, the first 1,000 queries from the log were then used as training data to set configuration parameters such as B, the number of brokers (Section 7.4.2), and assignments of shards to machines (Section 7.4.3). The next 10,000 queries were used for testing. Together, the queries cover a time period of 2006-03-01 00:01:03 to 2006-03-01 01:31:45. For Gov2, the timestamp-sorted AOL query stream is also filtered to form a query stream where there is at least one .gov-domain result click recorded for each query. The first 1,000 queries were again used as training data, and the next 10,000 queries used for testing, covering the time period 2006-03-01 00:01:08 to 2006-03-01 20:09:09. Two further test query sets were also extracted from the AOL log: 10,000 queries starting from 2006-03-08, one week after the main query stream (TestW); and another 10,000 queries commencing 2006-04-01, one month after the main query stream (TestM). These two additional test sets were used in the experiments described in Section 7.4.3.

Another pair of query streams was created using the TREC Million Query Track (MQT) queries. The MQT queries have no timestamps and were used in the order they appear in the files provided by the National Institute of Standards and Technology (NIST). For Gov2, the first 1,000 queries of the 2007 query set were used for training; 10,000 queries from the 2008 query set were used for testing. For CW09-A, both the 1,000 training and 10,000 testing queries were extracted from the TREC 2009 MQT sequence. In total, 84,000 queries were used at various stages of the evaluation, split across the categories summarized in Table 16.

### 7.4 Experimental Results

Our experiments investigate the efficiency and load characteristics of a selective search architecture under a variety of conditions. The first set of experiments evaluates an environment similar to the one proposed by Kulkarni [2013], but allowing for parallel execution of queries and tasks (Section 7.4.1). The second suite of experiments explores resource selection costs, resource selection algorithms, and how to overcome any computational load imbalances originating from resource selection (Section 7.4.2). The third round of experiments investigate the load and throughput effects of two different policies for assigning shards to machines (Section 7.4.3). The last experi-
The experimental results section compares index-spreading and mirroring strategies when using additional computational resources (Section 7.4.4).

### 7.4.1 Selective search efficiency

In this section, we address Research Question 7 – Is selective search more efficient than exhaustive search in a parallel query processing environment? The selective search architecture is compared to a typical exhaustive search architecture in small-scale environments similar to those examined by Kulkarni and Callan [2010a] and Kulkarni [2013]. Each test collection was divided into topical shards and the shards randomly distributed across all machines, with each machine receiving the same number of shards. The same collections were also used to build an exhaustive search baseline by constructing \( C \) (the total number of cores) evenly sized shards via a random assignment of documents, and then allocating one shard per core to each machine; \( c = 8 \) in our experiments, thus 8 shards were assigned to each machine. This allows exhaustive search to minimize latency by making use of all cores for every query. In both configurations, only one machine \( B = 1 \) accepted broker tasks, so as to emulate previous arrangements as closely as possible.

Table 17 summarizes the average number of shards selected by the resource selection algorithms with parameters as discussed in Section 7.3.2. Note the high variance in the number of shards selected by Taily for the AOL query set used with CW09-A.

When the AOL query log is used with the CW09-A collection, the distribution of Taily shard scores levels off quickly and becomes flat, which makes it difficult for Taily to distinguish between shards. This causes the mean and standard deviation of the number of shards selected to be much higher than in other configurations. The AOL query set used with Gov2 does not experience the same issues; these queries are cleaner due to the filtering process documented in Section 7.3.3. Even with these variances affecting behavior, selective search examines only a small fraction of available shards and produces significant efficiency gains regardless of the resource selection method deployed.

Figure 17 shows the throughput of two selective search variants, compared to exhaustive search. In each of the frames the vertical axis shows the median time to process each query, plotted as a function of the query arrival rate on the horizontal axis. Alternative summary metrics such as the mean or the 95% percentile processing times were also explored, and produced similar outcomes. Each plotted curve represents a combination of query set and processing regime, and the two frames correspond to the two collections. The elbow in each plotted curve indicates the point at which that system configuration approaches saturation. To avoid extreme results, each

### Table 17: Average number of shards selected by Taily and Rank-S for the Test logs, measured using the configurations depicted in Figure 17.

<table>
<thead>
<tr>
<th>Query log</th>
<th>Resource selection</th>
<th>Gov2 avg</th>
<th>Gov2 sddev</th>
<th>CW09-A avg</th>
<th>CW09-A sddev</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQT</td>
<td>Taily</td>
<td>2.5</td>
<td>1.4</td>
<td>3.6</td>
<td>2.9</td>
</tr>
<tr>
<td></td>
<td>Rank-S</td>
<td>4.2</td>
<td>1.8</td>
<td>4.4</td>
<td>1.7</td>
</tr>
<tr>
<td>AOL</td>
<td>Taily</td>
<td>2.9</td>
<td>1.6</td>
<td>11.9</td>
<td>31.3</td>
</tr>
<tr>
<td></td>
<td>Rank-S</td>
<td>4.6</td>
<td>1.9</td>
<td>4.2</td>
<td>1.7</td>
</tr>
</tbody>
</table>
Figure 17: Exhaustive search and selective search using CW09-A and Gov2 and random shard assignment for the AOLW Test and MQT W Test, AOLG Test and MQTG Test query sets (shown as AOL and MQT in the labels for brevity). In these experiments, \( M = 2 \), \( B = 1 \), and \( S = 2 \).

The curve was truncated at the point at which the median processing time for queries exceeded twice the median processing time for queries in an unloaded system, determined by the latency at 0.01 queries per second, at the left-hand end of the curve. Query arrival rates were chosen dynamically for each curve, so that the region of greatest gradient can be clearly distinguished. Configurations where the curve is lower have lower latency. Configurations with elbows that are further to the right require fewer resources per query and attain higher throughput rates when the system is approaching saturation.

To describe the variation of query processing times, Table 18 presents the 50th, 75th and 99th percentile, mean, and max values of query processing time for several configurations of selective search. The dispersion of query processing time is computed at the last point in the plotted curves of Figure 17 – the point where the system is approaching saturation. Table 19 presents the breakdown of average costs for queries at the system saturation point for the configurations studied. The network costs and merging costs are minimal. As expected of a system under full load, queries spend a significant amount of time in the central query queue, and in queues internal to each system. Note the differences in the time spent in resource selection between Rank-S and Taily. Systems using Rank-S, a sample-based algorithm, spent a larger portion of time performing resource selection, particularly when using CW09-A. This also produces higher latency for Rank-S systems as seen in Figure 17. This result is expected because sample-based resource selection methods are more expensive than term-based methods. This topic is explained in more detail and further explored in Section 7.4.2.

When querying the CW09-A collection (Figure 17b), selective search outperforms exhaustive search by a factor of more than ten, because only a small fraction of the 884 shards are searched. Query latency is also lower in selective search, despite the two-step process of resource selection followed by shard search. This is due to fewer resources being used by selective search, and the fact that at low query loads, latency is largely determined by the slowest partition that is polled. Topical shards are smaller than random shards, thus they can be searched more quickly. A larger fraction of the fifty possible shards are searched in the Gov2 collection (Figure 17a), so
Table 18: Dispersion of per-query processing times for selected configurations of exhaustive, Rank-S and Taily methods, at the last point on each of the corresponding curves in Figure 17. Columns display the 50th, 75th, 99th percentiles, the mean, and the maximum value of individual query processing times, where time is measured in simulation milliseconds. In these experiments, $M = 2$. Recall that the simulation was tuned to model the costs associated with Indri, a research search engine; a commercial system would likely have lower processing times.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Queries Method</th>
<th>$1/\lambda$ (qry/s)</th>
<th>Percentile</th>
<th>Mean</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>50%</td>
<td>75%</td>
<td>99%</td>
</tr>
<tr>
<td>Gov2</td>
<td>MQT Exh</td>
<td>0.28</td>
<td>2,268</td>
<td>4,409</td>
<td>11,676</td>
</tr>
<tr>
<td></td>
<td>Rank-S</td>
<td>3.20</td>
<td>1,632</td>
<td>2,651</td>
<td>6,206</td>
</tr>
<tr>
<td></td>
<td>Taily</td>
<td>5.50</td>
<td>1,189</td>
<td>1,901</td>
<td>4,579</td>
</tr>
<tr>
<td>AOL</td>
<td>MQT Exh</td>
<td>0.42</td>
<td>250</td>
<td>1,126</td>
<td>5,016</td>
</tr>
<tr>
<td></td>
<td>Rank-S</td>
<td>7.50</td>
<td>397</td>
<td>865</td>
<td>2,606</td>
</tr>
<tr>
<td></td>
<td>Taily</td>
<td>12.00</td>
<td>265</td>
<td>612</td>
<td>1,868</td>
</tr>
<tr>
<td>CW09-A</td>
<td>MQT Exh</td>
<td>0.04</td>
<td>5,819</td>
<td>26,510</td>
<td>110,232</td>
</tr>
<tr>
<td></td>
<td>Rank-S</td>
<td>3.20</td>
<td>1,468</td>
<td>3,537</td>
<td>13,082</td>
</tr>
<tr>
<td></td>
<td>Taily</td>
<td>9.00</td>
<td>510</td>
<td>1,012</td>
<td>3,340</td>
</tr>
<tr>
<td>AOL</td>
<td>MQT Exh</td>
<td>0.03</td>
<td>21,736</td>
<td>50,525</td>
<td>172,978</td>
</tr>
<tr>
<td></td>
<td>Rank-S</td>
<td>1.80</td>
<td>3,263</td>
<td>6,220</td>
<td>18,351</td>
</tr>
<tr>
<td></td>
<td>Taily</td>
<td>0.75</td>
<td>902</td>
<td>3,869</td>
<td>21,350</td>
</tr>
</tbody>
</table>

the performance improvement is not as high. Even so, selective search of Gov2 handles four to five times the rate of queries as exhaustive search before saturating.

Taily has better throughput and latency than Rank-S for the majority of settings tested. The only exception is for the AOL$_W$ Test query set in CW09-A, where the larger number of shards searched by Taily eclipses the lower resource selection costs, resulting in better throughput for Rank-S. More broadly, selective search delivers markedly better performance characteristics than exhaustive search in all of the configurations investigated, reaffirming the findings of Kulkarni and Callan [2010a] and Kulkarni [2013] that selective search is a viable framework for improving the efficiency of parallel query processing.

7.4.2 Resource allocation

Figure 17 indicates that Taily is more efficient than Rank-S, with significantly better query latency and throughput in the majority of configurations tested. The difference is a direct consequence of the two approaches to resource selection. Aly et al. [2013] demonstrated that resource selection costs can be a substantial portion of the total retrieval cost in a system that uses sample-based resource selection algorithms, such as ReDDE and Rank-S. We now examine in detail how the different resource selection computations affect throughput and load of selective search, across a range of different system configurations in order to answer Research Question 8 – How does the...
Table 19: Breakdown of the average time spent processing queries, at the last point on each of the corresponding curves in Figure 17. Central queue indicates the percentage of time spent in the central query queue; network indicates the delays caused by network congestion; internal queues indicates time spent in system queues such as a machine’s shard search queue or merge queue; resource selection indicates time spent performing resource selection; and merge is time spent performing the final merge operation. The remaining time not included in these categories was spent in shard search.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Queries Method</th>
<th>1/λ (qry/s)</th>
<th>Central queue</th>
<th>Network</th>
<th>Internal queues</th>
<th>Resource selection</th>
<th>Merge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gov2 MQT</td>
<td>Exh</td>
<td>0.28</td>
<td>21.88%</td>
<td>0.06%</td>
<td>14.91%</td>
<td>-</td>
<td>0.09%</td>
</tr>
<tr>
<td></td>
<td>Rank-S</td>
<td>3.20</td>
<td>32.16%</td>
<td>&lt;0.01%</td>
<td>3.35%</td>
<td>13.38%</td>
<td>0.01%</td>
</tr>
<tr>
<td></td>
<td>Taily</td>
<td>5.50</td>
<td>30.38%</td>
<td>&lt;0.01%</td>
<td>3.44%</td>
<td>1.58%</td>
<td>0.01%</td>
</tr>
<tr>
<td>AOL MQT</td>
<td>Exh</td>
<td>0.42</td>
<td>21.41%</td>
<td>0.07%</td>
<td>16.92%</td>
<td>-</td>
<td>0.01%</td>
</tr>
<tr>
<td></td>
<td>Rank-S</td>
<td>7.50</td>
<td>25.67%</td>
<td>0.01%</td>
<td>2.58%</td>
<td>14.67%</td>
<td>0.04%</td>
</tr>
<tr>
<td></td>
<td>Taily</td>
<td>12.00</td>
<td>20.55%</td>
<td>0.01%</td>
<td>2.68%</td>
<td>2.50%</td>
<td>0.03%</td>
</tr>
<tr>
<td>CW09-A MQT</td>
<td>Exh</td>
<td>0.04</td>
<td>31.57%</td>
<td>&lt;0.01%</td>
<td>25.24%</td>
<td>-</td>
<td>&lt;0.01%</td>
</tr>
<tr>
<td></td>
<td>Rank-S</td>
<td>3.20</td>
<td>25.55%</td>
<td>&lt;0.01%</td>
<td>1.72%</td>
<td>53.44%</td>
<td>0.01%</td>
</tr>
<tr>
<td></td>
<td>Taily</td>
<td>9.00</td>
<td>26.36%</td>
<td>0.01%</td>
<td>5.10%</td>
<td>1.66%</td>
<td>0.02%</td>
</tr>
<tr>
<td>AOL MQT</td>
<td>Exh</td>
<td>0.03</td>
<td>30.86%</td>
<td>&lt;0.01%</td>
<td>31.08%</td>
<td>-</td>
<td>&lt;0.01%</td>
</tr>
<tr>
<td></td>
<td>Rank-S</td>
<td>1.80</td>
<td>22.40%</td>
<td>&lt;0.01%</td>
<td>1.93%</td>
<td>59.67%</td>
<td>&lt;0.01%</td>
</tr>
<tr>
<td></td>
<td>Taily</td>
<td>0.75</td>
<td>36.77%</td>
<td>&lt;0.01%</td>
<td>7.54%</td>
<td>0.46%</td>
<td>0.02%</td>
</tr>
</tbody>
</table>

Figure 18: Average utilization of machines using Rank-S, the CW09-A dataset, the AOL-W Test queries, M = 2, B = 1 and S = 2. Each point is the mean over 10 sequences of 1,000 queries; error bars represent 95% confidence intervals.

The choice of resource selection algorithm affects throughput and load distribution in selective search, and how can any imbalances originating from resource selection be overcome?
In the experiments illustrated in Figure 17, only one machine acted as broker, and the shards were evenly distributed across the two machines \((B = 1, S = 2)\). This is similar to the configuration described by Kulkarni [2013]. However, this configuration is not optimal for selective search, and produces an uneven machine load, especially when using Rank-S to select shards. Figure 18 plots the machine utilization of this configuration, using Rank-S, \(M = 2\) machines, and searching CW09-A with the AOL\_W Test query set. All broker tasks are handled by machine \(m_1\), and this machine is also a searcher. Consequently, \(m_1\) is heavily loaded when compared to the other machine, and the majority of its load is generated by resource selection. A detailed examination of the \(m_1\) workload reveals that resource selection is responsible for more than 70% of the machine’s computational load when the arrival rate is 2.5 queries per second. Merging requires near-zero load. A similar imbalance is observed when other simulator parameters such as query streams, collections, and hardware are varied.

We then configured the machines in the Rank-S system to allow more broker processes. When tuning a selective search system, this decision can be based on training queries. We compare the results of \(B = 1\) to \(B = 2\), a configuration where both machines perform resource selection. Figure 19 shows these parameter settings when applied to the Test queries and the Rank-S MQT\_G Test and AOL\_W Test configurations. When \(M = 2\), the best Rank-S settings were \(B = 2\) for Gov2 with an even distribution of shards for both MQT\_G Train and AOL\_G Train logs. This configuration gives small improvements in throughput over the baseline of using \(B = 1\). For Rank-S on CW09-A, \(B = 2\) was also the best configuration for AOL\_W Train and MQT\_W Train, but the throughput improvements are much larger. The difference in behavior for these datasets is due to the size of the CSI. Rank-S uses a 1% sample of the corpus to make decisions. For Gov2, a 1% sample is about half the size of an average shard, but for CW09-A, 1% is about eight times the size of an average shard. Most of the work for CW09-A is devoted to resource selection while for Gov2 most of the work is dedicated to shard search, as was seen in Table 19. Similar results were obtained using Rank-S with other settings.

Taily requires far less computation for resource selection than does Rank-S, and the best setting was \(B = 1\) for all query logs, and all datasets. At \(M = 2\) and \(B = 1\), resource selection for Taily
accounted for less than 2% of \( m_1 \)'s processing capability. This illustrates the advantage of term-based algorithms – efficiency. In most configurations, Taily also consumes fewer computational resources overall than Rank-S. As is shown in Figure 17b, the resulting performance gains can be large.

Overall, the choice of resource selection algorithm has a significant impact on the throughput, latency, and load distribution of selective search; and unless care is taken, load imbalances can emerge even when just a few machines are being used. In the larger CW09-A dataset, Taily had significant latency and throughput advantages compared to Rank-S, because of the high cost of searching the larger Rank-S CSI. However, sample-based algorithms have other advantages, including the ability to run structured queries.

7.4.3 Shard assignment policies

The preceding experiments assigned shards to machines randomly. With two machines (\( M = 2 \)), random assignment of shards to machines distributes query traffic evenly across machines, since there are many more shards than machines (\( P \gg M \)). However, this assumption does not hold as \( M \) increases because selective search deliberately creates shards that have skewed term distributions and topical coverage. So, in this section we examine Research Question 9 – How do different methods of shard allocation affect throughput and load distribution across multiple machines?

Figure 20 shows the relative popularity of the 80 most used CW09-A shards, and demonstrates that the skewed term distributions means that some shards are selected more often than others. For example, when the Rank-S resource selection algorithm is used to select topic-based shards for the Gov2 queries from AOLC Train query set, the five most frequently selected shards account for 29% of all shards selected – 1,302 out of 4,491 shard selections. In CW09-A, the AOL query log also exhibits a strongly skewed popularity pattern. The MQT query set displays a milder, but still noticeable, skew. This unevenness of popularity has the potential to adversely affect throughput in a system distributed over multiple machines by overloading machines that are responsible for popular shards and creating bottlenecks that then starve other machines of work. Even a small increase in the number of machines to \( M = 4 \) can lead to a load imbalance.
The next set of experiments investigates the effectiveness of Random assignment, and compares it with a Log-based algorithm, which uses training data to estimate and balance the average load across machines. The Log-based approach takes a set of training queries, performs resource selection, and computes the sum of the postings costs for each shard. That sum is used as an estimate of the shard’s load. Shards are then ordered most to least expensive, and assigned to machines one by one, in each case choosing the machine that currently has the lowest estimated load. Finally, any remaining shards that were not accessed by the training queries are assigned based on size. In these experiments a set of 1,000 training queries was employed. Using Taily’s shard selections, Gov2 had no unaccessed shards for either query set, and 7% and 4% of shards were unaccessed for AOLW Train and MQTW Train respectively in CW09-A.

In this experiment, we focus only on the performance of Taily. The results for Rank-S are similar. Figure 21 shows the effect of the shard assignment policies on machine utilization as query arrival rates are varied for the Gov2 dataset using the Million Query Track queries. Machine utilization (or load) varies from 0–100% (shown as 0.0 to 1.0). The range of machine loads show that the Random policy produces an uneven utilization of machines, and saturates relatively quickly due to a bottleneck on m4. In comparison, the Log-based policy reduces variance of load noticeably, and produces a more even utilization of the machines. Detailed results for other configurations are given in Table 20, and display similar usage patterns. In all settings, Log-based assignment produces more uniform resource utilization than does Random.

The risk of using Log-based allocations is that the learned attributes may become outdated as a result of changes in the query stream. Table 21 investigates this potential shortcoming by showing machine usage when processing three query sets each containing 10,000 queries: one that arrived immediately after the training queries; a second set that arrived one week after the training queries; and a third set that arrived one month after the training queries. Average utilization is similar in each case, and variance increases marginally as the query stream evolves, but the changes are generally small. Results for the AOL log for Gov2 one month after assignment are an
Table 20: Average load and range (max load - min load) of shard search machine utilization as the training data ages, using the Log-based shard allocation policy. Other simulation settings are as described in Table 20. Test queries must begin immediately after the training queries. TestW queries begin from 1 week and 1 month after the training queries, respectively. The MQT queries do not have timestamps and so were not used in this experiment.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Queries</th>
<th>Allocation method</th>
<th>Avg load and range of load for each query arrival rate T</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>avg  range avg range avg range avg range</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6 qry/s 8 qry/s 10 qry/s 12 qry/s</td>
</tr>
<tr>
<td>Gov2</td>
<td>MQT_G Test Random</td>
<td>0.32 0.30 0.42 0.40 0.53 0.49 0.61 0.58</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Log-based</td>
<td>0.32 0.10 0.42 0.14 0.53 0.17 0.63 0.21</td>
<td></td>
</tr>
<tr>
<td>Gov2</td>
<td>AOL_G Test Random</td>
<td>0.40 0.19 0.59 0.28 0.73 0.34 0.78 0.36</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Log-based</td>
<td>0.40 0.14 0.59 0.20 0.73 0.25 0.78 0.26</td>
<td></td>
</tr>
<tr>
<td>CW09-A</td>
<td>MQT_W Test Random</td>
<td>0.36 0.08 0.53 0.11 0.70 0.15 0.81 0.17</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Log-based</td>
<td>0.36 0.05 0.53 0.07 0.70 0.09 0.82 0.10</td>
<td></td>
</tr>
<tr>
<td>CW09-A</td>
<td>AOL_W Test Random</td>
<td>0.39 0.07 0.52 0.10 0.65 0.12 0.76 0.14</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Log-based</td>
<td>0.39 0.01 0.52 0.02 0.65 0.02 0.77 0.03</td>
<td></td>
</tr>
</tbody>
</table>

Table 21: Average load and range (max load - min load) of shard search machine utilization as the training data ages, using the Log-based shard allocation policy. Other simulation settings are as described in Table 20. Test queries must begin immediately after the training queries. TestW and Test_M queries begin from 1 week and 1 month after the training queries, respectively. The MQT queries do not have timestamps and so were not used in this experiment.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Query log</th>
<th>Average load and range of load for each query arrival rate T</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>avg  range avg range avg range avg range</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20 qry/s 30 qry/s 40 qry/s 50 qry/s</td>
</tr>
<tr>
<td>Gov2</td>
<td>AOL_G Test</td>
<td>0.40 0.14 0.59 0.20 0.73 0.25 0.78 0.26</td>
</tr>
<tr>
<td></td>
<td>AOL_G Test_W</td>
<td>0.38 0.13 0.56 0.19 0.71 0.24 0.76 0.25</td>
</tr>
<tr>
<td></td>
<td>AOL_G Test_M</td>
<td>0.32 0.14 0.47 0.20 0.62 0.26 0.72 0.30</td>
</tr>
<tr>
<td>CW09-A</td>
<td>AOL_W Test</td>
<td>0.39 0.01 0.52 0.02 0.65 0.02 0.77 0.03</td>
</tr>
<tr>
<td></td>
<td>AOL_W Test_W</td>
<td>0.39 0.01 0.51 0.02 0.63 0.02 0.75 0.02</td>
</tr>
<tr>
<td></td>
<td>AOL_W Test_M</td>
<td>0.40 0.02 0.53 0.02 0.66 0.02 0.77 0.03</td>
</tr>
</tbody>
</table>

exception and have a markedly lower utilization due to a burst of shorter queries that occurred at this time (2.18 words on average versus 2.41 and 2.44 for Test_W and Test queries respectively), but
7.4 EXPERIMENTAL RESULTS

Figure 22: The variance of the load spread ($\text{max load}_i - \text{min load}_i$) of ten Random shard allocations, with $M = 4$, $B = 1$. The midline of the box indicates the median, the outer edges the 25th and 75th percentile values, and the whiskers the most outlying values. The load spread for Log-based shard allocation is also plotted as a reference point. Note the different horizontal and vertical scales in the two panes.

Figure 23: The variance of the load spread ($\text{max load}_i - \text{min load}_i$) of ten Random shard allocations, now with $M = 8$, $B = 8$. All other parameters are as for Figure 22. Note the different horizontal scales in the two panes.

the variance still remains similar. Shard assignments should be periodically revised to maximize throughput, but it is not necessary to do it frequently, and the cost of periodically refreshing shard assignments can be amortized.

An additional concern is that the Random allocations used in Figure 21 and Table 20 may not be representative of the distribution of all possible allocations, as there could be significant variation between two Random allocations. One allocation could have nearly perfect organization of shards by coincidence alone, while another might group the popular shards on a single machine, causing a severe load imbalance. In order to explore the variability of Random allocations, we generated
ten different allocations per combination of collection and query stream, and measured the variance of the loads. Figure 22 presents the results for the AOL-G Test query set of Gov2 and the MQT_W Test query set of CW09-A. Also plotted on the graph is the load spread of the Log-based allocation method. While there was at least one Random allocation that produced a more uniform load than the Log-based method, there was also significant variability in the load spread for different Random allocations, and the median values were all well above the corresponding value for the Log-based method. Figure 23 shows that when the number of machines is increased to $M = 8$, Log-based allocation produces the best load distribution even when compared to ten different Random allocations. That is, the Log-based method reliably picks an allocation with low spread when compared to Random allocation.

Balanced assignment of shards and higher utilization of machines can lead to higher query throughput, as shown in Figure 24. The Random policies shown in this figure are the median performing assignment from Figure 22 and Figure 23. At $M = 4$, the Log-based policies are distinctively better than a Random policy for the Gov2 dataset (especially for MQT). The difference between Random and Log-based policies widens when $M = 8$ machines are used. The differences between the allocation strategies for CW09-A dataset are much smaller due to the smaller differences in load variance between the Random and Log-based assignment as can be seen in Table 20, Figure 22, and Figure 23. Correcting greater load imbalances produces a larger impact in throughput.

Finally, Log-based assignment produces better utilization than Random allocation even after one month in most settings, further supporting the stability of the assignment. It is clear that the method of allocating shards to machines has an impact in load distribution of selective search, and hence throughput.

7.4.4 Scalability

The main focus of this work is on low-resource environments (typically two to eight machines). But one advantage of a simulation model is that other configurations can also be explored, and we are also able to examine large-scale environments. Distributed IR systems typically achieve scalability in two ways: shard replication and/or increasing the number of machines available in the cluster so that each is responsible for a smaller volume of data. We explore the scalability of selective search using these two approaches, and consider Research Question 10 – Does selective search scale efficiently when adding more machines and/or shard replication? Note that the experiments in this section are for CW09-A only, because Gov2 only has 50 shards.

Figure 25 compares selective search and exhaustive search with $M = 64$ machines in operation, and uses the methods for load balancing described in Section 7.4.2 and Section 7.4.3. With more machines in use, the latency of exhaustive search is decreased because the shard sizes are smaller. Selective search’s latency remains the same because the number of topical shards is not determined by the number of machines. There are 512 random shards and 884 topical shards, thus one might expect selective search to have lower latency than exhaustive search, but exhaustive search is faster. Query term posting lists are shorter in the random shards than in the topical shards selected by Taily or Rank-S, which allows an exhaustive shard to be searched more quickly than a topical shard. However, selective search still provides substantially higher throughput than exhaustive search; selective search may take longer to search one shard, which produces higher latency, but it also searches many fewer shards, which produces higher throughput.
Figure 26 further demonstrates the throughput advantage held by selective search. In this experiment, the total time in simulation seconds required to process 10,000 queries is measured as a function of the number of machines available. In all configurations, selective search remains a better option for throughput, requiring less time to process queries. This experiment also shows that selective search scales with the number of machines available. Both exhaustive and selective search are nearly parallel to the red dash-dot line, which represents the ideal case where throughput is doubled with doubled machine capacity.

In Figure 25 and Figure 26, an index spreading strategy was assumed, in which the shards are distributed across the larger number of machines, and then the number of machines allocated to resource selection and shard search are adjusted to make the best use of the new hardware. However, when there are more machines than shards, index spreading is no longer feasible and a different method must be used. One alternative is to increase throughput by using additional space and adopting a mirrored architecture. For example, if the number of machines is doubled, two copies of each shard, twice as many resource allocation cores, and twice as many shard search cores can be used. Mirroring requires double the disk space of index spreading, but may have other advantages that make it desirable. Either approach can roughly double throughput.

Figure 27 compares the effect of index spreading and mirroring when $M = 64$ machines are employed, again plotting median query latency as a function of query arrival rate. In most configurations, mirroring has a small throughput advantage, derived from the fact that as the number of machines increases, it becomes increasingly less likely that an evenly balanced shard assignment will occur. For example, at $T = 280$, the standard deviation of the machine load in the CW09-A MQT Test queries is 5.2% for 64 machines, and 3.9% for a 32 machine mirrored configuration using the index spreading strategy with Log-based allocation. While the throughput differences are small, mirroring provides additional benefits, such as fault tolerance in case of failure. In addition, as mentioned previously, mirroring is clearly the best approach when there are more machines than total shards.

The best overall solution in environments with high query traffic, or many machines may be a mix of index spreading and strategic shard mirroring (replication), an option also noted by Moffat et al. [2006]. This is may be an interesting topic for future work.

7.5 SUMMARY

Selective search has been shown to be substantially more efficient than the standard distributed architecture if computational cost is measured by counting postings processed in a one-query-at-a-time environment [Kulkarni and Callan 2010a,b; Kulkarni 2013]. We have extended those findings using a simulator that models a realistic parallel processing environment and a wider range of hardware configurations to be explored; and long query streams extracted from the logs of web search engines.

Previous investigations also demonstrated that selective search is as effective as exhaustive search, based on experiments with two large datasets [Kulkarni and Callan 2010a,b; Aly et al. 2013; Kulkarni 2013]. Although our work focused primarily on selective search efficiency, we refined the experimental methodology used to measure effectiveness, and achieved stronger baseline results for the CW09-A dataset. One consequence of these changes was the discovery that in some cases selective search is less effective than exhaustive search, regardless of which resource selection algorithm is used. Effectiveness was not our focus in this work, but our findings are recorded to benefit other researchers that wish to pursue this topic.
Our investigation makes it clear that selective search is more efficient than conventional distributed architectures for parallel query processing. In a low-resource environment with few machines, the two-step selective search architecture – resource selection followed by shard access – can deliver greater total throughput (number of queries processed per time interval) as well as lower latency (faster response time) than a conventional exhaustive architecture. The latter result is one that some readers may find surprising, and is a consequence of the small size of the shards used by selective search, about 500K documents per shard in our experiments. When more machines are available, exhaustive search has lower latency but selective search always has substantially higher throughput.

Other studies have shown that sample-based and term-based resource selection algorithms have different advantages and costs [Aly et al. 2013]. Our experiments investigated the effects of the resource selection algorithm on load distribution, latency, and throughput. We found that Rank-S was more accurate than Taily, but also much more computationally expensive, usually resulting in higher latency and lower throughput than Taily. If one chooses to use Rank-S to obtain greater effectiveness, the computational costs cause a skew in the workload of machines that must be corrected by replicating resource selection on multiple machines.

Selective search uses topical shards that are likely to differ in their popularity. We discovered that typical random assignments of shards produce large imbalances in machine load, even when as few as \( M = 4 \) machines are assigned. A Log-based assignment policy using training queries provided higher throughput and more consistent query processing times than a random assignment policy. The Log-based assignment is also resistant to temporal changes, and even after a delay of a month, throughput degradation was minimal.

Previous studies only investigated selective search in computing environments that contained a small number of machines (low resource computing environments). With the aid of the simulator we also examined the behavior of selective search using clusters of up to \( M = 64 \) machines, each with eight processing cores. We found that selective search remains a viable architecture with high throughput in these larger-scale computing environments. When additional processing resources are available, mirroring (replicating all index shards) provides slightly better throughput in addition to advantages such as fault-tolerance and the ability to use more machines than there are shards when compared to index spreading (no replication). It may be sufficient to replicate a few high-load shards rather than the entire index, which would greatly reduce storage costs without greatly reducing throughput. We reserve this interesting problem for future work.

After a thorough investigation of the efficiency characteristics of the selective search architecture, we conclude that it is highly efficient in low resource environments typical of academic institutions and small businesses, and that the load imbalances of a naive configuration can be readily addressed. At larger scale, the latency advantages may be lost, but selective search continues to provide substantially higher throughput. For batch-oriented or offline tasks where latency is less of a concern than total computational cost, selective search remains the better choice even at large scales.
Figure 24: The effect of shard assignment policies on throughput of selective search, using Taily with $M = 4$, $B = 1$ (left column) and $M = 8$, $B = 8$ (right column). Note the differing horizontal scales in each pair.
Figure 25: Selective search compared to exhaustive search with $M = 64$, $B = 64$ using CW09-A and log-based shard assignment for the AOL Test and MQT Test query sets (shown as AOL and MQT in the labels for brevity).

Figure 26: The total time required by a selective search system to process 10,000 queries as a function $M$, the number of machines. Assumes an infinite query arrival rate and $B = M$ in all configurations. The two query streams used are MQT Test and AOL Test, denoted as MQT and AOL respectively. The red dash-dot line represents the ideal case, where throughput is doubled with doubled machine capacity.
Figure 27: Throughput achieved with Log-based shard assignment and $M = 64$, $B = 64$ to process queries against CW09-A. The mirrored configuration is a doubled $M = 32$, $B = 32$ system, including disk space. The two query streams are MQT$_G$ Test and AOL$_G$ Test, denoted as MQT and AOL respectively.
The mapping of documents into shards, called a shard map, can be generated using many different partitioning schemes (e.g. text similarity, source domain, geography, random) [Kulkarni and Callan 2010a; Baeza-Yates et al. 2009b; Cambazoglu et al. 2010; Brefeld et al. 2011] and many resource selection algorithms are available for selective search [Aly et al. 2013; Kulkarni et al. 2012; Dai et al. 2017].

The shard map can greatly affect the accuracy of the end-to-end selective search system and is one of multiple such components [Dai et al. 2015]. Thus, it is desirable to be able to measure the quality of shard maps independent of the other components in the selective search system. In prior work, the effectiveness of shard maps were evaluated with end-to-end selective search systems using queries with relevance judgements [Kulkarni and Callan 2010a; Dai et al. 2016]. This method is affected by the choices made in other system components. Building an end-to-end system and gathering relevant judgements also can be costly. A measurement of shard map quality independent of other system components would provide better diagnostic information and allow a selective search system to be tuned more easily and quickly.

This chapter¹ introduces a new method for estimating shard map effectiveness called Area Under Recall Curve (AUReC). AUReC is calculated independent of other selective search system components. It requires only a set of queries and the results of the full collection retrieval for those queries; from this data, AUReC can then be used to quickly identify good shard maps for an efficient selective search system. To our knowledge, AUReC is the first to fully decouple shard map evaluation from other selective search components.

We compare AUReC against end-to-end system evaluations using three different resource selection algorithms with relevance judgements to answer the following research questions:

**RQ 11** Can AUReC generate a relative ordering of multiple shard maps that is consistent with orderings generated by end-to-end system evaluations?

**RQ 12** Is AUReC consistent with end-to-end system evaluations on whether differences between two shard maps are statistically significant?

**RQ 13** Is AUReC robust when compared to end-to-end systems using different resource selection algorithms; compared to a variety of IR metrics at different retrieval depths; when the search engine generating the top-k retrieval is weaker; and when it is calculated with a different set of queries than the end-to-end evaluation queries?

Section 8.1 reviews prior work in evaluating shard maps, gives background on selective search, and details the motivation for this work. Section 8.2 details the process for calculating AUReC.

¹ This work is currently under review.
Section 8.3 describes the experimental methodology. Section 8.4 presents the experimental results and Section 8.5 concludes.

8.1 RELATED WORK AND MOTIVATION

A shard is a search index that contains a subset of the total document collection. A shard map is the mapping that describes to which shard each document in the collection belongs, which is determined by the partitioning scheme, a method or technique for generating shard maps, e.g. text-similarity clustering, grouping documents by geography, or splitting a collection randomly.

Three areas of prior research created and used shard maps. Cluster-based retrieval systems sought to improve accuracy by dividing the corpus into topical clusters and searching just within cluster(s) most relevant to each query. Distributed retrieval systems partition a large corpus into shards so that processing effort can be divided among multiple processors to improve response time (latency). Selective search systems are a type of cluster-based retrieval system that reduces computational effort by only searching the most relevant shard(s) for each query [Kulkarni and Callan 2010a]. Below we discuss each of these areas in more detail. We omit federated search systems because they do not create shard maps.

Cluster-based retrieval systems organize a corpus into topical clusters (shards). Different partitioning schemes were measured by their effect on search accuracy [Croft 1980; Voorhees 1985; Griffiths et al. 1986; Willett 1988]. A common method of assessing partition quality was to measure search results produced by a complete (“end-to-end”) system that selects k optimal clusters [Tombros et al. 2002; Griffiths et al. 1986]. This method has the advantage of separating evaluation of the shard map from the problem of identifying the best shard(s) (resource selection). However, Dai et al. [2015] argued that optimal selection gives misleading information because it measures only how well relevant documents are grouped together but ignores whether they are grouped in a way that allows them to be found. This method also requires manual relevance assessments and is sensitive to parameter choices for the end-to-end system.

Distributed retrieval divides a large collection into shards to improve efficiency. Term-based and document-based partitioning schemes have been suggested and the efficiency enabled by different partitioning schemes is well-studied [Moffat et al. 2007; Cambazoglu et al. 2013; Baeza-Yates et al. 2009b; Cambazoglu et al. 2010; Brefeld et al. 2011]. However, the effectiveness of different shard maps has received little attention.

Selective search aims to improve the efficiency of large scale search while maintaining the same quality as searching the entire collection (exhaustive search). Like cluster-based retrieval systems, selective search systems rely on shard maps that group similar documents together so that the documents relevant to a query are likely to appear in just a few shards. Random elements (e.g., sampling, random seeds) cause moderate variance in shard map quality that affects search accuracy [Dai et al. 2015].

Prior work on selective search evaluated shard maps using the end-to-end retrieval accuracy (e.g., MAP, NDCG@k, or P@k) of a complete system using that shard map. This methodology has the advantage of being realistic. However, as noted above, it requires manual relevance assessments and because the performance of a selective search system is affected by multiple components [Dai et al. 2015], often the end-to-end result is more representative of the effectiveness of the particular choice of components and how well they are tuned. Thus, a thorough system designer would generate many shard maps and test each one against a variety of system configurations,
using different resource selection algorithms and different efficiency levels [Dai et al. 2016]. This is cumbersome, so it is not done often. It is also more difficult to do well than one might expect.

Consider a case where shard maps containing a different numbers of shards are compared. Given two Gov2 shard maps containing 50 shards and 194 shards, the parameter $B = 3$ in Rank-S, a resource selection algorithm with dynamic shard cut-offs, selects $3.7/50$ shards and $4.0/194$ shards on average for the TREC Terabyte Track queries, leading to very different efficiency and accuracy. The parameters of the resource selection algorithm must be tuned for each shard map such that approximately the same fraction of documents are searched. An end-to-end evaluation of these two shard maps requires two parameter sweeps for this one parameter.

There is a need for shard map evaluation that is decoupled from other system components and parameters, and does not require relevance judgements. Such a tool would give better diagnostic information about the quality of the shard maps alone and enable rapid tuning of new selective search systems.

One method of decoupling evaluation from relevance judgments is to compare the results of a less thorough (less expensive) search to the results of a more thorough (more expensive) search that exhaustively searches the entire index. The more thorough, exhaustive search system is treated as the ‘gold standard’ that the less thorough system is intended to mimic. Exhaustive search has been used to measure a new system’s accuracy [Carmel et al. 2001] and efficiency [Kim et al. 2017]. It has also been used as a target to train a supervised resource selection algorithm [Dai et al. 2017]. Clarke et al. [2016] used overlap with exhaustive to evaluate the effectiveness of early-stage filters in a multi-stage retrieval system. In a selective search context, this is equivalent to evaluating the end-to-end selective search pipeline by overlap with exhaustive results. We are the first to apply this concept to scoring shard maps, independent of the other stages in selective search.

We borrow ideas from French and Powell [2000], who describe a recall-like measure, $\hat{R}_k(E, B)$, used to compare the performance of resource selection algorithms in federated search. Let $B$ represent an ideal ordering for query $q$ of a shard map containing $n$ shards; and $E$ the estimated ordering produced by a resource selection algorithm. Let $B_i$ represent the number of relevant documents in the $i$’th ranked shard in $B$, and similarly for $E_i$ and $E$. It should be the case that $B_i \geq B_{i+1}$ and this ordering of shards is called relevance-based ranking (RBR). It is not generally the case that $E_i \geq E_{i+1}$.

$$\hat{R}_k(E, B) = \frac{\sum_{i=1}^{k} E_i}{\sum_{i=1}^{n} B_i}$$

French and Powell plot $\hat{R}_k(E, B)$ from $k = \{1 \ldots n\}$ to visualize and compare the effectiveness of various resource selection algorithms. Our work develops a variation of the $\hat{R}_k$ curve to evaluate shard maps quantitatively, thereby transforming a visualization aid into a powerful diagnostic tool.

8.2 aurec

Our goal is a method of evaluating shard maps that that does not require relevance judgements and is independent of the resource selection algorithm, other system components, and parameter settings. While different resource selection algorithms have some differences, their shared goal is to find the shards with the most relevant documents and they tend to return similar shards. Thus,
the quality of a shard map can be treated as an inherent property that can be measured on its own.

Selective search reduces costs by searching as few shards as possible while trying to produce results that are as accurate as searching all the index shards (exhaustive search). For a given query \( q \), an ideal shard map i) groups the important documents \( D_q \) in as few shards as possible, and ii) places them in shards that contain many similar documents. The first condition enables most shards to be ignored. The second condition makes the right shards easy to recognize.

There are different ways to define \( D_q \). French and Powell [2000] defined it as documents that have been judged relevant by a human assessor. However, we would like to avoid reliance on human relevance judgements. In addition, considering only the relevant documents when evaluating shard maps creates data sparsity issues. Often only a small number of queries with relevance judgements are available and some queries have very few relevant documents. The sparsity of data can produce high variance results [Dai et al. 2015].

Alternatively, note the goal of selective search to meet, rather than to exceed, the accuracy of exhaustive search, just at a much lower cost. Thus, \( D_q \) can be defined as the documents that an exhaustive search system would return for the query. For the exhaustive search system, a well-tuned, strong ranker is preferred (e.g., a trained learning to rank system) for more accurate signals on quality of the documents. This method avoids the sparsity issues of relevance judgements; our experiments showed that the additional data generated by this method increases the robustness of AUReC.

Given a shard map \( p \) containing \( n_p \) shards, a query \( q \), and \( D_q \), the documents that should be retrieved for query \( q \), we can calculate the maximum possible recall of \( D_q \) when searching up to a limit of \( k \) shards as follows.

For query \( q \), let \( \text{count}(D_q, s^p_i), i = \{1 \cdots n_p\} \) represent the number of documents in \( D_q \) present in shard \( s^p_i \), where the shards were ordered such that \( \text{count}(D_q, s^p_i) > \text{count}(D_q, s^p_{i+1}) \). \( R_q(p,k) \) is the percentage of documents in \( D_q \) that appear in the first \( k \) shards of shard map \( p \), that is:

\[
R_q(p,k) = \begin{cases} 
1 & |D_q| = 0 \\
0 & |D_q| > 0, k = 0 \\
\frac{\sum_{i=1}^{k-1} \text{count}(D_q, s^p_i)}{|D_q|} & |D_q| > 0, k = \{1 \cdots n_p\}
\end{cases}
\]

\( R_q \) loosely represents recall (what portion of relevant documents can be found in the given shards) and \( k \) represents efficiency (searching less shards is more efficient). The relationship between the recall and efficiency in selective search can be described in graphical form by plotting \( k \) vs. \( R_q(p,k) \) for \( k = \{0 \cdots n_p\} \). It is a convex, monotonically increasing curve, and the x-axis is normalized to be between \([0,1]\) by dividing by \( n_p \). When comparing multiple shard maps for a given query, the curves of effective shard maps will be higher and to the left of less effective shard maps. An example is presented in Figure 28. In this example, shard map A is more effective than B, achieving higher \( R_q \) at lower values of \( k \). C is a shard map where documents in \( D_q \) were scattered evenly across all shards and thus \( R_q \) increases linearly with \( k \).

Our method calculates the area under this recall versus fraction of shards searched curve and is named Area Under Recall Curve (AUReC). AUReC of query \( q \) for shard map \( p \) can be calculated as follows, using the formula for calculating the area of trapezoids:
AUReC scores have the range $[0.5, 1.0]$. 0.5 indicates that the documents from $D_q$ are completely evenly distributed across the shards and is the worst possible score, shown by shard map C in Figure 28. Scores closer to 1.0 indicate that the documents from $D_q$ are tightly clustered in very few shards. 1.0 is a degenerate case that occurs when $D_q$ is empty.

$R_q(p, k)$ is comparable to $\hat{R}_k(B, B)$ [French and Powell 2000], where $B$ is the ideal ordering of shards in shard map $p$ (refer to Section 8.1). It differs by using exhaustive search runs rather than human-judged relevant documents. It also has defined values when $|D_q| = 0$ and $k = 0$, whereas the equivalent in $\hat{R}_k$ are undefined. The $k = 0$ case for $R_q$ is critical when comparing shard maps that contain a different number of shards, and it enables us to determine that a shard map $p_{100}$ consisting of 100 shards with $R_q(p_{100}, 1) = 1$ is better than a shard map $p_2$ consisting of 2 shards with $R_q(p_2, 1) = 1$, since the important documents are more densely clustered. Without this modification, AUReC plotted from $k = \{1...n_p\}$ (as opposed to $k = \{0...n_p\}$) is biased towards shard maps containing fewer shards.

AUReC is calculated on a per query basis similar to traditional information retrieval metrics and can be averaged across queries to generate a summary value. Paired significance testing is possible. In this chapter, unless otherwise specified, the AUReC score refers to the AUReC$_q$ averaged across all queries in the query set.

Note that no relevance judgements are used to calculate AUReC and no other component of selective search needs to be implemented. In particular, there is no need to tune a shard cut-off value or to pick an efficiency level for resource selection. Instead, the shard-recall curve describes how well a given shard map performs for a query over all shard cut-offs. Using only the results of
an exhaustive search engine, AUReC can be used to quickly tune a new, effective selective search system to replace the less efficient exhaustive search system.

8.3 Experimental setup

We investigate the effectiveness of AUReC by comparing it to full end-to-end system evaluations of selective search of various configurations; shard maps were evaluated with both methods on two large web data sets and the results are compared. Generating an end-to-end evaluation requires queries with relevance judgements and choosing a resource selection algorithm. AUReC requires the results of a strong retrieval engine. This section describes these necessary components and the metrics used to compare the two methods of evaluation.

8.3.1 Data and query sets

The experiments were conducted on two large web data sets. Gov2 is a TREC collection containing 25 million .gov domain websites\(^2\). ClueWeb09 Category B (ClueWeb09 B) is the first 50 million documents of the ClueWeb09 web crawl\(^3\). The shard maps used in evaluation were generated by Dai et al. [2016] using three different methods: KLD-Rand, QKLD-Rand and QKLD-Qinit, where each successive method produces shard maps of higher quality than the last. These partitioning schemes have random components and different random seed initializations produce different shard maps. Using 10 different random seeds, the authors generated 10 shard maps for each method for a total of 30 shard maps per data set, 60 shard maps overall\(^4\). This data set contains both shard maps that should be of similar qualities and different qualities, and is ideal for testing a new evaluation measure which should be able to distinguish the two cases.

To generate an end-to-end system evaluation, we utilize queries and relevance judgements published by TREC. Gov2 was evaluated with the TREC Terabyte Track queries spanning 2004–2006. ClueWeb09 B was evaluated with the TREC Web Track queries spanning 2009–2012. The relevance judgements for the TREC Web Track queries were filtered such that only documents in Category B were present. Furthermore, queries with zero judged relevant documents were removed from consideration (topics 20, 112, 143, 152 in the Web Track queries), since an end-to-end system evaluation of these queries cannot be used to rank shard maps. AUReC is calculated using the same queries but the relevance judgements are not used.

8.3.2 Document retrieval

Calculating AUReC requires knowing a set of documents that should be retrieved for a given query \(q\), \(D_q\). We create \(D_q\) using the top 1000 results of a strong retrieval engine to create the best possible substitute for relevance judgements. To create competitive near state-of-the-art results, SlideFuse-MAP [Anava et al. 2016], a data fusion technique was used to fuse the top 10 runs submitted to TREC in each year as ordered by MAP. The accuracy of the resulting retrieval runs is summarized in Table 22. The scores of SlideFuse-MAP were also used to rank the documents in the end-to-end selective search system. Thus, the evaluation of an end-to-end system uses

\[^2\]http://ir.dcs.gla.ac.uk/test_collections/access_to_data.html
\[^3\]https://lemurproject.org/clueweb09/
\[^4\]Downloaded from http://boston.lti.cs.cmu.edu/appendices/CIKM2016-Dai/
retrieval results of quality on par with the information used by AUReC, enabling apple-to-apple comparisons.

<table>
<thead>
<tr>
<th>Query set</th>
<th>Best submitted run (MAP)</th>
<th>Fused run (MAP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009</td>
<td>0.20</td>
<td>0.24</td>
</tr>
<tr>
<td>2010</td>
<td>0.24</td>
<td>0.28</td>
</tr>
<tr>
<td>2011</td>
<td>0.25</td>
<td>0.31</td>
</tr>
<tr>
<td>2012</td>
<td>0.30</td>
<td>0.34</td>
</tr>
<tr>
<td>Average</td>
<td>0.25</td>
<td>0.29</td>
</tr>
</tbody>
</table>

(a) ClueWebB

<table>
<thead>
<tr>
<th>Query set</th>
<th>Best submitted run (MAP)</th>
<th>Fused run (MAP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2004</td>
<td>0.31</td>
<td>0.35</td>
</tr>
<tr>
<td>2005</td>
<td>0.40</td>
<td>0.44</td>
</tr>
<tr>
<td>2006</td>
<td>0.42</td>
<td>0.45</td>
</tr>
<tr>
<td>Average</td>
<td>0.38</td>
<td>0.41</td>
</tr>
</tbody>
</table>

(b) Gov2

Table 22: Effectiveness of the top submitted TREC runs compared to the runs created from fusing the top 10 submitted runs for each query set with SlideFuse-MAP [Anava et al. 2016].

8.3.3 Resource selection and baselines

To compare AUReC to the current state-of-the-art, we must create several different configurations of end-to-end selective search systems, requiring a few different resource selection algorithms. Three resource selection algorithms were used in the experiments: Taily [Aly et al. 2013], a term-based algorithm; Rank-S [Kulkarni et al. 2012], a sample-based algorithm; and oracle method, relevance-based ranking (RBR) at various static shard cut-offs. As discussed in Section 8.1, RBR orders shards based on the relevance judgements of the query. Shards with the more relevant documents are ranked higher.

The parameters for Rank-S and Taily were set using the values from Kim et al. [2017] and are summarized in Table 23. When searching the CSI for Rank-S, SlideFuse-MAP is used to rank the documents. While Rank-S and Taily produce dynamic shard cut-offs (i.e. number of shards to search per query is built-in to the method and varies by query), RBR does not. Therefore, we evaluate RBR selective search systems at three different static shard cut-offs, \( k = \{1, 3, 5\} \), which is a common range for shard cut-offs used in prior work [Kim et al. 2017] and expresses a range of efficiency levels.

The three algorithms used for end-to-end evaluation present two different families of resource selection, two different strategies for shard cut-offs, and the upper bound of resource selection performance. By experimenting with three dissimilar resource selection algorithms, we demonstrate the robustness of AUReC.
<table>
<thead>
<tr>
<th>Data set</th>
<th>Algorithm</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gov2</td>
<td>Taily</td>
<td>$n = 400, v = 50$</td>
</tr>
<tr>
<td></td>
<td>Rank-S</td>
<td>1% CSI sample, $\text{base} = 3$</td>
</tr>
<tr>
<td>CW09-B</td>
<td>Taily</td>
<td>$n = 400, v = 50$</td>
</tr>
<tr>
<td></td>
<td>Rank-S</td>
<td>1% CSI sample, $\text{base} = 5$</td>
</tr>
</tbody>
</table>

Table 23: Parameter settings for Rank-S and Taily used in the end-to-end selective search system evaluations.

As AUReC is the first method to completely decouple shard map evaluation from other selective search components, there are no apple-to-apple baselines to which it can be directly compared. However, a common heuristic used in cluster-based retrieval systems is optimal cluster effectiveness, the effectiveness of performing retrieval on the one shard that contains the most relevant documents [Tombros et al. 2002]. This is exactly equivalent to an end-to-end system evaluation using RBR resource selection with $k = 1$. Although this method has dependencies on relevance judgements and shard cut-offs, we include this common option as a baseline in our experiments to provide context and a point of comparison. Note that variations of this baseline that do not use relevance judgements can be easily created as noted in Section 8.1. However, these baselines are weaker because they use less information. Thus, we chose to present the strictly stronger baseline of optimal cluster effectiveness.

8.3.4 Metrics

To establish AUReC as a robust method, we compare it to end-to-end selective search evaluations of multiple different settings. An end-to-end evaluation score $E_{tE}(p)$ of shard map $p$ is generated by evaluating queries on a selective search system based on shards dictated by shard map $p$. Multiple versions of $E_{tE}(p)$ are possible dependent on the type of resource selection algorithm used, and the evaluation metric used to score the final result list of selective search.

The primary evaluation metric used in this work is Precision at 1000 ($P@1000$); when used as a first-stage retrieval system, selective search must return deeper result lists to support later stage re-rankers [Macdonald et al. 2013]. Other metrics are explored in Section 8.4.2.

AUReC and the end-to-end system evaluation are compared in two major ways: list-wise and pair-wise. In the list-wise comparison, the correlation of the scores of $E_{tE}(p_i)$ and AUReC$(p_i)$ are calculated using Pearson’s $r$. Pearson’s $r$ was chosen over non-parametric, ordinal methods such as Kendall’s $\tau$ so that the correlation in the difference in the magnitude of the scores were measured, not just the relative rankings. This is important as the shard maps used in the experiments were generated with three different methods, with each method generating ten shard maps each. Thus, shard maps generated by the same method are expected to have similar scores, whereas shard maps generated by different methods should have larger score differences.

In the pair-wise comparison, for each pair of shard maps $(p_i, p_j)$ where $i, j = (1 \ldots 30), i \neq j$, we determine the ordering of AUReC$(p_i)$ and AUReC$(p_j)$ and whether the difference is statistically significant under a paired two-tailed t-test. This is repeated for $E_{tE}(p_i)$ and $E_{tE}(p_j)$. Given that there are 30 shard maps for a data set, the total number of pair-wise comparisons is $\binom{30}{2} = 435$. Based the decisions of AUReC and end-to-end evaluation on the direction and significance of the
8.4 experimental results

AUReC is compared against a full end-to-end system evaluation to investigate the correlation between the two metrics and see whether it can substitute for a full end-to-end evaluation. We then investigate the robustness of AUReC by exploring its behaviour when compared against different retrieval metrics and depths; when calculated with \(D_q\), the set of documents that should be retrieved for query \(q\), that was generated by a weaker retrieval engine; and when calculated with queries from a different query set.

8.4.1 Comparison with end-to-end systems

AUReC and optimal cluster effectiveness (equivalent to end-to-end evaluation with RBR, \(k = 1\)) are compared against end-to-end selective search systems utilizing various resource selection techniques. The results are presented in Table 24.

---

Figure 29: Contingency table for pair-wise comparison of index partitions using AUReC and end-to-end system evaluation (EtE).
### AUREC

<table>
<thead>
<tr>
<th>RS</th>
<th>r</th>
<th>Pairs recall</th>
<th>Overlap+Additional pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RBR k = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rank-S</td>
<td>0.94</td>
<td>176/236 = 0.75</td>
<td>0.84 + 0.03 = 0.86</td>
</tr>
<tr>
<td>Tally</td>
<td>0.90</td>
<td>161/184 = 0.88</td>
<td>0.89 + 0.06 = 0.94</td>
</tr>
<tr>
<td>RBR (k = 1)</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RBR (k = 3)</td>
<td>0.95</td>
<td>181/228 = 0.79</td>
<td>0.88 + 0.01 = 0.89</td>
</tr>
<tr>
<td>RBR (k = 5)</td>
<td>0.91</td>
<td>173/215 = 0.80</td>
<td>0.87 + 0.03 = 0.90</td>
</tr>
</tbody>
</table>

**AURSC**

<table>
<thead>
<tr>
<th>RS</th>
<th>r</th>
<th>Pairs recall</th>
<th>Overlap+Additional pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rank-S</td>
<td>0.96</td>
<td>221/236 = 0.94</td>
<td>0.71 + 0.21 = 0.93</td>
</tr>
<tr>
<td>Tally</td>
<td>0.92</td>
<td>181/184 = 0.98</td>
<td>0.65 + 0.23 = 0.88</td>
</tr>
<tr>
<td>RBR (k = 1)</td>
<td>0.94</td>
<td>185/187 = 0.99</td>
<td>0.66 + 0.27 = 0.93</td>
</tr>
<tr>
<td>RBR (k = 3)</td>
<td>0.98</td>
<td>219/228 = 0.96</td>
<td>0.72 + 0.23 = 0.96</td>
</tr>
<tr>
<td>RBR (k = 5)</td>
<td>0.96</td>
<td>206/215 = 0.96</td>
<td>0.69 + 0.25 = 0.94</td>
</tr>
</tbody>
</table>

(a) CW09-B

<table>
<thead>
<tr>
<th>RS</th>
<th>r</th>
<th>Pairs recall</th>
<th>Overlap+Additional pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rank-S</td>
<td>0.95</td>
<td>169/226 = 0.75</td>
<td>0.84 + 0.03 = 0.87</td>
</tr>
<tr>
<td>Tally</td>
<td>0.86</td>
<td>115/149 = 0.77</td>
<td>0.77 + 0.15 = 0.92</td>
</tr>
<tr>
<td>RBR (k = 1)</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RBR (k = 3)</td>
<td>0.93</td>
<td>169/224 = 0.75</td>
<td>0.85 + 0.02 = 0.87</td>
</tr>
<tr>
<td>RBR (k = 5)</td>
<td>0.89</td>
<td>166/229 = 0.72</td>
<td>0.82 + 0.03 = 0.85</td>
</tr>
</tbody>
</table>

**AURSC**

<table>
<thead>
<tr>
<th>RS</th>
<th>r</th>
<th>Pairs recall</th>
<th>Overlap+Additional pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rank-S</td>
<td>0.95</td>
<td>197/226 = 0.87</td>
<td>0.89 + 0.05 = 0.93</td>
</tr>
<tr>
<td>Tally</td>
<td>0.84</td>
<td>121/149 = 0.81</td>
<td>0.71 + 0.20 = 0.92</td>
</tr>
<tr>
<td>RBR (k = 1)</td>
<td>0.93</td>
<td>166/181 = 0.92</td>
<td>0.85 + 0.12 = 0.96</td>
</tr>
<tr>
<td>RBR (k = 3)</td>
<td>0.93</td>
<td>193/224 = 0.86</td>
<td>0.87 + 0.05 = 0.92</td>
</tr>
<tr>
<td>RBR (k = 5)</td>
<td>0.92</td>
<td>195/229 = 0.85</td>
<td>0.87 + 0.03 = 0.90</td>
</tr>
</tbody>
</table>

(b) Gov2

Table 24: Comparison of AURSC against the evaluation of end-to-end systems with various resource selection algorithms. The end-to-end systems used P@1000 to measure the effectiveness of the partitions. Also presented an end-to-end system evaluation using RBR k = 1, a commonly used heuristic in prior work.

The experiments demonstrate that AUReC is highly correlated with all end-to-end system evaluations in both list-wise and pair-wise comparisons. The high Pearson’s r indicates that the AUReC and end-to-end evaluation scores order shard maps similarly throughout the entire score range. Figure 30 visualizes the correlation. In addition, the best shard map selected by AUReC
AUReC also had a high *Pairs recall*, indicating that it recovered most of the statistically significant differences between shard map pairs found by the end-to-end systems. Finally, the high *Overlap+Additional pairs* sum shows that AUReC and end-to-end evaluations mostly agree on the direction of differences between pairs of shard maps. In addition, while not shown in the table, there were no pairs in which AUReC and end-to-end evaluations disagreed in the direction of statistical significance in any of the settings; that is, there were no pairs of shard maps \((p_i, p_j)\) where AUReC believed that \(p_i\) was statistically significantly better than \(p_j\) but an end-to-end evaluation believed vice versa.

Furthermore, AUReC correlates better with end-to-end systems than optimal cluster effectiveness, across both datasets and nearly all settings and comparisons, including systems that use RBR with different cut-offs. In particular, AUReC is superior in replicating statistically significant differences between pairs of shard maps (*Pairs recall*). This is because optimal cluster effectiveness produces high query variance. Other resource selection algorithms select 3–5 shards on average. Selecting only one shard means optimal cluster effectiveness sees fewer relevant documents and has less information to make judgements. Consequently, compared to AUReC, optimal cluster effectiveness produces higher variance and is less able to distinguish statistically significant differences and is less correlated with other systems overall. While often used in prior research [Tombros et al. 2002], it is a less reliable metric.

There are two places where optimal cluster effectiveness correlates better. First is the pair-wise comparisons against a Taily system in ClueWeb09 B, where optimal cluster effectiveness has a higher overall agreement (*Overlap+Additional pairs*) than AUReC. This was because Taily also produces high variance results [Dai et al. 2015] and most of the *Overlap* was from pairs where both methods did not find significant differences (58% of *Overlap* pairs). In contrast, most of
the Overlap between AUReC and Taily system was from concordance on statistically significant differences (64% of Overlap pairs), which is more informative.

Optimal cluster effectiveness also produced a slightly higher Pearson’s $r$ than AUReC when compared to the Taily system in Gov2. However, this difference is less meaningful; due to the high variance of in the Taily systems, most of the pair-wise differences between shard maps in the end-to-end evaluation are not significant. This means that in an ordered list of shard maps, there may be sections where multiple shard maps are interchangeable in order because the confidence intervals of the shard maps’ scores overlap each other. This introduces noise when calculating Pearson’s $r$ against a shard map ordering created by a Taily system and thus small differences become less meaningful.

The TREC Web Track queries have shallow judgement pools and produce high variance due to the dearth of signals [Lu et al. 2016]. In the work that described the shard maps used in our experiments, Dai et al. [2016] discovered that while the differences in the means for the end-to-end evaluation scores of the three categories of partitioning schemes (KLD-Rand, QKLD-Rand, and QKLD-QInit) were greater in ClueWeb09 B than Gov2, the differences were sometimes not statistically significant in ClueWeb09 B due to the higher variance. It was unclear which data set benefits more from the improved partitioning scheme. This can now be answered using AUReC.

AUReC is calculated based on 1000 retrieved documents and uses more information than evaluating an end-to-end retrieval with sparse relevance judgements. Thus, AUReC finds significant differences where an end-to-end system evaluation cannot. Note the amount of Additional pairs found by AUReC on the ClueWeb09 B dataset. These are shard map pairs where the end-to-end evaluation saw the same direction of difference as AUReC but could not conclude their statistical significance. The end-to-end evaluation found as few as 42% of shard map pairs to be significantly different (the denominator of the Pairs recall column), but AUReC found significant differences in 76% of shard map pairs in ClueWeb09 B (cells $a + d + g + c + f + i$ in Figure 29). A visual representation can be seen in Figure 30b. The three groups of shard maps are clearly distinguished when projected along the x-axis (AUReC) while the groups overlap when projected along the y-axis (P@1000). In contrast, Terabyte Track queries for Gov2 have deeper pooling for the relevance judgements, producing stable results. AUReC finds fewer Additional pairs in Gov2 and found significant differences in 50% of shard map pairs in Gov2, similar to end-to-end evaluations and lower than ClueWeb09 B. Figure 30a illustrates this. The differences between the groups of shard maps are similar for both AUReC and P@1000, and the QKLD-QInit and QKLD-Rand shard maps are less separated in Gov2 compared to ClueWeb09 B. Thus, thanks to evidence supplied by AUReC, we can now conclude that the choice of partitioning scheme affects ClueWeb09 B more strongly than Gov2.

AUReC typically generates fewer Additional pairs in Gov2. However, when the resource selection algorithm used by the end-to-end system has high variance (i.e. Taily and RBR $k = 1$) AUReC finds significant differences that the end-to-end systems miss.

With this experiment, Research Question 11, RQ 12 and a portion of Research Question 13 were answered. The ordering of shard maps generated by AUReC is highly correlated with the ordering determined by end-to-end system evaluations (Research Question 11). In pair-wise comparisons, AUReC found significant differences in pairs of shard maps where the end-to-end system found differences. In addition, AUReC was able to discover additional pairs of significantly different shard maps, where end-to-end systems were unable to ascertain the significance (Research Question 12). Finally, AUReC performed well across two data sets and three different resource selection algorithms, and substantially better than the common baseline heuristic used
8.4 Experimental results

by prior cluster-based retrieval research (Research Question 13). Further sections elaborate on Research Question 13 and examine AUReC under less ideal scenarios to test its robustness.

8.4.2 Different metrics and evaluation depths

So far P@1000 was used for the end-to-end system evaluation, which is a metric directly comparable to AUReC. AUReC is calculated with a top 1000 retrieval ordering of shards that maximizes the recall. Because there are 1000 documents in $D_q$, in most cases this is equivalent to maximizing P@1000 (where a document is considered ‘relevant’ if it is in $D_q$). However, not all metrics are so directly related to AUReC. If a metric takes rank position into account, a recall-oriented ordering of shards is not always optimal. In fact, for metrics such as NDCG and MAP, the optimal solution set of $k$ shards is not always a subset of the optimal set for $k + 1$ shards. To explore how well AUReC correlates with metrics that are less directly related, AUReC is compared against end-to-end systems evaluations using two rank sensitive metrics, MAP and NDCG at two different evaluation depths, 1000 and 100. The results are presented in Table 25. The table shows results for Rank-S end-to-end systems for brevity. Other resource selection choices behaved similarly, to the exception of RBR $k = 1$.

Optimal cluster effectiveness (i.e. RBR $k = 1$) was much less robust than AUReC and experienced a sharp drop in correlation in every setting and method of comparison. Both the change of metric and the change of evaluation depth adversely affected the correlation and a combination of both produced dismal results, especially in ClueWeb09 B (Pairs recall was 0.04 for MAP@100).

AUReC remains highly correlated with MAP and NDCG, and metrics evaluated at a shallower depth. In particular, differences in the type of evaluation metric did not impact the correlation in a major way. In Gov2, AUReC was slightly better correlated with NDCG, a rank sensitive metric, than Precision.

Evaluation depth had a stronger effect on the correlation of AUReC, yet still not necessarily negative. The results show that at shallower evaluation depths, the variance of the end-to-end systems increase, as seen by the lower number of statistically significant shard map pairs found by the end-to-end system, e.g. a P@1000 Gov2 system found 226 pairs with significant differences whereas the P@100 system found only 173. When compared against the shallower, more variable system evaluations, AUReC found more Additional pairs and Pearson’s $r$ was reduced. This is in line with the observations from Table 24 where similar effects were present when AUReC was compared to a Taily-based end-to-end system which has higher variance.

The increased variance was more pronounced in Gov2 than ClueWeb09 B. The queries for Gov2 have more relevant documents per query than those of ClueWeb09 B (182 vs. 50) and the pooling during the assessing phase was deeper. Gov2 queries have more judged documents further down in the ranked list than ClueWeb09, and thus were impacted more by the loss of this information with the use of a shallower evaluation depth.

Results for shallower depths were not reported because the increase in variance makes the results less interesting. For example, in ClueWeb09 B with P@10, more than 75% of shard map pairs have no significant differences in the Rank-S system. With such noise, list-wise comparison using Pearson’s $r$ becomes less informative and any cluster effectiveness metric that is high in variance would have a good pair-wise overlap with the end-to-end evaluation.

This experiment demonstrates that AUReC is well-correlated with different metrics and different evaluation depths. AUReC is a robust, low-variance method that can identify significant
<table>
<thead>
<tr>
<th>Metric</th>
<th>r</th>
<th>Pairs recall</th>
<th>Overlap+Additional pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baseline</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P@1000</td>
<td>0.94</td>
<td>176/236 = 0.75</td>
<td>0.84 + 0.03 = 0.86</td>
</tr>
<tr>
<td>NDCG@1000</td>
<td>0.89</td>
<td>97/248 = 0.39</td>
<td>0.64 + 0.01 = 0.65</td>
</tr>
<tr>
<td>MAP@1000</td>
<td>0.78</td>
<td>17/217 = 0.08</td>
<td>0.53 + 0.01 = 0.54</td>
</tr>
<tr>
<td>P@100</td>
<td>0.86</td>
<td>116/204 = 0.57</td>
<td>0.76 + 0.03 = 0.79</td>
</tr>
<tr>
<td>NDCG@100</td>
<td>0.75</td>
<td>10/214 = 0.05</td>
<td>0.53 + 0.00 = 0.53</td>
</tr>
<tr>
<td>MAP@100</td>
<td>0.62</td>
<td>8/190 = 0.04</td>
<td>0.57 + 0.01 = 0.58</td>
</tr>
<tr>
<td><strong>AURSC</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P@1000</td>
<td>0.96</td>
<td>221/236 = 0.94</td>
<td>0.71 + 0.21 = 0.93</td>
</tr>
<tr>
<td>NDCG@1000</td>
<td>0.94</td>
<td>229/248 = 0.92</td>
<td>0.72 + 0.19 = 0.91</td>
</tr>
<tr>
<td>MAP@1000</td>
<td>0.91</td>
<td>204/217 = 0.94</td>
<td>0.68 + 0.22 = 0.90</td>
</tr>
<tr>
<td>P@100</td>
<td>0.90</td>
<td>194/204 = 0.95</td>
<td>0.66 + 0.21 = 0.88</td>
</tr>
<tr>
<td>NDCG@100</td>
<td>0.90</td>
<td>196/214 = 0.92</td>
<td>0.65 + 0.23 = 0.88</td>
</tr>
<tr>
<td>MAP@100</td>
<td>0.88</td>
<td>175/190 = 0.92</td>
<td>0.61 + 0.27 = 0.88</td>
</tr>
</tbody>
</table>

(a) CW09-B

<table>
<thead>
<tr>
<th>Metric</th>
<th>r</th>
<th>Pairs recall</th>
<th>Overlap+Additional pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baseline</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P@1000</td>
<td>0.95</td>
<td>169/226 = 0.75</td>
<td>0.84 + 0.03 = 0.87</td>
</tr>
<tr>
<td>NDCG@1000</td>
<td>0.94</td>
<td>153/221 = 0.69</td>
<td>0.78 + 0.06 = 0.84</td>
</tr>
<tr>
<td>MAP@1000</td>
<td>0.94</td>
<td>126/228 = 0.55</td>
<td>0.73 + 0.03 = 0.77</td>
</tr>
<tr>
<td>P@100</td>
<td>0.84</td>
<td>68/173 = 0.39</td>
<td>0.73 + 0.03 = 0.76</td>
</tr>
<tr>
<td>NDCG@100</td>
<td>0.84</td>
<td>50/131 = 0.38</td>
<td>0.77 + 0.05 = 0.81</td>
</tr>
<tr>
<td>MAP@100</td>
<td>0.80</td>
<td>36/115 = 0.31</td>
<td>0.79 + 0.03 = 0.82</td>
</tr>
<tr>
<td><strong>AURSC</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P@1000</td>
<td>0.95</td>
<td>197/226 = 0.87</td>
<td>0.89 + 0.05 = 0.93</td>
</tr>
<tr>
<td>NDCG@1000</td>
<td>0.96</td>
<td>196/221 = 0.89</td>
<td>0.89 + 0.05 = 0.94</td>
</tr>
<tr>
<td>MAP@1000</td>
<td>0.96</td>
<td>197/228 = 0.86</td>
<td>0.88 + 0.05 = 0.93</td>
</tr>
<tr>
<td>P@100</td>
<td>0.91</td>
<td>145/173 = 0.84</td>
<td>0.77 + 0.16 = 0.93</td>
</tr>
<tr>
<td>NDCG@100</td>
<td>0.88</td>
<td>112/131 = 0.85</td>
<td>0.71 + 0.23 = 0.95</td>
</tr>
<tr>
<td>MAP@100</td>
<td>0.86</td>
<td>95/115 = 0.83</td>
<td>0.67 + 0.27 = 0.95</td>
</tr>
</tbody>
</table>

(b) Gov2

Table 25: Comparison of AURSC against a Rank-S end-to-end system at different rank depths.
differences better than end-to-end system evaluations that have high variance configurations, e.g., using a shallow evaluation depth.

8.4.3 Weaker retrieval engine

<table>
<thead>
<tr>
<th>Type</th>
<th>MAP</th>
<th>$r$</th>
<th>Pairs recall</th>
<th>Overlap+Additional pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDM</td>
<td>0.21</td>
<td>0.95</td>
<td>213/236 = 0.90</td>
<td>0.82 + 0.12 = 0.93</td>
</tr>
<tr>
<td>BOW</td>
<td>0.20</td>
<td>0.95</td>
<td>214/236 = 0.91</td>
<td>0.83 + 0.11 = 0.94</td>
</tr>
</tbody>
</table>

(a) CW09-B

<table>
<thead>
<tr>
<th>Type</th>
<th>MAP</th>
<th>$r$</th>
<th>Pairs recall</th>
<th>Overlap+Additional pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDM</td>
<td>0.32</td>
<td>0.95</td>
<td>204/226 = 0.90</td>
<td>0.89 + 0.06 = 0.94</td>
</tr>
<tr>
<td>BOW</td>
<td>0.29</td>
<td>0.92</td>
<td>185/226 = 0.82</td>
<td>0.86 + 0.04 = 0.90</td>
</tr>
</tbody>
</table>

(b) Gov2

Table 26: Comparison of AURSC scores and Rank-S end-to-end system evaluation using P@1000, when the pool of desired documents were generated from weaker rankers.

In previous sections, AUReC was calculated based on a very high-quality ranker. However, results from a strong retrieval engine may not be easy to obtain. In this experiment, we investigate the results of using a good, but less accurate retrieval engine to generate $D_q$.

We use the Indri search engine to generate $D_q$ using default search parameters. The indexes for both Gov2 and ClueWeb09 B were stopped and stemmed using the Indri stopword list and the Krovetz stemmer. Runs of two different types were generated: bag-of-word queries and sequential dependency model (SDM) queries with 0.8 weight given to the original query and 0.1 to the bigrams and 0.1 to the unordered windows. In a post-retrieval step, ClueWeb09 B results were filtered for spam, removing any documents that had a Waterloo Fusion spam score of below 50. AUReC scores were generated these weaker runs and were compared to a full end-to-end selective search evaluation, unchanged from Table 24. The accuracy of the Indri-based runs and correlation of the resulting AUReC scores are presented in Table 26. For brevity, only Rank-S end-to-end system results are shown. Other resource selection methods displayed similar trends.

Despite the fact that the Indri-based runs were significantly worse (approximately $-30\%$ change in MAP scores) than the data fusion run, the resulting AUReC scores were well-correlated with the end-to-end evaluation. However, there is some degradation of results. The bag-of-words Gov2 run especially saw a reduction in correlation scores across all methods of comparison.

The ClueWeb09 B results were also degraded, but in a more subtle way. While the final correlation scores were mostly similar, the composition of the overall agreement scores in the Indri runs shifted; the weaker Indri based runs were less able to uncover additional significant differences and instead agreed more with the end-to-end system that the differences were not significant. For example, AUReC scores based on the data fusion run and Indri SDM run both agreed with the

---

5 https://www.lemurproject.org/indri/
6 http://www.lemurproject.org/stopwords/stoplist.dft
7 http://plg.uwaterloo.ca/ gvcormac/clueweb09spam/
end-to-end system evaluation on 93% of all shard map pairs. However, 21% of pairs were *Additional pairs* in the data fusion run whereas only 12% of pairs were new significance discoveries in the Indri SDM run.

Although AUReC using the Indri-based runs produced slightly worse correlation with the end-to-end evaluations, the reduction of effectiveness was much less than the difference in accuracy of the runs. This implies that while strong retrieval results can improve AUReC, the robustness of metric indicates that a good out-of-the-box retrieval engine is sufficient to calculate reliable scores.

8.4.4 *Different queries*

The final experiment further tests the robustness of AUReC by experimenting with queries that lack relevance judgements and high accuracy runs, using an out-of-the-box search engine. The experiments use 10,000 queries from the 2009 TREC Million Query Track (MQT). Queries which were duplicates of the Terabyte Track query set or Web Track query set were removed. The queries were then expanded into SDM queries and 1000 results were retrieved per query from the Indri index using settings and post-retrieval spam filtering for ClueWeb09 B as described in Section 8.4.3.

In order to explore the effect of the number of queries on the reliability of AUReC, AUReC scores were generated using various sized subsets of the MQT queries. These runs were then compared against end-to-end systems that are, as usual, evaluated with queries that have relevance judgements. The results are presented in Table 27. For brevity, only the Rank-S end-to-end system results are shown. Other resource selection methods displayed similar trends.

AUReC generalizes well to different queries and the scores calculated are well-correlated with a full system evaluation. When the best result of Table 27 is compared to the Rank-S entry of Table 24, in ClueWeb09 B, AUReC computed with MQT queries converges to a slightly lower Pearson’s $r$ than what could be achieved if the TREC queries are used. In Gov2, the convergence value is slightly higher than in Table 24. Both values remain high and suggest that AUReC shard map rankings remain robust when queries change.

As the number of queries increase, there is a general increase in correlation between AUReC and the end-to-end system evaluation. However, there is a clear diminishing returns effect on list-wise comparisons as more queries are used to calculate AUReC scores. Pearson’s $r$ hits a saturation point quickly and moves slowly or not at all with increasing queries. The pair-wise correlation comparisons continue to grow somewhat and *Additional pairs* in particular continues to grow more with additional queries. This is expected. As additional queries are utilized, the mean of the AUReC scores for a given shard map converge quickly. Thus, the shard map’s position in an ordered list shifts rarely. However, with more evidence, the confidence interval around the mean continues to shrink and AUReC uncovers more significant differences. Note that Overlap decreases because $e$ in Figure 29 decreases (i.e. cases where both AUReC and end-to-end believe the differences between the two shard maps are not significant).

We have calculated AUReC with queries without relevance judgements or high accuracy runs, using an out-of-the-box search engine. The results were highly correlated with full end-to-end system evaluations. When this less optimal configuration is compared to the optimal cluster effectiveness baseline from Section 8.4.1 which used the same queries and relevance judgements as the end-to-end system evaluations, AUReC had better pair-wise correlation and better or near-equal list-wise correlation while using a different set of queries and no relevance judgements, demonstrating that it is robust and reliable.
Table 27: Comparison of AURSC and Rank-S end-to-end system evaluation using P@1000, when using varying number of MQT queries. The end-to-end system was evaluated with TREC queries, as usual.

8.5 conclusion and recommendations

Prior work evaluated shard maps by measuring the accuracy of end-to-end selective search systems. This is a cumbersome method that relies on relevance judgements and is sensitive to the specific system configuration. This chapter introduces AUReC, a new way to measure the effectiveness of shard maps that does not require gathering relevance judgments and is the first to completely decouple shard map evaluation from other components and parameters of a selective search system. By freeing shard map quality from other system components, AUReC provides robust diagnostic information that can be used to quickly sort through a large number of shard maps to tune a new selective search system, a process which was previously time-consuming and difficult.

AUReC evaluates shard maps by the area under a recall curve using the retrieval results of an exhaustive search system. It is highly-correlated to end-to-end selective search system evaluations while being simple to implement and not requiring: the implementation of other selective search components; picking a fixed efficiency level; or human-assessed relevance judgements.
An examination of the effectiveness and robustness of AUReC found it produces scores that are highly-correlated with the evaluation of end-to-end systems under a variety of configurations.

Given a set of shard maps, the ordering of the shard maps determined by AUReC scores closely resembled the ordering by different end-to-end evaluations, usually with Pearson’s $r > 0.9$. When pairs of shard maps were compared, most shard maps that had significant differences under an end-to-end evaluation also were significantly different when compared with AUReC scores. AUReC scores are calculated from easy-to-generate, plentiful data points and therefore produces stable results. Thus, AUReC was able to ascertain significant differences in pairs of shard maps where end-to-end system evaluations could not due to the scarcity of relevance data.

The robustness of AUReC was tested in several ways. First, the end-to-end system evaluation methodology was altered to explore how well AUReC generalized. End-to-end system evaluations were conducted with three different resource selection algorithms, and it was found that AUReC was well-correlated with the results of all three systems. AUReC was also compared to rank-sensitive metrics NDCG and MAP in addition to Precision and was found to track all three metrics closely. Correlation was affected when the evaluation depth was decreased, but the slight drop in Pearson’s $r$ was largely due to the increased variance in the end-to-end system evaluations.

In addition, $D_q$ was generated with different methods used to investigate how sensitive AUReC is to changes in the source of information. When $D_q$ was generated with a weaker search engine, the correlated degraded somewhat but not nearly to the degree of differences in the accuracy of the search engines. When $D_q$ was generated from different queries, the resulting AUReC scores remained highly correlated. More queries produced better correlated results, but there were diminishing returns.

AUReC allows system designers to quickly test a large number of shard maps to tune the accuracy of a new selective search system, a task which used to be prohibitively expensive. We end the chapter with some guidelines on using AUReC. First, to generate $D_q$, the set of documents that should be retrieved for query $q$, the strongest search engine available is preferred. However, an out-of-the-box retrieval still produces reliable results. More queries generate more consistent results with less variance. However, there are diminishing gains after about 800 queries.
CONCLUSION

This dissertation describes the selective search architecture and identifies and addresses several outstanding questions on the suitability of its use in large, practical search and text analysis pipelines. The dissertation provides assurances on: the variance in effectiveness of different selective search instances; selective search performance in combination with dynamic pruning optimization techniques; the ability of selective search to serve as a high-recall first-stage retrieval system through accurate resource selection; and selective search performance related to parallel query processing, including load balancing. It demonstrates that selective search is an effective distributed search architecture that greatly reduces the computational cost of search in realistic production environments. It also provides useful tools for systems administrators to evaluate and test selective search to encourage wider adoption. This chapter details the findings and the wider impact of this dissertation, and suggests future research directions.

9.1 SUMMARY OF FINDINGS

Selective search uses non-deterministic processes for shard creation and for sample-based resource selection algorithms. This work explores the impact of these random decisions on selective search effectiveness. Overall, the results of the system were stable at the top most ranks and still reasonable at deeper ranks. Most of the variance came from a small number of queries, where relevant documents were placed into clusters that were not representative of them. In particular, the queries with the highest variance were typically rare queries, suggesting that query type can influence the effectiveness of selective search.

Another outstanding issue explored is the possible effects of combining selective search with dynamic pruning optimization methods such as WAND, which are common in practical systems. This dissertation shows that selective search and WAND have independent cost savings. In addition, with good shard selection, selective search and WAND has better-than-additive gains due to the long, skewed posting lists generated by the clustering process. In experimenting with different ways of combining WAND and selective search, it was shown that when shards are searched in sequence and the WAND state is passed on between shards, significant computational cost savings can be realized. While a strict sequential shard search is likely undesirable for interactive use-cases, it is practical for offline tasks and it opens the possibility of a tiered shard search that can realize some of the savings while retaining competitive latency.

New light-weight methods of resource selection were presented, which utilize existing collection statistics gathered by Block-Max WAND. For single-term queries, these methods are statistically non-inferior to exhaustive search, an improvement over prior methods. This suggests that with a good way to combine individual term data, it would be possible to perform resource selection using existing statistics. Furthermore, it is possible predict a reasonably sized set of
shards which would exactly reproduce the exhaustive search results to a given depth in the rank. This creates an avenue for performing exact search with selective search, i.e. a guaranteed exact reproduction of exhaustive search, a new use-case that is currently unsupported.

This thesis improves the overall accuracy of selective search by introducing a new resource selection algorithm, learning to rank resources. This supervised method is suitable for quickly ranking hundreds of resources. The presented method does not require human judged training data and is able to produce good rankings by using exhaustive search results as a training target. Experimental results showed that learning to rank resources is able to closely replicate exhaustive search results in deep, recall-oriented metrics such as MAP@1000, which was not previously possible in selective search. Finally, in a feature ablation study it was shown that the learning to rank resources can be run with efficient features while largely maintaining the accuracy of the full set of slower features. The high accuracy in recall-oriented metrics enables the use of selective search as a first-stage retrieval in a multi-stage text analysis pipeline.

Prior work has shown that selective search is very efficient in an environment evaluating a single query at a time. In this thesis, we extend this claim to a more realistic fully parallel processing environment, where multiple queries can be processed by the system in parallel. Using a long query stream and large datasets, we evaluate selective search under load. It was found that in a wide range of hardware configurations, selective search had higher throughput than a typical distributed search setup.

We also address concerns of a potential load imbalance in selective search in a multi-machine, parallel processing environment under load. The shards produced by selective search are topically focused. Some topics are naturally more popular than others, causing certain shards to be selected at a higher frequency. This causes hotspots in the machines hosting those shards, which become more severe as the number of machines are increased. This issue was addressed using training queries to estimate shard popularity and by assigning the shards to available machines in a greedy fashion. This method outperforms 10 different random shard partitions. In addition, experiments with queries a week and a month into the future from the initial shard allocation suggest that the broad trends of topic popularity are relatively stable and shards do not need to be reassigned on a frequent basis.

Combining this load balancing method with an efficient resource selection algorithm such as Taily, selective search displays higher throughput and lower latency than typical distributed search in the parallel processing environment containing a small number of machines. We also briefly explored the use of selective search in medium scale environments and found that selective search maintains its throughput advantage.

Finally, this work presents AUREC, a metric for evaluating the effectiveness of shard maps. AUREC does not require human judged relevance assessments to calculate and can be calculated using only the exhaustive retrieval from an out-of-the-box search engine. It is independent of resource selection algorithms and does not require an end-to-end evaluation of selective search to calculate, making it efficient. This is especially useful when comparing a large number of shard maps. AUREC is a robust metric; it is highly correlated with the end-to-end accuracy of selective search using several different resource selection algorithms evaluated with multiple traditional IR metrics at different depths, even when AUREC was calculated from similar, but different queries. AUREC enables the rapid evaluation of different selective search configurations, allowing system administrators to make the best choices for their circumstances.
9.2 CONTRIBUTIONS AND WIDER IMPACT

With the tools and insights developed by this thesis, an organization can deploy a selective search system with confidence that it will have high efficiency and effectiveness. This both benefits immediate, practical users of selective search and empowers future research.

Prior work in selective search presented a proof-of-concept of a new architecture and demonstrated that it works in laboratory settings. It was initially billed as an architecture aimed primarily for small institutions such as research labs and start-ups and was meant to enable these organizations to work with large, web-scale corpora.

The research insights in this thesis established that selective search is effective in realistic production environments of larger scale that use parallel query processing, dynamic optimization and have large query loads. This thesis also demonstrated, for the first time, that selective search has high accuracy in recall-oriented metrics. This is a critical feature for modern multi-stage information retrieval systems that use slower, more complex re-rankers on top of a fast first-stage retrieval. The newly established accuracy in recall enables selective search to be used as a fast first-stage retrieval process in modern retrieval pipelines. In short, we now have confidence that selective search will increase the efficiency of real production systems.

This thesis also provided two useful new tools for prototyping a selective search system. First, it presented a detailed simulator software package that models different hardware configurations for selective search with configurable I/O and network costs, and accurately characterizes its efficiency. With a few measurements of their current system, this allows an architect to quickly prototype and evaluate the efficiency of a wide variety of selective search configurations without the prohibitive cost of building multiple real implementations.

In addition, it presented AUReC, a simple, efficient, and accurate evaluation metric for shard maps. AUReC can be calculated from the result list of an exhaustive search retrieval from an out-of-the-box search engine for a reasonable number of queries with no relevance judgements required. This can be used to quickly evaluate the effectiveness of a large number of shard maps so that the best possible way to partition the collection can be found. Together, these critical tools enable system administrators to compare and evaluate a large variety of configurations of selective search to find the settings for optimal efficiency and effectiveness without laborious implementation work.

Adoption of selective search may have significant impact on industry and academia alike. This thesis paved the path for the acceptance of selective search in production environments in industry. The key advantage of selective search in this setting is its high throughput obtained from its low consumption of total computational resources. With more powerful computing resources are becoming available to search increasingly larger data collections, electricity consumption has become a significant cost of large scale search in industry. Selective search, with its low total computational cost, is an alternative to traditional distributed search with a smaller carbon footprint.

In academia, adoption of selective search enables organizations with fewer computing resources such as university research labs to search larger collections. This thesis provided assurances for high accuracy in recall-oriented metrics. This is especially important for tasks that build on top of large scale text search such as text mining or question answering. With these assurances, researchers who use information retrieval as a first stage in a longer pipeline can use a selective search architecture. Selective search empowers research labs with limited resources and gives them tools to innovate in the big data space that may be difficult to access otherwise.

The contributions of this thesis also has wider impact on future research directions as well. AUReC, the newly proposed metric for evaluating the effectiveness of shard maps, is not limited
to selective search. It can be used as a criterion for general clustering algorithms, especially if there are a small subset of objects that are more important than others. Because calculating AUReC is simple and fast, it can be used as an efficient optimization target, empowering clustering research. Finally, confidence in selective search as an architecture gives credence to future research directions that adapt selective search to specialized domains and tasks. Researchers can extend selective search in interesting ways with the assurance that the underlying core architecture is sound.

9.3 Future Work

Selective search has been shown to be a robust architecture suitable for a practical, large-scale environments. This dissertation established that selective search is capable of being the first-stage retrieval in a more complex pipeline. For wider adoption of selective search, continued concrete investigation must be conducted into this setup.

While research in this dissertation has been conducted with web data, there are many other different domains and tasks where selective search could be of benefit, and there may be additional accuracy and effectiveness gains to be had in these areas if the selective search architecture is judiciously adapted. By supporting additional types of data beyond web documents, it increases the appeal of selective search. This section presents two possible research directions that specializes selective search to different types of corpora and purposes: extending selective search to faceted, hierarchical data; and selective search in large, online data streams.

9.3.1 Selective search in pipelines

Research in this dissertation achieved good performance on recall-oriented metrics for selective search; showed that it plays well with other dynamic optimization; established that it can be load-balanced with easy solutions; and showed that the variance of different instances are small. This opens the avenue for selective search to be used as a fast, first-stage retrieval system in complex text analysis pipelines, which would positively impact a wide range of use-cases such as modern multi-stage information retrieval systems, text mining, and question answering. This possibility warrants more substantive research and a thorough investigation. In addition, by better understanding the downstream processes using selective search and studying the interaction between them, we may be able to better tailor selective search for each use-case and increase its effectiveness.

In order to increase confidence that selective search would be truly effective in a large-scale, multi-stage pipeline, some questions must be answered. First, the interaction of selective search with other common search components must be investigated in detail. In efficiency, selective search should be studied in combination with tiering and replication, which are common strategies for improving efficiency of a large-scale system. Various configurations of these combinations are possible and discovering the best setting is an open research problem. For example, selective search could be implemented within a tier, or tiers could be created within topical shards. The differences in topic popularity of selective search can influence replication strategies; perhaps it would be easier to predict which shards will need dynamic replication with topically focused shards rather than randomly distributed shards.

There are components that affect accuracy that must be studied in combination with selective search as well, such as machine learned re-rankers. While this dissertation showed that selective
search can achieve good recall, the exact interactions between the results from selective search and downstream re-rankers are still unknown. The documents that selective search retrieve are often from a small number of topical shards. The narrower focus of topic in the candidate set may help or hinder re-rankers.

Finally, after individual component interactions have been examined, it is important to conduct integration tests, i.e. studying a whole pipeline of multiple components in combination with selective search, to ensure that secondary interactions between components and selective search are understood. A careful examination of the interaction of components may open the possibility of adjusting selective search to create further benefits in the pipeline.

While in general good recall is necessary for multi-stage text analysis pipelines, exactly how deep the result list must be is application dependent. By studying how much recall is needed, efficiency gains could be made by dynamically adjusting the level of recall of selective search. However, currently there is no well-defined way to adjust the level of recall of selective search. While resource selection parameters can be changed, the relationship between the parameters and the recall it generates is not well-understood and even small adjustments in parameters can result in large changes in recall. In addition, selective search also lacks guarantees for its level of recall; while the overall recall accuracy is good, the recall of the results may vary from query to query. Providing adjustable knobs for recall up to and including complete recall with varying levels of guarantees would be advantageous.

Because selective search is powered by creating shards, some of the filtering effort that would normally be further downstream in the text pipeline, such as temporal or reading level filtering, may be easily shifted to the initial retrieval stage by adjusting the features used to create the selective search shards. The different types of tasks may also benefit from different organization of shards, e.g. from documents clustered by popularity, time, vocabulary difficulty etc. Determining what similarity features best benefit which tasks is an open problem.

Similar adjustments can also be made to the resource selection and result merging stage of selective search to better fit the needs of a downstream process. For example, to increase diversity in the results, resource selection could be modified to select topically diverse set of shards and in the merging step preferentially show documents from many different shards rather than focusing on documents from a single shard.

9.3.2 Selective search in hierarchical corpora

Extant selective search research has dealt with web documents and assumed a single bag-of-words representation. This is a major conceptual limitation that prevents selective search from benefiting from richer, structured data commonly present in applications such as news, product search in e-commerce, medical search, and legal search. In these domains, documents are often organized in hierarchies and have well-defined attribute labels in addition to free text descriptions. For example, an iPhone is categorized under Cell phones, which is under Electronics. Selective search may then be modified to be aware of the multi-leveled nature of the hierarchy in various processes. Extending selective search to take advantage of the additional information allows us to apply the efficiency benefits of selective search to these new data sources.

Rather than using a clustering algorithm such as K-means to create topical clusters, selective search can leverage the hierarchical organization to create shards; each leaf-level node in the hierarchy becomes a shard, with shards retaining the parent-child relationship from the hierarchy. If the dataset has very fine-grained categories, interesting research questions arise on the level
of granularity to create the shards, because very small shards can cause a significant search overhead.

Resource selection in a hierarchical setting requires specialization. Rather than ranking shards, a resource selection algorithm for hierarchical selective search must generate candidate hierarchy branches to search. In addition to determining which and how many different branches to search, hierarchical resource selection also needs to consider the granularity of search. For example, if the query is broad, coarser grained categories may be searched than if a query is very specific, in which case finer grained categories may be chosen. Interesting research questions remain in what level of hierarchy should be selected for a particular query; whether the entire branch should be searched or a select few sub-branches should be searched; and the effect of these policies on the efficiency and accuracy of the system. Conversely, it may be that efficiency constraints drive these decisions as well. When merging the results from the shard search, the categorical hierarchy of the shards may inform strategies such as diversification; e.g. distance in the hierarchy could be used as a similarity metric.

In addition, selective search does not need to be limited to one particular shard map for a dataset. At the cost of some disk space, which is typically cheap, clustering on different attributes of structured data can lead to creating multiple different ways of partitioning or views of the data which may be advantageous for different types of queries. Queries can then be routed to a specific view of the corpus or multiple views may be used for a query and the results combined. For example, one view of product corpora could cluster on price in addition to product hierarchies. This price-based view of the corpus could help eliminate problems such as iPhone search results being flooded with cheap iPhone accessories. Determining how many views to create and which view to use for a given query are open problems.

9.3.3 Selective search in streaming datasets

While viewing corpora as static is convenient, there are an increasing number of important data sources that temporally-oriented and rapidly updated, e.g. Twitter. These data sources are often high volume so search efficiency in this space is much needed and would enable systems to hold and search more data, efficiently; i.e. this domain is potentially a good match for selective search. There are many challenges in adapting selective search to online data because of the characteristics of the data and because user search behavior in Twitter differ significantly from web search [Teevan et al. 2011]. Users search Twitter for temporally relevant information and to monitor changes. In other words, the data that the users are most interested in are in a stage of constant flux.

Selective search must be modified to be able to organize constant updates into shards. Streaming clustering algorithms [Ailon et al. 2009] can be used for this purpose. However, there are still interesting research problems surrounding the issue. One such problem is determining policies on how long to keep the data. Simple policies may just use a strict time cut-off (i.e. only keep data from the past n months), one can imagine using quality as a component in the decision to keep or discard a tweet. For example, tweets that have been highly ranked in many queries may be kept longer than tweets were never relevant to any queries. Sudden bursts in a topic due current events may also pose challenges. This may cause a topic shard to become size-skewed which is undesirable for efficiency reasons and may warrant splitting the topic into further shards. Conversely, there may be times when shards with less incoming documents should be merged. Finally, as the shards are being updated, the resource selection database must be updated as well; policies on
how frequently this must occur are an open question and would represent a trade-off in accuracy and efficiency.

Online corpora often have a strong temporal component that is critical for accurate search [Teevan et al. 2011] and selective search must be modified to be aware of this attribute throughout its process. During clustering, the time stamp of the data should inform the similarity metric so that items from similar times are more likely to be clustered together. During resource selection, careful attention should be paid to the temporal component of the shards as relevant documents may occur around the same time period. Finally, during result merging the final ranking of the documents should take their time stamps into account.

Finally, in online data sets, using predetermined shard assignments to machines may not a practical method of load balancing because the sizes of the shards are changing constantly as current events influence the volume of tweets in different topics. An online search system must be able to respond to rapidly changing data and queries. To achieve this, selective search must be able to load balance on-the-fly. It must be able to detect sudden increases in load on a particular topic and alleviate the load through methods such as dynamic replication of the hot shard.


Anagha Kulkarni and Jamie Callan. Topic-based index partitions for efficient and effective selective search. In *SIGIR Workshop on Large-Scale Distributed Information Retrieval*, 2010b.


