Efficient Web Search Diversification via Approximate Graph Coverage

Jared Niederhauser
Dept. of Computer Science
ETH Zürich, Switzerland
njared@ethz.ch

Carsten Eickhoff
Dept. of Computer Science
ETH Zürich, Switzerland
carsten.eickhoff@inf.ethz.ch

Aurelien Lucchi
Dept. of Computer Science
ETH Zürich, Switzerland
aurelien.lucchi@inf.ethz.ch

Thomas Hofmann
Dept. of Computer Science
ETH Zürich, Switzerland
thomas.hofmann@inf.ethz.ch

ABSTRACT

In the case of general or ambiguous queries, retrieval systems rely on result set diversification techniques in order to ensure an adequate coverage of underlying topics such that the average user will find at least one of the returned documents useful. Previous attempts at result set diversification employed computationally expensive analyses of document content and query intent. In this paper, we instead rely on the inherent structure of the Web graph. Drawing from the locally dense distribution of similar topics across the hyperlink graph, we cast the diversification problem as optimizing coverage of the Web graph. In order to reduce the computational burden, we rely on modern sketching techniques to obtain highly efficient yet accurate approximate solutions. Our experiments on a snapshot of Wikipedia as well as the ClueWeb'12 dataset show ranking performance and execution times competitive with the state of the art at dramatically reduced memory requirements.

Categories and Subject Descriptors

H.5 [**Information Retrieval**]: Retrieval models and ranking—*Information retrieval diversity*; F.5 [**Streaming, sublinear and near linear time algorithms**]: Design and analysis of algorithms—*Sketching and sampling*

General Terms

Theory, Experimentation

Keywords

Web search, Web graph, Diversification, Sketching

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ACM CIKM Big Network Analytics Workshop 2016, Indianapolis, IN, USA
Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...$15.00.

1. INTRODUCTION

Modern search engines aim to make massive amounts of information accessible to broad audiences. Search engines rely on user-specified queries to quickly produce a relevant set of results. Unfortunately, short keyword queries are imperfect proxies for the complex nature of a user's true intent; a fact that becomes increasingly apparent when looking at *broad or ambiguous queries*. In cases where user intent is unclear, search engines attempt to satisfy multiple possible intents by producing a result set covering a *diverse* group of topics. As a motivating example, consider the query "mercury" illustrated in Figure 1. On its own, "mercury" can represent one of several different topics. Without additional qualifying information, it becomes very difficult for search engines to intuit the user's true intent. Compounding this difficulty is the fact that some topics are more popular than others. Serving *all* the needs of a broad audience, replete with numerous niche interests, is not as important as presenting typical users with the most commonly sought results. For example, in Figure 1, results relevant to mercury (the element) and Mercury (the planet) may be more popular than the now defunct car manufacturer or the archaic Roman God. This example alludes to the central problem in diversity-aware search: balancing relevance and diversity. For many modern search engines, results to ambiguous queries eschew diversity in favor of maximizing expected topical relevance. For certain ambiguous queries such as "apple", hot topics such as Apple (the company) are very well represented but there are still numerous other possible interpretations that are often neglected. The Wikipedia disambiguation page for "apple" contains 46 potential matches spanning 7 distinct categories, most of which are under-represented in typical search results. In this paper, we investigate a novel approach to diversity-aware search that attempts to improve diversity by increasing coverage of the collection's hyperlink-induced Web graph. Intuitively, one would expect pages located spatially near one another to contain similar content, such that selecting multiple documents from the same "pocket of topicality" would increase redundancy and decrease diversity. We explore this intuition and produce diverse result sets by maximizing the number of nodes that can be reached within a certain number of hops on the Web graph. Since modern Web graphs can consist of tens of billions of nodes, travers-

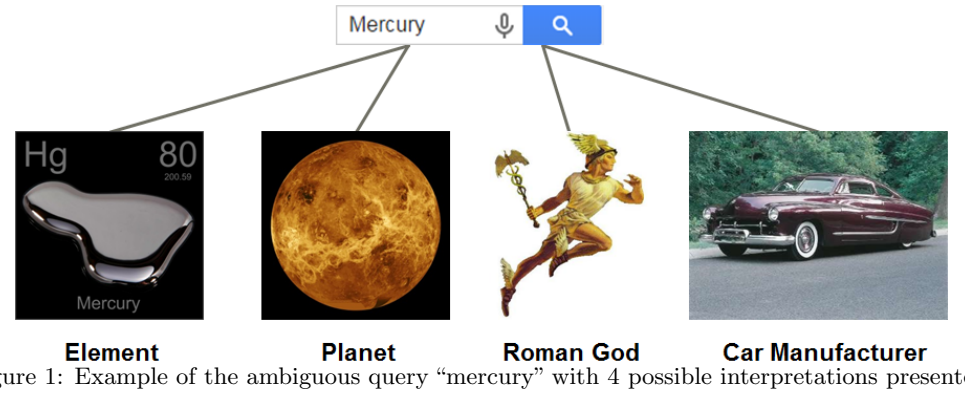


Figure 1: Example of the ambiguous query "mercury" with 4 possible interpretations presented.

ing and even storing the complete graph is often infeasible. As such, this paper makes use of approximate graph sketching techniques to efficiently measure Web graph coverage at a small and controllable loss in accuracy.

The novel contributions of this paper are threefold: 1) For the first time, instead of analyzing document content, our method diversifies result lists solely on the basis of Web graph structure. 2) In order to address scalability concerns we rely on graph sketching techniques that allow for fast and memory efficient processing. 3) Evaluating on a real-world dataset, we demonstrate competitive ranking performance and speed at greatly reduced memory footprints.

The remainder of this paper is organized as follows: Section 2 investigates previous work on diversity-aware search, Subsection 3.1 covers the necessary technical background of our approach, Subsection 3.2 formally defines the problem and discusses factors impacting performance, Section 4 discusses evaluation techniques and our experiments, and Section 5 concludes the paper with a summary of our results and a preview of future work.

2. RELATED WORK

This section gives a brief overview of existing approaches to diversity-aware retrieval models as well as metrics for evaluating diversity-aware search.

Agrawal et al. [1] attempt to "minimize the risk of dissatisfaction for the average user" by reranking a set of documents in accordance with an underlying topic distribution. The relevance of a document whose topics have already been represented in the result set is penalized by how well its topics have already been covered. Both Capannini et al. [3] and Radlinski and Dumais [15] propose approaches that utilize query logs to examine common user search behavior. These approaches rerank search results by observing a catalog of typical query "refinements" (e.g., apple → Apple corporation). Vallet and Castells later revisit this relationship between diversity and personalization, finding that both can serve a joint goal rather than taking antagonistic positions [19]. Gollapudi and Sharma begin from a carefully selected set of axioms, building towards the core definition of diversity in search results [11]. Clarke et al. [6] divide ambiguous queries into a set of sub topics, so-called "information nuggets". The intuition is that if one information nugget has already been answered by a document, all future documents that answer this particular nugget are redundant. Cronen-Townsend et al. [7] do not explicitly address diversity-aware search in their paper, rather they attempt to quantify how ambiguous a given query is. To this end, the authors develop a function for calculating a query's *clarity score* by looking at the KL-divergence between a query's language model and the corpus' language model. They reason that if the relative entropy between the query and collection language models is high then the query itself is unambiguous. There have been several parallel efforts into substituting the previously used topical scales by a search intent framework and subsequently trying to maximise coverage across individual intent classes [4, 17]. Yin et al. [21] introduce the use of survival analysis theory for the purpose of search result diversification by modelling result list versatility and topic coverage. Dang and Croft [8] propose to balance the mixture of individual facets on the result page according to their popularity, motivating a proportionality based approach. Rather than relying on a single given scale (topics, nuggets, intents, etc.) along which to diversify, Dou et al. [9] present a multi-dimensional diversification scheme, demanding diversity in a potentially high-dimensional space.

Traditional information retrieval metrics are often ill-suited when evaluating the diversity of a set of documents. These common metrics (e.g., NDCG or MAP) eschew the importance of diversity and rely solely on relevance judgments to evaluate a result set. As previously motivated in Section 1, algorithms can score highly on these evaluation metrics by producing a highly-relevant yet non-diverse result set. Agrawal et al. [1] introduce a number of intent-aware (IA) extensions to traditional search metrics (IA-NDCG, IA-MAP, and IA-MRR). These evaluation metrics incorporate a topic distribution model in their assessments such that a result set can be evaluated on the individual document's relevance to the query terms and the collective set's coverage of topics relevant to the query. Clarke et al. propose a modification to the standard NDCG evaluation metric which they label α-NDCG [6]. α-NDCG looks at the cumulative gain vector of a collection of documents and weights them based on how well those documents answer a series of information nuggets relevant to the query. From 2009 to 2014, the α-NDCG metric as well as the IA-metrics have been used as standard evaluation metrics within the TREC Diversity Track [5]. Rafiei et al. [16] introduce another evaluation technique which utilizes Wikipedia disambiguation pages to implicitly link ambiguous terms with a set of relevant pages. Using this method, the authors quantify diversity as the number of these relevant pages which are included in the result set.

In contrast to existing methods, the proposed approach re-

quires no initial preprocessing (e.g., topic model generation), query logs, or ancillary metadata to run. Instead, we rely exclusively on a hyperlink Web graph, implicitly given by the document collection, in order to perform diversification. Our work is similar in spirit to Li et al. [13], who demonstrate general graph diversification using Flajolet-Martin sketches. In this paper, we describe a similar setup, relying on HyperLogLog counters, targeting a specific application to Web search scenarios. To the best of our knowledge, there has been no previous attempts to diversify search results by using Web graph coverage information.

3. METHODOLOGY

3.1 Preliminaries

In the following we briefly explain various notation and background material used throughout the rest of the paper.

3.1.1 Ad-hoc information retrieval

Ad-hoc information retrieval (IR) takes as input a user query $Q = \{q_1, \ldots, q_m\}$ comprised of one or more terms and returns an ordered result set of documents $S^\star = \{s_1^\star, \ldots, s_n^\star\}$ that have been taken from a larger corpus of documents $S^\star \subseteq \mathcal{D} = \{s_1, \ldots, s_N\}$. A document's value, with respect to the query Q, is given by an objective function $V(s|Q)$, and the documents that compose the result set are ordered such that:

$$\forall s_i^\star, s_j^\star \in S^\star, V(s_i^\star|Q) \geq V(s_j^\star|Q) \iff i \leq j$$

While there are many different ways to create an objective function, traditional ad-hoc IR applications assume a document independence model wherein a document's score is not impacted by the inclusion or exclusion of another document in the result set. This, however, is not the case with diversity-aware IR, in which one document's inclusion in the result set can impact the score of all other documents. Put more generally, a document's diversity-aware value is given by some objective function $V(s|Q, S)$. In Subsection 3.2, we explore in detail the specific diversity-aware objective function used in this paper.

3.1.2 HyperLogLog counters

Due to their size, exact storage and manipulation of modern Web graphs is often intractable. To combat this challenge, various graph sketching techniques exist that store and perform approximate calculations on large graphs by sacrificing accuracy in favor of space efficiency. *HyperLogLog counters* are one such sketching technique used throughout this paper, and in this section we preview the HyperLogLog algorithm and subsequently explain what it accomplishes as well as why it is useful in the context of diversity-aware search. Before presenting the algorithm, we first explain some of the notational constructs used in the calculation:

- $\boldsymbol{h : V \rightarrow 2^\infty}$ is a fixed hash function mapping each graph node $v \in V$ into an infinite binary sequence
- $\boldsymbol{h_t(x)}$ denotes, for a given $x \in 2^\infty$, the sequence made by the leftmost t bits of $h(x)$, and $\boldsymbol{h^t(x)}$ is the sequence of remaining bits of x
- $\boldsymbol{h_t}$ is identified with its corresponding hash function integer value in the range $\{0, 1, \ldots, 2^t - 1\}$

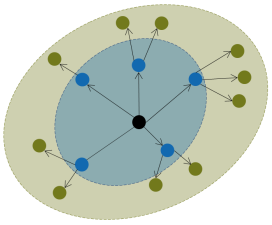


Figure 2: An example of two hyperballs surrounding a graph node (in black). The hyperball in blue represents a radius of 1 whereas the green hyperball represents a radius of 2.

- $\boldsymbol{\rho^+(w)}$ is, given a binary sequence w, the number of leading zeroes in w plus one (e.g., $\rho^+(01001) = 2$)
- $\boldsymbol{p = 2^b}$ is the number of registers that compose the HyperLogLog counter.
- Unless otherwise specified, all logarithms are base 2

A HyperLogLog counter, M, is a probabilistic data structure composed of an array of $p = 2^b$ registers that approximately represents a set. While HyperLogLog counters do not allow us to answer the question "Is this object contained in the set?" they do allow us to ask "Approximately how many distinct objects are contained in the set?" via the `size`(M) function. In the context of Web graphs, HyperLogLog counters can be used to approximately represent the set of neighbors for each node in the graph and the `size`() function allows us to approximate that set's cardinality. Formally, for any digraph $G(V, E)$ we can approximate the value $|\{u \in V | v \rightarrow u\}|$ for each $v \in V$ where $v \rightarrow u$ is the notation we use for the directed edge $(v, u) \in E$. As previously stated, the HyperLogLog algorithm is particularly useful for our purposes because it allows accurate set cardinality prediction with only scant memory usage. In [10], the authors explain how HyperLogLog can estimate cardinalities well beyond 10^9 with a standard error of 2% using only 1.5 kilobytes of memory. For further mathematical and technical details, we refer the reader to [10].

3.1.3 HyperBall algorithm

In our proposed approach we want to "cover" as much of the Web graph as possible. We formalize the notion of coverage by introducing the *hyperball*.

A hyperball, as defined by Boldi and Vigna [2], is the ball of radius r around a given node

$$\mathcal{B}_G(x, r) = \{y | d(x, y) \leq r\}$$

Here the function $d(x, y)$ is merely the shortest distance between two nodes x and y following the directed edges E of the Web graph $G = (V, E)$. Figure 2 presents an example of two hyperballs with varying radii. The basic idea used by the HyperBall algorithm[1] is that the hyperball of radius r satisfies the following properties

$$\mathcal{B}_G(x, 0) = \{x\} \tag{3.1}$$

[1]The HyperBall implementation used throughout this paper is part of the WebGraph framework found at http://webgraph.di.unimi.it/

$$\mathcal{B}_G(x, r+1) = \bigcup_{x \to y} \mathcal{B}_G(y, r) \cup \{x\} \tag{3.2}$$

Using the preceding properties in conjunction with HyperLogLog counters described in Subsection 3.1.2, the HyperBall algorithm creates a probabilistic hyperball with radius r for each node in the Web graph by means of iterative expansion. The `union` appearing in Eq. 3.2 is performed by register-by-register maximization of the corresponding counters. Note that in HyperBall and throughout the rest of the paper we adopt the notation M_v to represent the HyperLogLog counter assigned to node v.

3.2 Diversity-aware Web Graph Search

Recall the goal of ad-hoc IR: given a sequence of query terms we want to provide the user with a result set of documents ranked in order of decreasing *utility*. Utility in this context depends highly on the clarity of the initial query. In the case of unambiguous queries, equivocating a document's value with its relevance to the query is often sufficient and can produce good results. However, when queries are ambiguous, and the user's intent is unclear, a document's value can be interpreted not only in terms of how relevant it is to the query, but also how different it is from other documents in the result set. This notion leads us to the central problem of *diversity-aware search*, namely constructing a result set whose document values reflect a tradeoff between relevance and diversity.

3.2.1 Relevance scores

In many IR applications, topical relevance quantifies how suitable a given document is for a query. This relevance score can be achieved in a number of different ways and is a well-researched area of IR. For the purposes of this project, we rely on the popular open-source IR platform, Apache Lucene, to estimate topical relevance. Although document relevance is not the major focus of this paper, it is worth noting the differences between calculating relevance and diversity scores. Common relevance models assume that document relevance is independent from other documents added to the result set. For example, the default scoring function used by Lucene utilizes a tf-idf approach to calculate relevance. More specifically, its scoring function is given by:

$$R(s|Q) = \frac{1}{Z_Q} \cdot \sum_{w \in Q} \frac{\text{tf}(w, d) \cdot \text{idf}(w)}{Z_s},$$

where $\text{tf}(w, d)$ is the frequency of the word w in document d and $\text{idf}(w) = \log \frac{|\mathcal{D}|}{|\{d \in \mathcal{D} : w \in d\}|}$ is the inverse document frequency, i.e., the number of documents in the entire corpus containing w. For the sake of brevity, we omit several technical details such as the role of the normalization constants Z_s and Z_Q; however, a comprehensive description of the Apache Lucene scoring function is readily available on their website[2]. A document's final score is given by a weighted mixture of relevance and diversity. This process is explained in detail in Subsection 3.2.3

3.2.2 Diversity: Increasing Web graph coverage

Recall our initial assumption for increasing diversity, a set of documents covering a greater portion of the Web graph intuitively suggests that those documents are more widespread throughout the graph and thereby topically more diverse. We now seek to achieve this goal by counting *how many distinct elements a union of hyperballs contains*. The example in Figure 3 illustrates the diversity objective wherein we want to add a new document, a or b, to a pre-existing result set, v. While document b has the greater overall connectivity with 12 documents reachable within 2 hops (instead of 9 for document a), document a introduces a greater number of new distinct documents (6 instead of 4) that were previously not reachable from v. This section explains how we use the HyperLogLog and HyperBall algorithms to construct a document diversity function that achieves our goal of efficiently quantifying Web graph coverage. From Subsection 3.1.3, the HyperBall algorithm outputs a probabilistic representation of a hyperball of radius r around each node, $\hat{\mathcal{B}}_G(v, r)$, and these hyperballs are stored as an array of registers (collectively referred to as a counter), M_v. Consider two nodes in the graph, v & u, and their corresponding hyperballs, $\hat{\mathcal{B}}_G(v, r)$ & $\hat{\mathcal{B}}_G(u, r)$ respectively. We represent the set of nodes covered by these two hyperballs by taking their union. More specifically, the union of two arbitrary node hyperballs is given by

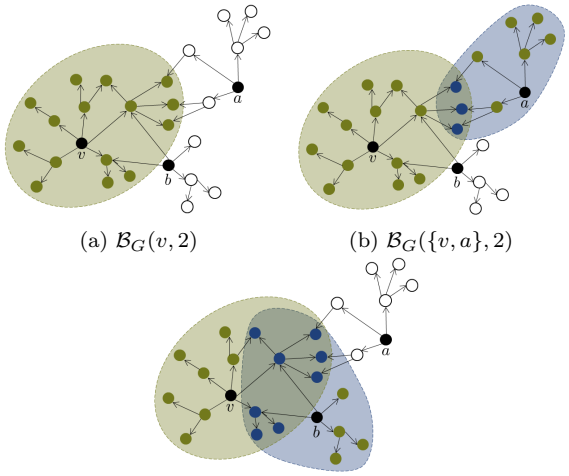
$$\hat{\mathcal{B}}_G(v, r) \cup \hat{\mathcal{B}}_G(u, r) = \texttt{union}(M_v, M_u)$$

Given the preceding formula, we can easily estimate the number of distinct elements contained in the union of the hyperballs by simply using the `size` function presented in Subsection 3.1.2

$$|\hat{\mathcal{B}}_G(v, r) \cup \hat{\mathcal{B}}_G(u, r)| = \texttt{size}\left(\texttt{union}(M_v, M_u)\right)$$

By extension, the number of distinct elements in the union of a set of hyperballs can be approximated via

$$|\hat{\mathcal{B}}_G(S, r)| = \left| \bigcup_{s \in S} \hat{\mathcal{B}}_G(s, r) \right| = \texttt{size}\left(\texttt{union}(M_S)\right)$$



(a) $\mathcal{B}_G(v, 2)$ (b) $\mathcal{B}_G(\{v, a\}, 2)$

(c) $\mathcal{B}_G(\{v, b\}, 2)$

Figure 3: Example of two competing hyperball unions in 3b & 3c where white nodes are uncovered, green nodes are covered and non-overlapping, and blue nodes are covered but overlapping. Despite the fact that node b's hyperball contains more elements than node a's (12 v. 9), it has fewer distinct nodes from v (4 v. 6).

[2] http://bit.ly/1bLnktw

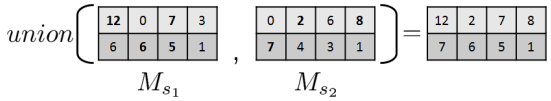


Figure 4: The register-by-register maximization of two HyperLogLog counters. In this example, each document is represented by 8 registers, however, in practice, documents have 16 to 1024 registers each.

It is important to note that $\texttt{union}(M_S)$ returns not just a single number, but rather an array of registers whose values correspond to the register-by-register maximization of all HyperLogLog counters in the set S. Figure 4 presents an example of how this operation works on a set of two documents, $S = \{s_1, s_2\}$.

Finally, the equation we use to calculate an individual document's diversity when added to an existing set of documents, S, is

$$D(s|S; M) = |\hat{\mathcal{B}}_G(S \cup s, r)| = \texttt{size}\,(\texttt{union}(M_S, M_s))$$

3.2.3 HyperBall-Diversify *algorithm*

Supposing that users are only interested in the top k results of a search, the objective becomes:

> WebGraph-Search(k): Given a set of documents $S \subseteq D$, a set of query terms Q, relevance scores of the documents $R(s)$, the r-radius hyperball counters of the documents M, and a mixture parameter $\lambda \in [0, 1]$. Find a set of documents $S^\star \subseteq S$ where $|S^\star| \leq k$ that maximizes:
>
> $$V(S^\star|Q) = \sum_{s \in S^\star} \lambda \cdot R(s|Q) + (1 - \lambda) \cdot D(s|S^\star \setminus s; M)$$

This objective function formalizes the previously stated problem of diversity-aware search. We want to find a result set whose overall value is merely the cumulative value of its composite documents. As one may expect, optimizing the preceding objective function is NP-hard. Fortunately, WebGraph-Search(k) falls within the category of *submodular functions*[3] for which theoretical guarantees derived in [14] ensure that a greedy maximization yields a bounded approximation of the order of $(1 - 1/e)$ to the optimal solution. Consequently, we propose a greedy algorithm to the problem titled HyperBall-Diversify whose pseudocode is given in Algorithm 1.

It is important to note that document diversity is being normalized by dividing by the maximum possible diversity value (the number of distinct elements covered by the union of every document in the result set, D_{max}). This additional step is taken to avoid scaling issues when mixing document relevance and diversity scores due to their large differences in magnitude. For example, relevance scores calculated from Apache Lucene are normalized such that $R(s) \in [0, 1]$, however, the raw document diversity function can estimate cardinalities greater than 10^9, many orders of magnitude larger than the document's relevance score. By means of normalization, we ensure that both relevance and diversity lie within the range $[0, 1]$ which greatly improves our ability to analyze

[3]Intuitively, this follows the principle of diminishing returns wherein the benefit of adding a document to a larger set is less than that of adding it to a smaller set.

Algorithm 1 HyperBall-Diversify

Input: set of documents S, set of HyperBall counters M, Q, k, λ
Output: reranked set of documents $S^\star$

$S^\star \leftarrow \emptyset$
$D_{max} \leftarrow \text{size}(\text{union}(S))$
while $|S^\star| < k$ **do**
  for $s \in S$ **do**
    $v(s) \leftarrow \lambda \cdot R(s) + (1 - \lambda) \cdot \frac{D(s|S^\star; M)}{D_{max}}$
  end for
  $s' \leftarrow \arg\max v(s)$
  $S^\star \leftarrow S^\star \cup s'$
  $S \leftarrow S \setminus s'$
end while

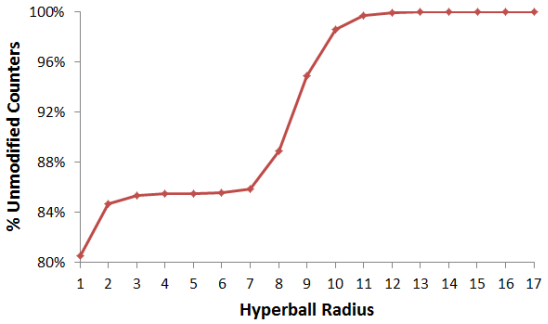


Figure 5: The number of unmodified HyperLogLog counters after each iteration of the HyperBall algorithm. The maximum hyperball size, $r = 17$, occurs when all counters are in a stable state.

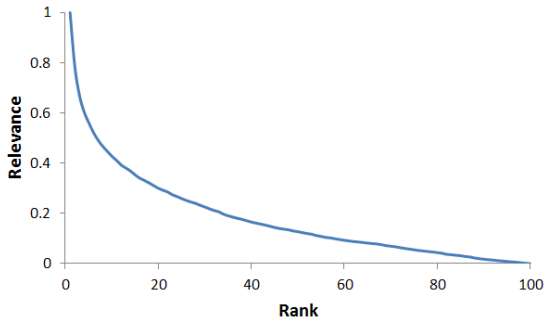
results and choose a suitable λ.

3.3 Parameter selection

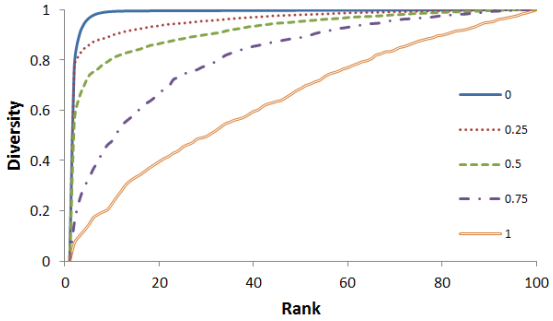
The two most important parameters that dictate diversification performance are λ, which governs the tradeoff between relevance and diversity, and r, the hyperball radius. While we tune these parameters via empirical analysis in Section 4, this section briefly explores *why* these parameters are important and *how* altering them impacts the objective function $V(S|Q)$. Additionally, we discuss how the number of registers per HyperLogLog counter, b, impacts the accuracy of the graph sketching approach as well as the overall memory usage and execution time of the HyperBall-Diversify algorithm.

3.3.1 *Hyperball radius:* r

Recall that HyperBall-Diversify takes, as input, an array of HyperLogLog counters, M, one for each node in the Web graph. These HyperLogLog counters represent hyperballs with radius r as generated by Algorithm 1. The importance of this parameter becomes apparent when looking at two opposing extremes. Setting the radius too large implies that each node's counter, M_i, becomes nearly identical and the corresponding hyperballs cover almost the entire Web graph. Conversely, setting r too small has the opposite effect meaning hyperballs cover too small a portion of the Web graph to be meaningful. To help analyze the behavior of r, we look at how the array of HyperLogLog counters changes when we increase the radius. To this end, we run Algorithm 1 on the Wikipedia Web graph from 3rd February



(a) Relevance decreases rapidly with rank



(b) Diversity increases rapidly with rank for $\lambda < 0.5$

Figure 6: Relevance v. diversity change with document rank.

2014 and observe what percentage of the ∼4.5 million counters were unmodified after each iteration of the algorithm; the results are presented in Figure 5. The two plateaus in the chart suggest that hyperball expansion occurs in two stages. We interpret the first plateau as a period of semi-stable graph coverage wherein the majority of hyperballs are sufficiently large such that they cover all nodes in their immediate vicinity. The second plateau occurs when further hyperball expansion yields no changes which, in the case of strongly connected graphs this means that all counters are identical and cover the graph in its entirety. We previously analogized regions of the Web graph as "pockets of topicality" and we interpret this first plateau as the stage when the hyperballs have covered these pockets. We thus intuit that the optimal hyperball size is when this first plateau occurs, i.e. $r = \{4, 5, 6\}$. Indeed, through empirical testing (explained in more detail in Section 4) we confirm the assumption that hyperballs with $r = 5 \pm 1$ perform best, depending on the evaluation metric being used.

The results displayed in Figure 5 seem to follow our initial intuition that Web graphs are composed of several heavily interconnected cliques with small diameters and similar content.

3.3.2 *Relevance and diversity tradeoff: λ*

This section focuses on the parameter λ which explicitly governs the tradeoff between relevance and diversity, $R(s|Q)$ and $D(s|S; M)$ respectively, as documents are added to the result set. We observe this tradeoff by looking at the average relevance and diversity score per document rank for 364 different ambiguous[4] queries and with a fixed hyperball radius, $r = 4$. Such analysis should illuminate the impact of λ and how these separate components work together to create a diversified result set. We start by investigating how document relevance changes with rank position. Recall that a document's relevance score is precomputed by Apache Lucene and thus HyperBall-Diversify takes, as input, an already ordered original result set S. Figure 6a visualizes how relevance scores in the original result set decrease rapidly with respect to the document's rank position. Contrast the behavior of document relevance with that of its diversity shown in Figure 6b. In this particular chart, we are interested in the percentage of the Web graph covered (diversity) with each addition to the result set. There are five lines corresponding to different lambda values, $\lambda = \{0, 0.25, 0.5, 0.75, 1\}$. The two extreme values, $\lambda = \{0, 1\}$, correspond to greedily selecting documents with the highest diversity and relevance respectively. In the case where *only* relevance is considered, we observe the unsurprising behavior that Web graph coverage increases slowly and almost uniformly over time as documents are added to the result set. However, we notice that for any $\lambda \leq 0.5$, the first ten documents added to the result set cover $> 80\%$ of the Web graph. From the aforementioned charts, it is apparent that relevance and diversity follow inverse behavior with respect to document rank. Furthermore, we notice that, for the majority of λ values, the Web graph is covered *very* quickly. This quick graph coverage means 1) the objective function will initially favor diversity over relevance and 2) marginal gains in diversity decrease very quickly such that the relative difference in diversity between documents not yet added to the result set becomes negligible. Neither of these outcomes are very favorable as 1) we still desire highly relevant documents early in the result rather than documents that purely increase diversity and 2) when marginal gains in diversity become too small (i.e. the Web graph is almost entirely covered), diversity scores between documents are very similar and diversity in general becomes irrelevant. To avoid these effects, the lambda value should be set sufficiently high such that there is a balance between relevance and diversity as documents are added to the result set. We verify this through parameter tuning in Section 4 where $\lambda \approx 0.65$ tends to produce the best results.

3.3.3 *HyperLogLog registers: b*

Recall the HyperLogLog algorithm presented in Subsection 3.1.2 wherein we define each HyperLogLog counter as an array of $p = 2^b$ registers. While this parameter b is not explicitly listed in HyperBall-Diversify, it is implicitly contained in the diversity function $D(s|S; M)$. More specifically, it is used in the creation of the HyperLogLog counters M and ultimately impacts the fidelity of our Web graph coverage approximation with respect to the true Web graph coverage. Altering b affects our approximation scheme in three ways: 1) Web graph coverage accuracy, 2) memory usage, and 3) execution speed.

To quantify Web graph coverage accuracy we look at the relative error in Web graph coverage between approximate hyperballs generated via the HyperBall algorithm, $\hat{\mathcal{B}}_G$, and exact hyperballs, $\mathcal{B}_G$, for a given graph, G. We calculate this relative error by iteratively choosing a random node from G and measuring the relative error in Web graph coverage after each successive iteration. More specifically, we define the relative error of the i-th iteration with the

[4]Explanation of how these queries were chosen is given in Section 4

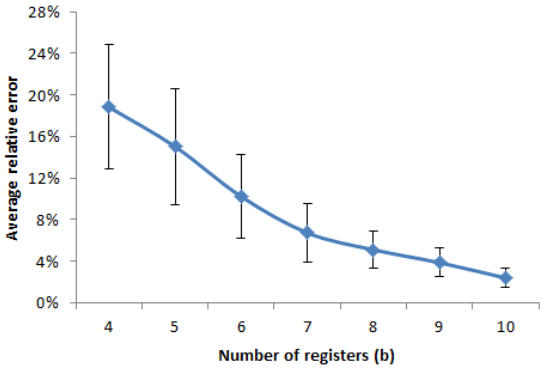


Figure 7: Increasing the number of HyperLogLog registers, b, results in a increased estimate accuracy.

following notation:

$$E_G^{(i)} = \frac{|\hat{\mathcal{B}}_G(S^{(i)}, r)| - |\mathcal{B}_G(S^{(i)}, r)|}{|\mathcal{B}_G(S^{(i)}, r)|}$$

Where the set $S^{(i)} = \{s_1, \ldots, s_i\}$ contains the first i nodes selected from G, the value $|\mathcal{B}_G(\cdot)|$ is the exact number of nodes covered by the union of selected node hyperballs, and the value $|\hat{\mathcal{B}}_G(\cdot)|$ is the approximate number of nodes covered as defined by the diversity function $D(\cdot)$ presented in Subsection 3.2.2. Note, for this experiment and all subsequent experiments we use a fixed hyperball radius, $r = 4$. Nodes are continually selected until the union of hyperballs entirely covers the graph and the total relative error for graph G is calculated via:

$$E_G = \frac{1}{n} \sum_{i=0}^{n} \left| E_G^{(i)} \right|$$

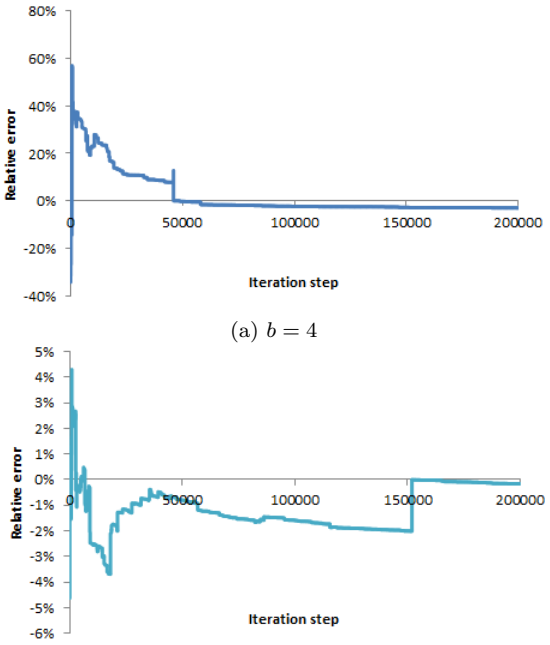
We perform this experiment and calculate the average relative error on a collection of 50 subgraphs generated by randomly sampling the 3rd February 2014 Wikipedia dump [20]. Each graph in the set of subgraphs, $G_{\mathcal{S}} = \{G_1, \ldots, G_{50}\}$, contains 1.5 to 2 million nodes and 3 to 3.8 million edges as opposed to the complete Wikipedia dataset which contains ∼4.5 million nodes and ∼40 million edges. We finally calculate the average relative error across all subgraphs using the following formula:

$$\bar{E} = \frac{1}{|G_{\mathcal{S}}|} \sum_{g \in G_{\mathcal{S}}} E_g$$

We repeat this same experiment for varying values of b. The results of these experiments are shown in Figures 7 and 8.

As expected, both the average relative error and the standard deviation decrease as the number of registers per HyperLogLog counter increases. This empirically mirrors the results of Flajolet et al. [10] who state that the relative standard deviation (i.e., the ratio between the standard deviation of the estimated cardinality and the true cardinality) for a given HyperLogLog counter is at most $\frac{\beta_p}{\sqrt{p}} \leq \frac{1.06}{\sqrt{p}}$ where $p = 2^b$ and β_p is a suitable constant. In other words, a larger value of b produces a better Web graph approximation. Indeed, Figure 7 suggests that for $b = 10$ we can expect an average relative error of $2.38\% \pm 0.93\%$.

As one may expect, increasing the number of registers per HyperLogLog counter results in an increased memory footprint. To illustrate this impact on memory, we run the



(a) $b = 4$

(b) $b = 10$

Figure 8: The relative iteration error, $E_G^{(i)}(\cdot)$, accrued during the first 200,000 iterations for $b = 4$ and $b = 10$. Although both charts reach a somewhat asymptotic state, the relative error for $b = 4$ lies in a *much* larger range than the error for $b = 10$

HyperBall algorithm on the subgraph G_1 with varying values of b and measure how much memory is required to store the resultant set of probabilistic Web graph hyperballs. As previously stated, the total number of registers for a HyperLogLog counter p is equal to 2^b and thus increasing the value of b results in an exponential increase in the memory required when constructing Web graph hyperballs. The results of this experiment are presented in Figure 9 and confirm an exponential increase. Additionally, Figure 9 presents the memory required to store the exact representation of G_1. Exact memory use falls between $b = 4$ and $b = 5$, and for all $b \geq 6$ the approximate solution's memory requirements greatly outstrip those of the exact solution. It is worth noting that for the subgraph G_1, the average degree of a node is ∼2, meaning G_1 is much sparser than the complete Wikipedia Web graph it was derived from which has an average node degree of ∼9. This distinction is important as the exact solution's memory use is determined by the number of nodes and edges in the Web graph whereas the approximate solution's memory use is dictated by the number of nodes and the parameter b. Thus Web graph density plays a large role in the amount of memory required by the exact solution but not the memory use of the approximate solution.

Finally, we are interested in how b impacts HyperBall-Diversify's execution speed. To observe this, we use the same experimental setup as when calculating average relative error. Namely, we take the collection of 50 subgraphs, $G_{\mathcal{S}}$, and calculate the average time required to completely cover each subgraph for differing values of b. The results of this experiment are shown in Figure 10 where we can see an exponential increase in the time required to cover a graph as the parameter value b increases. This is again unsurpris-

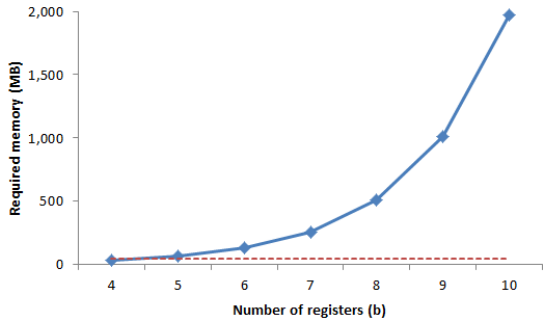


Figure 9: Increasing the number of registers per counter results in an exponential increase in memory. The memory usage of the exact solution is given by the dashed horizontal line. Note, the chosen subgraph being stored, G_1, consists of 1,648,614 nodes and 3,310,091 edges.

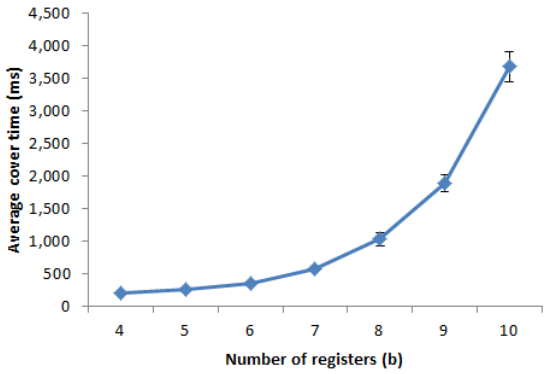


Figure 10: Increasing the number of registers per counter results in an exponential increase in execution time.

ing because, at every iteration step, the cardinality estimation function performs a register-by-register maximization, thus doubling the number of registers should double the execution time. Finally, we are interested in comparing the execution time required to generate a probabilistic approximation with the execution times of calculating the exact solution. Because calculating the exact solution does not depend on the number of HyperLogLog registers b, its execution time remains fairly constant with an average value of ∼98,000 ms. From Figure 10 we note that the slowest probabilistic approach tested, $b = 10$, completed in ∼3,670 ms, approximately 2,700% faster than the exact solution. It is worth noting that the exact solution used throughout these experiments is an unoptimized, naive solution and more sophisticated algorithms and data structures could undoubtedly improve its execution speed. However, it is unsurprising that the approximate solution is significantly faster as the exact solution relies on traversing large Web graphs, an operation that is *very* time consuming and, for the purposes of ad-hoc IR where generating a result set quickly is important, exact solutions seem impractical. Finally, we note a difference between the exact and approximate approaches with regard to execution time. For the approximate HyperLogLog approach, speed is dictated primarily by the number of registers per counter, b, and is unaffected by the choice of hyperball radius, r. For a fixed radius and Web graph, the HyperBall algorithm is performed once at which point the resulting array of HyperLogLog counters, M, does not change. Consequently, for a fixed value of b, we expect HyperBall-Diversify to have similar execution times regardless the choice of hyperball radius, r. However, this is not the case for the exact approach wherein the choice of r greatly impacts the amount of time spent traversing the Web graph and covering nodes. As mentioned previously, we fix $r = 4$ for all experiments performed in this section, however for use cases when an even larger hyperball radius is required, the disparity in execution time between the exact and approximate solutions becomes significantly more pronounced. In summary, we demonstrate through a series of experiments how the number of HyperLogLog registers, b, impacts speed, memory usage, and Web graph coverage fidelity. For the duration of this paper, we assume the value $b = 10$ as we find it provides favorable results while keeping both speed and memory usage reasonably small. Further details on speed and memory use will be explained in Section 4.

4. EXPERIMENTS

4.1 Datasets

We use two sizeable datasets as the basis of our empirical evaluation: Wikipedia [20] as well as the ClueWeb'12 collection. In the case of Wikipedia, we filter out all non-article pages (*e.g.*, discussion pages, category pages, etc.) resulting in a corpus of ∼4 million articles and ∼40 million edges. We further gather a collection of query/relevance set pairs, (Q, R), from the set of disambiguation pages containing at least 10 links to different articles. This approach follows Rafiei et al. [16] who assume query terms to be given by a disambiguation page's title and the relevance sets to be derived from the set of linked articles.

For ClueWeb'12, no additional preprocessing was required and we used the TREC Web Track topics and relevance judgements from the years 2013 and 2014.

4.2 Evaluation metrics

We evaluate HyperBall-Diversify using a collection of standard diversity-aware evaluation measures: α-NDCG, NDCG-IA, & MAP-IA. α-NDCG relies on a set of *information nuggets*, $n_i \subseteq N$ relevant to a query. While the notion of nuggets is a fairly broad concept, we define information nuggets using Wikipedia categories. Each Wikipedia page belongs to a set of manually-annotated categories. Thus, for an ambiguous query, Q, we define its set of relevant information nuggets, N, by the set of categories specified in the relevance set, R. The intent-aware measures, NDCG-IA and MAP-IA rely on a topic model to compute topic distributions for queries as well as documents [1]. For these metrics we generate an LDA topic model with 20 topics using the most current version of the DMOZ dump[5] as a training set. Each experiment is run on an identical 16-core, 128 GB machine.

4.3 Baseline methods

Throughout our experiments, we compare the performance of HyperBall-Diversify with two alternative approaches as well as the original non-diversified ranking produced by Apache Lucene's default search algorithm.

[5] http://rdf.dmoz.org/

4.3.1 Topic model diversification

The first baseline method we compare against, proposed by Agrawal et al. [1], relies on a topic model to construct a result set that covers many different topics in the topic model. Formally put, this approach attempts to find a set of documents, S, that maximizes the following objective function:

$$P(S|Q) = \sum_c P(c|Q) \left(1 - \prod_{s \in S} (1 - V(s|Q, c))\right)$$

The function $P(c|Q)$ is the probability distribution of categories for the given query and the function $V(s|Q, c)$ is the retrieval status value of a document given the query and category (i.e., a document's relevance to the query weighted by categorical importance). The topic model used to predict topic distributions for this baseline is the same LDA topic model described in Subsection 4.2.

4.3.2 Language model diversification

The second baseline method, proposed by Cronen-Townsend et al. [7], relies on the construction of language models to quantify how ambiguous the query is given a set of documents. It is formally given by the following formula:

$$\text{clarity score} = \sum_{w \in V} P(w|Q) \log_2 \frac{P(w|Q)}{P(w|S)}$$

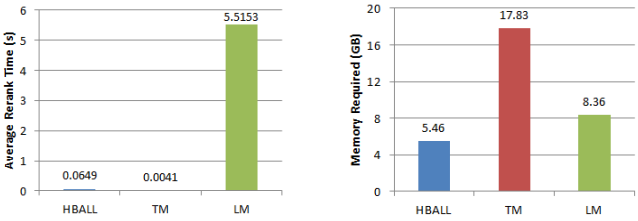
Cronen-Townsend argues that queries with a higher clarity score are less ambiguous (i.e., higher clarity scores are better). While Cronen-Townsend propose the preceding formula as a means of identifying vague queries, we instead implement a greedy algorithm that constructs a result set via a tradeoff between document relevance and maximizing clarity. Language models in this baseline are simple unigram models created in accordance with the methods outlined in [18, 12, 7].

4.4 Results

Table 1 presents the results of each experiment. The rows correspond to the four methods being evaluated: HyperBall-Diversify (HBALL), the two baseline methods topic model (TM) and language model (LM), and the original ranking obtained from Lucene (ORIG). Statistically significant improvements over all competing methods are determined by means of a two-tailed t-test at $\alpha < 0.05$ level and are denoted by an asterisk.

For Wikipedia, the best diversification method changes from metric to metric, but there are a few notable takeaways. First, every diversification method outperformed the original ranking, confirming that each method is indeed improving result set diversity. Secondly, there is no best-performing algorithm across all evaluation metrics. Indeed, each diversification scheme is the best performing in one of the three metrics although only the HyperBall and language model approaches outperform their competitors at significance level.

In the case of the ClueWeb corpus, the picture is much clearer. The HyperBall approach significantly outperforms all baseline methods across the three metrics by a solid margin. We assume that the considerably greater size and less uniform connectivity of the ClueWeb graph help graph-coverage methods in unfolding their full potential. This is encouraging especially with respect to applicability for diversification on the Web.



(a) Average Rerank Time (b) Memory Usage

Figure 11: Reranking speed and required memory for each of the three diversification methods.

4.5 Speed and memory usage

Throughout the experiments we measure the maximum memory requirements and average re-ranking speed of the three diversification schemes. Figure 11 presents these measures when performing diversification on the entire 4 million-node Wikipedia Web graph (The same tendencies are observed on The ClueWeb'12 collection). We note that HyperBall-Diversify requires the least amount of memory (5.46 GB) whereas the topic model and language model approaches require approximately 227% and 53% more memory respectively as storing a node's HyperLogLog counter is more efficient than storing its topic distribution or language model. With regards to reranking time, the topic model approach is slightly faster than HyperBall-Diversify while the otherwise well-performing language models are prohibitively slow. When taken as a whole, we see that HyperBall-Diversify provides the best scalability in terms of memory whereas the topic model approach has the fastest re-ranking performance. While there is no clear victor in terms of both speed and memory usage, we conclude that 1) the language model approach is not viable for standard IR applications as it is too slow (execution times of several seconds) and 2) as Web graphs expand, memory usage favors HyperBall-Diversify whereas re-ranking time remains constant. This implies that, among all methods proposed, HyperBall-Diversify is the most scalable option if slightly slower re-ranking times are permissible.

4.6 Qualitative differences

Apart from the quantitative assessments, there are a number of qualitative differences between HyperBall-Diversify and the other diversification methods. Most notably, we observe that HyperBall-Diversify poses no additional data requirements in that it relies solely on the properties inherent to the dataset. There is no need to train a model or pre-process supplementary query logs which can add significant complexity in terms of parameter selection and tuning. Additionally, due to its lean memory footprint and structure, HyperBall-Diversify is scalable and lends itself well to distributed environments and parallelization since each Web graph node is encapsulated entirely within an independent HyperLogLog counter. The baseline methods discussed in this paper require a much greater degree of redundancy across parallel nodes. Furthermore, changes to the underlying graph structure can be quickly accounted for via a simple update to the offending node's HyperLogLog counter.

In spite of its simplicity, HyperBall-Diversify does require that the chosen data source can be represented as a graph. While this requirement can be met by most Web corpora, it nonetheless means that this method cannot be ap-

Table 1: Intent-aware ranking performance.

	Wikipedia			ClueWeb'12 (WT'13)			ClueWeb'12 (WT'14)		
Method	MAP-IA	NDCG-IA	α-NDCG	MAP-IA	NDCG-IA	α-NDCG	MAP-IA	NDCG-IA	α-NDCG
HBALL	0.000199	**0.2554***	0.0853	**0.000137***	**0.3782***	**0.1684***	**0.000141***	**0.3594***	**0.1494***
TM	**0.000248**	0.2324	0.1017	0.000094	0.3295	0.1255	0.000096	0.3121	0.1104
LM	0.000197	0.2365	**0.1164***	0.000089	0.3386	0.1493	0.000093	0.3299	0.1229
ORIG	0.000196	0.2324	0.0685	0.000089	0.3195	0.1107	0.000092	0.3077	0.0868

plied to any and all data sets. This Web graph dependency is not present for the two baseline methods (topic model and language model). Finally, HyperBall-Diversify is blind to changes in user search behavior and individual user preference. Section 2 discusses other diversification schemes that incorporate query logs to track user search behavior and incorporate personalization in their result sets. However, as query logs are not always available, the detriment of not personalizing results comes with the benefit of fewer dataset requirements.

5. CONCLUSION

In this paper, we introduced a novel ranking scheme that incorporates diversity into result sets by maximizing Web graph coverage. Due to the massive scale of modern Web collections, exact solutions are intractable at query time. Instead, our implementation named HyperBall-Diversify utilizes graph sketching techniques to deliver an approximate solution that is fast and memory efficient, yet accurate. The submodular nature of our implementation guarantees a mathematically bounded, worst-case result through a simple greedy algorithm. We evaluated our algorithm against two state-of-the-art diversification approaches using a range of standard evaluation metrics and two sizeable Web corpora.

Our experimental results conclude that the re-ranking performance of all competing methods was closely tied, with a significant improvement for HyperBall-Diversify on the ClueWeb collection. In a second set of experiments, the language model approach was shown to be prohibitively slow in execution while our proposed method approaches the time efficiency of the topic model scheme. Finally, we showed that HyperBall-Diversify has the smallest memory footprint of all compared approaches, resulting in up to 200% reduced space requirements even at high accuracy ($b = 10$). This is especially interesting for applications in distributed settings since our method represents each document entirely by its HyperLogLog counter, requiring no additional redundancy as compared to the single-machine setting.

In this first exploratory analysis, we relied on a straight forward linear combination of relevance and diversity scores in order to understand the general dependencies at play in the domain. In the future, we aim to explore alternative objective functions for combining the two constituent scores in a more principled way. Additionally, the Web graph coverage framework gives us an exciting set of yet untapped tools for search result diversification. Relying on the appropriate sketching methods, notions such as degree centrality can more accurately describe the coverage and location of the topical pockets relative to the center of graph mass. Lastly, we hope to explore improved personalized search via a user-defined or dynamic lambda value.

6. REFERENCES

[1] Rakesh Agrawal, Sreenivas Gollapudi, Alan Halverson, and Samuel Ieong. Diversifying search results. In *WSDM 2009*. ACM.

[2] Paolo Boldi and Sebastiano Vigna. In-core computation of geometric centralities with hyperball: A hundred billion nodes and beyond. *CoRR*, 2013.

[3] Gabriele Capannini, Franco Maria Nardini, Raffaele Perego, and Fabrizio Silvestri. Efficient diversification of web search results. *VLDB Endowment 4, 2011*.

[4] Olivier Chapelle, Shihao Ji, Ciya Liao, Emre Velipasaoglu, Larry Lai, and Su-Lin Wu. Intent-based diversification of web search results: metrics and algorithms. *Information Retrieval 6, 2011*.

[5] Charles L Clarke, Nick Craswell, and Ian Soboroff. Overview of the trec 2009 web track. Technical report, DTIC Document, 2009.

[6] Charles L Clarke, Maheedhar Kolla, Gordon V. Cormack, Olga Vechtomova, Azin Ashkan, Stefan Büttcher, and Ian MacKinnon. Novelty and diversity in information retrieval evaluation. In *SIGIR 2008*. ACM.

[7] Steve Cronen-Townsend, Yun Zhou, and W. Bruce Croft. Predicting query performance. In *SIGIR 2002*. ACM.

[8] Van Dang and W Bruce Croft. Diversity by proportionality: an election-based approach to search result diversification. In *SIGIR 2012*.

[9] Zhicheng Dou, Sha Hu, Kun Chen, Ruihua Song, and Ji-Rong Wen. Multi-dimensional search result diversification. In *WSDM 2011*. ACM.

[10] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and et al. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In *Proceedings of the 2007 Interntaional Conference on Analysis of Algorithms*.

[11] Sreenivas Gollapudi and Aneesh Sharma. An axiomatic approach for result diversification. In *WWW 2009*. ACM.

[12] Victor Lavrenko and W. Bruce Croft. Relevance based language models. In *SIGIR 2001*. ACM.

[13] Rong-Hua Li and Jeffery Xu Yu. Scalable diversified ranking on large graphs. *IEEE Transactions on Knowledge and Data Engineering*, 25(9):2133–2146, 2013.

[14] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions-i. *Mathematical Programming*, 14(1), 1978.

[15] Filip Radlinski and Susan Dumais. Improving personalized web search using result diversification. In *SIGIR 2006*. ACM.

[16] Davood Rafiei, Krishna Bharat, and Anand Shukla. Diversifying web search results. In *WWW 2010*. ACM.

[17] Rodrygo LT Santos, Craig Macdonald, and Iadh Ounis. Intent-aware search result diversification. In *SIGIR 2011*.

[18] Fei Song and W. Bruce Croft. A general language model for information retrieval. In *CIKM 1999*. ACM.

[19] David Vallet and Pablo Castells. Personalized diversification of search results. In *SIGIR 2012*. ACM.

[20] Wikipedia. Wikipedia english offline edition 2014-02-03. 2014.

[21] Xiaoshi Yin, Jimmy Xiangji Huang, Zhoujun Li, and Xiaofeng Zhou. A survival modeling approach to biomedical search result diversification using wikipedia. *Knowledge and Data Engineering, IEEE Transactions on*, 25(6), 2013.