Indexing Word-Sequences for Ranked Retrieval

SAMUEL HUSTON, University of Massachusetts Amherst
J. SHANE CULPEPPER, RMIT University
W. BRUCE CROFT, University of Massachusetts Amherst

Formulating and processing phrases and other term dependencies to improve query effectiveness is an important problem in information retrieval. However, accessing word-sequence statistics using inverted indexes requires unreasonable processing time or substantial space overhead. Establishing a balance between these competing space and time trade-offs can dramatically improve system performance.

In this paper, we present and analyze a new index structure designed to improve query efficiency in dependency retrieval models. By adapting a class of (ϵ, δ)-approximation algorithms originally proposed for sketch summarization in networking applications, we show how to accurately estimate statistics important in term dependency models with low, probabilistically bounded error rates. The space requirements for the vocabulary of the index is only logarithmically linked to the size of the vocabulary.

Empirically, we show that the sketch index can reduce the space requirements of the vocabulary component of an index of n-grams consisting of between 1 and 4 words extracted from the GOV2 collection to less than 0.01% of the space requirements of the vocabulary of a full index. We also show that larger n-gram queries can be processed considerably more efficiently than in current alternatives, such as positional and next-word indexes.

Categories and Subject Descriptors: H.3.4 [**Information Storage and Retrieval**]: Systems and software—*performance evaluation*; H.3.1 [**Content Analysis and Indexing**]: Indexing methods; H.3.3 [**Information Search and Retrieval**]: Information filtering

General Terms: Algorithms, Experimentation, Performance

Additional Key Words and Phrases: Sketching, Indexing, Scalability, Term Dependency Models

ACM Reference Format:
Samuel Huston, J. Shane Culpepper, W. Bruce Croft, 2013. Indexing Word-Sequences for Ranked Retrieval. ACM Trans. Inf. Syst. 31, 0, Article 0 (2013), 27 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

Term dependency models are a compelling new approach to improving effectiveness in ranked document retrieval. A term dependency is any relationship between two or more terms. Examples of term dependencies include noun phrases, verb phrases, ordered windows, unordered windows, spans of text, or any sequence of n-grams. Many recently developed retrieval models depend on statistics extracted for dependencies

This work was supported in part by the Center for Intelligent Information Retrieval, in part by NSF CLUE IIS-0844226, in part by NSF grant #CNS-0934322 and in part by the Australian Research Council. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsor.
Author's addresses: S. Huston and W. B. Croft, Center for Intelligent Information Retrieval, University of Massachusetts Amherst; J. S. Culpepper, School of Computer Science & Information Technology, RMIT University.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

between query terms [Metzler and Croft 2005; Lu et al. 2006; Bendersky and Croft 2008; Xue and Croft 2010; He et al. 2011; Broschart and Schenkel 2012]. While these methods have been shown to significantly improve the *effectiveness* of the retrieval model, little prior work has addressed how to *efficiently* generate the necessary statistics at query time. Broschart and Schenkel [2012] present an efficiency and effectiveness trade-off for proximity-based ranking algorithms. By selectively indexing terms that co-occur within a fixed window length, Broschart and Schenkel are able to prune the inverted files to include only co-occurrences above a pre-defined impact threshold, thus controlling the space used by the index.

Several successful approaches to the problems of plagiarism, duplicate and local text reuse detection also rely on phrase or word sequence features extracted from documents [Bernstein and Zobel 2006; Seo and Croft 2008; Hamid et al. 2009]. These approaches all assume there is some significant redundancy in instances of duplication or reuse. They exploit this assumption by defining discard policies that reduce the set of features extracted from each document. These policies reduce the total space requirements and allow the system to efficiently retrieve duplicated text. However, they each introduce the risk that vital features for some query document or passage may have been removed by the feature discard policy.

In this paper, we focus on the problem of calculating document level statistics for *all* word sequences (n-grams) in the collection in a space and time efficient manner, while minimizing the risk that important features will be inaccurately stored. We will use the term n-*gram* as any sequence of n sequential terms extracted from a document. We select this terminology in order to avoid confusion with linguistic definitions of various types of phrases and shingle extraction techniques. This type of positional term dependency is often used as a surrogate for more complex, linguistic term dependencies. Improving the efficiency of the calculation of this type of retrieval model feature can improve the efficiency of all the retrieval models that use this type of term dependency.

There are existing approaches to storing and retrieving document level n-gram statistics. For example, many current open source search engines provide access to such positional information of terms in inverted indexes [Witten et al. 1999]. These indexes support the calculation of n-gram statistics by processing n term posting lists simultaneously, and comparing position offset information for each term in each document. The space requirements for this approach are not substantial – usually a fraction of the size of the collection. However, this approach to the query-time calculation of n-gram statistics can result in inefficient query processing.

An obvious solution for improving the n-gram query processing efficiency is to construct an inverted index of all n-gram term dependency data. This approach stores a mapping for each term dependency to a posting list of documents and document frequencies. Processing term dependency queries using this type of index is fast; the inverted list for a query can be extracted and processed by direct lookup. However, previous research has shown that n-gram can create extremely large vocabularies. For example, about half of the 5-grams extracted from Clueweb-Part-B are unique in the collection [Huston et al. 2011]. The size of the vocabulary is the main drawback of this approach. By the nature of n-gram, each term in the collection occurs in n vocabulary entries. In the worst case, the vocabulary component of a full index would require as much as n times the storage required for the collection.

An alternative approach is to only store a subset of the n-grams. A frequent item index [Huston et al. 2011] is one approach that uses significantly less space than a full inverted index. As Huston et al. note, there are two possible execution modes for this index structure. First, missing n-grams could be ignored. Second, missing n-grams could be recreated using a positional index. The structure provides the guarantee that any missing term dependency will occur in a limited number of documents. However,

this subset of documents is likely to be relevant to the query. A different, dynamic subset selection algorithm is to cache recently queried n-grams and phrases as an intersected posting list [Ozcan et al. 2011]. This structure provides no guarantees about the sparsity or utility of missing n-grams. Therefore, n-grams that are present in queries but not in the index must be recomputed from a positional index or ignored.

Next-word indexing [Williams et al. 1999] was originally proposed as an attractive trade-off between index space and retrieval efficiency for processing phrase queries. A next-word index stores position data for all pairs of adjacent terms in the collection. Phrases are processed in a fashion similar to a positional index. The query is segmented into a set of adjacent term pairs, and the posting list for each term pair is processed in parallel. Positional data stored in each posting list is then used to determine if the phrase is present in a document. We note that this structure is strictly limited to processing phrase or n-gram queries. We will show that our new sketch index data structure is able to process larger phrase queries significantly faster than next-word indexes. In follow-on work, Williams et al. [2004] investigate the time and space trade-offs of indexing common phrases as a term, and using a next-word index to resolve less common phrase queries. Assuming the correct phrases can be identified, the hybrid approach was shown to be significantly more efficient than traditional phrase resolution using positional offsets. The idea of selectively indexing a subset of term dependency components to improve efficiency served as inspiration for the approaches of [Huston et al. 2011] and [Broschart and Schenkel 2012]. However, these approaches always result in selective index tuning in order to account for variance across collections and query streams. Our new approach is designed to be robust and efficient regardless of the query input.

In this study, we present an indexing structure using data stream sketching techniques to estimate n-gram statistics. Our sketch index is derived from a **CountMin** sketch [Cormode and Muthukrishnan 2005b], and designed to minimize space usage while still producing accurate statistical estimates. This strategy also ensures that the space required by the index is independent of the number of indexed n-gram terms, while still supporting efficient query processing. This work significantly extends and analyzes the ideas presented in Huston et al. [2012].

Conceptually, our summary sketch is an (ϵ, δ)-approximation of a full inverted index structure. So, the index representation is capable of estimating collection statistics for a specific n-gram with bounded performance. We show that the relative error of extracted term dependency statistics can be probabilistically bounded and describe how these bounds minimize the space requirements of the structure in practice. We establish that the retrieval efficiency of the sketch index is comparable to full indexes, and notably faster than positional or next-word indexes. Finally, our experiments demonstrate that our estimator does not significantly alter the query effectiveness when using current state-of-the-art term dependency models.

This paper is structured as follows: Section 2 presents the necessary background on data stream sketching techniques; Section 3 presents the algorithmic framework for our term dependency statistics estimator, and outlines the probabilistic error bounds ensured by the representation; Efficient approaches to constructing the new sketch data structure are discussed in Section 4; Section 5 evaluates the performance of our new estimator empirically. We discuss future work in Section 6; and we conclude in Section 7.

2. FREQUENCY-BASED SKETCHING

Algorithms for approximating the frequency of items in a collection or a stream have advanced dramatically in the last twenty years [Cormode and Hadjieleftheriou 2008, 2010]. This line of research is based on the streaming model of computation, and has

widespread applications in networking, databases, and data mining [Muthukrishnan 2005]. Much of the work in the networking community using these tools has focused on identifying "heavy-hitters", or top-k items (see [Berinde et al. 2009] or [Cormode and Hadjieleftheriou 2008] and the references therein). If only the k most frequent items must be accurately estimated, counter-based approaches work well in practice. However, counter-based methods are generally not sufficient if estimates for *all* the items in a stream are desirable since the number of counters is limited to the top-k subset of items in the stream. For frequency estimation of any item in a stream, various "sketching" methods are an appropriate alternative.

Here, we limit our discussion to sketching algorithms as these data structures are able to bound the allowable error of approximation for all items in a stream. A *sketch* is a hash-based data structure that represents a linear projection of the streaming input. Two general approaches to sketching are present in the literature: **AMS** and **CountMin**.

The **AMS** sketch was first proposed by Alon et al. [1999] to estimate the second frequency moment ($\mathcal{F}_2$), relative the collection size ($|\mathcal{C}|$), with error $\epsilon\sqrt{\mathcal{F}_2} \leq \epsilon|\mathcal{C}|$ with probability at least $1-\delta$ for a sketch using $\mathcal{O}(\frac{1}{\epsilon^2}\log\frac{1}{\delta})$ bits of space. However, the original representation of **AMS** is not efficient in practice since the whole sketch must be updated for each new item.

To address this shortcoming, Charikar et al. [2002] proposed a modification that ensures each update only affects a small subset of the entire summary. Charikar et al. [2002] refer to this approach as a **CountSketch**. The key idea of a **CountSketch** is to create an array of $r \times w$ counters, with independent hash functions for each row r. The hash functions map each update to set of counters, one in each row r. In addition, another independent hash function maps the value $\{-1, +1\}$ to each update. This approach ensures that collisions over the entire distribution are likely to cancel out. Increasing the number of rows used (r) lowers δ. So, to match the same ϵ and δ bounds of the **AMS**, the values $r = \log\frac{4}{\delta}$ and $w = \mathcal{O}(\frac{1}{\epsilon^2})$ are used. Using these parameters, the space bound for the **CountSketch** is now identical to **AMS**, but update time is reduced to $\mathcal{O}(\log\frac{1}{\delta})$.

Another sketching alternative was recently proposed by Cormode and Muthukrishnan [2005b]. The **CountMin** sketch is similar in spirit to **CountSketch**, in that the method uses an array of $r \times w$ counters. The key difference is the omission of the secondary hashing function which maps $\{-1, +1\}$ onto each update. Instead, **CountMin** always increments each counter. In streams which do not include deletions, this ensures that the frequency of any item $f(i)$ in the sketch is an overestimate. The expected number of collisions for i on any row is $\sum_{1\leq i'\leq\sigma, i'\neq i} f(i')/w$. **CountMin** can be used to estimate $\hat{f}_i$ with error at most ϵn with probability at least $1-\delta$ using $\mathcal{O}(\frac{1}{\epsilon}\log\frac{1}{\delta})$ bits. The time per update is $\mathcal{O}(r)$ where $r = \log\frac{1}{\delta}$ and $w = \mathcal{O}(\frac{1}{\epsilon})$.

An example **CountMin** sketch is shown in Figure 1. When an item, i, is added to or removed from the sketch, one counter is incremented or decremented in each row of the **CountMin** sketch. The correct cell is determined by the corresponding hash function. Formally:

$$\forall_{j<r} : count[j, h_j(i)] := count[j, h_j(i)] \pm 1$$

If the stream contains only positive frequencies, the frequency of item i can be estimated by returning the minimum count in the set.

$$\hat{a}_i = \min_j \ count[j, h_j(i)]$$

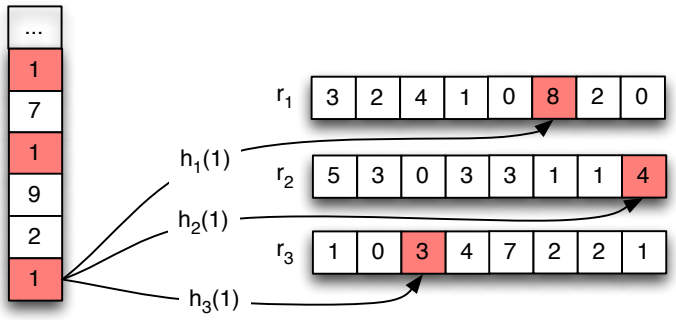


Fig. 1. Example **COUNTMIN** Sketch containing frequency data for a stream of integers. Where the highlighted integer, 1, is hashed to each of the highlighted cells. The frequency of 1 in this stream is estimated as the minimum value of the highlighted cells $f_1 = 3$.

For streams allowing positive or negative frequencies, the frequency of i is estimated by returning the median of the r counts.

$$\hat{a}_i = \underset{j}{\text{median}}\ count[j, h_j(i)]$$

In general, the ϵ-bound of the **COUNTMIN** sketch cannot be directly compared to the ϵ-bound of the **AMS** sketch [Cormode and Muthukrishnan 2004]. The **COUNTMIN** sketch provides bounds relative to the L_1-norm, and **AMS** style sketches provide bounds relative to the L_2-norm. In practice, this is not a huge limitation, and both sketching approaches have been shown to be effective and efficient for skewed data collections [Charikar et al. 2002; Ganguly et al. 2004; Cormode and Muthukrishnan 2005a; Cormode and Hadjieleftheriou 2008].

Additional enhancements and applications of **COUNTMIN** have been proposed in the literature. Conservative update, originally presented by Estan and Varghese [2002], is a heuristic method used to improve the frequency estimates produced by **COUNTMIN** by minimizing collisions. It operates by only updating the minimum set of rows in the **COUNTMIN** sketch. Using this approach, the update function for each row j for the event i becomes:

$$count[j, h_j(i)] := \max(count[j, h_j(i)], \min_{k<r}(count[k, h_k(i)] + 1))$$

3. SKETCHING STATISTICS FOR N-GRAMS

3.1. Inverted-Sketch-Index

Let $\mathcal{C}$ be a text collection partitioned into l documents $\{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_l\}$ containing at most σ_n unique terms. Here, a term t can be an n-gram, a sequence of n adjacent words. So, σ_1 represents the total number of unique 1-grams in the collection. An *inverted index*, $\mathcal{I}$ counts the number of times each term t appears in each document $\mathcal{D}_j$. Conceptually, this can be represented as a $\sigma_n \times l$ matrix, $\mathcal{M}$, as each indexed term t may appear in any document $mdoc_j$. Figure 2 shows an example of the matrix representation of an inverted index.

	$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_3$	$\mathcal{D}_4$	$\mathcal{D}_5$	$\cdots$
t_0	1	3	2	6	0	$\cdots$
t_1	2	0	3	1	2	$\cdots$
t_2	2	5	1	3	7	$\cdots$
$\vdots$						

Fig. 2. Example matrix representation, $\mathcal{M}$, of a term-level, non-sparse inverted index, $\mathcal{I}$.

The following notation will apply for our discussion:

— $f_{d,t}$, the frequency of term t in document $\mathcal{D}_d$;
— $f_{q,t}$, the frequency of term t in the query;
— f_t, the number of documents containing one or more occurrences of term t;
— F_t, the number of occurrences of term t in the collection;
— l, the number of documents in the collection;
— σ_n, the number of indexed terms in the collection (vocabulary size); and
— $|\mathcal{C}| = \sum_{i=0}^{\sigma_1} F_{t_i}$, the total number of tokens in the collection.

In practice, $\mathcal{M}$ is sparse, and each row in $\mathcal{I}$ is often stored as a compressed posting list. An inverted index is a mapping of keys to a list of document counters. For each document identifier j, $f_{d,t}$ is maintained. Traditionally, each term in the vocabulary is stored explicitly in a lookup table.

$\vdots$				
$t_0\ t_1\ t_2$	$(\mathcal{D}_1, 1)$	$(\mathcal{D}_3, 3)$	$(\mathcal{D}_{10}, 2)$	$\cdots$
$t_0\ t_1\ t_3$	$(\mathcal{D}_2, 2)$	$(\mathcal{D}_3, 5)$	$(\mathcal{D}_4, 7)$	$\cdots$
$t_0\ t_1\ t_4$	$(\mathcal{D}_1, 1)$	$(\mathcal{D}_6, 5)$	$(\mathcal{D}_7, 1)$	$\cdots$
$\vdots$				

Fig. 3. Example n-gram inverted index. For each n-gram, a list of documents and document frequencies are stored. Documents are sorted by identifier. If the n-gram is not present in a document, then the document is omitted from the index structure. Integer compression techniques can be used to reduce the total space requirements of the data structure.

Now, consider the case of constructing an inverted index of n-grams. The collection $\mathcal{C}$ can contain at most $(|\mathcal{C}| - n + 1)$ distinct n-grams. This number is often less than the σ_1^n possibilities, but still significantly larger than σ_1, thus increasing the number of potential rows in $\mathcal{M}$. Figure 3 shows an example of $\mathcal{I}$ when using n-gram terms.

We investigate how to apply the ideas presented by Cormode and Muthukrishnan [2005b] to fix the number of rows in $\mathcal{M}$ and still provide accurate statistical information. Interestingly, l is already static for a given collection, and $l \ll |\mathcal{C}|$. But, the number of rows, σ_n, increases with n, and we would like to minimize this overhead. Note that the total number of rows required in the sketch is proportional to $r \cdot f_t$. So, if we reduce $\mathcal{M}$ to a linear projection of f_t, we can use **CountMin** to accurately approximate $\hat{f}_t$. Recall that the expected number of collisions for t on any row in the sketch is $\sum_{1 \leq t' \leq \sigma, t' \neq t} f_{t'}/w$. Using a Markov inequality argument, Cormode and Muthukrishnan [2005b] show that by setting $w = 2/\epsilon$ and $r = \log 1/\delta$ in the sketch, the estimate $\hat{f}_t$ is at most $\epsilon \mathcal{F}_1$ with probability at least $1 - \delta$, where $\mathcal{F}_1$ is the first frequency moment $\sum_{1 \leq t' \leq \sigma_n} f_{t'}$, the sum of all of the frequencies. Note that this proof assumes that the hash functions selected are from a pair-wise independent family of hash functions.

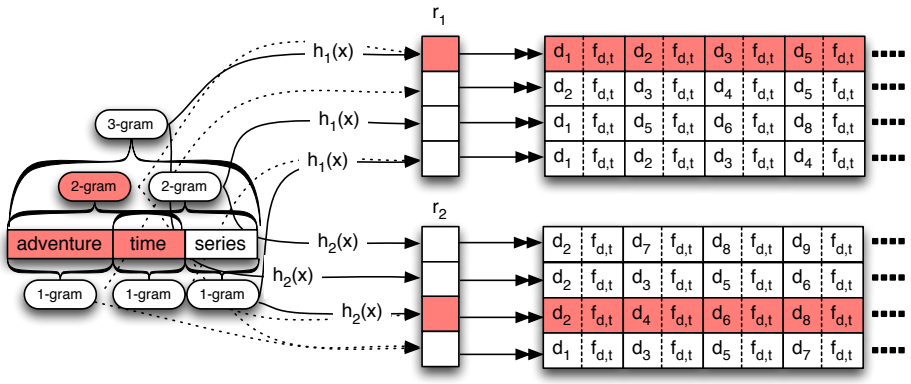


Fig. 4. Example index representation of a sketch index data structure composed of two rows, $r \in \{0, 1\}$, each hash function is required to be pair-wise independent, and returns values in the range $[0, w-1]$. Note that the postings lists in each row may contain hash collisions.

In a sketch representation of an inverted index, each distinct term is replaced with a hash value where each hash value *may* represent more than one term. This reduction means that the vocabulary of n-grams no longer needs to be stored with the index. Figure 4 shows an example of our sketch-based indexing representation. If a simple hashing representation were used, then there is no mechanism available to resolve collisions unless each term string is accessible to the table. However, using the collision mitigation strategy of a sketch, such as the method described for **CountMin**, we are able to reduce the probability that hashing collisions will result in incorrect results.

We note that it may be possible to use a perfect hash function to avoid collisions entirely for this type of index. However, one of the key advantages of a non-perfect hash function is that there is no requirement to retain the original vocabulary, thus avoiding unrealistic space requirements. We leave the investigation of this type of hash function to future work.

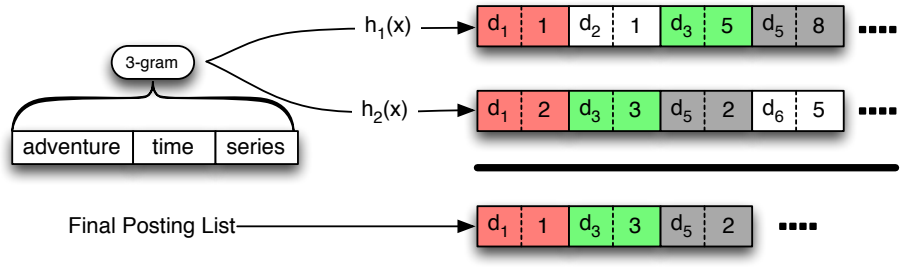


Fig. 5. Example extraction of the statistics for a single term dependency in our sketch representation. The first two posting lists represent the 3-gram, `"adventure time series"`, extracted from the sketch using the corresponding hash functions. The final posting list is simulated by intersecting r posting lists (in this case, $r = 2$). Colors identify matching documents for each $\hat{f}_{d,t}$ counter. The final posting list contains the minimum $\hat{f}_{d,t}$ from each matching document across r lists. For any document not represented in all r lists, the minimum is assumed as $\hat{f}_{d,t} = 0$.

Our new indexing structure is composed of an $r \times w$ matrix of pointers to $r \times w$ postings lists. Conceptually, this matrix is equivalent to a **COUNTMIN** sketch designed to estimate $\hat{f}_t$ with one twist: we do not simply use a single counter to aggregate $\hat{f}_t$, but rather allow multiple document counters attached in list-wise fashion to each cell in the **COUNTMIN** sketch to form posting lists, as illustrated in Figure 4. These document counters are then used to aggregate $\hat{f}_{d,t}$.

This approach allows us to fix the size of the lookup table independent of the order of the n-grams being indexed. We do not attempt to fix the number of $\hat{f}_{d,t}$ counters. As in a standard inverted index, every term could appear in every document, producing a maximum of $l \cdot |\sigma_n|$ counters in the worst case. But, in practice, the distribution is skewed, and many terms have few non-zero $\hat{f}_{d,t}$ counters. Note that since the width of $|\sigma_n|$ is fixed in our approach, the number of counters is largely independent of the order of n, but rather some percentage of the counters are redistributed in the redundant postings lists.

We now discuss how to estimate the frequency of a particular n-gram using our approach. By using the biased estimation of a **COUNTMIN** sketch of only positive counts, our estimates of $\hat{f}_t$, and subsequently $\hat{f}_{d,t}$, are guaranteed to be an overestimate of the true term counts. Furthermore, the same formal arguments using the Markov inequality and Chernoff bounds can be made for bounding $\hat{f}_t$, and subsequently $\hat{f}_{d,t}$, we could reasonably expect for each cell. So, to estimate $\hat{f}_t$ using **COUNTMIN**, we would take $\min_j \ count[j, h_j(x_i)]$. But, each counter $count[\ ]$ is actually a pointer to a postings list, containing approximately $\hat{f}_t$ counters. When the posting list for any t is requested, r posting lists are extracted and the *intersection* of the lists represents the $\min_j$ of the r postings lists. Figure 5 shows an example of the intersection process that represents the $\min_j$ for a given t.

This index is only intended to store and return document-level statistics, and to enable the ranked retrieval of documents for an input query. The use of hashing functions in the structure prohibit tasks that require vocabulary exploration. The proposed structure also cannot store positional data for n-grams, prohibiting the use of the structure in common post-retrieval tasks such as snippet generation. It is also possible to augment sketch-indexes to store positional data, but we do not explore this alternative here. The sketching techniques discussed here can be directly applied in a frequent index [Huston et al. 2011] and in term-pair indexes as proposed by Broschart and Schenkel [2012]. Currently, the vocabulary of the sketch index and single-hash index are stored in memory with computed hash values used to index an array of file pointers. It is also possible to store this table implicitly, by using a B+Tree to map each hash value to the posting list data. This approach permits fine grained control of the memory requirements of the structure, but may introduce additional disk-seeks when executing queries. In future work we intend to investigate the best combinations of sketch-indexes and other term selection techniques as well as other trade-offs in managing the hash value mappings.

4. SKETCH INDEX CONSTRUCTION

Construction of an n-gram sketch index is similar to the construction of an inverted index. Inverted index construction algorithms, as discussed by Witten et al. [1999], require only minor modifications in order to be adapted to term dependency estimator construction. The SORT-BASED SKETCH-INVERSION algorithm details one possible sort-based inversion algorithm to construct the estimator representation discussed here.

Algorithm 1: SORT-BASED SKETCH-INVERSION

Input: Text collection, $\mathcal{C}$
Output: Inverted Sketch Index, $\mathcal{I}_S$
for each $\mathcal{D}_d \in \mathcal{C}$ **do**
  for each *term dependency,* $t_j \in (\mathcal{D}_d)$ **do**
    for $k \leftarrow [1 \dots r]$ **do**
      append $(h_k(t_j), \mathcal{D}_d, f_{d,t_j})$ to the output file O_k
    end
  end
end
for $k \leftarrow [1 \dots r]$ **do**
  sort O_k lexicographically
end
write the data from each row, $(O_1 \dots O_r)$, to sketch index structure $\mathcal{I}_S$

In the simplest case, to generate an index of n-grams, a linear pass over the text collection with a sliding window of size n is performed. Each of the r hash functions are applied to each n-gram extracted to generate a set of r *term-ids*. The *term-id* data for each row is sorted. Then the algorithm writes the sorted data directly to the sketch index structure $\mathcal{I}_i$.

The cost of constructing our term dependency estimator is therefore equivalent to the cost of constructing an inverted index of n-grams with r repetitions. Specifically, we can bound the cost of constructing a sketch index to $\mathcal{O}(r \cdot |C| \log |C|)$, as the set of term postings must be sorted r times, once for each row.

Modifications can be made to any distributed term-level indexing algorithm, such as the MapReduce indexing algorithm described by McCreadie et al. [2009], to produce a sketch index structure. For example, the algorithm proposed by McCreadie et al. [2009] would be modified by augmenting each key-value pair to include the id of the sketch row as part of the key. The MapReduce key-value pairs, $(t, posting\text{-}list\text{-}data)$, are substituted with, $((r_k, h_{r_k}(t)), posting\text{-}list\text{-}data)$. Then the map and reduce functions are modified accordingly.

As sketch-based algorithms were originally designed for streaming data applications, the approach is also amenable to dynamic construction and maintenance of the index. A fully dynamic version of the index can be constructed by applying the dynamic indexing algorithms presented by Büttcher et al. [2010]. One approach is to maintain an in-memory version of the sketch index as documents are added to the collection. Periodically the in-memory sketch index is written to disk as an index shard. Index shards on disk are periodically merged to control the total number of index shards. In order to provide retrieval over the entire collection at any time, the posting list data extracted from the memory-shard and each disk index shard are merged at query time.

5. EXPERIMENTS

5.1. Experimental Setup

We investigate the performance trade-offs of our new index structure using three TREC collections: Robust-04, GOV2 and ClueWeb-B. Statistical properties for each collection are shown in Table I. In each of our experiments, we measure index properties and retrieval performance on n-gram data. An n-gram is defined as any sequence of n sequential words. In the literature, each distinct sequence is sometimes referred to as a phrase, or a shingle. An n-gram can often used as an approximation of a more complex linguistic term dependency.

Table I. Statistics for TREC Collections Robust-04, GOV2 and ClueWeb-B. Disk space is the space requirements of the uncompressed collections. Collection length, document count and vocabulary statistics are presented in millions (M).

	Robust-04	GOV2	ClueWeb-B
Disk Space	1.9 GB	426 GB	1,460 GB
Collection Length	252 M	23,000 M	39,800 M
Document Count	0.5 M	25 M	50 M
1-gram Vocab.	0.775 M	35 M	98 M
2-gram Vocab.	24.5 M	449 M	1,370 M
3-gram Vocab.	95.1 M	2,110 M	5,960 M
4-gram Vocab.	166 M	4,600 M	11,600 M
5-gram Vocab.	204 M	6,520 M	15,800 M

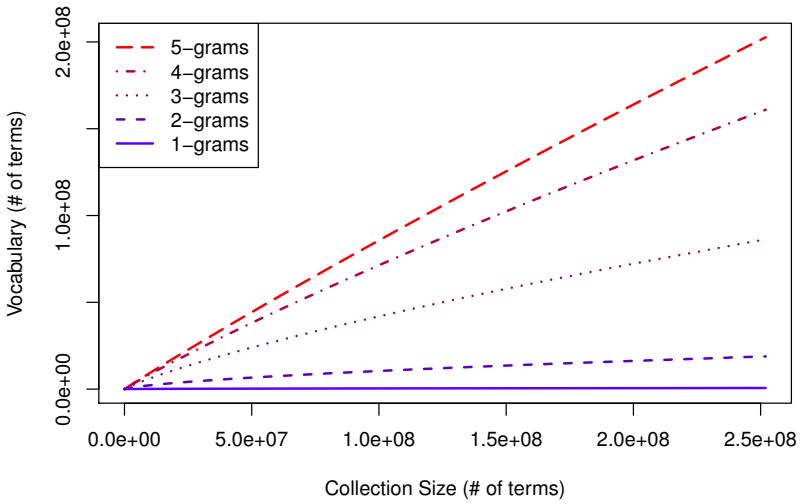


Fig. 6. Heaps' law graph showing the vocabulary growth of n-grams extracted from the Robust-04 collection.

It is important to note that as n increases, the vocabulary size increases dramatically. Figure 6 shows vocabulary growth rates for 1-to-5-grams for the Robust-04 collection. This data clearly shows that there is an unreasonable space requirement when storing the full vocabulary of n-grams for large collections.

A second important observation is that increasing n can have an affect on the skew of the statistical distribution of terms in a collection. Figure 7 shows the distribution of n-grams extracted from the Robust-04 collection. It is clear from this data that the skew of an n-gram distribution decreases as n increases.

We compare the performance of our statistical n-gram estimator with four previously proposed index structures capable of storing and returning document-level statistics of n-gram term dependencies. We compare our approach with positional indexes, full indexes of n-grams, frequent indexes, query-log-based indexes and next-word indexes. We also compare the sketch index to a single-hashed index.

To ensure a fair comparison, all baseline index structures are implemented using the same set of modern index compression techniques, including d-gap and *vbyte* integer compression for posting list data, and prefix-based vocabulary compression for b-tree

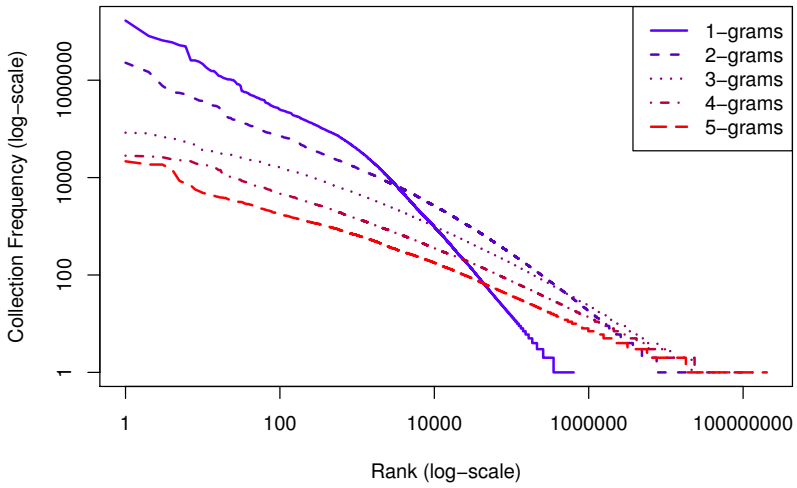


Fig. 7. Zipfian graph showing the distribution of n-grams extracted from the Robust-04 collection.

blocks. We use 32 kB b-tree blocks. Witten et al. [1999] provides a good description of standard compression techniques amenable to text indexing and retrieval.

Positional indexes are a commonly used index structure that store a mapping from each term in the collection to a list of documents, and the set of document positions for the term [Witten et al. 1999]. The n-gram frequency data for each document can be computed by comparing positional data for each term in the queried n-gram.

A full inverted index of n-grams stores a direct mapping from each n-gram in the collection to an inverted list of documents, and associated document frequencies. Unlike the positional index, the n-gram frequency data can be extracted directly from the index with a single lookup, and requires no further processing. This approach represents one of the fastest methods of accessing pre-calculated statistical properties of term dependencies.

Huston et al. [2011] propose a simple pragmatic approach: frequent indexes. This type of index can be viewed as a subset of a full index of n-gram data. Only n-grams that occur more frequently are present in the index. Using an appearance threshold, h, selection can be done at indexing time, and may be collection dependent. The authors suggest that the index should be used in conjunction with a standard positional index of 1-grams in order to compute statistics of infrequent n-grams. For efficiency experiments, we use a positional index to supplement this index structure.

We also test a second method of selecting a subset of n-grams to index, based on a simulated query log. Using queries extracted from the AOL query log,[1] we build an index of recently queried n-grams. This method is analogous to the construction of a cache of intersected posting lists [Ozcan et al. 2011]. After ordering the query log by timestamps, we index all n-grams extracted from the first 90% of the AOL query log. The remaining 10% is reserved to test the retrieval efficiency of each of the index

[1] http://www.gregsadetsky.com/aol-data/

structures. Note that n-grams are extracted from each query, as if to be executed using the n-gram retrieval models presented in Section 5.3.

Next-word indexes, originally proposed by Williams et al. [1999], store a positional mapping for every word pair. The structure is divided into two files, a lexicon file and a vector file. The vector file stores positional posting lists for every word pair found in the collection. The lexicon file stores mappings from each word to a list of next-words and vector file offsets, and the vector file stores all posting lists. The auxiliary data structures in the next-word index can *only* be used for n-gram or phrase queries.

As a final sanity check and baseline, a singly-hashed index is also used. The single-hash indexing structure is equivalent to a single row of the sketch index. Each indexed n-gram is hashed to a b bit integer value. An index is constructed by associating each hash value with a posting list. This structure is implemented using a hash-indexed array of offsets into a file containing all posting lists. Recall that the likelihood of collisions decreases exponentially with respect to the number of independent hash functions used in any sketch-based data structure. Therefore, there is an implicit trade-off in the number of hashes used and the bounded error rate. Our approach works with a single hash or many hashes in a similar manner.

Our experiments focus on four key aspects of our new statistical term dependency estimator: relative statistical error, retrieval effectiveness, disk and memory space requirements, and retrieval efficiency. A key component of this study is the investigation of the relationships between retrieval effectiveness, space requirements and retrieval efficiency for the sketch index, in comparison to each of the baseline data structures. We show that the sketch index provides valuable new trade-offs between efficiency, effectiveness and space requirements.

In each experiment, ϵ and δ are reported for each sketch index. These parameters determine the width and depth of the sketch used in the sketch index. The depth of the sketch is determined as $\lceil \log \frac{1}{\delta} \rceil$. In these experiments, we focus on 1, 2, and 3 row sketch indexes, specified by $\delta \in \{0.5, 0.25, 0.125\}$, respectively. The width of the sketch is determined as $w = \frac{2}{\epsilon}$. So, where $\epsilon = 2.9e-06$, the width of the sketch is $554{,}751$ cells. The width of the hash values required from the hash function, for this value of ϵ, is at least $\lceil \log(554{,}751) \rceil = 20$ bits.

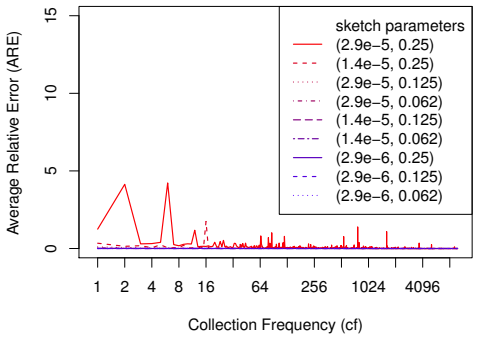
Each index structure we investigate in this section is implemented as an extension to the Galago package, provided by the Lemur Toolkit [Croft et al. 2001-2012]. All timing experiments were run on a machine with 8-core Intel Xeon processors, with 16 GB of RAM, running the CentOS distribution of Linux, using a distributed, network-attached, 4-node Luster file system to store index data. We measure the CPU time taken for at least 10 consecutive runs, and report the average in each experiment.

5.2. Estimation of Collection Frequency

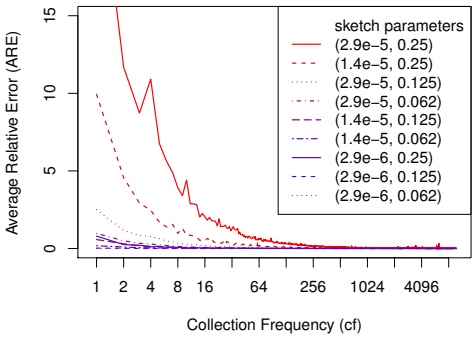
As discussed previously, sketch indexes provide an attractive trade-off between space usage and accuracy. In this section, we investigate the relationship between the (ϵ, δ) parameters and the quality of approximation by comparing the relative error of our approach to the true collection statistics. We show results computed on indexes created over n-grams, extracted from the TREC Robust-04 collection. The Average Relative Error (ARE) is defined as the average of the absolute difference between the true value and the estimated value. In this case:

$$ARE_n = \frac{1}{|\sigma_n|} \sum_{t \in \mathcal{T}_n} \frac{|F_t - \hat{F}_t|}{F_t},$$

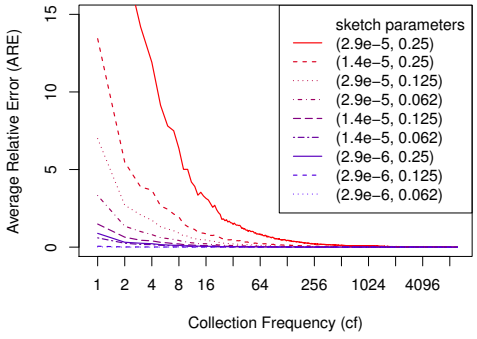
where $\mathcal{T}_n$ is the set of all unique terms (n-grams) of size n and $\sigma_n = |\mathcal{T}|$.



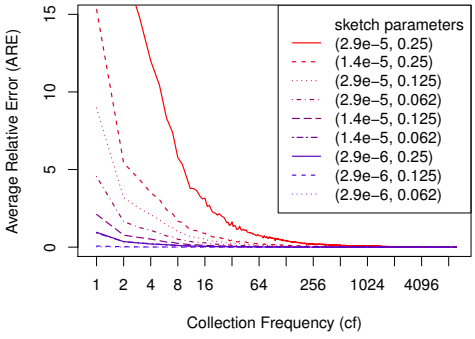
(a) 1-gram sketch index



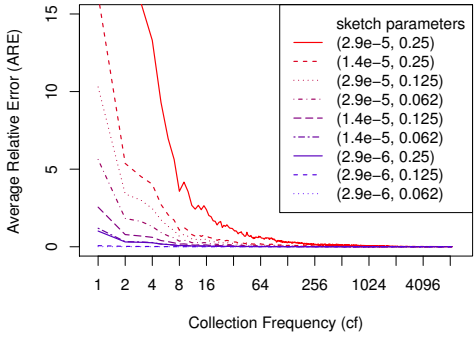
(b) 2-gram sketch index



(c) 3-gram sketch index



(d) 4-gram sketch index



(e) 5-gram sketch index

Fig. 8. Average relative error of n-gram frequency statistics extracted from 10 instances of sketch indexes over Robust-04 data, using each set of parameters. Sketch index parameters, (ϵ, δ), shown are $\epsilon \in \{2.9 \cdot 10^{-5}, 1.4 \cdot 10^{-5}, 2.9 \cdot 10^{-6}\}$, and $\delta \in \{0.25, 0.125, 0.062\}$. Note

Figure 8 shows ARE_n values grouped by F_t for several different n-grams using our approach with a variety of parameters. Data shown in this graph is aggregated from 10 instances of sketch indexes with each parameter setting. The x and y axes are identical for each graph, allowing direct comparison.

First, this data shows that conservative settings for (ϵ, δ) can ensure a low error rate in the estimation of collection statistics. Additionally, we can see that using overly restrictive values of ϵ and δ can degrade our estimates, particularly for infrequent items. This insight is not surprising, since summary sketching has primarily be applied in networking scenarios that require only the top-k items in a set to be accurately estimated. This problem is referred to as the "heavy-hitter" problem in the literature. Nevertheless, accurate estimates are possible using these approaches if conservative ϵ and δ values are used.

The graphs also show the relationship between error rate and the skew of the indexed data. Recall from Figure 7 that increasing n decreases the skew of the term frequency in the collection. Figure 8 shows that as the skew of indexed data decreases, ϵ must also be reduced in order to minimize the relative error.

In particular, we see that $\epsilon \leq 2.9 \cdot 10^{-6}$ and $\delta \leq 0.062$ produce a low relative error for all n-grams tested in the Robust-04 collection. Relative error for other important statistics including document count and document frequency for each n-gram were also evaluated. The graphs showing error rates for these statistics are omitted from this paper as the overall trends remain the same.

We now relate these observations to the theoretical bounds discussed previously. Recall that the expected error rate is controlled by ϵ and δ. Taking an example from our above graphs, we set $\epsilon = 2.9 \cdot 10^{-6}$, and $\delta = 0.25$. Theoretically the expected error for this type of sketch is $\epsilon \cdot |C| \leq 735$, for the Robust-04 Collection. We expect to see that no more than 25% of estimated collection frequencies overestimate the true statistic by more than 735. In practice, we obtain a much smaller observed error rate. In the data collected and summarized in the ARE graphs for these parameters, no n-gram collection frequency values are overestimated by 735. The largest overestimation of the collection frequency of an n-gram, observed in this data, is 261. From this data, it's clear our observations do not contradict the probabilistic error bounds. The difference between theory and practice here is that the theory assumes that each sketch cell, in this case a posting list, just stores a single collection frequency value. By intersecting the stored posting lists, we obtain a more conservative observed error.

5.3. Retrieval Effectiveness

We now investigate the effect of the use of sketch indexes on information retrieval effectiveness. We focus on testing that sketch indexes can be used to store and retrieve n-gram features for information retrieval without degrading retrieval effectiveness. We seek to investigate if this data structure introduces a risk of compromised retrieval effectiveness. For comparison, the relationship between hash-table size and retrieval effectiveness for the single-hashed index is also evaluated in these experiments. All the other benchmark index structures provide accurate collection statistics, thus they are not specifically evaluated in this section.

Given that the indexes we are focusing on in this study are n-gram indexes, it is appropriate to use an n-gram based retrieval model. We now define an n-gram retrieval model using the Markov Random Field retrieval model framework [Metzler and Croft

Table II. Details of TREC topic descriptions used for each collection for retrieval effectiveness experiments.

Collection	# of Queries	TREC Query IDs	# of Judgments
Robust-04	250	301 - 450, 601 - 700	311,410
GOV2	150	701 - 850	135,352

Table III. Example tuned parameters for each retrieval model for the GOV2 collection. Parameters shown here are tuned using 150 TREC description topics for the GOV2 collection.

Retrieval Model	Parameters (Λ)	Values
`sdm`	$(\lambda_T, \lambda_O, \lambda_U)$	$(0.837, 0.102, 0.061)$
`n1-4`	$(\lambda_{N1}, \lambda_{N2}, \lambda_{N3}, \lambda_{N4})$	$(0.95, 0.11, 0.01, -0.07)$

2005]. The `n1-4` retrieval model is defined as:

$$
\begin{aligned}
P_{n1-4}(D|Q) \stackrel{rank}{=} & \sum_{i \leq |Q|} \lambda_{n1} \log P(q_i|D) \\
& + \sum_{i \leq |Q|-1} \lambda_{n2} \log P(\#1(q_i, q_{i+1})|D) \\
& + \sum_{i \leq |Q|-2} \lambda_{n3} \log P(\#1(q_i, q_{i+1}, q_{i+2})|D) \\
& + \sum_{i \leq |Q|-3} \lambda_{n4} \log P(\#1(q_i, q_{i+1}, q_{i+2}, q_{i+3})|D),
\end{aligned}
$$

where the model is parameterized using 4 weights ($\Lambda = \lambda_{n1}, \lambda_{n2}, \lambda_{n3}, \lambda_{n4}$), and q_i is the i^{th} term in query Q. The $\#1$ operator, originally defined by Metzler and Croft [2005], is an ordered window operator, it matches instances of n-grams in each scored document, and returns the number of matches found, where n is as the number of terms provided to the operator.

To determine the retrieval performance of this model, we compare the n-gram retrieval model to two commonly used benchmark retrieval models, the query likelihood model (QL) [Ponte and Croft 1998], and the sequential dependency model (SDM) [Metzler and Croft 2005]. We start by tuning the parameters for each model on each collection. A summary of the TREC query and relevance data used in this experiment is shown in Table II. For each TREC collection – Robust-04 and GOV2 – coordinate ascent is used to tune query parameters, where mean average precision (MAP) is the metric optimized. A example of the tuned parameter values used for this experiment is shown in Table III. We omit the ClueWeb-B collection from these experiments due to the exceedingly large space requirements made by constructing a large number of indexes over a wide range of sketch parameters.

Since we use all available query and judgment data to tune each model, for each collection, the observed retrieval performance is considered the oracle performance for each collection. Observe that in these parameters, the weight for some of the longer n-gram features are negative. During tuning we observed that a positive weight for these features negatively impacts retrieval effectiveness. This observation implies that an exact match between the query phrasing and a particular document is weakly correlated with the document not being relevant to the query. Bendersky and Croft [2012] further discusses how negative-features can be incorporated into the Markov Random Field model in this manner.

Table IV shows the oracle model effectiveness for each collection using the MAP, nDCG20 and P@20 evaluation metrics. The `n1-4` model shows significant improve-

Table IV. Oracle retrieval effectiveness using MAP, nDCG@20 and P@20 retrieval metrics. Retrieval models shown include query likelihood (ql), sequential dependence model (sdm) and the n1-4 retrieval model. Results were produced using the Robust-04 and GOV2 collections. Significant improvements($^+$) over query likelihood (ql) are computed using the Fisher randomization test ($\alpha = 0.05$).

Retrieval Model	MAP	nDCG@20	P@20
Robust-04			
ql	0.245	0.390	0.333
sdm	0.262^+	0.412^+	0.352^+
n1-4	0.260^+	0.410^+	0.348^+
GOV2			
ql	0.249	0.367	0.464
sdm	0.276^+	0.404^+	0.515^+
n1-4	0.273^+	0.405^+	0.507^+

ments over the query likelihood retrieval model for each retrieval metric, using Fisher's randomization test, (using $\alpha = 0.05$). This statistical test does not show any significant differences between the n1-4 model, and the sequential dependency model, for each metric and each collection.

We now investigate the effect n-gram sketch indexes have on the effectiveness of the n1-4 retrieval model. We explore the relationship between different sketch parameters and retrieval performance (MAP). Sketch indexes used in this experiment each contain statistics for all n-grams $(1 \leq n \leq 4)$ in the collection. Other retrieval metrics (nDCG@20 and P@20) were also evaluated, and similar trends were observed.

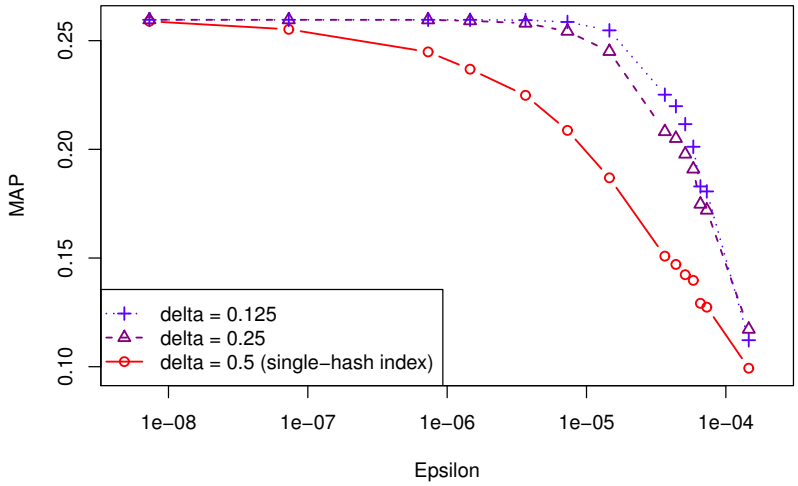
Figure 9 shows that retrieval performance (MAP) degrades as the ϵ parameter is increased and the sketch table width decreases correspondingly. We can see that for each value of δ, there is a threshold value of ϵ, below which retrieval effectiveness is identical to the oracle. A summary of these threshold parameter settings is shown in Table V. It is important to observe that the sketch parameters only need to grow sub-linearly in the size of the collection. GOV2 is almost 1,000 times longer than Robust-04 (as measured by collection length), and we observe that sketch parameters grow only sub-linearly, even while ensuring no change in retrieval effectiveness. Sketches of 3 rows are observed not to dramatically change the threshold settings. As such, we focus on the 2 row sketch in the following experiments.

It is important to note that the n1-4 retrieval model used here is not the only retrieval model that can use sketch indexes to improve retrieval efficiency. There are a variety of models that more selectively extract key concepts and other linguistic features that contain 2 or more terms from queries [Bendersky and Croft 2008; Maxwell and Croft 2013; Park et al. 2011; Xue and Croft 2010]. The efficiency of each of these models could benefit from direct access to the n-gram statistics provided by the sketch index. We use the n1-4 model in this study, as it provides similar retrieval performance to a commonly used benchmark, SDM, without requiring term proximity statistics to be computed from positional data. These experiments demonstrate that larger n-gram-based features for retrieval models can be efficiently computed using this structure.

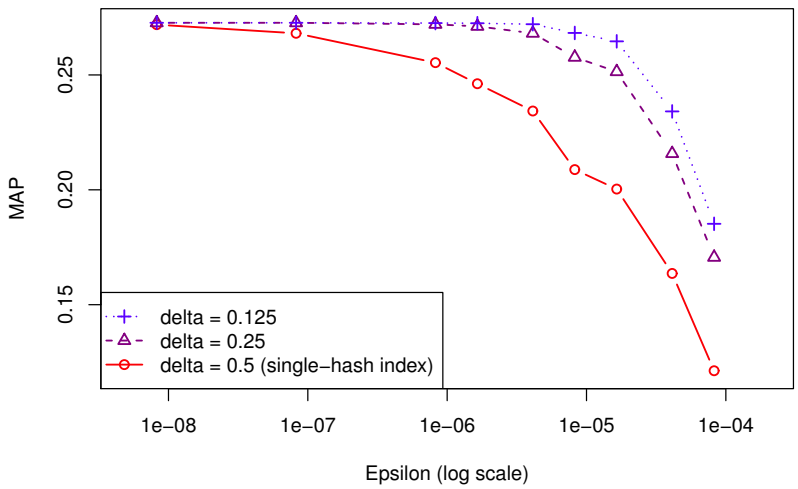
5.4. Memory and Disk Space Requirements

In this section memory and disk space requirements are evaluated for sketch indexes over a variety of sketch parameters and across different collections. We then compare disk and memory requirements to benchmark index structures.

Figure 10 shows memory requirements for a range of sketch parameters. Figure 11 shows the specific memory requirements of sketch indexes that maintain accurate re-



(a) n1-4 model, Robust-04



(b) n1-4 model, GOV2

Fig. 9. Retrieval effectiveness using sketch indexes measured using MAP, varying the (ϵ, δ) parameters, for each collection. Note that the sketch index where $\delta = 0.5$ is equivalent to a single-hash index, $\lceil \log(1/0.5) \rceil = 1$. Each data point is the average retrieval performance from 10 sketch index instances.

Table V. Sketch parameters and the corresponding sketch table sizes. Observed retrieval effectiveness (MAP) for each of the parameter settings in this table is within 1% of observed retrieval effectiveness for the `n1-4` retrieval model, using oracle-tuned parameters.

Collection	Delta	Epsilon	Sketch Width	Sketch Depth
Robust-04	Single-hashed	$7.3 \cdot 10^{-8}$	$27,457,393$	1
Robust-04	0.25	$3.6 \cdot 10^{-6}$	$554,752$	2
Robust-04	0.125	$1.4 \cdot 10^{-5}$	$143,067$	3
GOV2	Single-hashed	$8.1 \cdot 10^{-9}$	$247,116,529$	1
GOV2	0.25	$1.6 \cdot 10^{-6}$	$1,235,582$	2
GOV2	0.125	$8.1 \cdot 10^{-6}$	$247,116$	3

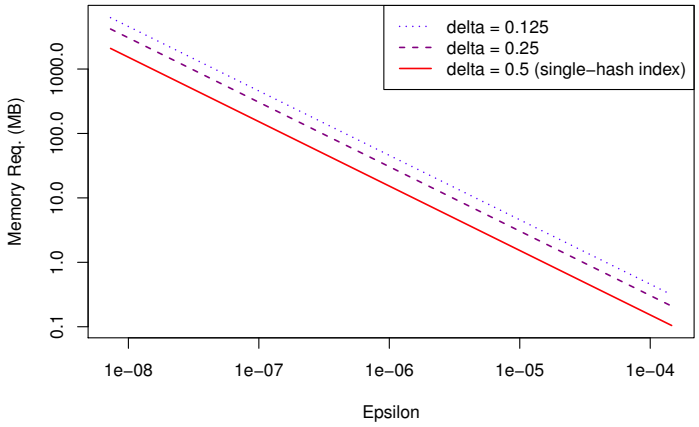


Fig. 10. Memory requirements for a range of sketch index parameters. Note both x and y axes are in log scale.

trieval effectiveness for the Robust-04 and GOV2 collections (see Table V). We observe that the memory requirements for sketch indexes grows much slower than the single hash index structure. If each sketch cell stores an 8 Byte file offset for the postings data, the memory requirement for a sketch index of all 1-to-4-grams in the GOV2 collection is just 19 MB. Over the same collection, using similar 8 Byte offset values, the single-hashed index requires almost 2 GB of RAM. This data shows that the sketch index is able to scale to large collections without introducing unreasonable memory requirements.

Figure 12 shows the disk requirements of sketch indexes on the Robust-04 collection for a range of sketch parameters. This data clearly shows that naive use of sketch indexes can result in inefficient use of disk resources. Figures 13 and 14 show disk requirements for sketch indexes compared to the disk requirements of the baseline index structures for the Robust-04 and GOV2 collections. This data shows that the sketch index uses significantly less space than the full index, but more than each of the other baseline methods. The vocabulary data for a 2-row sketch index is less than 0.01% of the disk requirements of the vocabulary data for a full index of 1-to-4-grams extracted from the GOV2 collection. However, the postings data stored in the same sketch index grows by a factor of 1.2 over the postings data stored in the full index.

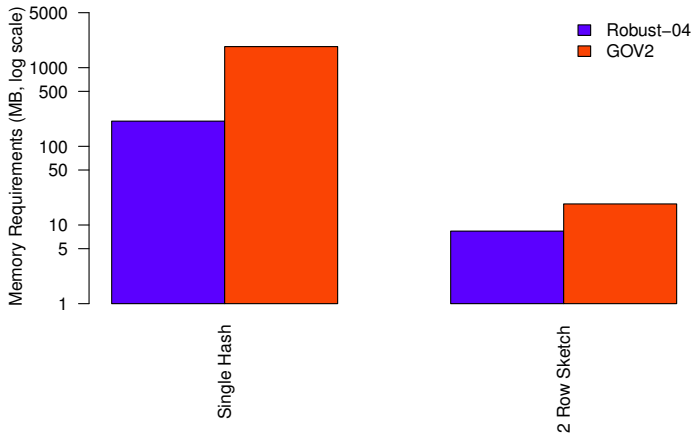


Fig. 11. Memory requirements for single hash structure vs a 2 row sketch index, using parameters specified in Table V. This graph assumes that each cell in the hash tables is an 8 Byte file offset to the posting list data. Note y axis is in log scale.

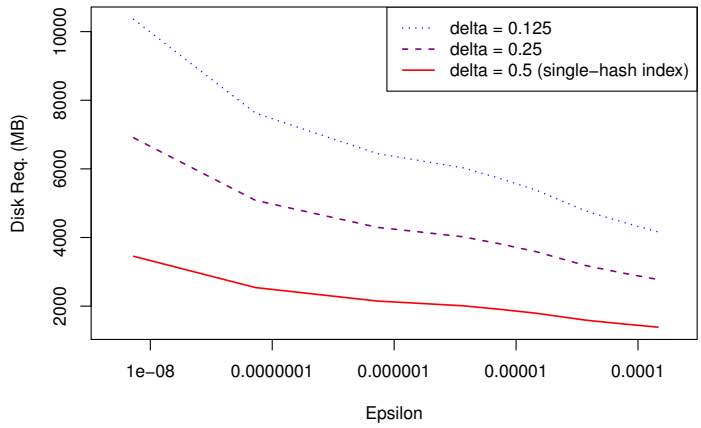


Fig. 12. Disk requirements for sketch indexes of the Robust-04 collection. Note the x axes is in log scale.

This implies that the sketch index is most effective when storing the vocabulary data is a large cost of the inverted index, as in an inverted index of term dependencies.

5.5. Retrieval Efficiency

We now evaluate the retrieval efficiency of our statistical estimator relative to the other index structures for n-gram queries. For this experiment, we sample queries from the AOL query log. The AOL query log consists of over 20 million unique web queries that users submitted to the AOL search engine in 2006. In these experiments, we execute queries on indexes of the ClueWeb-B collection. By using a web collection

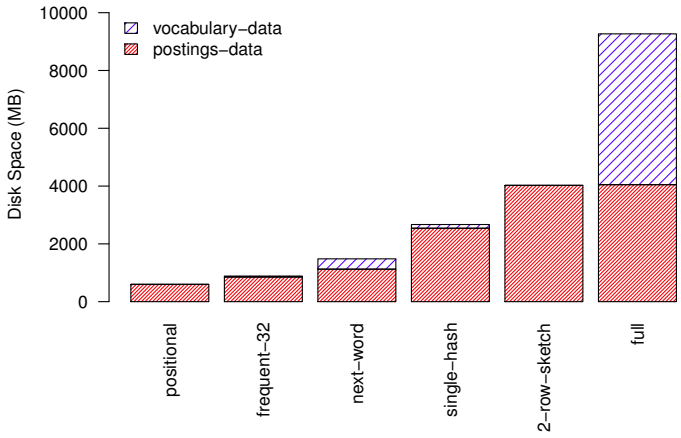


Fig. 13. Disk requirements for sketch indexes of the Robust-04 collection. Sketch index parameters are selected such that retrieval effectiveness is not compromised (see Table V).

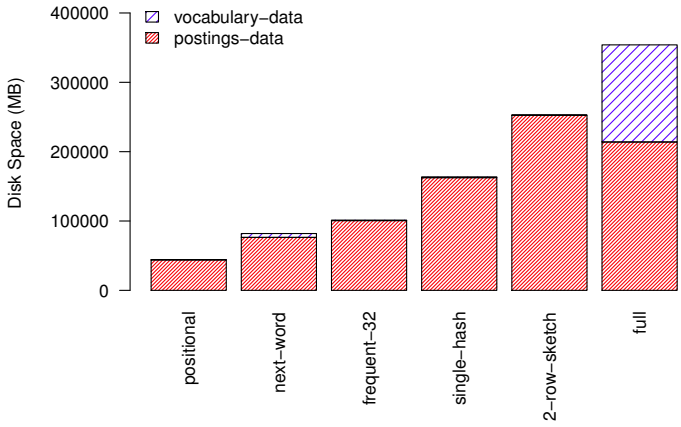


Fig. 14. Disk requirements for sketch indexes of the GOV2 collection. Sketch index parameters are selected such that retrieval effectiveness is not compromised (see Table V).

to target the queries, no query translation methods [Webber and Moffat 2005] are required. We omit the single-hash index from this experiment, as it operates identically to the full index structure.

The first 90% of the time-ordered query log is used to create the query log cache index structure. Test queries are sampled from the remaining 10% of the log. From this subset of the query log, we uniformly at random sample $10,000$ n-grams extracted from queries for each size, $n \in \{1, 2, 3, 4, 5\}$. The n-grams extracted from this query log represent a random sample of query features that are required to be computed for the n-gram based retrieval model used in Section 5.3. We note that this sampling technique

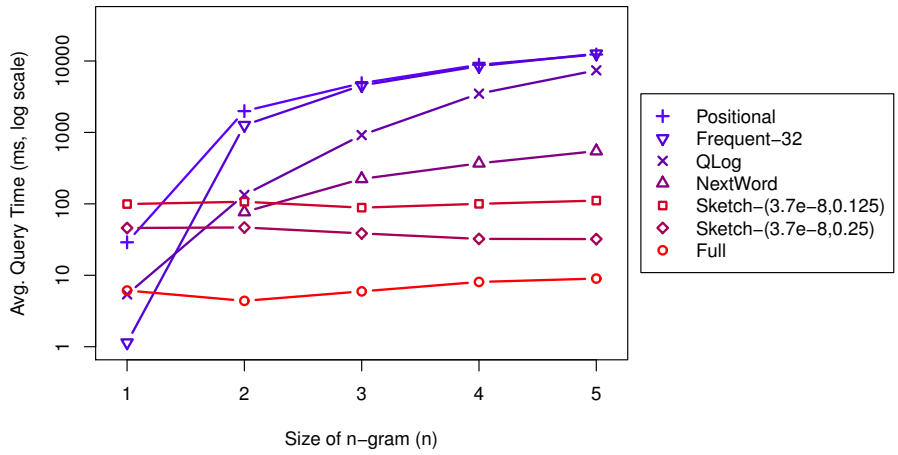


Fig. 15. Query processing times for indexes over the ClueWeb-B collection. Each data point is the average of 10 runs of 10,000 phrase queries.

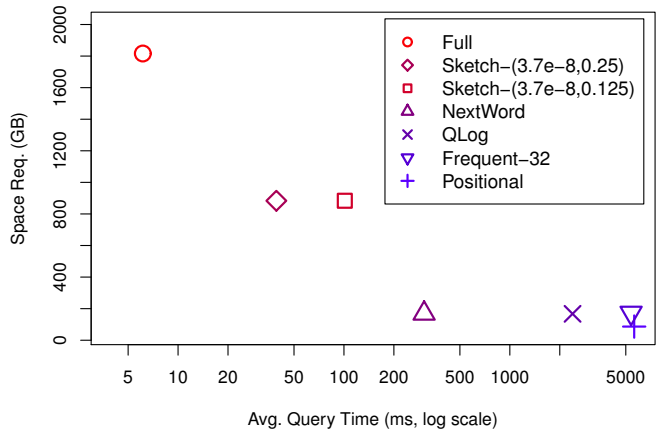


Fig. 16. Query processing time versus space usage. Query processing time as a function of index space requirements. Each data point is the average of 10 runs of 10,000 of n-gram queries. The size of the query for each data point is indicated on the graph.

would also be appropriate for several other types of retrieval models. Several linguistic and machine learned techniques segment or classify lists of terms extracted from a query into potentially valuable phrases, windows, n-grams, and other dependent sets of terms [Bendersky and Croft 2008; Bergsma and Wang 2007; Shi and Nie 2010]. These retrieval models each require collection statistics for all candidate n-grams or candidate query term sets. Therefore, each of these models requires that index struc-

tures provide access to statistics for any n-gram that may be queried, including the ability to efficiently determine if the n-gram occurs in the collection.

The processing speed for each index structure over a each size of n-gram is measured as the average of 5 timed runs of the corresponding sample of queries. To ensure the order of extracted n-grams does not affect the results, the order of each run is randomized. The retrieval system is initialized for each experiment by running a randomly selected sub-sample of $2{,}000$ queries. This process ensures that a portion of the index data is held in memory-based file buffers, as it would be in a live retrieval system.

Figure 15 shows query processing time as the length of the query increases for each index structure. Note that times shown in this graph are displayed in log scale. All data points in the graph are significantly different from each of the other index structures, $\alpha = 0.05$ for all pairs using the Fisher randomization test. This graph shows that the query processing time of the sketch index data structure is significantly faster than the positional, frequent, query-log and next-word indexes. Unlike position-based indexes, we can see that the sketch index is scalable in the length of the n-gram, since as n increases, the time to process n-gram queries does not increase with n.

The full index processes an average 4-gram in around 10 ms, while the sketch index processes the same average 4-gram in 33 ms. The other index structures (positional, frequent and next-word indexes) all process longer n-grams between 1 to 3 orders of magnitude slower on average than the sketch index structure. In particular, sketch indexes can be over 400 times faster than a positional index, and 15 times faster than next-word indexes, for processing 5-gram queries.

Figure 16 shows the trade-off between query processing speed and space usage. Data shown is the total space requirements to process 1-to-5-gram queries for each index structure, and the average time to process all of the sampled queries used in the timed experiments above. Query processing times are shown in log scale. Data structures that provide the best trade-off will approach the origin - that is the query processing time, and space requirements are minimized. This graph clearly demonstrates that our n-gram statistical estimator offers a new and effective trade-off between space usage and query efficiency compared with all other baselines.

6. RELATED AND FUTURE WORK

One of the key benefits of the sketch index structure examined here is that the size and count of indexed items is only weakly correlated with the space requirements of the index. This structure could be an effective structure to use when indexing more general term dependencies, including the window-based proximity features used in several term dependency retrieval models [Metzler and Croft 2005; Büttcher et al. 2006; Peng et al. 2007; Bendersky et al. 2010]. System space restrictions in our experimental configuration have made the construction of full indexes of proximity-based term dependencies for comparison infeasible at the current time. In future work, we intend to investigate the application of this index structure to proximity-based term dependencies.

Sketch indexes provide efficient access to n-gram features for information retrieval. In this work, we proposed retrieval models that make use of all n-grams extracted from an input query. There are several other information retrieval models and frameworks that make use of n-gram features. In particular, linguistic and machine learning techniques can be applied to identify various subsets of all possible n-gram features [Bergsma and Wang 2007; Bendersky and Croft 2008; Guo et al. 2008]. Retrieval models based on these subsets have shown the potential to improve retrieval effectiveness over models that use all possible n-grams of a particular size or pattern. Future work will include the investigation of the effect of sketch indexes on other retrieval models that make use of n-gram features.

It is also important to consider how sketch indexes may be used in different query processing models. MaxScore [Turtle and Flood 1995] is a well understood algorithm to reduce the number of documents processed during the evaluation of a query. It operates by estimating the maximum contribution from each of the query components, and using this information to short circuit the scoring of each document, as compared to the current top k documents. WAND, and related improvements [Broder et al. 2003; Ding and Suel 2011] present an approach that only scores a document when the total score is estimated to be higher than some threshold.

In order to operate efficiently, both of these algorithms require the estimation of the upper and lower bounds for each query component contribution. Recently, several new approaches to improving the estimation of the bounds for term dependency contributions have been proposed [Tonellotto et al. 2010; Macdonald et al. 2011]. Sketch indexes can provide direct access to a tight upper bound on term dependency statistics. Therefore, the use of this index structure has the potential to dramatically improve the effectiveness of each of these query processing algorithms over complex retrieval models. In future work, we intend to quantify any benefit gained from sketch indexes of term dependencies on the processing of term dependency based retrieval models.

Sketch indexes can also be applied to multi-phase query processing models. The cascade ranking model [Wang et al. 2011] incorporates the processing cost, in addition to an estimate of the score contribution for each query feature when learning each of the cascade's ranking functions. As Wang et al. [2011] suggest, the learned query processing model can delay the processing of slower query features until the document set has been significantly reduced.

It is also possible to consider a simple two-phase query processing model. The first phase scores each document in the collection using a bag-of-words retrieval model. The top k documents are returned from this phase. The second phase scores each of the the top k documents according to a more complex ranking function that includes term dependency features. The output of the second phase is returned.

In both of these multi-phase query processing models, fast access to term dependency statistics, as provided by sketch indexes, would allow term dependency features to be used in the initial ranking function. This may allow a smaller initial set of documents to be processed first and thus the total query processing time can be further reduced. The query processing time of the second and subsequent passes could also be improved using sketch indexes.

Interestingly, phrase queries can also be resolved efficiently using a variety of suffix-based indexing data structures collectively referred to as *self-indexes* [Navarro and Mäkinen 2007]. These indexing structures have attractive worst case efficiency bounds when doing "grep-like" occurrence counting in text. Fariña et al. [2012] show how to extend these indexing structures to term-based alphabets. However, the basic self-indexing framework does not directly address the *document listing problem* whereby a listing of the documents containing the search pattern in some frequency ordering is required. Muthukrishnan [2002] provided the first bounded approach to these and other related counting problems using a *document array*. Subsequent research has steadily progressed the time and space efficiency of top-k document retrieval using a single search pattern (term or phrase) [Sadakane 2007; Hon et al. 2009; Culpepper et al. 2010; Patil et al. 2011; Hon et al. 2012].

Unfortunately, the body of work surrounding top-k document retrieval using self-indexes focuses primarily on singleton pattern querying in order to derive the best possible efficiency bounds, and all use character-based instead of term-based vocabulary representations which often results in indexes that are 2 to 3 times larger than the text collection being indexed, all of which must be maintained in memory. Culpepper et al. [2011] investigated the viability of using a self-indexing configuration for

multi-term bag-of-words querying using a BM25 similarity computation instead of frequency-based ordering. Culpepper et al. [2012] extended these bag-of-words querying capabilities to include term-based indexes and the pre-computation enhancements of Hon et al. [2009] and Navarro and Valenzuela [2012]. They show that term-based self-indexes with a variety of auxiliary data structures to support ranked document retrieval are competitive with classic inverted indexes in both effectiveness and efficiency.

Despite the advances in self-indexing, the approach is still hampered by two problems: the indexes must be entirely in memory; and index construction requires full suffix array construction as an intermediate step. Suffix array construction is notoriously memory hungry, requiring around $9 \cdot |C|$ in-memory space for large collections [Puglisi et al. 2007]. In summary, self-indexing approaches for ranked document retrieval are a promising and active area of research, but current methods are limited by the amount of physical RAM available, which translates well to only modest-sized document collections in the IR domain. A variety of in-memory inverted indexing methods also exist [Strohman and Croft 2007; Transier and Sanders 2010; Fontoura et al. 2011], some of which attempt to selectively include phrasal components directly within the index [Transier and Sanders 2008]. While all of these indexing approaches provide compelling efficiency gains and can be constructed using significantly less memory than current suffix-based approaches, physical memory limitations still bound the size of collection that can be supported and were therefore not considered in this study.

Goyal et al. [2012] and Goyal and Daumé [2011] have proposed various sketch structures for use in a variety of natural language processing (NLP) tasks. An important distinction between these NLP tasks, and the information retrieval (IR) tasks discussed in this study, is that these NLP tasks require collection level statistics to compute word-pair association measures, such as point-wise mutual information, and log-likelihood ratios. However, IR systems require document level statistics for use in document scoring functions. Similar to results observed in our study, Goyal et al. [2012] observe new trade-offs between space requirements and estimation accuracy using sketch structures. For example, they are able to sketch 75 million word pairs in around 80 MB of memory, with minimal error [Goyal et al. 2012].

7. CONCLUSION

In this paper, we have investigated the problem of accurately estimating document and collection level n-gram statistics in large data collections. Existing solutions for this problem either require large amounts of disk space, or are inefficient for query processing in practice. We have presented a novel approach to estimating n-gram statistics for information retrieval tasks. By using frequency sketching techniques developed for data streaming applications, we can accurately estimate collection and document level statistics, and provide an attractive trade-off between space and relative error. Furthermore, we show how to bound the space usage of the data structure. Importantly, the number of distinct n-grams stored in the sketch is logarithmically linked to the size of the sketch, allowing us to scale up to very large collections, as empirically observed in experiments focusing on memory space requirements.

We have demonstrated that our approach is efficient in both time and space requirements, and can provide a low error rate for all of the statistics being estimated. Empirically, we have shown that the sketch index can reduce the space requirements of the vocabulary component of an index of 1-to-4-grams extracted from the GOV2 collection to less than 0.01% of the requirements of an equivalent full index. We have shown that sketch indexes can process queries considerably faster than both positional indexes, and next-word indexes. Unlike frequent indexes and query-log cache approaches, our method does not require an auxiliary index to calculate statistics for unseen or infre-

quent n-grams. This new index representation provides an attractive alternative to other state-of-the-art approaches depending on n-gram statistics for retrieval tasks.

The sketch index data structure provides a new and useful trade-off between query processing time and space requirements for n-gram queries. Importantly, we also have shown that this index structure is scalable in both query processing time and space requirements for the size of the queried n-gram, n, and for the size of the collection, $|C|$.

Finally, the index structures described encourage the exploration of n-grams as query features. We have initiated this exploration through a simple n-gram based retrieval model in this paper. Our retrieval model can be executed efficiently using sketch index structures. We have empirically shown these models to be significantly more effective than the query likelihood retrieval model, and not perform significantly differently than the sequential dependence model. Furthermore, using the sketch index data structure, n-gram retrieval models can be executed more efficiently than positional index-based implementations of the sequential dependence model.

REFERENCES

N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58:137–147, 1999.

M . Bendersky and W. B. Croft. Discovering key concepts in verbose queries. In *Proc. of the 31st ACM SIGIR*, pages 491–498, 2008.

M. Bendersky and W. B. Croft. Modeling higher-order term dependencies in information retrieval using query hypergraphs. In *Proc. of the 35th ACM SIGIR*, pages 941–950, 2012.

M. Bendersky, D. Metzler, and W. B. Croft. Learning Concept Importance using a Weighted Dependence Model. In *Proc. of the 3rd ACM WSDM*, pages 31–40, 2010.

S. Bergsma and Q.I. Wang. Learning noun phrase query segmentation. In *Proc. of the EMNLP-CoNLL*, pages 819–826, 2007.

R. Berinde, G. Cormode, P. Indyk, and M. J. Strauss. Space-optimal heavy hitters with strong error bounds. In *Proc. of the 28th ACM PODS*, pages 157–166, 2009.

Y. Bernstein and J. Zobel. Accurate discovery of co-derivative documents via duplicate text detection. *Journal of Information Systems*, pages 595–609, 2006.

A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In *Proc. of the 12th CIKM*, pages 426–434, 2003.

A. Broschart and R. Schenkel. High-performance processing of text queries with tunable pruned term and term pair indexes. *ACM TOIS*, 30(1):5–1–5–32, February 2012.

S. Büttcher, C. Clarke, and B. Lushman. Term proximity scoring for ad-hoc retrieval on very large text collections. In *Proc. of the 29th ACM SIGIR*, pages 621–622, 2006.

S. Büttcher, C. Clarke, and G.V. Cormack. *Information Retrieval: Implementing and evaluating search engines*. The MIT Press, 2010.

M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. *Proc. of the 28th ICALP*, pages 693–703, 2002.

G. Cormode and M. Hadjieleftheriou. Finding frequent items in data streams. *PVLDB*, 1(2):1530–1541, 2008.

G. Cormode and M. Hadjieleftheriou. Methods for finding frequent items in data streams. *The VLDB Journal*, 19(1):3–20, 2010.

G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. In *Proc. of the 6th LATIN*, pages 29–38, 2004.

G. Cormode and S. Muthukrishnan. Summarizing and mining skewed data streams. In *Proc. of the 5th SDM*, 2005a.

G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005b.

W. B. Croft, J. Callan, and http://lemurproject.org/contrib.php. The lemur project. http://www.lemurproject.org/, 2001-2012. Lemur Toolkit, Indri, Galago, ClueWeb09.

J. S. Culpepper, G. Navarro, S. J. Puglisi, and A. Turpin. Top-k ranked document search in general text databases. In *ESA, Part II*, LNCS 6347, pages 194–205, 2010.

J. S. Culpepper, M. Yasukawa, and F. Scholer. Language independent ranked retrieval with NeWT. In *ADCS*, pages 18–25, December 2011.

J. S. Culpepper, M. Petri, and F. Scholer. Efficient in-memory top-k document retrieval. In *SIGIR*, pages 225–234, 2012.

S. Ding and T. Suel. Faster top-k document retrieval using block-max indexes. In *Proc. of the 34th ACM SIGIR*, pages 993–1002, 2011.

C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *SIGCOMM Comput. Commun. Rev.*, pages 323–336, 2002.

A. Fariña, N. R. Brisaboa, G. Navarro, F. Claude, Á. S. Places, and E. Rodríguez. Word-based self-indexes for natural language text. *ACM TOIS*, 30(1):1–1–1–33, 2012.

M. Fontoura, V. Josifovski, J. Liu, S. Venkatesan, X. Zhu, and J. Zien. Evaluation strategies for top-k queries over memory-resident inverted indexes. *Proceedings of the VLDB Endowment*, 4(12):1213–1224, 2011.

S. Ganguly, M. N. Garofalakis, and R. Rastogi. Processing data-stream join aggregates using skimmed sketches. In *Proc. of the 9th EDBT*, pages 569–586, 2004.

Amit Goyal and Hal Daumé, III. Approximate scalable bounded space sketch for large data nlp. In *Proc. of the 2011 EMNLP-CoNLL*, pages 250–261, 2011.

Amit Goyal, Hal Daumé, III, and Graham Cormode. Sketch algorithms for estimating point queries in nlp. In *Proc. of the 2012 EMNLP-CoNLL*, pages 1093–1103, 2012.

J. Guo, G. Xu, H. Li, and X. Cheng. A unified and discriminative model for query refinement. In *Proc. of the 31st ACM SIGIR*, pages 379–386, 2008.

O.A. Hamid, B. Behzadi, S. Christoph, and M. Henzinger. Detecting the origin of text segments efficiently. In *Proc. of the 18th WWW*, pages 61–70, 2009.

B. He, Huang J. X., and Zhou X. Modeling term proximity for probabilistic information retrieval models. *Journal of Information Sciences*, 181(14):3017 – 3031, 2011.

W.-K. Hon, R. Shah, and J. S. Vitter. Space-efficient framework for top-k string retrieval problems. In *FOCS*, pages 713–722, 2009.

W.-K. Hon, R. Shah, and S. V. Thankachan. Towards an optimal space-and-query-time index for top-k document retrieval. In *CPM*, pages 673–184, 2012.

S. Huston, A. Moffat, and W. B. Croft. Efficient indexing of repeated n-grams. In *Proc. of the 4th ACM WSDM*, pages 127–136, 2011.

S. Huston, J. S. Culpepper, and W. B. Croft. Sketch-based indexing of n-words. In *Proc. of the 21st ACM CIKM*, pages 1864–1868, 2012.

W. Lu, S. Robertson, and A. MacFarlane. Field-weighted xml retrieval based on bm25. In *Proc. of the 4th INEX*, pages 161–171, 2006.

C. Macdonald, I. Ounis, and N. Tonellotto. Upper-bound approximations for dynamic pruning. *ACM Transactions on Information Systems*, 29(4):17:1–17:28, December 2011.

K. Tamsin Maxwell and W. Bruce Croft. Compact query term selection using topically related text. In *Proc. of the 2011 ACM SIGIR*, pages 583–592, 2013.

R. M. C. McCreadie, C. Macdonald, and I. Ounis. On single-pass indexing with Map-Reduce. In *Proc. of the 32nd ACM SIGIR*, 2009.

D. Metzler and W.B. Croft. A Markov random field model for term dependencies. In *Proc. of the 28th ACM SIGIR*, pages 472–479, 2005.

S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *SODA*, pages 657–666, 2002.
S. Muthukrishnan. *Data streams: Algorithms and applications*. Now Publishers, 2005.
G. Navarro and V. Mäkinen. Compressed full-text indexes. *ACM Computing Surveys*, 39(1):2–1 – 2–61, 2007.
G. Navarro and D. Valenzuela. Space-efficient top-k document retrieval. In *SEA*, LNCS 7276, pages 307–319, 2012.
R. Ozcan, I.S. Altingovde, B.B. Cambazoglu, F.P. Junqueira, and O. Ulusoy. A five-level static cache architecture for web search engines. *Information Processing & Management*, 2011.
J. H. Park, W. B. Croft, and D. A. Smith. A quasi-synchronous dependence model for information retrieval. In *Proc. of the 20th ACM CIKM*, pages 17–26, 2011.
M. Patil, S. V. Thankachan, R. Shah, W.-K. Hon, J. S. Vitter, and S. Chandrasekaran. Inverted indexes for phrases and strings. In *SIGIR*, pages 555–564, 2011.
J. Peng, C. Macdonald, B. He, V. Plachouras, and I. Ounis. Incorporating Term Dependency in the DFR Framework. In *Proc. of the 30th ACM SIGIR*, pages 843–844, 2007.
J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *Proc. of the 21st ACM SIGIR*, pages 275–281, 1998.
S. J. Puglisi, W. F. Smyth, and A. H. Turpin. A taxonomy of suffix array construction algorithms. *ACM Computing Surveys*, 39(2):4.1–4.31, 2007.
K. Sadakane. Succinct data structures for flexible text retrieval systems. *J. Discr. Alg.*, 5(1):12–22, 2007.
J. Seo and W. B. Croft. Local text reuse detection. In *Proc. of the 31st ACM SIGIR*, pages 571–578, 2008.
Lixin Shi and Jian-Yun Nie. Using various term dependencies according to their utilities. In *Proc. of the 19th ACM CIKM*, pages 1493–1496, 2010.
T. Strohman and W. B. Croft. Efficient document retieval in main memory. In *SIGIR*, pages 175–182, 2007.
N. Tonellotto, C. Macdonald, and I. Ounis. Efficient dynamic pruning with proximity support. *Large-Scale Distributed Systems for Information Retrieval*, pages 33–37, 2010.
F. Transier and P. Sanders. Out of the box phrase indexing. In *SPIRE*, LNCS 5820, pages 200–211, 2008.
F. Transier and P. Sanders. Engineering basic algorithms of an in-memory text search engine. *ACM TOIS*, 29(1):2–1–2–37, 2010.
H. Turtle and J. Flood. Query evaluation: strategies and optimizations. *Information Processing & Management*, 31:831–850, November 1995.
L. Wang, J. Lin, and D. Metzler. A cascade ranking model for efficient ranked retrieval. In *Proc. of the 34th ACM SIGIR*, pages 105–114, 2011.
W. Webber and A. Moffat. In search of reliable retrieval experiments. In *Proc. of the 10th ADCS*, pages 26–33, 2005.
H. E. Williams, J. Zobel, and P. Anderson. What's next? Index structures for efficient phrase querying. In *Proc. of the 10th ADC*, pages 141–152, 1999.
H. E. Williams, J. Zobel, and P. Anderson. Fast phrase querying with combined indexes. *ACM TOIS*, 22(4):573–594, 2004.
I.H. Witten, A. Moffat, and T.C. Bell. *Managing Gigabytes: Compressing and indexing documents and images*. Morgan Kaufmann, 1999.
X. Xue and W.B. Croft. Representing queries as distributions. In *Proc. of SIGIR Workshop on Query Representation and Understanding*, pages 9–12, 2010.