

When Close Enough Is Good Enough: Approximate Positional Indexes for Efficient Ranked Retrieval

Tamer Elsayed
King Abdullah University of
Science and Technology
Thuwal, Saudi Arabia
tamer.elsayedaly@kaust.edu.sa

Jimmy Lin
The iSchool
University of Maryland
College Park, MD
jimmylin@umd.edu

Donald Metzler
Information Sciences Institute
Univ. of Southern California
Marina del Rey, CA
metzler@isi.edu

ABSTRACT

Previous research has shown that features based on term proximity are important for effective retrieval. However, they incur substantial costs in terms of larger inverted indexes and slower query execution times as compared to term-based features. This paper explores whether term proximity features based on *approximate* term positions are as effective as those based on *exact* term positions. We introduce the novel notion of *approximate positional indexes* based on dividing documents into coarse-grained *buckets* and recording term positions with respect to those buckets. We propose different approaches to defining the buckets and compactly encoding bucket ids. In the context of linear ranking functions, experimental results show that features based on approximate term positions are able to achieve effectiveness comparable to exact term positions, but with smaller indexes and faster query evaluation.

Categories and Subject Descriptors: H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval

General Terms: Algorithms, Performance

Keywords: term proximity, efficiency, learning to rank

1. INTRODUCTION

In information retrieval, efficiency (speed) and effectiveness (quality) are often in tension, since good relevance signals are frequently expensive to compute during ranking. For example, term proximity features are good predictors of relevance but are costly to compute compared to term-based features. This paper proposes to reduce the cost of computing term proximity features by introducing *approximate positional indexes*, a novel method of encoding term positions in the payload of postings lists.

There are two aspects of cost associated with term proximity features. The first is index size: in order for a ranking model to support term proximity features, term positions must be encoded in postings payloads. The second is the speed of computing term proximity features at retrieval

time. For example, to determine if two query terms occur within w terms of each other during ranking, term positions must be decoded and checked. This is computationally expensive compared to term-based features.

The central hypothesis explored in this paper is the following: are term proximity features based on approximate term positions as effective as those based on exact term positions? The answer turns out to be yes, in that a fuzzy notion of term proximity yields ranking functions, trained in a learning to rank framework, that are comparable in effectiveness to similar models that use exact term proximity features. The basic insight is that instead of recording exact term positions, we divide documents into coarse-grained buckets and record in which bucket the term occurs. We explore different approaches to defining and compressing the buckets. Our best variant algorithm yields substantial reductions in both index size and query evaluation time.

2. RELATED WORK

It is widely recognized that term proximity features contribute to improvements in retrieval effectiveness over term-based features (i.e., “bag of words”) alone. Early studies date back to the 1980s [4], and there has been no shortage of studies on the topic since then [3, 9, 1]. More recently, Markov random fields [7] provide a theoretically-motivated framework for understanding and reasoning about term dependencies using undirected graphical models.

Of course, to support term proximity features, it is necessary to store term position information in the postings. Existing compression techniques for encoding positional information [10] can be considered lossless, and they achieve different tradeoffs between compression rate and decoding speed. Our work tackles an orthogonal issue related to *representing* term positions, where we can take advantage of previous work on compression.

One common approach to faster query evaluation is index pruning [2]. Experiments have shown that it is possible to discard postings that are unlikely to contribute much to a document’s score, thus yielding a smaller index—translating into faster ranking. Index pruning differs from our work in that we retain all postings, but encode positional information in a more compact (but lossy) manner.

3. RETRIEVAL MODEL

Our work uses linear feature-based ranking functions [7] of the form $S(Q, D) = \sum_j \lambda_j f_j(Q, D)$, where Q is a query, D is a document, $f_j(Q, D)$ is a feature function, and λ_j is the weight assigned to feature j . More specifically, this work

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.

CIKM’11, October 24–28, 2011, Glasgow, Scotland, UK.

Copyright 2011 ACM 978-1-4503-0717-8/11/10 ...\$10.00.

uses a particular instantiation of the Markov random field model known as the *sequential dependence* (SD) model [7]:

$$S(Q, D) = \lambda_T \sum_{q \in Q} f_T(q, D) + \lambda_U \sum_{q_j, q_{j+1} \in Q} f_O(q_j, q_{j+1}, D) + \lambda_O \sum_{q_j, q_{j+1} \in Q} f_U(q_j, q_{j+1}, D)$$

where f_T is defined over query unigrams, and f_O and f_U are defined over query bigrams. The latter two are term proximity features: f_O corresponds to phrase matches and f_U corresponds to query term co-occurrence within a particular span. The λ 's are parameters that need to be learned. The original formulation of the SD model defines features in terms of language modeling query likelihoods with Dirichlet smoothing [7]. Here, we adopt an alternative formulation based on BM25 scores [8]. In practice, we have discovered that BM25 scores are slightly more effective, but the differences are not statistically significant in most cases.

3.1 Approximate Positions

The central hypothesis we explore in this paper is whether term proximity features based on approximate term positions are as effective as those based on exact term positions. Our basic idea is that instead of storing exact term positions in the index, we divide each document into coarser-grained *buckets* and record the id (i.e., the sequential number) of the bucket in which the term occurs.

We explored two different approaches to defining buckets:

Variable-Width Buckets. We can divide each document into a fixed number of buckets b , where each bucket represents a span of term positions within that document. Under this scheme, buckets in different documents will have different widths, which means that long documents receive a coarser-grained treatment than short documents.

Fixed-Width Buckets. As an alternative, we can fix the bucket size w and let the number of buckets in a document vary with the document length. This in effect selects a single level of granularity at which to model term positions in the entire collection.

We explored two different approaches for coding bucket ids:

Integer Coding. We treat bucket ids as an array of integers and use standard gap-based compression techniques to code them (using γ codes [10]).

Bit-Array Coding. Bucket positions are coded in a bit-array. That is, the k^{th} bit of the bit-array is set to one if the k^{th} bucket contains the term, or zero otherwise. This scheme works well if the number of buckets is equal to a machine word in the variable-width bucket scheme. Furthermore, as an optimization, checking for term proximity (e.g., if two terms are found in the same bucket) translates into efficient bitwise operations.

Note that in both cases, we only code the presence or absence of terms in a bucket, but not the number of terms that occur in that bucket. In other words, the approximate positions do not capture term frequency.

3.2 Proximity Features

To parallel the f_O and f_U term proximity features in the sequential dependence model, we introduce the three following approximate proximity operators. Each operator is defined over an adjacent pair of query terms (i.e., a query bigram), just as the original features are:

- **SAMEBUCKET**(q_j, q_{j+1}, D) matches any positional bucket that contains both q_j and q_{j+1} .
- **ORDADJBUCKETS**(q_j, q_{j+1}, D) matches if q_j occurs in a bucket at position i and q_{j+1} occurs in a bucket at position $i + 1$. That is, the two terms must appear in order across adjacent bins.
- **UNORDADJBUCKETS**(q_j, q_{j+1}, D) matches if q_j occurs in a bucket at position i and q_{j+1} occurs in the bucket at position $i - 1$ or $i + 1$, i.e., adjacent buckets. Unlike the previous operator, which requires the two terms to appear in the correct order, this operator allows for unordered matches.

These features share the same functional form (i.e., weighted using BM25) as their exact positional counterparts. To estimate the parameters of our linear models, we employ greedy feature selection—a simple but effective learning to rank approach that directly optimizes for the retrieval metric of interest (e.g., MAP) described by Metzler [6].

4. EXPERIMENTAL EVALUATION

We implemented the approximate positional indexes described above with Ivory [5]. Inverted indexes adopt a standard compression scheme: document id gaps are compressed with Golomb codes and term frequencies are coded with γ codes. For baselines, we created an index with no positional information and an index with exact term positional information. In the latter index, term positions are converted into gap differences and compressed with γ codes.

Indexes were constructed using Hadoop, and retrieval experiments were performed on a server with dual Intel Xeon quad-core processors (E5620 2.4GHz), 64GB RAM, and six 2TB 7.2K RPM SATA drives in RAID-6 configuration (running Red Hat Linux). All experiments used a single thread and performed retrieval on a single, monolithic index (i.e., no document partitioning), returning 1000 hits per query.

For evaluation, we used two TREC web collections: Gov2 (topics 701–850) and the first English segment of ClueWeb09 (topics 1–50). In each case, queries were split sequentially into training and test sets of equal size. All parameters were estimated on the training set and all results reported on the test sets. Retrieval effectiveness is measured in terms of mean average precision (MAP) and precision at 10 (P@10). A one-side paired t -test (with $p = 0.05$) was used to determine the statistical significance of differences in the metrics.

4.1 Baselines and Approximate Models

We used the following baselines as points of comparison: classic **BM25**; **BM25-*sd***, the basic sequential dependence model with BM25 scores and default parameters; and **BM25-*sd-t***, a variant of **BM25-*sd*** for which the model parameters were trained on a per-collection basis. This allowed the model to adapt to collection-specific characteristics, rather than using the same default parameters for all collections as is done with **BM25-*sd***. Our BM25 runs used

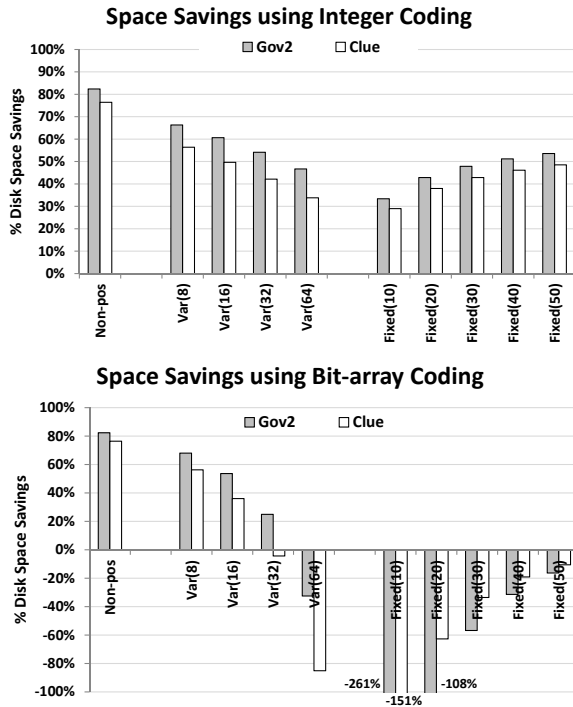


Figure 1: Percentage reduction in index size with respect to exact positional indexes. Non-positional indexes (left-most bars) provide upper bounds.

the non-positional indexes since they do not require any positional information for ranking documents, while the other two baselines used the exact positional indexes, which were needed to compute the exact term proximity features.

In our experiments, we used nine variants of approximate positional indexes that have different bucket types: variable-width buckets with number of buckets $b = 8, 16, 32, 64$ —each denoted as $\text{Var}(b)$ —and fixed-width buckets with bucket size $w = 10, 20, 30, 40, 50$ —each denoted as $\text{Fixed}(w)$.

4.2 Results

We now describe our experimental analysis of approximate positional indexes and proximity features in terms of index size, retrieval effectiveness, and query evaluation speed.

4.2.1 Index Size

Figure 1 compares the sizes of the approximate positional indexes with respect to exact positional indexes. Bars plot space savings, so higher is better. For reference, the left-most bars show the space savings of non-positional indexes (i.e., discarding positional information altogether), which ranges between 76% to 82%. This serves as an upper bound.

The two collections exhibit similar patterns. As expected, we notice that as the number of buckets per document (b) increases in the variable-width models or as the bucket size (w) decreases in the fixed-width models, the index size increases (and thus the space savings decrease). In some cases, the bit-array coding indexes are actually bigger than exact positional indexes.

The figures also show a clear advantage to the integer coding scheme in terms of index size (space savings range from 29% to 66%) compared with the bit-array scheme.

	MAP		P@10	
	Gov2	Clue	Gov2	Clue
Baselines				
BM25	0.3162	0.2251	0.5787	0.3800
BM25- <i>sd</i>	0.3505	0.2312	0.5920	0.3800
BM25- <i>sd-t</i>	0.3447	0.2340	0.6107	0.3840
Variable width				
$b = 8$	0.3223	<u>0.2252*</u>	0.5813	<u>0.3800*</u>
$b = 16$	0.3229	0.1933	0.5813*	0.2560
$b = 32$	0.3234	<u>0.2250*</u>	0.5813*	<u>0.3800*</u>
$b = 64$	0.3220	0.2067	<u>0.5880*</u>	0.3080
Fixed width				
$w = 10$	0.3376	<u>0.2256*</u>	<u>0.5920*</u>	<u>0.3760*</u>
$w = 20$	0.3426*	<u>0.2261*</u>	0.5787*	<u>0.3800*</u>
$w = 30$	0.3385	0.2247*	<u>0.5933*</u>	<u>0.3760*</u>
$w = 40$	0.3382	<u>0.2250*</u>	0.5867*	<u>0.3800*</u>
$w = 50$	0.3346	0.2243*	0.5853*	<u>0.3800*</u>

Table 1: Retrieval effectiveness. Bold values indicate stat. sig. over BM25; star-annotated and underlined values indicate *n.s.* compared to BM25-*sd* and BM25-*sd-t*, respectively.

4.2.2 Retrieval Effectiveness

Table 1 presents retrieval effectiveness in terms of MAP and P@10 for the test collections. We annotated the effectiveness values to indicate results of significance testing between the approximate positional models and the baselines: bold values indicate statistically-significant improvements over BM25, and star-annotated and underlined values indicate *no* statistically-significant difference compared with BM25-*sd* and BM25-*sd-t*, respectively.

MAP results reported in the table indicate that, in the Gov2 collection, all approximate positional models exhibit significant improvements over BM25. Moreover, all of the fixed-width models have no statistically-significant differences with BM25-*sd-t*. While the MAP results are generally in favor of our approximate models for Gov2, the Clue results are more mixed. The table shows that the baselines aren’t very different from each other, and this might explain the lack of significant improvement of our approximate models over BM25.

Although the P@10 results indicate that our approximate models are not significantly better than BM25, the models are also not statistically different from the other baselines either. This suggests that term proximity features are better at enhancing precision at lower ranks, since they consistently improve MAP.

Considering the absolute MAP and P@10 scores in addition to the results of the statistical significance tests, we see that the fixed-width models are consistently better than the variable-width models. We hypothesize that variable-width models introduce a great deal of variance (or noise) into the proximity features (such as SAMEBUCKET) because window sizes vary greatly depending on the document length (i.e., very short for short documents, very long for long documents)—or more accurately, on the *variance* of document lengths within a collection. If all documents have the same length (i.e., zero variance), then $\text{Var}(b)$ would behave exactly the same as $\text{Fixed}(avgdl/w)$. However, as the variance increases, the definition of proximity features becomes less “consistent”, which ultimately yields noisy feature values and thus less effective results.

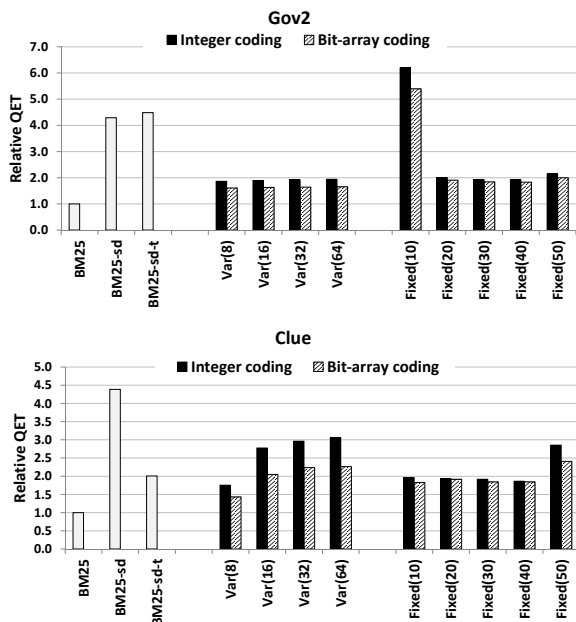


Figure 2: Average normalized query execution time of baselines and different approximations.

The results of the fixed-width models show that, as the width of the bucket decreases, the effectiveness improves. This is expected since smaller buckets yield more accurate approximate positional information. Overall, the fixed-width models Fixed(10) and Fixed(20) are the best among all the approximate positional models.

4.2.3 Retrieval Efficiency

Figure 2 shows the average (over three trials) query execution time (QET) for each collection across a variety of approximate positional indexing strategies. The QET figures are normalized with respect to baseline BM25, which uses the non-positional indexes, and thus the bars show how much slower (in relative terms) the other conditions are.

Comparing integer and bit-array coding, we notice that the latter is generally faster but not substantially. BM25-*sd-t* is comparable to BM25-*sd* on Gov2. For Clue, the learned BM25-*sd-t* model uses fewer features than BM25-*sd*, simply as an artifact of the greedy feature selection algorithm, and as a result it is substantially slower.

Rather than exhaustively explain all the results, we focus on the Fixed(10) and Fixed(20) models, which yielded consistently strong effectiveness (from the previous section). For Gov2, we observe a substantial reduction in QET with the Fixed(20) model (−57%) compared to both variants of the *sd* models. In the Fixed(10) model, the greedy feature selection algorithm uses more features, and thus is substantially slower. The results for Clue show that both Fixed(10) and Fixed(20) are significantly faster (−57%) than BM25-*sd* and exhibit comparable efficiency (−5%) to BM25-*sd-t*.

Overall, the Fixed(20) model not only exhibits effectiveness that is comparable to the BM25-*sd* model, but also strong efficiency characteristics—a substantial reduction in index size and query execution time. This particular setting nicely balances a small index footprint, good effectiveness, and fast query execution.

5. CONCLUSIONS

Term proximity features are useful for a variety of search tasks, especially those that deal with large collections such as web search. However, full positional indexes can consume a large amount of space and positionally-aware retrieval models that utilize such indexes are considerably slower than their non-positional counterparts. To help minimize the time and space costs of positional indexes, we proposed a novel indexing strategy called approximate positional indexes and a corresponding retrieval model. These indexes break documents into fixed or variable width buckets and encode which buckets a term occurs in, either using integer or bit-array coding.

Our experimental results, carried out over two TREC web collections, show that our proposed methods are able to achieve comparable effectiveness to term-proximity models based on full positional indexes, but with smaller indexes and faster query execution times. Therefore, the proposed methodology provides system designers with a viable “middle ground” in the efficiency-effectiveness tradeoff space to better balance quality, time, and space.

6. ACKNOWLEDGMENTS

This work has been supported by NSF awards IIS-0836560, IIS-0916043, and CCF-1018625. Any opinions expressed are the authors’. The second author is grateful to Esther and Kiri for their loving support and dedicates this work to Joshua and Jacob.

7. REFERENCES

- [1] S. Büttcher, C. Clarke, and B. Lushman. Term proximity scoring for ad-hoc retrieval on very large text collections. In *SIGIR*, 2006.
- [2] D. Carmel, D. Cohen, R. Fagin, E. Farchi, M. Herscovici, Y. Maarek, and A. Soffer. Static index pruning for information retrieval systems. In *SIGIR*, 2001.
- [3] W. Croft, H. Turtle, and D. Lewis. The use of phrases and structured queries in information retrieval. In *SIGIR*, 1991.
- [4] J. Fagan. Experiments in automatic phrase indexing for document retrieval: A comparison of syntactic and non-syntactic methods. Technical report, Cornell University, 1987.
- [5] J. Lin, D. Metzler, T. Elsayed, and L. Wang. Of Ivory and Smurfs: Loxodontan MapReduce experiments for web search. In *TREC*, 2009.
- [6] D. Metzler. Automatic feature selection in the Markov random field model for information retrieval. In *CIKM*, 2007.
- [7] D. Metzler and W. Croft. A Markov random field model for term dependencies. In *SIGIR*, 2005.
- [8] S. Robertson, S. Walker, M. Hancock-Beaulieu, M. Gatford, and A. Payne. Okapi at TREC-4. In *TREC*, 1995.
- [9] M. Srikanth and R. Srihari. Biterm language models for document retrieval. In *SIGIR*, 2002.
- [10] I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, 1999.