A Hybrid BitFunnel and Partitioned Elias-Fano Inverted Index

Xinyu Liu
liuxy@nbjl.nankai.edu.cn
College of CS, TJ Key Lab of NDST
Nankai University
Tianjin, China

Zhaohua Zhang
zhangzhaohua@nbjl.nankai.edu.cn
College of CS, TJ Key Lab of NDST
Nankai University
Tianjin, China

Rebecca J. Stones
becky@nbjl.nankai.edu.cn
College of CS, TJ Key Lab of NDST
Nankai University
Tianjin, China

Yusen Li
liyusen@nbjl.nankai.edu.cn
College of CS, TJ Key Lab of NDST
Nankai University
Tianjin, China

Gang Wang*
wgzwp@nbjl.nankai.edu.cn
College of CS, TJ Key Lab of NDST
Nankai University
Tianjin, China

Xiaoguang Liu*
liuxg@nbjl.nankai.edu.cn
College of CS, TJ Key Lab of NDST
Nankai University
Tianjin, China

ABSTRACT

Search engines encounter a time vs. space trade-off: search responsiveness (i.e., a short query response time) comes at the cost of increased index storage. We propose a hybrid method which uses both (a) the recently published mapping-matrix-style index BitFunnel (BF) for search efficiency, and (b) the state-of-the-art Partitioned Elias-Fano (PEF) inverted-index compression method. We use this proposed hybrid method to minimize time while satisfying a fixed space constraint, and to minimize space while satisfying a fixed time constraint. Each document is stored using either BF or PEF, and we use a local search strategy to find an approximately optimal BF-PEF partition. Since performing full experiments on each candidate BF-PEF partition is impractically slow, we use a regression model to predict the time and space costs resulting from candidate partitions (space accuracy 97.6%; time accuracy 95.2%). Compared with a hybrid mathematical index (Ottaviano et al., 2015), the time cost is reduced by up to 47% without significantly exceeding its size. Compared with three mathematical encoding methods, the hybrid BF-PEF index allows performing list intersection between around 16% to 76% faster (without significantly increasing the index size). Compared with BF, the index size is reduced by 45% while maintaining an intersection time comparable to that of BF.

CCS CONCEPTS

• **Information systems → Information retrieval query processing**; **Search engine indexing**; **Search index compression**.

KEYWORDS

Partitioned Elias-Fano; BitFunnel; hybrid index; compression; query processing

ACM Reference Format:
Xinyu Liu, Zhaohua Zhang, Rebecca J. Stones, Yusen Li, Gang Wang, and Xiaoguang Liu. 2019. A Hybrid BitFunnel and Partitioned Elias-Fano Inverted Index. In *Proceedings of the 2019 World Wide Web Conference (WWW '19), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3308558.3313553

WWW '19, May 13–17, 2019, San Francisco, CA, USA

ACM ISBN 978-1-4503-6674-8/19/05.
https://doi.org/10.1145/3308558.3313553

1 INTRODUCTION

Search engines face a trade-off between space and time [1, 7, 24]: if we compress the underlying index, we reduce the system's query responsiveness. The most common search-engine index structure is the *inverted index* where for each term, we store a *posting list* containing the docIDs of documents containing that term [6, 18].

The Partitioned Elias-Fano (PEF) [23] inverted-index compression method is a state-of-the-art method, which uses a shortest-path algorithm [8] to dynamically partition the posting list into variable-length blocks with clustered docIDs within each block; Elias-Fano encoding [12, 13] is used blockwise for random access.

Microsoft's Bing search engine uses a newly described index structure BitFunnel (BF) [15], which is a bitmap-like data structure: terms map to one or a few rows, while columns correspond to individual documents. If a term appears in the document, the corresponding bit positions are set to 1. In this way, intersection is performed using simple bitwise AND operations. However, the BitFunnel is much larger than the corresponding compressed inverted index.

In this paper, we combine the advantages of BF (intersection efficiency) and PEF (space efficiency), and propose a hybrid index. We partition the document corpus into two parts, with one part allocated to BF and the other allocated to PEF. The choice of partition affects the time-space trade-off, and can be chosen to suit the search engine's requirements. We envisage choosing this partition so that either a time threshold or a size threshold is not exceeded.

The major contributions of this paper are as follows:

(1) We use the local search method to find an approximate optimal partition of the document corpus while satisfying a pre-specified threshold condition.
(2) For a large dataset, it is impractical to repeatedly estimate the index's size and intersection time experimentally, so we develop and utilize a machine learning approach to estimate the index size and intersection time of the hybrid index with a given partition.
(3) We implement and conduct experiments on the proposed hybrid index. More specifically:

- we assess three machine learning methods (GBRT [14], SVR [11], and RF [5]) for predicting the space and time costs of the resulting BF-PEF hybrid index (given an assignment of the documents to BF and PEF), and ultimately select support vector regression (SVR) which consistently outperforms the other two methods in terms of error rate;
- we investigate the convergence of the local search process on the size and average intersection time of proposed hybrid index under various thresholds; and
- we compare the space-time trade-off of the proposed hybrid index against several mathematical encoders, along with a hybrid mathematical index.

This paper is structured as follows. In Section 2 we describe pilot experiments which motivate the design decisions throughout the paper. Section 3 gives a minimalistic background to the relevant material. Section 4 reviews alternative hybrid index methods, and their relevance to the proposed hybrid BF-PEF method. A detailed description of the method used to generate an approximately optimal BF-PEF partition is given in Section 5. Experimental results are given in Section 6. We conclude with some future research ideas in Section 7.

2 PILOT EXPERIMENTS

In this section, we describe some preliminary experiments for some seemingly natural approaches to choosing a *BF-PEF partition*, i.e., a partition of the documents into two parts, one stored using BF and the other stored using PEF. These results motivate the subsequent use of machine learning and local search in optimizing the partition. These pilot experiments use the GOV2 document corpus and MillionQuery set, described in Section 6.1. We also describe some alternative methods.

With N documents there are 2^N possible BF-PEF partitions (and N is large in most search engines, e.g., billions of documents). Exhaustive inspection of all 2^N partitions is unrealistic, and thus the goal is to find an approximately optimal partition. Further, it is practically far too time-consuming to generate a hybrid index and perform experiments on it (like in Table 1) for each partition we wish to test. Moreover, there is no single "best" partition, rather we seek a partition which is optimized according to criteria determined by the search engine. For example, if the search engine has X TB of memory available for index, we cannot violate this hard restriction. Similarly, a search-engine operator may have their own constraints regarding the system's responsiveness.

To summarize, we have three major design constraints:

(1) we cannot find an optimal BF-PEF partition, but only an approximation,
(2) it is impractical to perform fully fledged experiments for a large number of candidate BF-PEF partitions, and
(3) we anticipate hard restrictions on the index size and intersection time imposed e.g. by hardware or the operator.

An alternative straightforward approach is to randomly assign documents to either BF or PEF; we randomly choose a proportion, and randomly assign that proportion of documents to BF and the remainder are assigned to PEF, and we tabulate experimental results of partition in Table 1. While this randomized hybrid method combines the advantages of BF and PEF, the trade-off is unimpressive: we incur around 75% of the large space cost of BF while being almost 1.6 times as slow as BF. Thus, we need to use a more intelligent way of partitioning the documents.

Table 1: The index size (space) and average intersection time (time) on BF, PEF, and a BF-PEF hybrid method with documents randomly assigned to BF and PEF.

	BF	BF-PEF (rand.)	PEF
space (GB)	13.40	9.76	3.25
time (ms)	1.45	2.29	3.19

We also consider incorporating a bitmap compression method, such as EWAH [17]; we tested this approach and observed a minor space reduction of less than 5% when compressing the BitFunnel index of GOV2. We attribute this behavior to BitFunnel's high density of 1s (usually 0.1 to 0.2), which results in short all-0 and all-1 subsequences, leading to poor compression.

Recently, Liu et al. [19] proposed a dictionary-based compression method for BitFunnel with additional false positives. On the GOV2 dataset, we found that the compression rate of this method is around 30%, but its intersection time increases by about 50%, and its false positive rate is doubled which is consistent with the results in that paper. We thus abandon BF compression approaches.

A possible way to compress an inverted index is via mathematical encoding [2, 3, 25, 30]. However, PEF achieves a higher compression rate while simultaneously allowing efficient list intersection.

We investigate how the BF-PEF partition is influenced by some *basic features*:

- *URL*, the lexicographic order of the document URLs;
- *DL*, the length of the documents without duplicate terms (i.e., the number of unique terms in document);
- *DF*, the mean document frequency of terms in the document;
- *dispersivity*, which measures the lack of clustering of a document in the inverted index, as per f_8 in Section 5.1.1; and
- *TF-IDF*, a mean TF-IDF calculated as per f_{11} in Section 5.1.1.

We perform some toy experiments involving these basic factors from which we glean insights into how we might design a fully fledged hybrid BF-PEF index.

For each basic feature f, we create the set S_f of documents in GOV2 which have the highest 10% of that f's possible values. We create an artificial document corpus $\mathcal{D}_f$ for each f by randomly selecting 100 000 documents in S_f. For each $\mathcal{D}_f$, we generate the BF and PEF indexes, and compare their size and average intersection time. In order to compare features with each other, we measure the size according to

$$s_f := \frac{\text{size of index generated from } \mathcal{D}_f}{\text{total number of docIDs in posting lists in } \mathcal{D}_f}$$

and the time according to

$$t_f := \frac{\text{average intersection time on index generated from } \mathcal{D}_f}{\text{total number of docIDs in posting lists in } \mathcal{D}_f}.$$

We also perform the same computations for the lowest 10% of the possible f values, and distinguish between the two cases as "high" and "low". We plot the results in Figure 1.

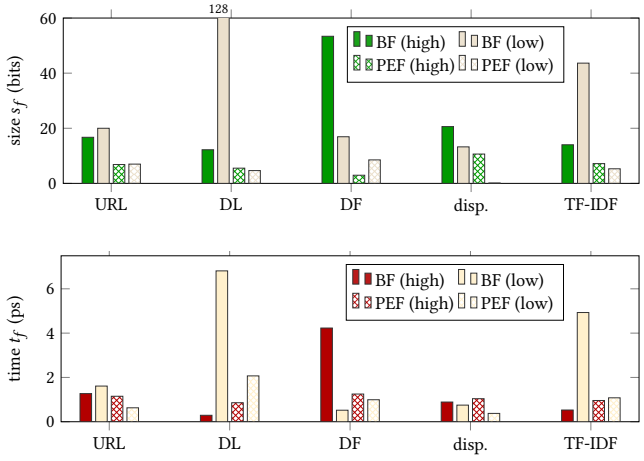

Figure 1: The size s_f and time t_f of the BF index and PEF index constructed for the toy document corpus $\mathcal{D}_f$ for the basic feature $f \in \{\textbf{URL}, \textbf{DL}, \textbf{DF}, \textbf{disp.}, \textbf{TF-IDF}\}$.

To investigate the sensitivity to the random document corpus $\mathcal{D}_f$, we also repeat this computation 5 times, and observe negligible difference between trials.

We make the following observations:

Observation 1. *Multiple basic features significantly influence the size and intersection time of the BF and PEF indexes.*

For example, for BF we observe that a greater DF leads to a greater s_{DF}, which we attribute posting list density: a dense posting list maps to unshared rows in the BitFunnel. We also see that for PEF, a low dispersivity results in a small index size and intersection time, indicating that docID clustering benefits PEF.

Observation 2. *For either the BF or PEF component of a hybrid BF-PEF index, a high (or low) value of a single feature may be simultaneously advantageous for size (resp. time) while disadvantageous for time (resp. size).*

For example, for PEF a greater DF leads to a smaller index, but a greater intersection time. We expect this is because a high DF implies that most of the PEF-compressed documents contain high-frequency terms in the corpus, resulting in longer posting lists being compressed (and decompressed during list intersection).

Observation 3. *A high (or low) value of a single feature may be simultaneously advantageous for size and time for both the BF and PEF components of a hybrid BF-PEF index.*

For example, a lower dispersivity results in a smaller $s_{\text{disp.}}$ and $t_{\text{disp.}}$ for both BF and PEF. Since PEF is designed to utilize clustering, it is apparent why a low dispersivity results in reduced $s_{\text{disp.}}$ and $t_{\text{disp.}}$ for the PEF index. For BF, a lower dispersivity also results in a smaller $s_{\text{disp.}}$ and $t_{\text{disp.}}$, although it is not apparent why.

Some of these observations are predictable, while others are not; and some of these observations result in a quandary, where there is not an apparent best way to assign documents to BF or PEF. In the subsequent experiments (Section 6; see in particular Section 6.5 for feature-based experiments), we incorporate additional features, and anticipate varying and unpredictable effects on BF-PEF indexes. Thus, we expect it is not straightforward to find a near-optimal BF-PEF partition using simple threshold methods.

This unpredictable feature behavior motivates the use of a machine learning regression approach, which predicts the index size and intersection time of hybrid BF-PEF indexes based on multiple document features. In particular, this is the motivation behind the feature vectors in Section 5.1.2.

3 BACKGROUND

In search engines, the most common index structure is the *inverted index* [6, 18], a database where each *term* t has a corresponding *posting list* ℓ_t, which stores an ordered list of the docIDs that the term t appears in, namely

$$\ell[t] = \langle \text{docID}_1, \text{docID}_2, \ldots, \text{docID}_n \rangle.$$

For conjunctive queries, the relevant documents for a query $\{t_i\}$ are given by the *list intersection* $\cap_i \ell[t_i]$, which is computed after a query is made (i.e., "online") and thus needs to be performed efficiently.

Ordinarily we use some form of compression to reduce the size of the inverted index. To avoid decompressing entire posting lists during query processing, we typically divide the posting lists into blocks which are individually compressed and decompressed (to facilitate random access).

Differing from the traditional fixed-length block encoding, Partitioned Elias-Fano (PEF) [23] encoding uses dynamic block sizes in order to make use of clustering in the inverted index. The authors of [23] also described a method for obtaining an approximate optimal block partition, where each block uses Elias-Fano encoding [12, 13].

BitFunnel [15] is a bitmap-like mapping matrix which is based on a Bloom filter [4, 28]. In a BitFunnel, each term is mapped to one or a few rows, with each column representing a document. If a term appears in a document, the corresponding entry (or entries) is set to "1" (otherwise "0"). For conjunctive queries, to obtain the documents containing all the query terms, we perform an AND operation on the rows that the query terms map to. A "1" in the result usually indicates a document which contains all the query terms, although the Bloom filter admits the possibility of false positives.

There are some properties of BitFunnel which affect its operation, but only indirectly affect the results in this paper. Specifically: (a) given a density, BitFunnel generates an index with a low false positive rate while approximately achieving that density; (b) BitFunnel usually has a false positive rate of less than 5%. False positives are not hugely problematic in practice, since e.g. they might be automatically eliminated during top-K ranking; (c) documents are divided into shards according to their length, and BitFunnel use "frequency-conscious" signatures (i.e., multiplexing) to reduce the storage space; and (d) to speed up the intersection process, rows are ascribed a rank, whereby high-ranked rows are shorter and are expanded (self-concatenated) before being intersected with low-ranked rows. In experiments in Section 6, we use BitFunnel's default settings, including setting the density to 0.15 and the number of shards to 11.

Compared with the list intersection approach, a BitFunnel index is much larger (e.g. 4.1 times larger in Table 1) but also much faster to query (2.2 times faster in Table 1). This results in an "all your

eggs (documents) in one basket (index)" trade-off, and this paper is about finding an in-between, hybrid approach.

Motivated by both hardware limits and operator-set limits, we consider two setups:

(1) minimizing the average intersection time subject to a fixed maximum index size, and
(2) minimizing the index size subject to a fixed maximum average intersection time.

4 RELATED WORK

To reduce the storage overhead, prior research proposed a variety of mathematical encoding methods for inverted indexes, such as PForDelta (PFD) [30], binary interpolative coding [21], Vbyte [25] (Varint-G8IU [27] exploiting SIMD operations), Simple9 and Simple16 [2, 3], and so on. We use OptPFD [29] (an improved version of PFD), binary interpolative coding, and Varint-G8IU as baselines in Section 6.4.

These mathematical encoding methods minimize the number of bits used to represent the docIDs, thereby achieving compression. However, the constant-size block division of the posting list is incompatible with clustering. To illustrate, if we split the toy posting list

$$\langle 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 21, 22, 23, 24, 30\rangle$$

into length-4 blocks of docIDs

$$\langle 1, 2, 3, 4\rangle, \langle 5, 6, 7, 8\rangle, \langle 9, 10, 20, 21\rangle, \langle 22, 23, 24, 30\rangle$$

we break apart the clusters of docIDs. To utilize this clustering among docIDs, and EF encoding for random access, in the proposed approach, we use PEF to compress one part of the document corpus. For the documents not compressed using PEF, we use BitFunnel. This proposed hybrid method deviates from the usual approach of using a single index structure or compression method for the whole corpus.

We also remark that the documents assigned to BF do not overlap with the documents assigned to PEF, thus the BF results and PEF results do not require a subsequent time-consuming "merging" operation.

We take advantage of the bitwise operation of BF for time efficiency and the dynamic block partitioning of PEF for space efficiency. PEF's compression is comparable to the best mathematical encoding method we consider in this work, while also significantly outperforming them in terms of intersection time (see Section 6.4).

Some hybrid indexes have been previously proposed, combining different types of indexes to obtain a smaller index or a shorter query time. Some methods partition the corpus by term [9, 20] where different type of index (or server) is selected for each posting list. Unlike their approaches, the proposed method partitions the document corpus by document. In this way, the partitioning may change the distribution of the inverted index (e.g., the length and the clustering of posting lists). We expect different document subsets adapt to different index type.

Ottaviano et al. [22] combined three mathematical encoding methods (PFD [29], binary interpolative coding [21], and Varint-G8IU [27]) to generate a hybrid index. Each block is assigned the compression method which minimizes the average query time through the greedy algorithm described in [26] while subject to a spatial threshold. Given a query log, Ottaviano et al. randomly sample a few blocks to generate the training set, and use a linear model to predict the decoding time of each block during query processing using this training set.

Unlike the proposed method, the method by Ottaviano et al. has a fixed block size, and does not take advantage of dynamic partitioning for inverted indexes like PEF. Moreover, it does not incorporate a bitmap-like index structure. We use [22] as a baseline (hybrid-PIV in Section 6.4).

Kane and Tompa [16] introduced "semi-bitvectors": the documents are grouped and reordered to increase the clustering in posting lists. They divide each posting list by a cut point into two parts: the denser front part for the bitvector and the remainder of the inverted index is compressed by PFD [30]. The cut point for each posting list is based on the bitvector density threshold F and the number of groups G: two user-defined parameters. In principle, we could apply a threshold on the intersection time or index size to the semi-bitvector method by repeatedly adjusting the parameters F and G and rebuild the index multiple times until the specified threshold is satisfied. However, this approach neither optimizes size nor time. In the proposed method, we use an iterative approach to find an approximately optimal BF-PEF partition; we minimize the intersection time while staying within a size threshold (or vise versa with "intersection time" and "size" switched).

A major structural difference is that the proposed method splits the document corpus into two parts, whereas Kane and Tompa's approach individually splits the posting lists. With semi-bitvectors, it is necessary to perform list intersection across two disparate index structures, which hinders parallelization and the application of Kane and Tompa's approach to e.g. distributed systems. Instead, partitioning the document corpus (as in the proposed approach) avoids this problem, since each partition can be intersected independently.

5 HYBRID BF-PEF PARTITION SELECTION

The proposed approach has three main components depicted in Figure 2:

- We use machine learning to develop regression models which compare BF-PEF partitions. We compare three different models, but find support vector regression to be the most suitable.
- Once a regression model is generated, we use a local search approach where we perturb the "current" partition (starting with an initialized partition), and replace it by the perturbed partition if the relevant regression model considers the perturbed partition better.
- We consider two ways of perturbing the current partition: one random, and one which prioritizes either PEF over BF or BF over PEF.

5.1 Prediction

5.1.1 Feature selection. We predict the size and intersection time of a hybrid index (arising from a BF-PEF partition) using machine learning. For an arbitrary document d, we have the following document features:

- the features f_1 and f_3 are respectively URL and DL, as used in the pilot experiments;
- the number f_2 of (not necessarily distinct) terms in d;

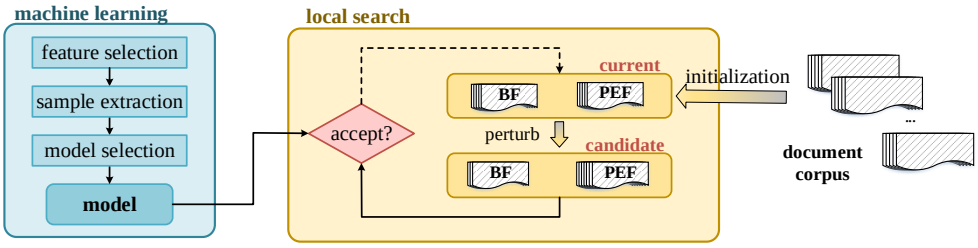

Figure 2: Flow chart for generating an approximately optimal BF-PEF partition. Machine learning is used to give a prediction model. The local search accepts a perturbed BF-PEF partition if it is predicted to outperform the current BF-PEF partition.

- the shardID f_4 of the document in the BF index, which varies in BitFunnel according to the number of unique terms in a document;
- the maximum f_5, minimum f_6, and mean f_7 of the document frequency among the terms t in the document (i.e., the length of the posting list for t);
- we define the *dispersivity* f_8 as

$$\operatorname*{avg}_{t\in d}\sqrt{\frac{1}{b-a+1}\sum_{j\in\{a,\ldots,b\}\setminus\{i\}}\frac{(\mathrm{docID}_j-\mathrm{docID}_i)^2}{\log_{10}(|j-i|+1)}}$$

  where i is the index for document d in the posting list ℓ_t, and $a=\max(1,i-32)$ and $b=\min(|\ell[t]|,i+32)$, as used in the pilot experiments;
- the maximum f_9, minimum f_{10}, and mean f_{11} of the TF-IDF score of the document d, restricted to the terms $t\in S$, where S is the smallest set of highest-frequency terms such that sum of the frequencies of terms in S accounts for 50% of the total frequency; this approach is inspired by [16] because high frequency terms usually play a more important role on index size and query time; and
- query count f_{12}, i.e., the number of query terms appearing in d.

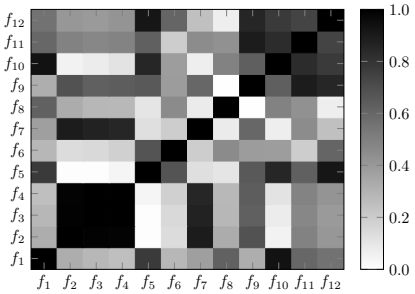

Figure 3: Feature correlation heat map.

We exclude some highly correlated features based on experiments much like the pilot experiments (Section 2). For each feature, we construct a vector of 8 values as in Figure 1. We normalize the feature values to $[0,1]$, then calculate the correlation [10] between each pair of features; we plot the results in Figure 3. We exclude the features with correlation exceeding 98%: document length f_2, document unique length f_3, and shardID f_4 have pairwise correlation exceeding 98%, so we exclude f_2 and f_4. While other pairs of features exhibit varying levels of correlation, there is virtually no meaningful distinction between f_2, f_3, and f_4, so we exclude two of them so as to not add undue weight. The remaining features, i.e., those in $\{f_1,f_3\}\cup\{f_5,f_6,\ldots,f_{12}\}$, are denoted $\hat{f}_1$ through $\hat{f}_{10}$.

5.1.2 Sample Extraction and Model Selection. In this section, we introduce how we define samples which we use in machine learning (to train the two prediction models). Generating a hybrid index based on a whole corpus is time consuming, so we construct a practical set of samples to train the model.

For the document corpus $\mathcal{D}$, for each $i\in\{1,\ldots,10\}$, we define $\mathcal{D}_i^{(\mathrm{low})}$ and $\mathcal{D}_i^{(\mathrm{high})}$ as the set of documents respectively with the least and greatest $\hat{f}_i$ values, both comprising approximately 25% of all documents. We define $\mathcal{D}_i^{(\mathrm{medium})}=\mathcal{D}\setminus(\mathcal{D}_i^{(\mathrm{low})}\cup\mathcal{D}_i^{(\mathrm{high})})$, which comprises about 50% of all documents. For the purpose of this paper (i.e., for use with the GOV2 dataset), we define the set of pairs

$$\mathcal{N}=\{(5,7.5),(7.5,10),(10,25),(25,50),(50,100),\\(100,150),(150,200),(200,250)\}\times 10^5,$$

so e.g. $(500\,000,750\,000)\in\mathcal{N}$, and so on.

Given a pair $(a,b)\in\mathcal{N}$ we randomly select a number $t\in\{0,\ldots,10\}$ and a random t-subset $F\subseteq\{1,\ldots,10\}$. Each $i\in F$ indexes feature $\hat{f}_i$, and we randomly select $\mathcal{D}_i\in\{\mathcal{D}_i^{(\mathrm{high})},\mathcal{D}_i^{(\mathrm{medium})},\mathcal{D}_i^{(\mathrm{low})}\}$, and for each document $d\in\mathcal{D}$, we assign it the score

$$\mathrm{score}(d):=|\{i\in F\colon d\in\mathcal{D}_i\}|\,. \tag{1}$$

We randomly choose $n\in\{a,\ldots,b-1\}$ and form a document corpus of size n consisting of the documents $d\in\mathcal{D}$ with the greatest score. If $t=0$, we randomly select any n documents from $\mathcal{D}$. Informally, we randomly choose a "high", "medium", or "low" feature condition for each of a random selection of features, and we choose n documents which satisfy the most conditions.

To reduce the time of sample extraction, we randomly choose pairs (a,b) from $\mathcal{N}$ so that lower-valued pairs have a greater probability of being selected. This gives a family $\mathcal{F}$ of document corpuses. We split $\mathcal{F}$ into training and validation sets.

We randomly choose two document corpuses $\mathcal{D}^{(\mathrm{BF})}$ and $\mathcal{D}^{(\mathrm{PEF})}$ from $\mathcal{F}$ to generate a partition (which might intersect, but we ignore this as we expect it has a negligible effect). We compute the average feature values for both of these corpuses, giving $\hat{f}_1^{(\mathrm{BF})}$ through

$\hat{f}_{10}^{(\mathrm{BF})}$ (from $\mathcal{D}^{(\mathrm{BF})}$) and $\hat{f}_1^{(\mathrm{PEF})}$ through $\hat{f}_{10}^{(\mathrm{PEF})}$ (from $\mathcal{D}^{(\mathrm{PEF})}$). We define the two sample vectors used for training: the *size vector*

$$\vec{\sigma}_{\mathrm{size}} = \Big\langle \hat{f}_1^{(\mathrm{BF})}, \ldots, \hat{f}_{10}^{(\mathrm{BF})}, |\mathcal{D}^{(\mathrm{BF})}|, \hat{f}_1^{(\mathrm{PEF})}, \ldots, \hat{f}_{10}^{(\mathrm{PEF})}, |\mathcal{D}^{(\mathrm{PEF})}|, s_{\mathrm{BF}} + s_{\mathrm{PEF}} \Big\rangle$$

and the *time vector*

$$\vec{\sigma}_{\mathrm{time}} = \Big\langle \hat{f}_1^{(\mathrm{BF})}, \ldots, \hat{f}_{10}^{(\mathrm{BF})}, |\mathcal{D}^{(\mathrm{BF})}|, \hat{f}_1^{(\mathrm{PEF})}, \ldots, \hat{f}_{10}^{(\mathrm{PEF})}, |\mathcal{D}^{(\mathrm{PEF})}|, t_{\mathrm{BF}} + t_{\mathrm{PEF}} \Big\rangle$$

where s_{BF} (resp. t_{BF}) is the size (resp. average intersection time) of the index formed when using BitFunnel on the corpus $\mathcal{D}^{(\mathrm{BF})}$ and s_{PEF} (resp. t_{PEF}) is the size (resp. average intersection time) of the index formed when using PEF on the corpus $\mathcal{D}^{(\mathrm{PEF})}$. Doing this repeatedly gives a set M_{size} of size vectors and a set M_{time} of time vectors.

The purpose of this elaborate generation process is to include a diverse selection of vectors in the training and validation sets, thereby helping the machine learning process to identify the (size and time) impact of the features.

For model selection, we use three common regression models: the gradient-boosted regression tree (GBRT) model [14], a support vector regression (SVR) model [11], and a random forest model [5] for training. Further, we use a validation set to tune the parameters. An experimental comparison of these models is given in Section 6.2.

Once the size and time regression models are generated, we apply these two models to the entire document corpus which assist local search to judge the "candidate" partition and the "current" partition. Noted that all the samples in test sets we use in experiments (see Section 6.2) are real BF-PEF partitions (of the whole document corpus, rather than artificial sets as for training and validation samples).

5.2 Partition

We use a local search approach to find an approximately optimal partition. We perturb the current partition, and accept the perturbed partition if the chosen model (derived via machine learning, as in Section 5.1) predicts the perturbed partition performs better (either in terms of size or time, depending on which setup is chosen). This is repeated some number of times; around 200 iterations seems sufficient with the present paper's experimental setup (see Section 6.3). The details are given in Algorithm 1.

There are two different methods for selecting the documents P in Algorithm 1 (Lines 3, 5, and 7):

(1) *Random selection* (used with probability 20%). Some subset F of the features is randomly selected. We select a random $n \in \{250\,000, \ldots, 1\,250\,000\}$ (n is the number of documents in P, account for 1% to 5% of the whole corpus), and choose the n documents with the highest score, as defined by eq. (1).

(2) *Priority selection* (used with probability 80%). This selection method is the same as random selection, except in calculating eq. (1), where we choose a collection of documents that are more friendly to the destination index. Specifically, we choose $\mathcal{D}_i$ as either "high" or "low" respectively whenever a greater or lesser value of the corresponding feature $\hat{f}_i$ benefits BF (resp. PEF) in terms of both size and time, and we refer to these features as *BF-friendly* features (resp. *PEF-friendly* features). When prioritizing BF, we randomly choose between "high" or "low" for non-BF-friendly features, and likewise for non-PEF-friendly features when prioritizing PEF.

Algorithm 1 Local search method

Input: Document corpus, the number of iterations T
Output: An approximately optimal partition

1: Randomly partition the document corpus into two parts $\mathcal{P} = \{\mathcal{P}^{(\mathrm{BF})}, \mathcal{P}^{(\mathrm{PEF})}\}$
2: **for** i from 1 to T **do**
3: Choose a subset $P \subseteq \mathcal{P}^{(\mathrm{BF})}$ ▷ two different methods
4: Replace the current partition $\mathcal{P}$ with the perturbed partition $\{\mathcal{P}^{(\mathrm{BF})} \setminus P, \mathcal{P}^{(\mathrm{PEF})} \cup P\}$ if the perturbed partition is predicted to perform better
5: Choose a subset $P \subseteq \mathcal{P}^{(\mathrm{PEF})}$ ▷ two different methods
6: Replace the current partition $\mathcal{P}$ with the perturbed partition $\{\mathcal{P}^{(\mathrm{BF})} \cup P, \mathcal{P}^{(\mathrm{PEF})} \setminus P\}$ if the perturbed partition is predicted to perform better
7: Choose a subset $P_1 \subseteq \mathcal{P}^{(\mathrm{BF})}$ and a subset $P_2 \subseteq \mathcal{P}^{(\mathrm{PEF})}$ ▷ two different methods
8: Replace the current partition $\mathcal{P}$ with the perturbed partition $\{(\mathcal{P}^{(\mathrm{BF})} \setminus P_1) \cup P_2, (\mathcal{P}^{(\mathrm{PEF})} \setminus P_2) \cup P_1\}$ if the perturbed partition is predicted to perform better
9: **end for**

For example, a lower dispersivity for PEF reduces both the index size and intersection time, and thus it is a PEF-friendly feature. In fact, dispersivity is also a BF-friendly feature. A non-example is document frequency for PEF: a higher value reduces the index size, while a lower value reduces the intersection time, so it is not PEF-friendly.

When using friendly features in Algorithm 1, when a subset of the documents are moved from one part to the other, "friendly" is defined according to the destination. Specifically, in Algorithm 1, we use PEF-friendly features in Line 3 and for P_1 in Line 7, and we use BF-friendly features in Line 5 and for P_2 in Line 7.

Suppose we apply a time threshold, i.e., an upper bound on the average intersection time. We choose between the current partition and the perturbed partition as follows:

(1) If the current partition and the perturbed partition both satisfy the time threshold, we choose the partition which minimizes index size.
(2) Otherwise, we choose the partition which minimizes time.

Above, both size and time are predicted using a machine-learning model. When instead applying a size threshold, we do the same thing with "time" and "size" swapped.

6 EXPERIMENTAL RESULTS

We experiment on the proposed hybrid BF-PEF index, which we abbreviate to *hybrid-BP*, under various circumstances. We explore the accuracy of the machine-learning regression models, the impact of the initial partition in the local search process, and the performance of hybrid-BP in terms of index size and average intersection time.

We compare hybrid-BP against the mathematical hybrid index proposed in [22], which we call *hybrid-PIV*; we also compare hybrid-BP against the mathematical encoders involved in hybrid-PIV. We also explore the feature distributions in hybrid-BP, and false positive rates.

6.1 Experiment Setup

Our experiments are performed using public GOV2 dataset, containing 25 205 183 documents with 6 220 848 172 postings. We remove extra information, such as HTML tags, while retaining the title and body parts. We do not remove any stop words. For query datasets, we use the public query sets (a) MillionQuery, containing 58 506 queries from the 2007, 2008, and 2009 TREC Million Query Tracks[1], and (b) TerabyteQuery, containing 99 322 queries from the 2006 TREC Terabyte Track[2]. In both query sets, the terms which do not exist in the GOV2 document set are filtered out, and the aforementioned query counts are after filtering. We do not remove the query that only have one term.

All experiments are run on a platform with 2× Xeon E5-2650 v4 CPUs at 2.20GHz with 512GB RAM; both CPUs have 12 physical cores (with 24 threads). The L1 instruction cache and data cache is 32KB, L2 cache is 256KB, and L3 cache is 30 720KB. All the programs are implemented in C++11 and complied with g++ version 5.4.0 with -O2 optimization. The operating system is Linux CentOS 6.5 with kernel version 2.6.32.

We use the downloadable PEF[3] and BF[4] source code for testing, with both using their default settings, and we set the density of BitFunnel to 0.15 for all 11 shards. In the experiments, we reorder the documents according to URL. For machine learning, we use sklearn[5] by python.

6.2 Model Selection

For training and validation of the prediction models, we generate a set of size vectors M_{size} and a set of time vectors M_{time} according to the procedure in Section 5.1.2. We randomly choose 380 corpuses from $\mathcal{F}$, of which 340 are used for generating the training set, and 40 are used for generating the validation set. The training set is generated from the sample vectors for 5000 random pairs of corpuses (where each pair acts as a BF-PEF partition for training), and the validation set is generated from the sample vectors for 1000 random pairs of corpuses. We also add two extra BF-PEF partitions to the training set: one where the whole document corpus is assigned to BF (and no documents are assigned to PEF), and one where the whole document corpus is assigned to PEF.

We generate prediction models for both "size" (i.e., total index size, $s_{\text{BF}} + s_{\text{PEF}}$) and "time" (i.e., average intersection time, $t_{\text{BF}} + t_{\text{PEF}}$). We measure time using the MillionQuery query set. For both size and time, we test three regression models, the gradient-boosted regression tree (GBRT) model, a support vector regression (SVR) model, and a random forest (RF) model.

The test set is from samples generated from 100 BF-PEF partitions (of the whole document corpus), which are randomly generated similar to the procedure described in Section 5.1.2. We randomly choose the number of documents n to assign to BF, and we assign BF the n documents which achieve the top score as per eq. (1) while other documents are assigned to PEF.

We compare the accuracy of the six prediction models (size and time; GBRT, SVR, and RF) against experimental measurements. When a value a is estimated as $\hat{a}$, the *relative error* is defined as $|(\hat{a} - a)/a|$. Table 2 records some statistics of the relative error: its mean, and first, second, and third quartiles.

Table 2 indicates that SVR is the best regression model among the three, in terms of both size and time. Consequently, we exclusively use SVR for the remaining experiments. Moreover, we use the SVR model as trained in this section. The small mean relative error (less than 5%) indicates the predictions are suitable for practical use.

6.3 Initial Partition

In this section, we investigate the influence of the initial partition on the local search method in Section 5.2. We choose initial partitions using the method for generating BF-PEF partitions in Section 6.2: we use 5 with a time threshold, and another 5 with a space threshold. We plot the predicted hybrid index size and average intersection time in Figure 4 where different colors correspond to different initial partitions. We also include points which indicate the actual (i.e., not predicted) values after 500 iterations.

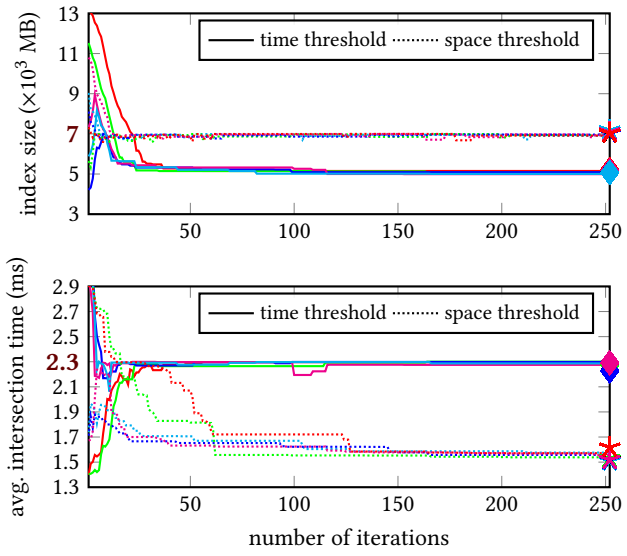

Figure 4: With 5 different initial partitions, after some number of iterations of the local search procedure (x axis), we measure the hybrid index size (top), and the average intersection time (bottom), with either a time threshold of 2.3ms (solid) or a space threshold of 7000MB (dotted). We also include marks to indicate the exact measurements after 500 iterations. (Note that the y axes do not start at 0.)

We make three observations from Figure 4:

(1) Different initial partitions have a negligible impact on the final result, i.e., they approximately converge. This suggests

[1] https://trec.nist.gov/data/million.query.html
[2] https://trec.nist.gov/data/terabyte06.html
[3] https://github.com/ot/partitioned_elias_fano
[4] https://github.com/BitFunnel/BitFunnel
[5] http://scikit-learn.org/stable/index.html

Table 2: The relative error when predicting the index size and average intersection time of the resulting BF-PEF index using GBRT, SVR, or RF; we list the first, second, and third quartiles, and the mean relative error.

	index size				average intersection time			
	1-st quart.	2-nd quart.	3-rd quart.	mean	1-st quart.	2-nd quart.	3-rd quart.	mean
GBRT	3.69%	6.66%	11.82%	8.03%	3.45%	9.49%	26.48%	15.12%
SVR	1.09%	2.03%	2.90%	2.35%	1.77%	3.61%	7.08%	4.81%
RF	3.73%	7.33%	25.16%	14.99%	3.24%	11.92%	24.23%	16.95%

that the local search is not getting stuck in local optima (where we would expect non-converging measurements from various local optima), which gives confidence that the process is reaching a close-to-optimal BF-PEF partition.

(2) Only a small number iterations are needed (200 iterations, say), before an approximately optimal partition is reached. Consequently, there is not a massive overhead introduced by this process.

(3) In these experiments, we see the prediction of the index size and average intersection time is accurate: within 5% of the actual values after 500 iterations. This is consistent with the observations in Section 6.2.

These three observations suggest that the local search method is a suitable approach to finding an approximately optimal BF-PEF partition.

6.4 Time and Space Trade-off

We investigate the trade-off between space (index size) and time (average intersection time). To generate the approximately optimal partition, we use the local search method (500 iterations) described in Section 6.3 with $n = 12\,500\,000$ (half of the whole document corpus) assigned to BF while others are in PEF for initial partition. The observations in Section 6.3 indicate this decision does not have a major impact on the size or time of the resulting BF-PEF partition. In this section the index size and average intersection time are measured after the index is generated rather than predicted by regression models. We do not normalize them as in Figure 1 .

6.4.1 Time threshold. We set a threshold on the average intersection time, and minimize the hybrid index size subject to that threshold. Figure 5 records the size and intersection time of the hybrid index subject to various time thresholds (for the MillionQuery query set). Here and in the next section, we experiment with both the MillionQuery and TerabyteQuery query sets (although model training only uses the MillionQuery query set).

Figure 5 indicates that as the time threshold increases, the size of the hybrid index decreases. This is as expected: as we relax the time threshold, more documents are assigned to PEF, which results in a smaller hybrid index. Indeed, when the time threshold is close to the PEF time, almost all documents are assigned to PEF, in which case there is negligible difference between the hybrid BF-PEF index and PEF itself.

One interesting observation is that when we reduce the time threshold to around the pure BF average intersection time, the size of the proposed hybrid index is reduced by 45% compared to BF. This significant improvement in space comes at virtually no cost in time. We expect this is partly because some documents are short and are infrequently queried: reassigning these from BF to PEF does not have a significant impact on the intersection time (it might even slightly reduce it), but it reduces the index size.

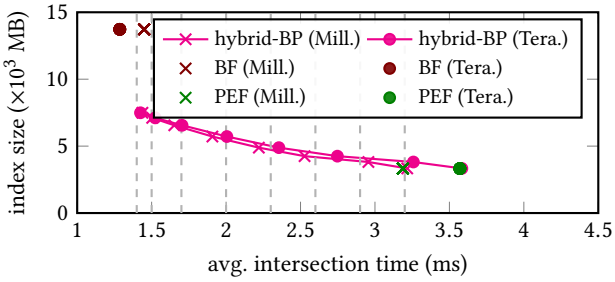

Figure 5: The index size vs. the average intersection time (on the MillionQuery query set (Mill.) and TerabyteQuery query set (Tera.)) of the hybrid BF-PEF index as the time threshold varies (indicated by vertical dashed lines). For comparison, we also include pure BF (red) and pure PEF (green) indexes. (Note that the x axis does not start at 0.)

In Figure 5, some time thresholds on MillionQuery query set are exceeded by a relatively small amount (within 3%), which we attribute to prediction errors in the regression model.

6.4.2 Size threshold. We set a threshold on the hybrid index size, and minimize the average intersection time subject to that threshold. Figure 6 records the size and intersection time of the hybrid index subject to various size thresholds.

For comparison, we also perform the same tests on another hybrid index (hybrid-PIV) proposed by [22][6]; see Section 4 for more about hybrid-PIV. For hybrid-PIV, we also use the MillionQuery query set for training, and the block sampling rate is set to 0.01. (Preliminary experiments with the block sampling rate set to 0.001 were not substantially different, so we do not report these results.)

From Figure 6 we observe that it is generally not possible to improve hybrid-PIV's average intersection time much beyond a certain point (e.g. 2.5ms for the MillionQuery query set) by relaxing the threshold on the index size. This implies the hybrid BF-PEF index has a functional advantage over hybrid-PIV: if memory is available, hybrid-BP can utilize it to reduce the intersection time. Thus, the proposed hybrid BF-PEF index offers greater flexibility in choosing a trade-off than hybrid-PIV.

[6] https://github.com/ot/ds2i

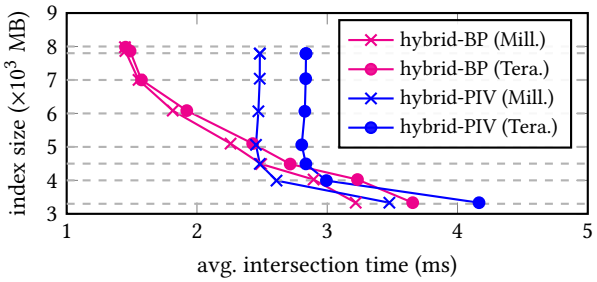

Figure 6: The index size vs. the average intersection time on the MillionQuery and TerabyteQuery query sets of the hybrid BF-PEF index (hybrid-BP) as the size threshold varies (indicated by horizontal dashed lines). There is no result for hybrid-PIV with a 8000MB threshold; its maximum index size is around 7800MB. (Note that both the x and y axes do not start at 0.)

We observe that for the 4000MB threshold, hybrid-PIV outperforms hybrid-BP by 11% and 8% on MillionQuery and TerabyteQuery, respectively. A situation like this amounts to the search engine only having a certain amount of memory they can use (not more nor less). If instead the operator has spare memory and e.g. is able to set a higher size threshold, then Figure 6 indicates that hybrid-BP reduces the average intersection time by up to around 47% with almost the same size.

Compared with PEF on the MillionQuery query set, the maximum improvement in the average intersection time achieved by hybrid-PIV and hybrid-BP is 23% and 55%, respectively; for the 4000MB threshold, the improvement is 18% and 9%, respectively.

In Table 3, we compare hybrid-BP with some indexes compressed by a single form of mathematical encoding. Specifically, we compress the entire GOV2 corpus using each of OptPFD [29], binary interpolative encoding [21], and Varint-G8IU [27], and generate three BF-PEF hybrid indexes with a size threshold approximately the same as the mathematical encoding index it is being compared with. Among them, the binary-interpolative encoded index has the best index size through recursive compression, but its intersection time is long due to recursive decompression. Varint-G8IU uses byte-aligned encoding and SIMD acceleration, making it faster for intersection but its compression is significantly worse; OptPFD offers a trade-off between these two extremes.

When we set the BF-PEF index size threshold equal to the binary interpolative encoding index size (3.13GB), the local search fails to find a BF-PEF partition satisfying the size threshold. The minimum index size of the hybrid BF-PEF index is close to the pure PEF index size, which is 3.25GB. In this case, the hybrid BF-PEF index likely assigns almost all documents to PEF (and acts much the same as pure PEF).

From Table 3, we observe that the hybrid BF-PEF index results in improving the intersection time (vs. the aforementioned mathematical encoding methods) from 16% to 75% on MillionQuery and from 20% to 76% on TerabyteQuery, while having almost the same index size.

6.5 Feature Analysis

We analyze the feature distributions of BF and PEF in the hybrid BF-PEF index in the approximately optimal BF-PEF partition under various threshold conditions. We consider five features: URL, DL, DF, dispersivity, and TF-IDF, as defined in Section 5.1.1. For a feature f, we define the *feature ratio* for $I \in \{\text{BF}, \text{PEF}\}$ as

$$\frac{\overline{f}_I - \overline{f}}{\overline{f}}$$

where $\overline{f}$ is the average value of feature f in the whole document corpus, and $\overline{f}_I$ is the average value of feature f among to the documents assigned to I. The results are plotted in Figure 7 (for time thresholds) and Figure 8 (for size thresholds) for the BF-PEF partitions arising in Section 6.4. For these BF-PEF partitions, we also tabulate the proportion of documents assigned to BF in Table 4.

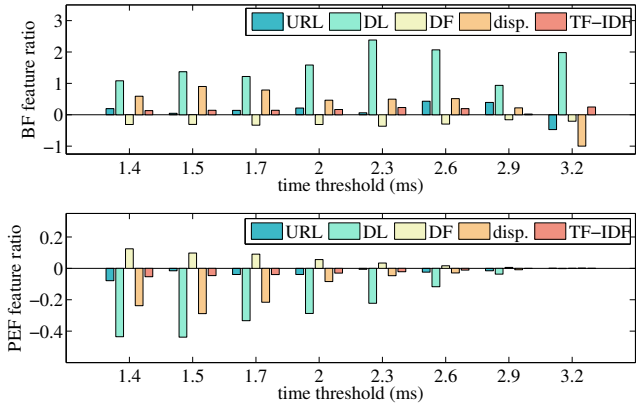

Figure 7: Feature ratios for the BF (top) and PEF (bottom) parts of the hybrid BF-PEF index, under various time thresholds.

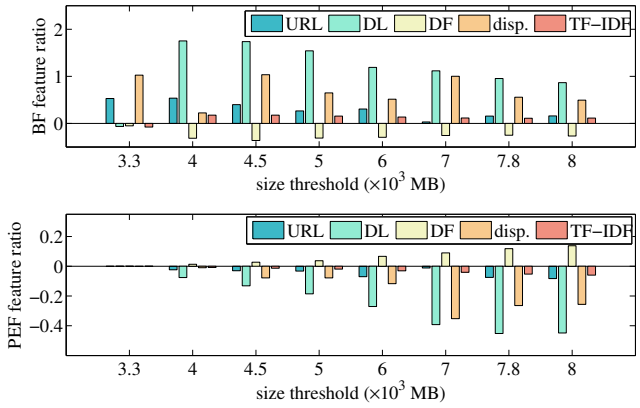

Figure 8: Feature ratios for the BF (top) and PEF (bottom) parts of the hybrid BF-PEF index, under various index size thresholds.

Table 3: The space time trade-off compared with mathematical encoding methods.

	OptPFD	hybrid	bin. int.	hybrid	Varint	hybrid
index size (GB)	3.66	3.77 (+3%)	3.13	3.27 (+4%)	7.74	7.77 (+1%)
avg. intersection time (ms) Mill.	3.65	3.08 (-16%)	12.95	3.28 (-75%)	2.42	1.42 (-41%)
avg. intersection time (ms) Tera.	4.25	3.42 (-20%)	15.06	3.67 (-76%)	2.78	1.43 (-49%)

Table 4: The proportion of documents in the BF part with a given time or size threshold.

time threshold (ms)	1.4	1.5	1.7	2.0	2.3	2.6	2.9	3.2
proportion	28.69%	24.23%	21.47%	15.33%	8.55%	5.36%	3.79%	0.10%
size threshold ($\times 10^3$MB)	3.3	4.0	4.5	5.0	6.0	7.0	7.8	8.0
proportion	0.09%	4.15%	7.02%	10.73%	18.50%	25.99%	32.16%	34.14%

In general, if a feature f has a positive feature ratio for $I \in \{\text{BF}, \text{PEF}\}$, we think of I as favoring feature f. For example, BF generally strongly favors document length (DL) and dispersivity. For a feature f with a negative feature ratio for $I \in \{\text{BF}, \text{PEF}\}$, we instead think of I as disfavoring feature f. For example, BF generally disfavors document frequency (DF). Some of these feature preferences are explainable (like in the pilot experiments in Section 2), while others are not. These results are fairly consistent with Figure 1 in the pilot experiments.

When we set the time threshold equal to the pure PEF intersection time (3.2ms), or the size threshold equal to the pure PEF index size (3300MB), almost all the documents are partitioned into PEF, therefore the feature distribution of the PEF part is almost identical to the whole document corpus; the feature distribution for BF in this setting is consequently unreliable, since it contains few documents.

Figures 7 and 8 further emphasize how a simple method alone is unlikely able to obtain a suitable BF-PEF partition; this is consistent with the observations in Section 2.

6.6 False Positive Analysis

A non-negligible problem with the proposed hybrid index is the false positives caused by BF. Table 5 lists the false positive rates (the proportion of false positive documents among all positive documents) for the proposed hybrid BF-PEF index under various thresholds (the time threshold is for the MillionQuery query set).

Table 5 indicates that, as time threshold increases, the false positive rate decreases; this is due to an increasing number of documents being assigned to PEF, which does not incur false positives. Conversely, as the size threshold increases, more documents are assigned to BF, which increases the false positive rate.

Compared with the pure BF index, the false positive rate of the hybrid BF-PEF index decreases by at least 40% and 23% on MillionQuery and TerabyteQuery, respectively.

Table 5: The false positive (fp.) rates for various thresholds for the MillionQuery (Mill.) and TerabyteQuery (Tera.) query sets. Note that the pure BitFunnel index has 2.48% and 1.70% false positive rates on MillionQuery and TerabyteQuery, respectively.

time thre. (ms)	1.4	1.5	1.7	2.0	2.3	2.6	2.9	3.2
Mill. fp.	1.39%	1.36%	1.03%	0.91%	0.74%	0.33%	0.14%	0.00%
Tera. fp.	1.25%	1.19%	1.01%	0.87%	0.71%	0.32%	0.15%	0.00%
size thre. ($\times 10^3$ MB)	3.3	4.0	4.5	5.0	6.0	7.0	7.8	8.0
Mill. fp.	0.00%	0.24%	0.36%	0.52%	0.86%	1.30%	1.43%	1.49%
Tera. fp.	0.00%	0.23%	0.36%	0.49%	0.84%	1.21%	1.27%	1.31%

7 CONCLUDING REMARKS

In the paper, we propose a hybrid BF-PEF index, combining the spatial efficiency of Partitioned Elias-Fano and intersection efficiency of BitFunnel by partitioning the documents into two disjoint parts. We use a local search to find an approximately optimal BF-PEF partition that meets a pre-specified threshold; during this process, we use machine learning to predict the size and intersection time of the hybrid index.

Experimental results indicate that the proposed hybrid BF-PEF index decreases the intersection time up to 47% compared with another hybrid index with the same index size. Compared with BitFunnel, it reduces the index size by 45% while maintaining almost the same intersection time.

In the future, it would be interesting to explore (a) combining multiple indexes, not just BF and PEF; (b) using more sophisticated thresholding techniques; (c) using alternative regression models; and (d) incorporating additional document features.

One limitation of the present approach is the implicit assumption of everything taking place on a single machine. Scaling up to a distributed system likely requires adapting the machine-learning method for each machine. While we expect this is feasible in practice, this matter is not addressed this in this paper.

8 ACKNOWLEDGEMENTS

This work is partially supported by National Science Foundation of China (61872201, 61702521, 61602266, U1833114); Science and Technology Development Plan of Tianjin (17JCYBJC15300, 16JCYBJC41900, 18ZXZNGX00140, 18ZXZNGX00200); and the Fundamental Research Funds for the Central Universities and SAFEA: Overseas Young Talents in Cultural and Educational Sector.

REFERENCES

[1] Rakesh Agrawal, Behzad Golshan, and Evangelos E. Papalexakis. 2015. A Study of Distinctiveness in Web Results of Two Search Engines. In *Proc. WWW*. 267–273.

[2] Vo Ngoc Anh and Alistair Moffat. 2005. Inverted Index Compression Using Word-Aligned Binary Codes. *Inf. Retr.* 8 (2005), 151–166.

[3] Vo Ngoc Anh and Alistair Moffat. 2010. Index Compression Using 64-bit Words. *Softw., Pract. Exper.* 40, 2 (2010), 131–147.

[4] Burton H. Bloom. 1970. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM* 13, 7 (1970), 422–426.

[5] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32.

[6] Stefan Büttcher, Charles L. A. Clarke, and Gordon V. Cormack. 2010. *Information Retrieval—Implementing and Evaluating Search Engines.* MIT Press.

[7] Berkant Barla Cambazoglu and Ricardo A. Baeza-Yates. 2015. *Scalability Challenges in Web Search Engines.* Morgan & Claypool Publishers.

[8] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms* (3 ed.). MIT Press.

[9] J. Shane Culpepper and Alistair Moffat. 2010. Efficient set intersection for inverted indexing. *ACM Trans. Inf. Syst.* 29, 1 (2010), 1:1–1:25.

[10] Jay L. Devore. 2011. *Probability and Statistics for Engineering and the Sciences* (8 ed.). Cengage Learning.

[11] Harris Drucker, Christopher J. C. Burges, Linda Kaufman, Alexander J. Smola, and Vladimir Vapnik. 1996. Support Vector Regression Machines. In *Advances in Neural Information Processing Systems.* 155–161.

[12] Peter Elias. 1974. Efficient Storage and Retrieval by Content and Address of Static Files. *J. ACM* 21, 2 (1974), 246–260.

[13] Robert Mario Fano. 1971. *On the Number of Bits Required to Implement an Associative Memory.* Massachusetts Institute of Technology, Project MAC.

[14] Jerome H. Friedman. 2001. Greedy Function Approximation: A Gradient Boosting Machine. *Ann. Stat.* 29, 5 (2001), 1189–1232.

[15] Bob Goodwin, Michael Hopcroft, Dan Luu, Alex Clemmer, Mihaela Curmei, Sameh Elnikety, and Yuxiong He. 2017. BitFunnel: Revisiting Signatures for Search. In *Proc. SIGIR.* 605–614.

[16] Andrew Kane and Frank Wm. Tompa. 2014. Skewed Partial Bitvectors for List Intersection. In *Proc. SIGIR.* 263–272.

[17] Daniel Lemire, Owen Kaser, and Kamel Aouiche. 2010. Sorting Improves Word-aligned Bitmap Indexes. *Data Knowl. Eng.* 69, 1 (2010), 3–28.

[18] Xiaozhu Liu and Zhiyong Peng. 2010. An Efficient Random Access Inverted Index for Information Retrieval. In *Proc. WWW.* 1153–1154.

[19] Xinyu Liu, Zhaohua Zhang, Boran Hou, Rebecca J. Stones, Gang Wang, and Xiaoguang Liu. 2018. Index Compression for BitFunnel Query Processing. In *Proc SIGIR.* 921–924.

[20] Claudio Lucchese, Salvatore Orlando, Raffaele Perego, and Fabrizio Silvestri. 2007. Mining query logs to optimize index partitioning in parallel web search engines. In *Proceedings of the 2nd International Conference on Scalable Information Systems.* 43–51.

[21] Alistair Moffat and Lang Stuiver. 2000. Binary Interpolative Coding for Effective Index Compression. *Inf. Retr.* 3, 1 (2000), 25–47.

[22] Giuseppe Ottaviano, Nicola Tonellotto, and Rossano Venturini. 2015. Optimal Space-time Tradeoffs for Inverted Indexes. In *Proc. WSDM.* 47–56.

[23] Giuseppe Ottaviano and Rossano Venturini. 2014. Partitioned Elias-Fano Indexes. In *Proc. SIGIR.* 273–282.

[24] Knut Magne Risvik, Trishul M. Chilimbi, Henry Tan, Karthik Kalyanaraman, and Chris Anderson. 2013. Maguro, a System for Indexing and Searching over Very Large Text Collections. In *Proc. WSDM.* 727–736.

[25] David Salomon. 2007. *Variable-length Codes for Data Compression.* Springer London.

[26] Prabhakant Sinha and Andris A. Zoltners. 1979. The Multiple-Choice Knapsack Problem. *Operations Research* 27, 3 (1979), 503–515.

[27] Alexander A. Stepanov, Anil R. Gangolli, Daniel E. Rose, Ryan J. Ernst, and Paramjit S. Oberoi. 2011. SIMD-based Decoding of Posting Lists. In *Proc. CIKM.* 317–326.

[28] Xiujun Wang, Yusheng Ji, Zhe Dang, Xiao Zheng, and Baohua Zhao. 2015. Improved Weighted Bloom Filter and Space Lower Bound Analysis of Algorithms for Approximated Membership Querying. In *Proc. DASFAA.* 346–362.

[29] Hao Yan, Shuai Ding, and Torsten Suel. 2009. Inverted Index Compression and Query Processing with Optimized Document Ordering. In *Proc. WWW.* 401–410.

[30] Marcin Zukowski, Sándor Héman, Niels Nes, and Peter A. Boncz. 2006. Super-Scalar RAM-CPU Cache Compression. In *Proc. ICDE.*