Improved compression techniques based on Partitioned Elias-Fano indexes

Chetan Arora
Department of Electronics & Instrumentation
BITS Pilani,Pilani Campus
Pilani, India
f2016346@pilani.bits-pilani.ac.in

Abhishek
Department of Computer Science
BITS Pilani, Pilani Campus
Pilani, India
abhishek@pilani.bits-pilani.ac.in

Abstract*— This paper is based on the paper Partitioned Elias-Fano Indexes accepted as the best paper of ACM-Sigir conference 2014. This paper describes improved compression techniques in Information Retrieval systems. In this paper, we have talked about the Elias-Fano encoding on inverted indexes and optimizing it by partitioning the inverted indexes. We have talked about many improvisation on the old compression schemes.

Keywords— Elias-Fano, Djikstra, Compression, Corpora.

I. Introduction

Information Retrieval includes preprocessing and storing of relevant information from large corpora so that it can be retrieved at ease. Storing information requires term-DocId mappings so that any conjunctive, disjunctive query can be executed on the data. Since large IR systems(like a web search) process billions of documents, storing the mapping itself requires too much space and therefore requires compression.

II. *History*

Information Retrieval includes preprocessing and storing of important information from large corpora so that it can be retrieved at ease. Storing of information requires the term-DocId mappings so that any conjunctive, disjunctive query can be executed on the data. Since large IR systems(like a web search) process billions of documents, storing the mapping itself requires too much space and therefore requires compression.

Compression usually comes at the cost of speed of query processing. We have talked about parameters like query speed, compression ratio and most important compression ratio/Query speed.

A. Inverted Indexes

Inverted indexes were an improvement over the docID-term matrix because the matrix was sparse due to large number of words in the english dictionary. Inverted indexes stores the docIds of all the documents that contain a term. Since different docIds(numbers) can be stored using a different number of bits, the process of storing can be further be optimized.

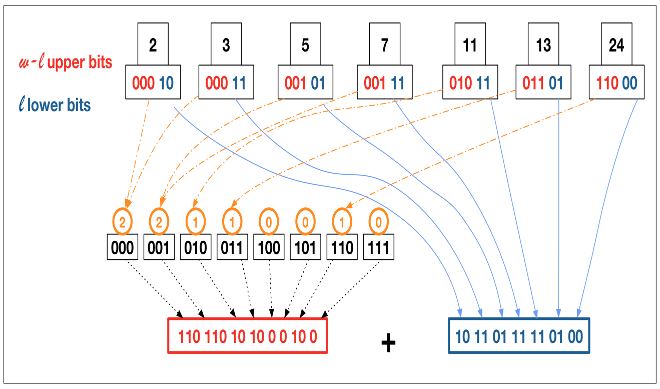
B. Compression

Assuming each docId does not need the same number of bits we can improve the storage using a different number of bits for each docId.

Another storage improvement was to store the gaps between docIds. This way we can store the gaps(small numbers) using less number of bits. But there are downsides of storing gaps as the whole list needs before a docId needs to be decoded for any search operations. There are several gap encodings that are efficient for storage.

III. Elias-fano Encoding

We are using Elias-Fano encoding for its efficient compression ratio and fast access and search operations. It is not a gap coding but still performs comparatively good. It works best only when the sequence does not contain outliers since each number uses the same number of bits. Elias Fano encoding uses 2+ log(u/m) bits for each of m numbers of series and upper bound u.



Elias-Fano encoding does not encode each number separately; rather it uses two bitvectors for storing the whole sequence. The sequence should be strictly increasing because it uses this property for better storing and fast access. Taking an example: Strict increasing sequence M[k] having k elements: [0,2,5,..., u] having the largest value of u. Elias-Fano divides the number into high-bits and low-bits for storing. As shown in the figure, each number is binary encoded with log(k) bits.These bits are divided into low-bits and high-bits.The encoding is optimal when the number of low-bits is [log(largest element u/num of elements k)] where [] represents greatest integer function, and the remaining are high bits.

The low-bits are directly stored in a bit vector of size k*log(u/k) bits while the high-bits are stored elegantly. 2^(logk) are total num of distinct high bits and would occur in increasing order. So we store only the number of elements that have distinct high bits like 2 elements have “000” as high bits. These counts are then stored using unary encoding as shown in the figure.

IV. Partitioned Elias-Fano Indexes

Partitioning the sequence into chunks helps in optimizing the compression ratio for numbers lying on the lower end of the sequence. Making a two-level data structure makes it possible to skip whole chunks of data helping in query speedup.Partitioning can be uniform and non-uniform.

A. Uniform Partition

In uniform partition scheme, a two-level structure is employed in which an inverted index is broken into equal size chunks. Assuming the chunk length being b and inverted index docIds sequence being M of size m, the index is broken into m/b number of chunks. The first level of the structure would contain only the last value of the chunk, while the whole chunk is stored in second level of structure encoded using Elias-Fano. The first level of the structure can also be encoded using Elias-Fano encoding.

As this is a two level structure for storing docIds in an inverted index, it can further optimized using Position Of Reference. The scheme is such that assuming the values in chunk being [c,c+2, c+5,c+14] can be stored as c + [0,2,5,14]. Since we are already storing c in the first level of structure, we can achieve more compression using this change of reference.

This way partitioning not only helps in optimizing the size of lower end of sequence but also using reference optimization can reduce the size on the higher end of the sequence.

B. Non-Uniform Optimal Partition

Since the uniform partitioning scheme does not take into account the patterns in the data itself, we can say it can still be improved using a non-uniform partitioning scheme. The problem at hand is to partition the inverted index sequence such that it is optimal in storage. Changing the size of chunks also affects the first level of data storage e.g. If we increase the number of chunks, it is possible to store the data using less number of bits due to less number of elements, but at the first level it would take a toll on storage and also the query searching have to skip more number of blocks. So, we have also taken into account the cost of storing data on first level in our optimal partitioning algorithm.

Assuming the sequence contains m elements in total, for finding the optimal solution we have to make a Directed Acylic Graph(DAG) considering possibility for each path that stores all numbers. We are using Directed graph because the sequence must be stored only in strictly increasing. The graph can contain m vertices representing all the elements and the edges can show the weights for the cost of encoding the chunk and also including cost of encoding endpoints in the first level. The cost can be found out different levels differently. In the first level, we have to encode the upper bound (Last value of chunk), the size of chunk(since we are using non-uniform encoding) and a pointer to the a second level of the structure. The cost of encoding the actual chunk at second level can be computed at run-time. Here, we can make use of locality of the numbers in chunks e.g. if the number are continuous, we don't use any bits, as we already have number of elements and largest value in the first level.
Since this Directed Acyclic graph have storage costs as weights, we can find the optimal solution using the shortest path for weighted DAG graph. Shortest distance algorithms like Topological sorting or Djikstra positive weights for graph traversal and have complexity of O(n^2) where n is number of vertices. Considering the number to be in millions, we can't directly apply such algorithms. We have to prune the graph using some constraints to ease the time complexity. Here we are not taking into account the problems with running such algorithms in main-memory and using Map-Reduce for such operations. Constraints for pruning the graphs are:

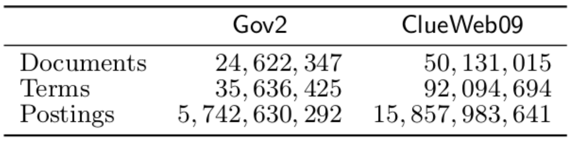
1.Weight(i,j) <F(1+epsilon) < Weight(i,k) if (i <= j< k) since cost function for chunk encoding depends upon the num of elements.
2. Initial vertex for the path should be the first element and we are calculating the shortest distance from this point.
3. Final vertex for any given path should end at vertex m-1 (last element in the sequence).

Mathematically, first statement says to consider only those chunk's weights that show considerable improvement over super sets of other chunks.
e. g. chunk A: [0,1,2,3,] and chunk B:[0,1,2,3,8] since chunk A would be very efficient at encoding. Here F shows a lower-bound for the improvement of cost.

This pruned graph will now have at max log(U/F) edges for each vertex giving complexity of O(m*log(U/F)). This can be further improved using some epsilon approximations such that complexity can be O(m*log1+e2(1/e1)) with solution being suboptimal only by factors of epsilon. This suboptimal solution depends upon adjustable e1 and e2(epsilons) that changes the bounds.

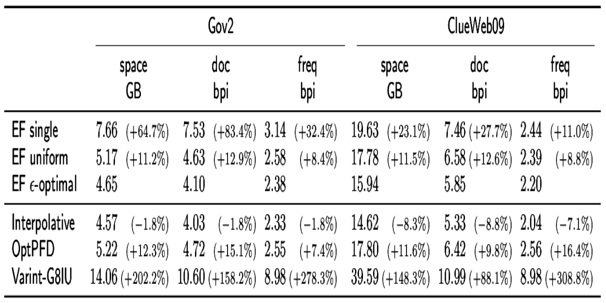
V. Results

Results produced by original writer of the paper (Partitioned Elias-Fano indexes ACM-sigir pasted here for understanding)
Information regarding Corpora:



	Gov2	ClueWeb09
Documents	$24,622,347$	$50,131,015$
Terms	$35,636,425$	$92,094,694$
Postings	$5,742,630,292$	$15,857,983,641$

Storage results:



	Gov2			ClueWeb09		
	space GB	doc bpi	freq bpi	space GB	doc bpi	freq bpi
EF single	7.66 (+64.7%)	7.53 (+83.4%)	3.14 (+32.4%)	19.63 (+23.1%)	7.46 (+27.7%)	2.44 (+11.0%)
EF uniform	5.17 (+11.2%)	4.63 (+12.9%)	2.58 (+8.4%)	17.78 (+11.5%)	6.58 (+12.6%)	2.39 (+8.8%)
EF ϵ-optimal	4.65	4.10	2.38	15.94	5.85	2.20
Interpolative	4.57 (−1.8%)	4.03 (−1.8%)	2.33 (−1.8%)	14.62 (−8.3%)	5.33 (−8.8%)	2.04 (−7.1%)
OptPFD	5.22 (+12.3%)	4.72 (+15.1%)	2.55 (+7.4%)	17.80 (+11.6%)	6.42 (+9.8%)	2.56 (+16.4%)
Varint-G8IU	14.06 (+202.2%)	10.60 (+158.2%)	8.98 (+278.3%)	39.59 (+148.3%)	10.99 (+88.1%)	8.98 (+308.8%)

Compression Ratio

Here we are using 3 state-of-art storage schemes namely

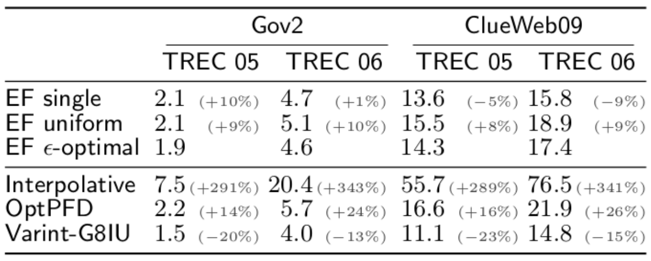
1. Binary Interpolative Encoding(Best Compression Ratio) not comparable at query speed.

2. OptPFD(Best (compression ratio/ query speed) tradeoff)

3. Variant-G8IU that uses machine oriented bytes (Good query speed) not comparable at compression ratio.

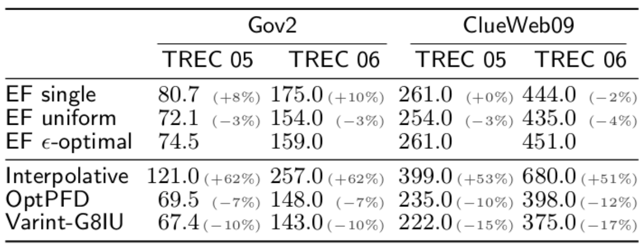
In terms of storage, EF-optimal gives competitive results to Binary Interpolative encoding. Also, EF-uniform results are satisfying.

Query Speed Results:



	Gov2		ClueWeb09	
	TREC 05	TREC 06	TREC 05	TREC 06
EF single	2.1 (+10%)	4.7 (+1%)	13.6 (−5%)	15.8 (−9%)
EF uniform	2.1 (+9%)	5.1 (+10%)	15.5 (+8%)	18.9 (+9%)
EF ϵ-optimal	1.9	4.6	14.3	17.4
Interpolative	7.5 (+291%)	20.4 (+343%)	55.7 (+289%)	76.5 (+341%)
OptPFD	2.2 (+14%)	5.7 (+24%)	16.6 (+16%)	21.9 (+26%)
Varint-G8IU	1.5 (−20%)	4.0 (−13%)	11.1 (−23%)	14.8 (−15%)

AND Queries



	Gov2		ClueWeb09	
	TREC 05	TREC 06	TREC 05	TREC 06
EF single	80.7 (+8%)	175.0 (+10%)	261.0 (+0%)	444.0 (−2%)
EF uniform	72.1 (−3%)	154.0 (−3%)	254.0 (−3%)	435.0 (−4%)
EF ϵ-optimal	74.5	159.0	261.0	451.0
Interpolative	121.0 (+62%)	257.0 (+62%)	399.0 (+53%)	680.0 (+51%)
OptPFD	69.5 (−7%)	148.0 (−7%)	235.0 (−10%)	398.0 (−12%)
Varint-G8IU	67.4 (−10%)	143.0 (−10%)	222.0 (−15%)	375.0 (−17%)

OR Queries

In terms of query speed on AND queries, we can see that EF-Uniform and EF-optimal performs better than other encoding schemes due to its fast Access and Search operations. We are going to discuss these operations in general in next section. EF-optimal encoding is made for these queries, it may take decompression time, but in long term, with such compression ratio, EF-optimal works best on AND queries.

In terms of query speed on OR queries, we can see that EF and specifically EF-optimal does not perform very well due to its partitioning nature. OR queries just measure the decompression oerations and due to different chunk size, this operation can be costly.

Elias-Fano normal may not come as fast in terms of query speed due to access and search time on large bit vectors however optimized can be expensive.

VI. Suggestions and Limitations

After reading the paper on Partitioned Elias-Fano indexes, we get many ideas and concepts to store the indexes efficiently. Applying the elegant idea of partioning to inverted indexes produce significant results. Reading on Elias-Fano indexes and its properties for fast access and search operations makes it comparable to delta-coding.

Another important and significant details that are mentioned vaguely in this paper are using position of referene in the chunks so that numbers on high spectrum can be decoded easily. These details make this paper and this system none other than a state-of-art model and very well deserve to be best accepted paper of ACM-sigil 2014 conference.

In my opinion, the concepts mentioned in this paper are worth employing to other storage models. We can also make a 2-level structure with chunks encoded using gamma-delta coding and the upper layer encoded using Elias-Fano coding.This model will not only perform better at compression ratio, since most of the data resides at second level. Also, the fast operations on Elias-Fano coding would help in skipping the chunks better. The second level chunks (being delta coded) can be decoded as a whole and still won't cause significant changes in the query speed.

Since the docIds that share a chunk are very related (e.g. being subdomains of same URLs it is possible to cluster and store them). This way, wherever we see a bi-word in these documents we can join them to both these words.

The idea behind is the transition from docIds to **cluster**(or chunks) of document ids that represents some idea. If these clusters become **standard** such that we can use the same cluster even some document in the cluster does not have the term, this improves storage significantly.

As we have already seen, the EF-Optimal requires epsilon parameters(e1 and e2) which work as bounds for suboptimal calculation. These parameters need to be set, and they depends highly on the given input data. This time for selecting optimal parameters and constructing the index is called construction time and it is significantly more as compared to EF-normal. We can make a machine learning model or any other heuristics to **converge to optimal parameters quickly.**

VII. References

[1] Giuseppe Ottaviano and Rossano Venturini. 2014. Partitioned Elias-Fano indexes. In Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'14). 273–282. *(Main Paper)*

[2] Elias-Fano encoding for sorted integers. https://www.antoniomallia.it/sorted-integers-compression-with-elias-fano-encoding.html