CorBit: Leveraging Correlations for Compressing Bitmap Indexes

Xi Lyu[1], Andreas Kipf[2,*,†], Pascal Pfeil[2,†], Dominik Horn[2,†], Jana Giceva[1] and Tim Kraska[3]

[1]Technical University of Munich, Boltzmannstr. 3, 85748 Garching bei München, Germany
[2]Amazon Web Services, Oskar-von-Miller-Ring 20, 80333 München, Germany
[3]Massachusetts Institute of Technology, 77 Massachusetts Ave, Cambridge, MA 02139, United States

Abstract

A bitmap index is a secondary index structure that supports equality and range predicates. In its simplest form, a bitmap index stores one bitmap per unique column value indicating qualifying tuples. To use such indexes in large-scale data warehousing, they need to be space efficient. Existing schemes such as Roaring can compress individual bitmaps but do not consider cross-column compression.

In this paper, we introduce CorBit, a technique that leverages column correlations to compress bitmap indexes on a given table. The high-level idea is to only store the bits that need to be flipped (the diff) when encoding the bitmaps of a column that correlates with another column that already has a bitmap index in place. CorBit automatically determines which columns to store an index for and which column bitmaps to diff-encode, minimizing the overall size of the index. Compared to Roaring, CorBit consumes 9.1% less space on the DMV dataset while incurring a 12.6% runtime overhead.

1. Introduction

Bitmap indexes are a well-known indexing technique in data warehousing. For example, YouTube's SQL engine, Google Procella, uses Roaring bitmaps for indexing advertising experiments. Such indexes can achieve tremendous speedups for selective queries, e.g., Procella claims 500× faster queries due to such indexes [1].

A straightforward implementation of a bitmap index is to store one bitmap per unique column value. For example, the Germany bitmap would indicate the positions (row numbers) of all tuples that have Germany as a column value. To trade precision for footprint, bitmap indexes can be used at the block level and individual column values can be binned together. However, even with these optimizations, bitmap indexes can consume too much space to use them in large-scale data warehousing. Bitmap compression schemes such as Roaring [2, 3, 4] or Tree-Encoded Bitmaps [5] do very well in compressing individual bitmaps while maintaining fast lookup performance, but they do not consider cross-column compression opportunities.

In this paper, we introduce CorBit, a new bitmap compression scheme that exploits column correlations to compress multiple bitmap indexes defined on a single table. On a high level, CorBit stores the *bit differences* between two given bitmap indexes and only fully materializes one of the two indexes. We also introduce a new correlation metric, *Total Reduced Popcnt*, that captures the compression opportunity between two columns. We further propose a greedy algorithm to decide which bitmap indexes to materialize and which indexes to diff-encode, minimizing overall index size.

We experiment with three real-world datasets and show that CorBit improves upon Roaring in space while incurring a small runtime overhead.

2. Compressing Bitmap Indexes

Basic Idea. If two columns in a table are highly correlated, we only materialize the bitmap index of one column. For each unique value in the other column, we find the closest bitmap in the materialized column w.r.t. Hamming distance [6] and only store the differences in its bitmap. If the Hamming distance is small, the diff-encoded bitmaps will be sparse, allowing various bitmap compression methods such as Roaring or run-length encoding to save more space.

For example, the two columns Language and Country in a user information table are usually highly correlated. A sample contingency table is shown in Table 1. The skewed frequency counts in the table indicate that if we would materialize the bitmaps of the Country column, using diff-encoding for the Language column would save space. E.g., for the value German in Language we choose the Germany bitmap as reference and store the difference of the two bitmaps, i.e., the XOR of the two bitmaps. In

Joint Workshops at 49th International Conference on Very Large Data Bases (VLDBW'23) — Workshop on Applied AI for Database Systems and Applications (AIDB'23), August 28 - September 1, 2023, Vancouver, Canada

[*] Corresponding author.

[†] Work done while at Massachusetts Institute of Technology.

xi.lyu@tum.de (X. Lyu); kipf@amazon.com (A. Kipf); pfeip@amazon.com (P. Pfeil); domhorn@amazon.com (D. Horn); jana.giceva@in.tum.de (J. Giceva); kraska@mit.edu (T. Kraska)

Table 1
Contingency table for Language and Country

Language	Country (materialized)			
(diff-encoded)	Germany	United States	China	Total
German	1000	200	100	1300
English	100	1050	150	1300
Chinese	50	100	1100	1250
Total	1150	1350	1350	3850

our experiments, we use Roaring as the bitmap encoding and serialization algorithm, which is faster and smaller than uncompressed bitsets [2, 3, 4].

Measuring Correlation. To decide which column pairs to consider for our cross-column compression scheme, we need to measure the correlation between columns. A naïve approach would be to consider all pairs of columns. More specifically, for each bitmap in one column, we find the closest one in the bitmaps of the other column w.r.t. Hamming distance [6], and finally sum up all the distances as a measure of correlation. The smaller the total, the higher the correlation between the two columns.

Unfortunately, the naïve approach is computationally expensive. A better method would be to find a suitable and efficient metric that can help us identify potentially correlated column pairs. Therefore, we have evaluated a number of metrics that measure the correlation of categorical features: e.g., *Cramer's V* [7], *Tschuprow's T* [8], *Pearson's Contingency Coefficient* [9], and *Approximate Functional Dependency Error* [10]. Additionally, we also designed our own metric called *Total Reduced Popcnt* based on the contingency table. It directly measures how much the total population count (i.e., the number of "1" bits), also known as Hamming weight, of the diff-encoded bitmaps can be reduced compared to materialization. This metric is defined as

$$\eta_{A,B} = \sum_{a \in U_A} \max\{0, \max_{b \in U_B}(2|\sigma_{A=a \wedge B=b}R| - |\sigma_{B=b}R|)\}$$

where A and B are two columns in relation R, and U_A or U_B denotes the set of unique values in column A or B. This metric can be easily calculated based on the contingency table of column A and B, and is highly positively related to the saved space. This approach only requires traversing the dataset once and then utilizing contingency tables for subsequent calculations.

We have conducted several experiments to evaluate the effectiveness of these metrics in our scenario. We define the effectiveness of a metric as follows: given a parameter k, for each column X in relation R, we select the top k columns with the highest correlation metric value as candidate reference columns. We calculate the ratio between the total potential space savings obtained by using these candidate reference columns and the total potential space savings obtained by using the actual optimal k candidate reference columns. Thus, we define the effectiveness of metric m as

$$E^X_{R,m,k} = \frac{\sum_{i=1}^{k} s(X, Y^m_{X,i})}{\sum_{i=1}^{k} s(X, Y^*_{X,i})}$$

where $Y^m_{X,i}$ represents the i-th best candidate reference column selected by metric m for column X, $Y^*_{X,i}$ represents the i-th optimal candidate reference column for column X, and $s(X,Y)$ denotes the amount of saved space for X when using Y as a reference column.

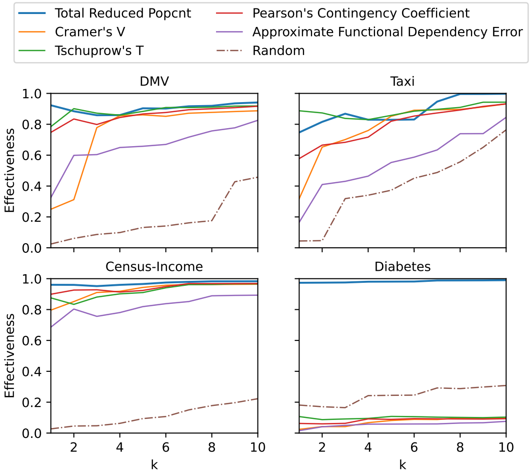


Figure 1: The effectiveness of correlation metrics

Furthermore, the overall effectiveness of metric m across the entire relation R is defined as the weighted average of the effectiveness on each column. Here we use the size obtained using Roaring encoding as weights for columns, giving higher weight to larger columns.

The results of the experiments that evaluate the effectiveness of all metrics are shown in Figure 1. The key observation is that for all evaluated datasets, the *Total Reduced Popcnt* metric consistently performs well and as such is a good proxy for potential savings.

Compressing a Bitmap Index. The previous section focused on measuring the correlation between two columns, but in real-world scenarios, there are often multiple categorical columns involved. What we ultimately need is a Directed Acyclic Graph (DAG) G^* that represents the dependency relationships between the columns. Specifically, each node in this DAG represents a column, with at most one outgoing edge pointing to another node to reference its bitmaps to exploit correlations and save space. Nodes without outgoing edges will be directly materialized. For performance reasons, we currently do not consider chained dependencies.

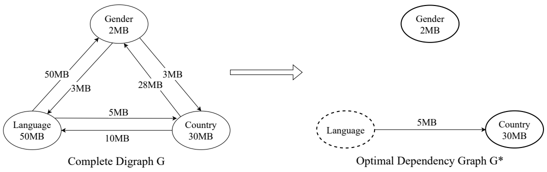


Figure 2: An example of an annotated complete digraph and its optimal dependency graph

To determine the optimal solution, we annotate the actual sizes of the materialized column for each node in a complete digraph G, with the size of the diff-encoded bitmaps of column A when referencing column B as cost for each directed edge (A, B), as shown in Figure 2. Finally, we can employ dynamic programming to obtain the optimal dependency graph G^*.

However, the aforementioned approach is excessively expensive. Fortunately, we have identified an appropriate correlation metric, namely Total Reduced Popcnt, which strongly correlates with saved space. What we annotate here is the Total Popcnt, the sum of the population counts of bitmaps of each column, which we can calculate easily.

Once we obtain an annotated complete digraph G, we can employ a greedy algorithm to obtain a near-optimal solution. This involves greedily selecting the best edge, which has the highest difference between the node cost and edge cost without violating the mentioned properties. We add this edge to G^* and continue the process as long as it results in a reduction in space consumption.

Finally, we compute and store the original bitmaps and XOR bitmaps based on the dependency graph. Since we use Total Reduced Popcnt, its intermediate results allow us to select, for each bitmap, the closest bitmap in its referenced column w.r.t. Hamming distance. This is also one of the advantages of employing our metric.

For performance considerations, we store a XOR bitmap only if it saves p of the space compared to the original bitmap. We choose 80% as the default value for the threshold p. By adjusting the parameter p, users can strike a balance between performance and space. To save more space, a smaller value of p can be chosen, such as 50%. On the other hand, if performance is of greater concern, a larger value of p, such as 90%, can be selected. We experiment with different thresholds in Section 3.

In summary, compressing the bitmap index works as follows:

1. Initialize a contingency table for each pair of columns.
2. Scan the data once while updating the frequencies in the contingency tables.
3. Calculate the Total Reduced Popcnt for each pair of columns based on their contingency tables.
4. Run the greedy algorithm to obtain an approximate optimal dependency graph G^*.
5. Compute the original bitmaps for preserved nodes and diff-encoded bitmaps for preserved edges in G^*, and persist them using Roaring.

Table 2
Datasets

Name	# Rows	# Columns	Average unique rate
DMV	12,300,116	19	0.047%
Taxi	12,741,035	15	0.012%
Diabetes	101,766	46	0.061%

Lookups. To retrieve the bitmap for a specific value in a column, if it is stored as an XOR bitmap, we read both the XOR bitmap itself and the referenced bitmap, and perform an XOR operation on them. If the value is stored as the original bitmap, we can directly read and return it.

3. Evaluation

We evaluate our method using three real-world datasets and compare it with Roaring in terms of space consumption and lookup latency.

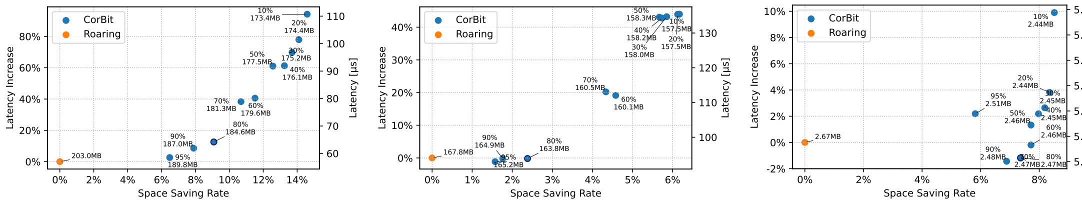
Experimental Setup. We conduct our experiments on a machine with 256 GiB of RAM, an Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20GHz, and a Toshiba MG03ACA100 1 TiB 7200 RPM Hard Drive. We utilize the official C++ implementation of Roaring[1] for serializing, deserializing, and XORing Roaring bitmaps.

We evaluate our approach using the DMV [11], Taxi [12], and Diabetes [13] datasets, as shown in Table 2. For each dataset, we select categorical columns with a proportion of unique values not exceeding 10%. For CorBit, we conduct experiments with various threshold values for parameter p ranging from 10% to 95%.

To measure the average lookup latency, for each lookup, we uniformly select a column from the dataset and then uniformly select a value from that column. We then ask CorBit or Roaring to return the corresponding bitmap and measure the latency.

Size versus Latency. The size and latency of the CorBit and Roaring compression methods on three real-world datasets are shown in Figure 3. On the DMV dataset (Figure 3a), CorBit ($p = 80\%$) achieves space savings of 9.1% compared to Roaring, with a 12.6% increase in latency. By adjusting the threshold value p, we can achieve a maximum space saving of up to 14.6%. However, if performance is more critical, by adjusting p to 95% we can still get a space saving of 6.5% while only experiencing a 2.6% increase in latency.

[1] CRoaring: https://github.com/RoaringBitmap/CRoaring



(a) DMV (12,300,116 rows) **(b) Taxi (12,300,116 rows)** **(c) Diabetes (101,766 rows)**

Figure 3: Size and lookup latency of different CorBit configurations compared to Roaring.

Table 3
Size per column under Roaring and CorBit ($p = 80\%$) (top 5 columns)

Column	Cardinality	Size in Roaring	Reference column	Size in CorBit	Saving rate	Total Reduced Popcnt
Scofflaw Indicator	2	1.52 MiB	Revocation Indicator	0.18 MiB	88.2%	12,207,722
Zip	11752	26.00 MiB	City	11.64 MiB	55.2%	8,115,043
County	63	20.67 MiB	City	18.20 MiB	11.9%	3,527,291
Record Type	4	1.97 MiB	Fuel Type	1.91 MiB	3.1%	10,936,841
Color	225	9.90 MiB	Body Type	9.75 MiB	1.5%	519,148

On the Taxi dataset (Figure 3b), CorBit ($p = 80\%$) achieved a space savings of 2.4% relative to Roaring, with similar query latency. Similarly, on the Diabetes dataset (Figure 3c), CorBit ($p = 80\%$) has comparable performance to Roaring with space savings of 7.4%.

Size per Column. Table 3 shows the top 5 columns with the highest space savings in the DMV dataset. We observe that the Scofflaw Indicator references the Revocation Indicator, and Zip and County reference City. These relationships are reasonable based on the meanings of these columns, and CorBit has successfully discovered the correlations, leading to space savings.

It is worth noting that there is a strong positive correlation between saving rate and Total Reduced Popcnt. In addition to demonstrating its effectiveness by the metric shown in Figure 1, it further substantiates its effectiveness when choosing reference columns.

For Taxi, the Total Amount column has the highest savings rate, by referencing Tip Amount. For Diabetes, the columns with the highest space savings are Acetohexamide, Glimepiride Pioglitazone, and Metformin Rosiglitazone. These columns reference Metformin Pioglitazone, because most of the time, none of the three drugs are prescribed with Metformin Pioglitazone. CorBit uses this observation to save space by exploiting the correlation.

Latency Breakdown. To further analyze latency, we profiled both CorBit ($p = 80\%$) and Roaring and identified the three main components contributing to the overall latency: disk access, Roaring bitmap deserialization, and the XOR operations. The breakdown of the time spent on each component is shown in Table 4.

Our observation is that CorBit ($p = 80\%$) reads only 1.7% more file content compared to Roaring on average. Therefore, it does not significantly impact performance in terms of disk access and bitmap deserialization. However, CorBit ($p = 80\%$) requires additional XOR operations between the XOR bitmaps and the referenced bitmaps, incurring a latency increase. On average, approximately 7.4 us are spent on XOR operations, accounting for 11.5% of the total latency.

Table 4
Latency breakdown on DMV

	CorBit ($p = 80\%$)		Roaring	
Component	µs	%	µs	%
Disk access	29.8	46.4%	30.1	52.7%
Roaring deserialization	26.5	41.4%	26.2	46.0%
XOR operation	7.4	11.5%		
Sum	64.2	100.0%	57.0	100.0%

4. Conclusions

We have introduced CorBit, a bitmap index compression scheme that can exploit cross-column correlations. We have shown that CorBit can achieve significant space savings on three real-world datasets. We have introduced a threshold parameter that allows for trading off compression rate for query performance. In future work, we plan to extend our work to support other compression schemes, such as run-length, frame-of-reference, and dictionary encoding.

Acknowledgments. This research is supported by Google, Intel, and Microsoft as part of DSAIL at MIT, and NSF IIS 1900933. This research was also sponsored by the United States Air Force Research Laboratory and the United States Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein. Pascal Pfeil and Dominik Horn were supported by a fellowship within the IFI programme of the German Academic Exchange Service (DAAD).

References

[1] B. Chattopadhyay, P. Dutta, W. Liu, O. Tinn, A. Mccormick, A. Mokashi, P. Harvey, H. Gonzalez, D. Lomax, S. Mittal, et al., Procella: Unifying serving and analytical data at youtube (2019).

[2] S. Chambi, D. Lemire, O. Kaser, R. Godin, Better bitmap performance with roaring bitmaps, Software: practice and experience 46 (2016) 709–719.

[3] D. Lemire, G. Ssi-Yan-Kai, O. Kaser, Consistently faster and smaller compressed bitmaps with roaring, Software: Practice and Experience 46 (2016) 1547–1569.

[4] D. Lemire, O. Kaser, N. Kurz, L. Deri, C. O'Hara, F. Saint-Jacques, G. Ssi-Yan-Kai, Roaring bitmaps: Implementation of an optimized software library, Software: Practice and Experience 48 (2018) 867–895.

[5] H. Lang, A. Beischl, V. Leis, P. Boncz, T. Neumann, A. Kemper, Tree-encoded bitmaps, in: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, 2020, pp. 937–967.

[6] R. W. Hamming, Error detecting and error correcting codes, The Bell system technical journal 29 (1950) 147–160.

[7] H. Cramér, Mathematical methods of statistics, volume 26, Princeton university press, 1999.

[8] H. Rietz, Aa tschuprow, grundbegriffe und grundprobleme der korrelationstheorie (1926).

[9] K. Pearson, Vii. mathematical contributions to the theory of evolution.—iii. regression, heredity, and panmixia, Philosophical Transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character (1896) 253–318.

[10] S. Kruse, F. Naumann, Efficient discovery of approximate dependencies, Proceedings of the VLDB Endowment 11 (2018) 759–772.

[11] J. Huang, B. Chen, L. Luo, S. Yue, I. Ounis, Dvm-car: A large-scale automotive dataset for visual marketing research and applications, in: 2022 IEEE International Conference on Big Data (Big Data), IEEE, 2022, pp. 4140–4147.

[12] N. Y. C. Taxi, L. C. (TLC), Yellow taxi trip records, https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page. accessed May 1, 2023, 2023.

[13] J. Clore, K. Cios, J. DeShazo, B. Strack, Diabetes 130-US hospitals for years 1999-2008, UCI Machine Learning Repository, 2014. DOI: https://doi.org/10.24432/C5230J.