

Indexing Incomplete Databases

Guadalupe Canahuat, Michael Gibas, and Hakan Ferhatosmanoglu

Department of Computer Science and Engineering,
The Ohio State University
{canahuat, gibas, hakan}@cse.ohio-state.edu

Abstract. Incomplete databases, that is, databases that are missing data, are present in many research domains. It is important to derive techniques to access these databases efficiently. We first show that known indexing techniques for multi-dimensional data search break down in terms of performance when indexed attributes contain missing data. This paper utilizes two popularly employed indexing techniques, bitmaps and quantization, to correctly and efficiently answer queries in the presence of missing data. Query execution and interval evaluation are formalized for the indexing structures based on whether missing data is considered to be a query match or not. The performance of Bitmap indexes and quantization based indexes is evaluated and compared over a variety of analysis parameters for real and synthetic data sets. Insights into the conditions for which to use each technique are provided.

1 Introduction

Real world applications using databases with missing data are common. Databases with missing data occur in a wide range of research and industry domains. Some examples of these are:

1. A census database that allow null values for some attributes
2. A survey database where answers to one question cause other questions to be skipped
3. A medical database that relates human body analyte (a substance that can be measured in the blood or urine) measurements to a number of diseases, or patient risk factors to a specific disease

The goal of this paper is to provide techniques that access databases efficiently in the presence of missing data.

There are a variety of reasons why databases may be missing data. The data may not be available at the time the record was populated or it was not recorded because of equipment malfunction or adverse conditions. Data may have been unintentionally omitted or the data is not relevant to the record at hand. The allowance for and use of missing data may be intentionally designed into the database. In some cases, the missingness of data is random, i.e. the missingness of some value does not depend on the value of another variable. In that case, the missingness is ignorable and the way of dealing with it is to “complete” the value

using regression or other statistical model and treat the data as if it was never missing. However, if the data are missing as a function of some other variable, a complete treatment of missing data would have to include a model that accounts for missing data. Consider the example of the analyte-disease database where diseases are the records and analyte ranges are the attributes. This database would contain values for analyte ranges if they are relevant for a specific disease, or null values if the analyte readings are not important in the diagnosis of that disease. We may query such a database with a patient's analyte readings to get a list of potential diagnoses. We do not want to discount diseases that do not have a value for an analyte included in the query, because the act of taking an analyte's measurement has no bearing on if a patient has a disease that is not relevant to that particular analyte. So in this case, missing data should be interpreted as a query match for that attribute. Alternatively, the intent of a query may not be to return records that *could* match query criteria, but to only return records that definitely match query criteria. In this case any missing data for a record that occurs in an attribute specified by the query search key means that the record does *not* match the query. An example of this could be a survey results query where the query asks for a count of respondents that answered question 5 with answer "A" and question 8 with answer "C".

This paper deals with data where missingness is not ignorable, in other words whether a data value is missing or not is important and we want to be able to distinguish between the real values and the absence of such values. In order to achieve this, we could assign a specific value for missing fields that is not in the domain of that particular attribute. For example, if the domain of an attribute is the positive integers, a value of -1 may be used to denote missing data. Then the transformed, complete multi-dimensional database could be indexed using traditional hierarchical multi-dimensional indexing techniques. However, this solution for indexing databases with missing data experiences significant performance issues when applied to hierarchical indexing techniques. To illustrate this point, we performed a set of experiments on two-dimensional data sets that are identical except that they vary with respect to their percentage of missing data. We built an R-tree index on the different datasets and executed 2-dimensional queries with a global selectivity of 25%. Figure 1 shows the effect on query execution time as missing data probability varies.

The graph shows time performance of a query using an R-tree built on the different data sets, normalized to the time to perform the query on a complete data set. This graph shows that even for a data set and index that is only two dimensions, we get far worse performance when the database contains missing data. Even when there is only 10% missing data for each attribute, the time performance is 23 times worse than if the data set were complete.

Multi-dimensional indexing techniques work best when records are mapped to non-overlapping hypercubes. When missing data are mapped to a single value, the overlaps associated with the index structure increase.

One technique to deal with this issue is to somehow randomize the values assigned to missing data so pruning potential results when traversing the index

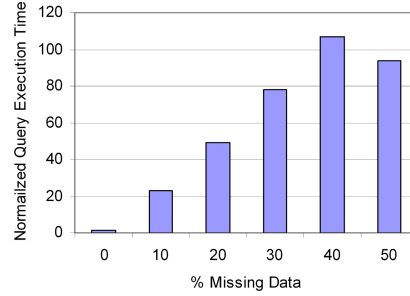


Fig. 1. Normalized Query Execution Time versus Percent Missing Data, Query Selectivity = 25%, 2-D Data Set

structure is not compromised. However, it becomes necessary to transform the initial query involving k attributes into 2^k subqueries. This is because there are 2^k possible combinations of missing and non-missing values among the k attributes in the search key. Therefore there are 2^k subspaces where query matching data can reside, and all of them must be searched. This fact causes query execution performance to become exponentially worse with respect to query dimensionality. Lastly, as described in [15] all hierarchical multi-dimensional index structures break down after a certain number of dimensions indexed.

Space partitioning multi-dimensional indexing techniques would also suffer from the same weaknesses in the presence of missing data. Records with missing data values would get mapped to lesser-dimensional spaces, and the full benefit of data space partitioning would not be realized. Again, partitioning the data space beyond a certain number of dimensions has limitations as discussed in [15].

Data repositories need techniques for indexing multi-dimensional data that work well in the presence of missing data. Further benefit is derived if the techniques also work for databases with higher dimensionality than can be achieved effectively using hierarchical or data partitioning indexes. The objective of this paper is to facilitate efficient access to and define query execution for databases with missing data in a way that even works well when the database dimensionality is high. The techniques introduced are evaluated in terms of performance against a number of parameters including database dimensionality, missing data frequency, query selectivity, and query semantics (whether missing data indicates a query match or not).

Contributions of this paper include the following:

1. Efficiently indexing databases with missing data using variations of bitmaps and VA-Files.
2. Demonstrating that missing data not only causes semantic problems but also degradation in the performance of queries.
3. Formalization of query processing operations for the proposed techniques in the presence of missing data.

4. Insights into the environments appropriate for each technique. Although bitmaps and quantization (VA-Files) have been extensively studied, and their applications are similar, we know of no work that compares and contrasts them.
5. Empirical study and evaluation of results over several analysis parameters.

The rest of this paper is organized as follows: Section 2 discusses related work, Section 3 defines the problem addressed in this paper, Section 4 describes the proposed solutions and Section 5 presents the experimental results. Finally, we conclude in Section 6 and provide directions for future work.

2 Related Work

Missing Data. Although databases commonly deal with or contain missing data, relatively little work has been performed for this topic. Formal definitions for imperfect databases, of which databases with missing data is a subset, and database operations are provided in [21]. Two techniques for indexing databases with missing data are introduced and evaluated in [12]. This is the only paper we are aware of that focuses on indexing missing data. These are the bitstring augmented method and the multiple one-dimensional one-attribute indexes technique, called MOSAIC.

For the bitstring-augmented index, the average of the non-missing values is used as a mapping function for the missing values. The goal is to avoid skewing the data by assigning missing values to several distinct values. However, by applying this method it becomes necessary to transform the initial query involving k attributes into 2^k subqueries, making the technique infeasible for large k . MOSAIC is a set of B+-Trees where missing data is mapped to a distinguished value. Similarly to the previous method, it becomes necessary to transform the initial query involving k attributes into $2k$ subqueries, one for each attribute.

What makes MOSAIC perform better than the Bitstring-Augmented index for point queries is that it uses independent indices for each dimension. However, by using several B+-Trees the query has to be decomposed and intersection and union operations need to be performed to obtain the final result. Queries that could gain a greater performance benefit by utilizing multiple-dimension indexes would not achieve it using this technique. Therefore, this method may not be useful for multiple-dimension range queries, or other queries where the number of matches associated with a single dimension is high.

Our work differs from [12] in that we introduce and evaluate techniques that do not suffer the same weaknesses as their techniques. In our approach the query need not be transformed into exponential number of queries and no extra expensive computation, such as set operations, needs to be performed in order to obtain the final result set. Moreover, even though VA-File is not a hierarchical index it benefits from pruning multiple dimensions in one pass through the structure. In addition, our solution using bitmaps and VA-Files is also scalable with respect to the data dimensionality.

Bitmaps. The topic of bitmap indexes was introduced in [10]. Several bitmap encoding schemes have been developed, such as equality [10], range [5], interval [5], and workload and attribute distribution oriented [9]. Several commercial database management systems use bitmaps [11, 3, 7]. Numerous performance evaluations and improvements have been performed over bitmaps [4, 17, 13, 8, 18, 19, 5, 20]. While the fast bitwise operations afforded by bitmaps are perhaps their biggest advantage, a limitation of bitmaps is the index size. Several compression techniques have been proposed [2, 16, 1, 13] to reduce the bitmap index size. Some of the most popular compression techniques such as Byte-Aligned Bitmap Code (BBC) [2] and Word Aligned Hybrid (WAH) code [16], use a hybrid between the run-length encoding and the literal scheme to compress the bitmap.

VA-Files. The motivation for VA-files is introduced in [15]. This paper showed theoretical limitations for the classes of data and space partitioning indexing techniques with respect to dimensionality. Since reading all database pages becomes unavoidable when the number of indexed dimensions is high, the authors suggest reading a much smaller approximate version, or vector approximation (VA), of each record in the database. An initial read approximately answers queries, and actual database pages are read to determine the exact query answer. VA-files are more thoroughly described in [14].

To the best of our knowledge this is the first paper that compares and contrasts bitmaps and VA-files and discusses them together and the first paper in which these techniques are used to index incomplete databases.

3 Problem Definition

Let D be a database with a schema of the form $(A_1, A_2, \dots, A_d)$. D is said to be incomplete if tuples in it are allowed to have missing attribute values. Without loss of generality, assume the domain of the attribute values is the integers from 1 to C_i , where C_i is the cardinality of attribute A_i . We assume that data retrieval is based on a k -dimensional search key, where k is less than or equal to d .

In range queries, a lower- and upper-bound is specified for each attribute in the search key. Each interval in the query is represented as $v_1 \leq A_i \leq v_2$, where v_1 and v_2 are between 1 and C_i . The query is said to be a point query if all lower bounds are equal to the corresponding upper bound for each attribute in the search key.

Given a range query Q with a k -dimensional search key, we have two ways to compute the results for Q . When missing data is considered to be a query match, a tuple t in the database is said to be an answer for Q if every attribute of t which appears in the search key that is not a missing value falls in the corresponding range defined in the query or is a missing value. When missing is not a match, a tuple t in the database is said to be an answer for Q if every attribute of t which appears in the search key is not missing and falls in the corresponding range defined in the query.

The performance of a query can be characterized by the time it takes to perform the query and the accuracy of the result. For this work we only consider techniques that provide accurate query results. The time it takes to perform a query when an index is used is made up of the time to read the index (if the index does not already reside in memory), the time to execute the query over the index, and the time to read the database pages indicated by the index. The goal of this work is to propose indexing techniques that exhibit better performance than existing techniques and sequential scan when the database attributes that are specified in a search key have missing data.

When measuring query performance we consider two metrics: index size and query execution time. Index size is simply measured as the size of the requisite index files on disk. It is indicative of the time required to initially load the index structures. Although this metric is not as critical for static read-only databases with ample disk-space available, it becomes important as database updates become more frequent or available disk space becomes limited. Query execution time is measured in milliseconds for a query set. Given that the indexes are in memory, this measurement indicates the time required to process a set of queries and arrive at a set of pointers to records in the database that *could* match the query criteria.

4 Proposed Solutions

Our proposed solutions are to apply the techniques of bitmap indexes and vector-approximation (VA) files modified appropriately to account for missing data and to execute the query according to the query's semantics. The reason is that we want to independently index each dimension and execute queries efficiently without needing to perform expensive operations to obtain the final result. Bit operations for bitmaps provide fast computation and VA-Files provide pruning in multiple dimensions at the same time using cheap comparisons.

4.1 Bitmap Indexes

We base one solution for the efficient access of incomplete databases on bitmap indexes. In the bitmap index context, records are represented by a bit string. Each attribute A_i would be represented by at most C_i bits of the string where C_i is the cardinality of A_i , i.e. the number of distinct non-null values among all records for attribute A_i . A bitmap is a column wise representation of each position of the bit string. Each bitmap would have n bits where n is the number of records in the dataset. Given a dataset $D = (A_1, A_2, \dots, A_d)$ for each A_i attribute we build a certain number of bitmaps depending on C_i . To handle missing data using bitmaps, we map missing values to a distinct value, i.e. 0. By doing this we are increasing the number of bitmaps for each attribute with missing data by 1. While mapping missing data to a distinct value fails for multi-dimensional indexes, it is acceptable for bitmaps because the attributes are indexed independently and we are not creating an exponential number of subspaces that must be searched to answer a query.

Let's denote the bitvectors or bitmap vectors for attribute A_i by $B_{i,j}$ where $0 \leq j \leq C_i$ if A_i has missing values and $1 \leq j \leq C_i$ otherwise. $B_{i,0}$ represents the bitvector for missing values.

Let's denote by $B_{i,j}[x]$ where $1 \leq x \leq n$ the bit value for record x in the bitmap for attribute A_i and value j .

Using bitmap indices, queries are executed by performing bit operations over the relevant bitmaps. OR, XOR, AND and NOT are commonly used.

An important aspect of a bitmap index is the type of encoding of the records. We explore two alternatives: equality and range encoding.

4.2 Bitmap Equality Encoding (BEE)

Using equality encoded bitmaps, bit $B_{i,j}[x]$ is 1 if record x has value j for attribute A_i and 0 otherwise. Using this encoding, if $B_{i,j}[x] = 1$ then $B_{i,k}[x] = 0$ for all $k \neq j$. If attribute A_i has missing values, we add the bitmap $B_{i,0}$ that behaves in the same manner explained above.

Adding an extra bitmap for each attribute with missing data is not a major burden with few records or few dimensions, but when we consider 1,000,000 records with 100 dimensions we are effectively adding 100,000,000 bits to our index which correspond to approximately 12 MB in size.

An intuitive solution that could be used to encode missing data without adding an extra bitmap would be to use different encodings depending on whether missing data is a match or not. In this alternative, when missing is a match we make $B_{i,j}[x] = 1$ for all j if record x has missing data in attribute A_i ; and when missing is not a match, we make $B_{i,j}[x] = 0$ for all j if record x has missing data in attribute A_i .

However, there are some problems with this approach. We will need to perform more bitmap operations when we use the NOT operator. The reason is that when we negate a bitmap when missing data is considered to be a query match, the resulting bitmap would have 0's for the missing records. In order to recover the records with missing data we will need to AND together two bit columns. We then need to OR that result with the original negated bitmap to arrive at a correct final result. When missing data does not imply a query match, we would need to OR together two bit columns to ensure we are eliminating the records with missing values and then AND this result with the negated bitmap to get correct results. Using this approach, it would also be impossible to distinguish between missing values and a real value when the cardinality of the attribute is 1. In addition, by making all bits 1 for the attribute when missing is a match we interrupt the runs of 0s and compression decreases dramatically for the attribute bitmaps.

Empirically, we realized that after compression using WAH, the addition of an extra bitmap to handle missing data did not introduce much overhead. For the same example of 1,000,000 records with 100 dimensions, and assuming 10,000 records with missing data, each bitmap for missing values would have approximately a compression ratio of 0.47 and overall the compression ratio for the dataset would also improve.

Table 1. Equality encoded with missing data

Record	Value	$B_{1,0}$	$B_{1,1}$	$B_{1,2}$	$B_{1,3}$	$B_{1,4}$	$B_{1,5}$
1	5	0	0	0	0	0	1
2	2	0	0	1	0	0	0
3	3	0	0	0	1	0	0
4	missing	1	0	0	0	0	0
5	4	0	0	0	0	1	0
6	5	0	0	0	0	0	1
7	1	0	1	0	0	0	0
8	3	0	0	0	1	0	0
9	missing	1	0	0	0	0	0
10	2	0	0	1	0	0	0

Table 2. Bitmap indices

Bitmap Vector	Value
$B_{1,0}$	0001000010
$B_{1,1}$	0000001000
$B_{1,2}$	0100000001
$B_{1,3}$	0010000100
$B_{1,4}$	0000100000
$B_{1,5}$	1000010000

$$v_1 \leq A_i \leq v_2 =$$

$$\left\{ \begin{array}{ll} \left(\bigcup_{j=v_1}^{v_2} B_{i,j} \right) \vee B_{i,0} & \text{if } v_2 - v_1 \leq \lfloor C_i/2 \rfloor \\ \bigcup_{j=1}^{v_1-1} B_{i,j} \vee \bigcup_{j=v_2+1}^{C_i} B_{i,j} & \text{otherwise} \end{array} \right. \quad \left\{ \begin{array}{ll} \left(\bigcup_{j=v_1}^{v_2} B_{i,j} \right) & \text{if } v_2 - v_1 \leq \lfloor C_i/2 \rfloor \\ \bigcup_{j=1}^{v_1-1} B_{i,j} \vee \bigcup_{j=v_2+1}^{C_i} B_{i,j} \oplus B_{i,0} & \text{otherwise} \end{array} \right.$$

(a) Missing Data is a Match (b) Missing Data is not a Match

Fig. 2. Interval Evaluation for Bitmap Equality Encoding

Query Execution With equality encoded bitmaps a point query is executed by ANDing together the bit vectors corresponding to the values specified in the search key. Bitmap Equality Encoded are optimal for point queries [5]. However, with missing data when missing data means a query match we need to use two bitmaps instead of one to answer the query, i.e. the bitmap corresponding to the value queried and the one for missing values.

Range queries are executed by first ORing together all bit vectors specified by each range in the search key and then ANDing the answers together. If the query range for an attribute queried includes more than half of the cardinality

then we execute the query by taking the complement of the ORed bitmaps that are not included in the range query.

With our approach we execute the query differently depending on whether missing data is a query match or not. Figure 2(a) shows how a query interval for one attribute is evaluated when missing data implies a query match. Figure 2(b) shows the same evaluation when missing data is not a match. The query execution time is a function of the number of bitvectors used to answer the query. The number of bitvectors used in the worst case to evaluate a single interval in the query is equal to $\min(AS_i, 1 - AS_i) * C_i + 1$ where AS_i is the attribute selectivity of attribute A_i for this query.

4.3 Bitmap Range Encoding (BRE)

For range encoded bitmaps, bit $B_{i,j}[x]$ is 1 if record x has a value that is less than or equal to j for attribute A_i and 0 otherwise. Using this encoding if $B_{i,j}[x] = 1$ then $B_{i,k}[x] = 1$ for all $k > j$. In this case the last bitmap B_{i,C_i} for each attribute A_i is all 1s. Thus, we drop this bitmap and only keep $C_i - 1$ bitmaps to represent each attribute. If attribute A_i has missing values we add the bitmap $B_{i,0}$ which has $B_{i,0}[x] = 1$ if record x has a missing value for attribute A_i . Also in this case $B_{i,j}[x] = 1$ for all j . We are treating missing data as the next smallest possible value outside the lower bound of the domain, in our case, the value 0. In total the set of bitmaps required to represent attribute A_i with missing values is C_i .

We also tried another kind of encoding in which instead of making missing data the smallest value we consider the extra bitmap to be a flag indicating whether the data is missing. In this alternative, if record x has a missing value for attribute A_i , $B_{i,0}[x] = 1$ and $B_{i,j}[x] = 0$ for all $j > 0$. However, by making $B_{i,C_i}[x] = 0$ when x has a missing value for attribute A_i , we can no longer drop it. This will effectively increase the number of bitmaps for attribute A_i to $C_i + 1$, and will not provide any advantage to the query evaluation logic.

Query Execution. With range encoded bitmaps the bitmaps used and the operations performed to execute a query depend on the range being queried. We identify three scenarios, depending on whether the range includes the minimum

$$v_1 \leq A_i \leq v_2 =$$

$\left\{ \begin{array}{ll} B_{i,1} & \text{if } v_2 = v_1 = 1 \\ B_{i,v_1} \oplus B_{i,v_1-1} \vee B_{i,0} & \text{if } 1 < v_1 = v_2 < C_i \\ \overline{B_{i,C_i-1}} \vee B_{i,0} & \text{if } 1 < v_1 = v_2 = C_i \\ B_{i,v_1-1} \vee B_{i,0} & \text{if } 1 < v_1 < C_i, \quad v_2 = C_i \\ B_{i,v_2} & \text{if } v_1 = 1, \quad 1 < v_2 < C_i \\ B_{i,v_2} \oplus B_{i,v_1-1} \vee B_{i,0} & \text{otherwise} \end{array} \right.$	$\left\{ \begin{array}{ll} B_{i,1} \oplus B_{i,0} & \text{if } v_2 = v_1 = 1 \\ B_{i,v_1} \oplus B_{i,v_1-1} & \text{if } 1 < v_1 = v_2 < C_i \\ \overline{B_{i,C_i-1}} & \text{if } 1 < v_1 = v_2 = C_i \\ B_{i,v_1-1} & \text{if } 1 < v_1 < C_i, \quad v_2 = C_i \\ B_{i,v_2} \oplus B_{i,0} & \text{if } v_1 = 1, \quad 1 < v_2 < C_i \\ B_{i,v_2} \oplus B_{i,v_1-1} & \text{otherwise} \end{array} \right.$
(a) Missing Data is a Match	(b) Missing Data is not a Match

Fig. 3. Interval Evaluation for Bitmap Range Encoding

Table 3. Sample data using Range encoding

Record	Value	$B_{1,0}$	$B_{1,1}$	$B_{1,2}$	$B_{1,3}$	$B_{1,4}$	$B_{1,5}$
1	5	0	0	0	0	0	1
2	2	0	0	1	1	1	1
3	3	0	0	0	1	1	1
4	missing	1	1	1	1	1	1
5	4	0	0	0	0	1	1
6	5	0	0	0	0	0	1
7	1	0	1	1	1	1	1
8	3	0	0	0	1	1	1
9	missing	1	1	1	1	1	1
10	2	0	0	1	1	1	1

Table 4. Range Encoded Bitmap indices

Bitmap Vector	Value
$B_{1,0}$	0001000010
$B_{1,1}$	0001001010
$B_{1,2}$	0101001011
$B_{1,3}$	0111001111
$B_{1,4}$	0111101111

value, or includes the maximum value, or is within the domain and includes neither the minimum or maximum.

Figures 3(a) and 3(b) show how the interval is evaluated for a single query attribute when missing data implies a match or does not imply a match respectively.

The first three conditions in Figures 3(a) and 3(b) refer to point queries. The other three refer to range queries.

In the presence of missing data, range encoded bitmaps are more efficient for range queries than equality encoded bitmaps in all but extreme cases.

In the case where missing data is a query match, we will need to access between 1 and 3 bitvectors per query dimension. In databases without missing data, we would need to access between 1 and 2 bitvectors per query dimension. We introduce some overhead to deal with the missing data case.

In the case where missing data is not a match, we need to access between 1 and 2 bitvectors per query dimension. This is also true for databases without missing data, but there are two conditions, specifically the conditions where the query range includes the minimum domain value, that require 1 extra bitvector access. This is due to the fact that missing values are encoded as 1's in all bitmaps and a XOR operation is required to eliminate missing data from the result set.

4.4 Bitmap Compression

One of the biggest disadvantages of bitmap indices is the amount of space they require. Several compression techniques have been developed in order to reduce bitmap size and at the same time maintain the advantage of fast operations [2, 16, 1, 13].

The two most popular compression techniques are the Byte-aligned Bitmap Code (BBC) [2] and the Word-Aligned Hybrid (WAH) code [16]. BBC stores the compressed data in Bytes while WAH stores it in words. WAH is simpler because it only has two types of words: literal words and fill words. In our implementation it is the most significant bit that indicates the type of word we are dealing with. Let w denote the number of bits in a word, the lower $(w-1)$ bits of a literal word contain the bit values from the bitmap. If the word is a fill, then the second most significant bit is the fill bit, and the remaining $(w-2)$ bits store the fill length. WAH imposes the word-alignment requirement on the fills. This requirement is key to ensure that logical operations only access words.

We chose WAH over BBC because the bit operations over the compressed WAH bitmap file are faster than BBC (2-20 times) [16]. However, we do sacrifice space since BBC gives better compression ratio.

Logical operations are performed over the compressed bitmaps resulting in another compressed bitmap.

4.5 VA-Files

For traditional VA-files, data values are approximated by one of 2^b strings of length b bits. A lookup table provides value ranges for each of the 2^b possible representations. For each attribute A_i in the database we use b_i bits to represent 2^{b_i} bins that enclose the entire attribute domain. In general $b_i \ll \lg C_i$ when the cardinality is high. We made $b_i = \lceil \lg(C_i + 1) \rceil$. For our purposes, we use $2^b - 1$ possible representations for data values and we use a string of b 0's to represent missing data values. A VA-file lookup table relates attribute values to the appropriate bin number. For VA-files we make a modification to the query based on the query semantics. For a range query where missing data is not a query match, we look for matches over the range of bins returned by the lookup table. In the case where missing data means a query match, we also include those records in the all 0's bin as a query match.

Tables 5 and 6 show a simple example of a VA-file using our missing data modification. If we perform a query “return all records where value is 4 or 5”,

Table 5. Database and VA-File representations

Record Number	Data Value	VA-File Representation
1	6	11
2	1	01
3	3	10
4	missing	00

Table 6. VA-file representations and data ranges

VA-File Representation	Range
00	missing
01	1-2
10	3-4
11	5-6

Table 7. Synthetic and Census Datasets Distribution

Synthetic Dataset						
Card	% of Missing Data					Total Columns
	10	20	30	40	50	
2	10	10	10	10	10	50
5	10	10	10	10	10	50
10	20	20	20	20	20	100
20	20	20	20	20	20	100
50	20	20	20	20	20	100
100	10	10	10	10	10	50
Total	90	90	90	90	90	450

Census Dataset						
Card	% of Missing Data					Total Columns
	0	≤10	≤50	≤90	>90	
<10	11	0	2	2	0	15
10-50	7	2	3	5	4	21
51-100	2	0	1	2	2	7
>100	0	0	1	2	2	5
Total	20	2	7	11	8	48

our VA-file technique will return the records in bins 00, 10, 11 as approximate answers in the case where missing data is a match. A filtering step would verify that record 1 does not answer the query. In the case where missing data is not a match, only the records in bins 10 and 11 would be returned in the first step.

Query translation is simple. When missing data implies a match, a range query in the form $v_1 \leq A_i \leq v_2$ is converted to $(VA(v_1) \leq VA(A_i) \leq VA(v_2)) \vee (VA(A_i) = 0^b)$, where $VA(x)$ is a function that converts values to their representative VA-file bit representation and b is the number of bits used to define an attribute.

These techniques are easy to apply and require little or no modification of the queries or query processing. As shown using empirical experiments, they are also scalable in terms of the number of data dimensions.

5 Experiments and Results

5.1 Experimental Framework

We performed experiments using both synthetic and real datasets. By using the synthetic data set we could control analysis parameters individually and gain insights into the behavior of the indexing techniques. We applied the techniques to a real data set to verify the effectiveness of the techniques on real scenarios.

For the synthetic data, we generated a uniformly distributed random dataset set with 450 attributes and 100,000 records. For the set of attributes we varied the cardinality and percent of missing data. Cardinality varied among 2, 5, 10, 20, 50, and 100 values and percent of missing data among 10, 20, 30, 40, and 50 percent.

The real data is census data with 48 attributes and 463,733 records. The attribute cardinalities widely vary from 2 to 165 (average of 37) and percent of missing data varies from 0% to 98.5% (average of 41%). Table 7 details the distribution for the synthetic and the real dataset.

We implemented query executors for both bitmaps and VA-Files in Java. We ran 100 queries for each type of experiment. Queries were executed in both scenarios when missing data is a query match and when missing data is not a query match. Since the graphs look very similar in both scenarios we present only results for queries executed where missing data is a match.

Given that we used the same precision (100%) for our implementations we compared bitmap indices and VA-Files in terms of:

- **Index Size.** Index Size is an important factor in any indexing technique. We are interested in indices that can fit into memory to ensure fast query execution without the overhead introduced when reading from disk.
- **Query Execution Time.** Query Execution Time is the time required to produce a query result set.

5.2 Index Size

In this section we evaluate how the attribute cardinality and the percentage of missing data affects index size.

Attribute Cardinality. For cardinality less than 10 there is not much room for compression and the index size is equal for both types of bitmap encoding and is not sensitive to the percent of missing data. For equality encoded bitmaps, as the attribute cardinality increases the compression ratio improves considerably, however, at the same time, bitmaps index size increases linearly with cardinality. For VA-Files the index grows very slowly with cardinality given our current quantization strategy. Index sizes are presented for attributes with 10% missing data in Figure 4(a). As can be seen, BRE does not benefit from WAH compression.

With real data, compression rate is highly variable with respect to attribute cardinality. Since real data can be far from uniform, an attribute that has low cardinality but frequently has one value can achieve high compression ratios. With our set of real data, those attributes which have cardinalities of between 1 and 10 and are not missing any data have a compression ratio between 0.002 and 1.03 using equality encoding and between 0.001 and 0.82 using bitmap range encoding. The wide range is attributable to the bit density (ratio of 1's) in the bit columns. As the bit density approaches 1 or 0, the compression ratio improves. Therefore, if one particular value is frequent, then the bit density for that value's column is close to 1 yielding good compression ratio for that column and the bit density for all other bit columns is close to 0, which results in good compression ratio for them.

Percent of Missing Data. For equality encoded bitmaps, as the percent of missing data increases the compression ratio decreases making the index smaller.

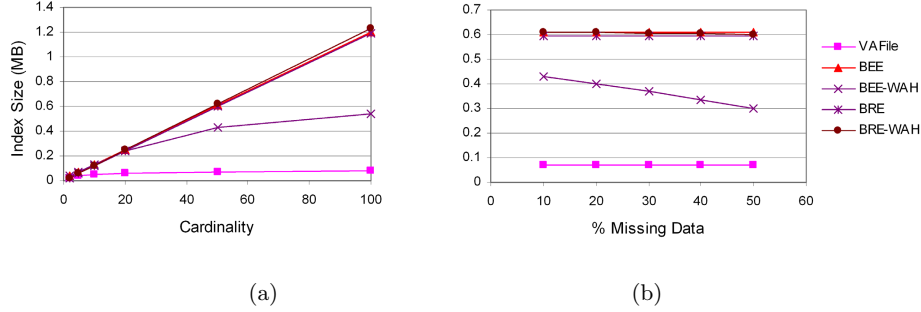


Fig. 4. Index Size Versus (a) Cardinality and (b) Percent of Missing Data

Range encoding does not get significant compression using WAH code. VA-File is not sensitive to the presence of missing data and its size is independent of it. In any case the index size for VA-Files is much smaller than bitmaps. Index sizes are presented for cardinality 50 in Figure 4(b).

Good compression is also obtained on the real dataset when an attribute has a high occurrence of missing data. The missing data bit column has a bit density close to 1 and all other columns are close to 0. This leads to very good compression ratios for equality encoded bitmaps (between 0.01 and 0.09 for each of the 8 attributes in our real data set which have more than 90% missing data) and decent compression ratios for range encoded bitmaps (between 0.11 and 0.44). Overall, this real data set had an equality encoded bitmap compression ratio of 0.17 and a range encoded bitmap compression ratio of 0.70.

5.3 Query Execution Time

To measure the effect of the various parameters over the query execution time of the 100 queries we needed to have control over the global query selectivity, i.e. the number of records that match the given query. The following formula relates Global Selectivity (GS), Attribute Selectivity ($AS = (v_2 - v_1 + 1)/C_i$) and Percent of Missing Data (P_{m_i}) of all the attributes involved in the queries:

$$GS = \prod_{i=1}^k ((1 - P_{m_i})AS_i + P_{m_i}),$$

where k is the number of dimensions involved in the query. In order to simplify this formula we assume equal attribute selectivity on all the attributes in the query. By doing this, individual attribute selectivities are easy to compute but we lose some precision on the global query selectivity. To measure query execution time we fixed the global query selectivity to 1 percent. Plugging in different values for the parameters into $GS = [(1 - P_m)AS + P_m]^k$ we compute the attribute selectivity for each attribute in the query. Note that the granularity of attribute selectivity is limited by C_i . In general, our estimate was very close to 1 percent but sometimes the actual global query selectivity went up to 3

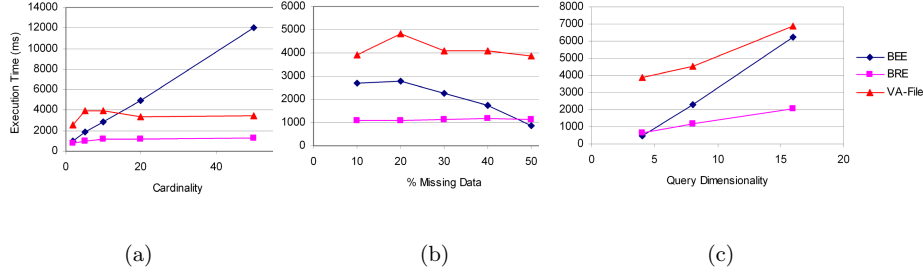


Fig. 5. Query Execution Time Versus (a) Cardinality, (b) Percent of Missing Data, and (c) Query Dimensionality

percent. Note that when we make the global selectivity constant and increase the percent of missing data, the attribute selectivity decreases. We tested the effect of attribute selectivity, percent of missing data, and query dimensionality against query execution time.

Attribute Cardinality. Figure 5(a) shows the query execution time of 100 queries over attributes with 10 percent missing data and various cardinalities. Also in this case the execution time for BRE and VA-Files remains somewhat constant with BRE being faster than VA-Files. For BEE, the execution time is linear since the number of bitmaps used to answer the queries depends on the cardinality of the attribute and its selectivity.

Percent of Missing Data. Figure 5(b) shows the results of these experiments for attributes with cardinality 10. For equality encoded bitmaps, the execution time decreases when the percent of missing data increases. This is because when we make the global selectivity constant and increase the percent of missing data, the attribute selectivity decreases and the number of bitmaps used in the query execution depends on the attribute selectivity for this kind of encoding. For range encoded bitmaps, the execution time remains somewhat constant. The small variations are due to the possibility of using between 1 and 3 bitmaps per dimension over the query execution. It turns out that as the percent of missing data increases the number of bitmaps used per dimension gets closer to 3. For VA-Files, the execution time is also somewhat constant. The variations are due to the actual global selectivity for cardinality 10 and 8 dimensions in the query. For cardinality 10 and 50 % missing data the global selectivity is 0.84%, for 30 and 40 is 1.28%, but for 20 is 1.7%. In general, BRE executes range queries faster than the other two. The only case in which BEE performs better than BRE is at 50% missing when the attribute selectivity is 10% and the range query becomes a point query.

Query Dimensionality. Figure 5(c) shows the query execution of 100 queries over attributes with cardinality 10 and 30 percent of missing data. For all indices the execution is linear in the number of query dimensions. BRE grows very slowly since we are only using between 1 and 3 bitmaps per query dimension. BEE grows

much faster since as we increase the number of dimensions with this percent of missing data the attribute selectivity get closer to 50 %. For smaller percents of missing data and same cardinality the attribute selectivity is greater than 50 %, around 70 % so effectively we only access the 30 % of the bitmaps and therefore the execution time does not increase linearly. For VA-Files the execution time also increases with the query dimensionality.

Results on Real Data. Experiments using this real data set yielded several conclusions. For this data set, the bitmap solutions were significantly faster than the VA-File solution (3 to 10 times faster). This was because the skewness of this particular data set allowed for very good compression of the bitmaps and while the VA-file implementation had to operate over about 500,000 vector approximations of the records, the bitmap implementations performed bit operations over substantially fewer words. The average compression ratio for the equality encoding bitmaps was 0.17 (with 23 attributes compressing to less than 0.1 times their original size). The average compression ratio for the range encoding bitmaps was 0.7 (with 18 attributes compressing to less than 0.5 times their original size and only 3 attributes not compressing at all).

Also of note is that whereas the presence of missing data can introduce a degradation of a couple of orders of magnitude in hierarchical multiple-dimension indexes as shown in the motivating example, there is not a large degradation associated with the presence of missing data using these techniques. Performance can be as high as two times slower with our techniques, and this is attributable to extra bit operations required to handle the missing data.

In our experiments with real data, the range encoded bitmaps performed faster than the equality encoded bitmaps. In these experiments we used range queries over 20% of the queried attribute possible values and would expect this result since range encoded bitmaps are tailored for range queries.

6 Conclusions

The techniques presented in this paper are easy to apply and allow the effective indexing of missing data. As opposed to traditional hierarchical indexing structures and previously proposed missing data indexing techniques, these techniques exhibit linear performance for query execution time with respect to database and query dimensionality. This is done by essentially indexing attributes independently. Our solutions take advantage of this independence by handling missing data for each attribute, and still maintain the linear performance associated with respect to dimensionality that bitmaps and VA-files have been known for.

These techniques exhibit a tradeoff between execution time and indexing space. The bit operations used to evaluate queries for bitmaps are fast, but the space required to represent an exact bitmap can be much higher than a corresponding exact VA-file.

The range encoded bitmaps typically offer the best time performance but, at least using the techniques we used, can not be compressed as much as equality

encoded bitmaps. They typically perform faster because there is a limit on the number of bit operations that must be performed to evaluate a query for each dimension.

Equality encoded bitmaps perform a maximum of $C/2+1$ bit operations per query dimension and can perform faster than range encoded bitmaps for point queries or range queries with small ranges. Equality encoded bitmaps can be compressed much more than range encoded bitmaps.

VA-files offer the least size to represent the same information offered by bitmaps, but the operations performed are not bit operations, they usually do not operate as fast as the range encoded bitmaps.

There are several areas in which the techniques proposed here could be improved. The biggest weakness of the range encoded bitmaps is the inability to compress them. We would like to explore techniques such as BBC compression and row reordering in order to achieve more compression of these bitmaps. The same modifications made to the basic VA-file to account for missing data could also be applied to the VA-plus file, a technique to quantize skewed data sets described in [6].

Acknowledgement. This research was supported by Department of Energy (DOE) grant DE-FG02-03ER25573 and National Science Foundation (NSF) grant CNS-0403342.

References

1. S. Amer-Yahia and T. Johnson. Optimizing queries on compressed bitmaps. In *The VLDB Journal*, pages 329–338, 2000.
2. G. Antoshenkov. Byte-aligned bitmap compression. In *Data Compression Conference*, Nashua, NH, 1995. Oracle Corp.
3. G. Antoshenkov and M. Ziauddin. Query processing and optimization in oracle rdb. *The VLDB Journal*, 1996.
4. C.-Y. Chan and Y. E. Ioannidis. Bitmap index design and evaluation. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 355–366. ACM Press, 1998.
5. C.-Y. Chan and Y. E. Ioannidis. An efficient bitmap encoding scheme for selection queries. *SIGMOD Rec.*, 28(2):215–226, 1999.
6. H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A. E. Abbadi. Vector approximation based indexing for non-uniform high dimensional data sets. In *Proceedings of the ninth international conference on Information and knowledge management*, pages 202–209. ACM Press, 2000.
7. S. Inc. *Sybase IQ Indexes.*, chapter Sybase IQ Release 11.2 Collection, chapter 5. Sybase Inc., March 1997.
8. T. Johnson. Performance measurements of compressed bitmap indices. In *Proceedings of the 25th International Conference on Very Large Data Bases*, pages 278–289. Morgan Kaufmann Publishers Inc., 1999.
9. N. Koudas. Space efficient bitmap indexing. In *Proceedings of the ninth international conference on Information and knowledge management*, pages 194–201. ACM Press, 2000.

10. P. O'Neil and D. Quass. Improved query performance with variant indexes. In *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 38–49. ACM Press, 1997.
11. P. E. O'Neil. Model 204 architecture and performance. In *Proceedings of the 2nd International Workshop on High Performance Transaction Systems*, pages 40–59. Springer-Verlag, 1989.
12. B. C. Ooi, C. H. Goh, and K.-L. Tan. Fast high-dimensional data search in incomplete databases. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 357–367. Morgan Kaufmann Publishers Inc., 1998.
13. K. Stockinger. Bitmap indices for speeding up high-dimensional data analysis. In *Proceedings of the 13th International Conference on Database and Expert Systems Applications*, pages 881–890. Springer-Verlag, 2002.
14. R. Weber and S. Blott. An approximation based data structure for similarity search, 1997.
15. R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24th International Conference on Very Large Databases*, pages 194–205, 1998.
16. K. Wu, E. Otoo, and A. Shoshani. Compressing bitmap indexes for faster search operations. In *SSDBM*, 2002.
17. K. Wu, E. Otoo, and A. Shoshani. On the performance of bitmap indices for high cardinality attributes. Technical Report LBNL-54673, Lawrence Berkeley National Laboratory, March 2004.
18. K. Wu, E. J. Otoo, and A. Shoshani. A performance comparison of bitmap indexes. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 559–561. ACM Press, 2001.
19. K. Wu, E. J. Otoo, A. Shoshani, and H. Nordberg. Notes on design and implementation of compressed bit vectors. Technical Report LBNL PUB-3161, Lawrence Berkeley National Laboratory, 2001.
20. M.-C. Wu. Query optimization for selections using bitmaps. In *Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, pages 227–238. ACM Press, 1999.
21. E. Zimanyi. *Incomplete and Uncertain Information in Relational Databases*. PhD thesis, Université Libre de Bruxelles, 1992.