Bitmap Index Design and Evaluation

Chee-Yong Chan
Department of Computer Sciences
University of Wisconsin-Madison
cychan@cs.wisc.edu

Yannis E. Ioannidis* †
Department of Computer Sciences
University of Wisconsin-Madison
yannis@cs.wisc.edu

Abstract

Bitmap indexing has been touted as a promising approach for processing complex adhoc queries in read-mostly environments, like those of decision support systems. Nevertheless, only few possible bitmap schemes have been proposed in the past and very little is known about the space-time tradeoff that they offer. In this paper, we present a general framework to study the design space of bitmap indexes for selection queries and examine the disk-space and time characteristics that the various alternative index choices offer. In particular, we draw a parallel between bitmap indexing and number representation in different number systems, and define a space of two orthogonal dimensions that captures a wide array of bitmap indexes, both old and new. Within that space, we identify (analytically or experimentally) the following interesting points: (1) the time-optimal bitmap index; (2) the space-optimal bitmap index; (3) the bitmap index with the optimal space-time tradeoff (knee); and (4) the time-optimal bitmap index under a given disk-space constraint. Finally, we examine the impact of bitmap compression and bitmap buffering on the space-time tradeoffs among those indexes. As part of this work, we also describe a bitmap-index-based evaluation algorithm for selection queries that represents an improvement over earlier proposals. We believe that this study offers a useful first set of guidelines for physical database design using bitmap indexes.

1 Introduction

While the query performance issues of on-line transaction processing (OLTP) systems have been extensively studied [7] and are pretty much well-understood, the state-of-the-art for Decision Support Systems (DSS) is still evolving as indicated by the growing active research in this area [3]. Current database systems, which are optimized mainly for OLTP applications, are not suitable for DSS applications due to their different requirements and workload [6]. In particular, DSS operate in read-mostly environments, which are dominated by complex adhoc queries that have high selectivity factors (i.e., large foundsets).

A promising approach to process complex queries in DSS is the use of bitmap indexing [8, 9, 10]. Bitmap manipulation techniques have already been used in some commercial products [12] to speed up query processing: a notable example is Model 204, a pre-relational DBMS from Computer Corporation of America [8]. More recently, various DBMS vendors, including Oracle, RedBrick, and Sybase, have introduced bitmap indexes into their products to meet the performance requirements of DSS applications [5, 6]. In its simplest form, a bitmap index on an indexed attribute consists of one vector of bits (i.e., bitmap) per attribute value, where the size of each bitmap is equal to the cardinality of the indexed relation. The bitmaps are encoded such that the ith record has a value of v in the indexed attribute if and only if the ith bit in the bitmap associated with the attribute value v is set to 1, and the ith bit in each of the other bitmaps is set to 0. This is called a *Value-List index* [10]. An example of a Value-List index for a 12-record relation R is shown in Figure 1, where each column in Figure 1(b) represents a bitmap B^v associated with an attribute value v.

	$\pi_A(R)$	B^8	B^7	B^6	B^5	B^4	B^3	B^2	B^1	B^0
1	3	0	0	0	0	0	1	0	0	0
2	2	0	0	0	0	0	0	1	0	0
3	1	0	0	0	0	0	0	0	1	0
4	2	0	0	0	0	0	0	1	0	0
5	8	1	0	0	0	0	0	0	0	0
6	2	0	0	0	0	0	0	1	0	0
7	2	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	0	0	1
9	7	0	1	0	0	0	0	0	0	0
10	5	0	0	0	1	0	0	0	0	0
11	6	0	0	1	0	0	0	0	0	0
12	4	0	0	0	0	1	0	0	0	0
	(a)					**(b)**				

Figure 1: Example of a Value-List Index. (a) Projection of indexed attribute values with duplicates preserved. (b) Value-List Index.

Consider a high-selectivity-factor query with selection predicates on two different attributes. A conventional OLTP optimizer would generate one of the following three query plans: (P1) a full relation scan, (P2) an index scan (using the predicate with lower selectivity factor) followed by a partial relation scan to filter out the non-qualifying tuples, or (P3) an index scan for each selection predicate, followed by a "merge" of the results from the two index scans. Due to the compact sizes of bitmaps (especially for attributes with low cardinality) and the efficient hardware support for bitmap operations (AND, OR, XOR, NOT), plan (P3) using bitmap indexes is likely to be more efficient than a plan that re-

*Partially supported by the National Science Foundation under Grant IRI-9157368 (PYI Award) and by grants from DEC, IBM, HP, AT&T, Oracle, and Informix.

†Author's present address: Department of Informatics, University of Athens, Hellas (Greece).

quires a partial or full relation scan (plans (P1) and (P2)). A simple cost analysis shows that evaluating plan (P3) with bitmap indexes is more efficient than using the conventional RID-list based indexes for queries with selectivity factor above some threshold. Let N and n be the relation and query result cardinalities, respectively. Assume that each RID is 4 bytes long and that one bitmap is scanned per predicate. In terms of the number of bytes read, using bitmap indexes for plan (P3) is more efficient than using RID-list based indexes if $2\frac{N}{8} \leq 4(2n)$; i.e., $\frac{n}{N} \geq \frac{1}{32}$. Furthermore, operations on bitmaps are more CPU-efficient than merging RID-lists.

Various bitmap indexes [8, 9, 10, 13, 14] have been designed for different query types, including range queries, aggregation queries, and OLAP-style queries. However, as there is no overall best bitmap index over all kinds of queries, maintaining multiple types of bitmap indexes for an attribute may be necessary in order to achieve the desired level of performance. While the gains in query performance using a multiple-index approach might be offset by the high update cost in OLTP applications, this is not an issue in the read-mostly environment of DSS applications. Indeed, the multiple-index approach is adopted by Sybase IQ, a DBMS specifically designed for data warehousing applications, which supports five different types of bitmap indexes, and requires the database to be fully inverted [1]. Maintaining multiple indexes for an attribute, however, further increases the disk space requirement of data warehouse applications. Understanding the space-time tradeoff of the various bitmap indexes is therefore essential for a good physical database design.

In this paper, we study the space-time tradeoff of bitmap indexes for the class of *selection queries*, i.e., queries with predicates of the form "A *op* v", where A refers to the indexed attribute, *op* is one of six comparison operators $\{\leq, \geq, <, >, =, \neq\}$, and v is the predicate constant. We refer to selection predicates where *op* $\in \{=, \neq\}$ as *equality predicates*, and to selection predicates where *op* $\in \{\leq, \geq, <, >\}$ as *range predicates*. Information on bitmap indexes for other types of queries (e.g., star-join and group-by queries) can be found elsewhere [9, 10].

The main contributions in this paper are as follows:

- The presentation of a general framework to study the design space of bitmap indexes for selection queries. The framework not only captures the existing proposed ideas but reveals several other design alternatives as well.
- An analysis of the space-time tradeoff of bitmap indexes; in particular, we identify analytically or experimentally four interesting points in a typical space-time tradeoff graph, as shown in Figure 2:
  - (A) The space-optimal bitmap index.
  - (B) The time-optimal bitmap index under a given space constraint.
  - (C) The bitmap index with the optimal space-time tradeoff, i.e., the knee of the space-time tradeoff graph.
  - (D) The time-optimal bitmap index.
- An experimental study of the effects of bitmap compression on the space-time tradeoff issues.
- An analytical study of the effects of bitmap buffering on the space-time tradeoff issues.
- The proposal of a more efficient evaluation algorithm for bitmap indexes. The new algorithm reduces the number of bitmap operations by about 50% and incurs one less bitmap scan for a range predicate evaluation.

The rest of this paper is organized as follows. In Section 2, we present a framework to study the design space of bitmap indexes for selection queries. Section 3 presents an improved evaluation algorithm for bitmap indexes, including both analytical and experimental performance evaluations. Section 4 presents an analytical cost model for the space-time tradeoff study. Section 5 compares the space-time tradeoff of two basic bitmap encoding schemes. In Section 6, we examine the space-optimal (point (A)) and time-optimal (point (D)) bitmap indexes. In Section 7, we characterize the knee of the space-time tradeoff graph of bitmap indexes (point (C)). Section 8 presents an optimal as well as a heuristic approach to find the time-optimal bitmap index under space constraint (point (B)). In Section 9, we present an experimental study of the impact of bitmap compression on the space-time tradeoff issues. Section 10 presents an analytical study of the effect of bitmap buffering on the space-time tradeoff issues. Finally, we summarize our results in Section 11.

For the rest of this paper, we use the term index to refer to a bitmap index. Due to lack of space, the derivations of analytical results and the proofs of theorems and algorithms' correctness are omitted from this paper; full details are given elsewhere [2].

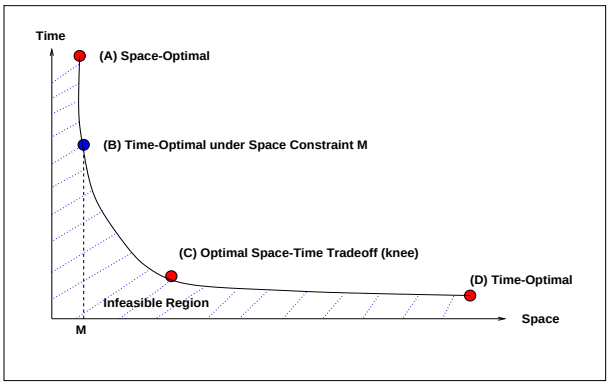


Figure 2: Space-Time Tradeoff Issues

2 Design Space of Bitmap Indexes for Selection Queries

In this section, we present a framework to examine the design space of indexes for selection queries. The framework has been inspired by the work of Wong et. al. [13, 14].

Let C denote the attribute cardinality; i.e., the number of distinct actual values of the indexed attribute. The attribute cardinality is generally smaller than the cardinality of the attribute domain; i.e., the number of all possible values of the indexed attribute. Without loss of generality and to keep the presentation simple, we assume in this paper that the actual attribute values are consecutive integer values from 0 to $C-1$. For the general case where the actual attribute values are not necessarily consecutive, a bitmap index can be built either on the entire attribute domain, or on C consecutive values by mapping each actual attribute value to its rank via a lookup table.

As described in Section 1, the Value-List index is a set of bitmaps, one per attribute value. In other words, if one views this set as a two-dimensional bit matrix (Figure 1(b)), the focus is on the columns. If the focus moves on the rows, however, then the index can be seen as the list of attribute values represented in some particular way. As there is a large number of possible representations for attribute values (integers in this case), this clearly opens up the way for defining a whole host of bitmap indexing schemes. In classifying all these representations, we identify two major orthogonal parameters that define the overall space of alternatives. In particular, considering the Value-List index again, we observe that each attribute value is represented as a single digit (in base-C

arithmetic), this digit being encoded in bits by turning exactly one out of C bits on. The arithmetic we choose for the value representation, i.e., the *decomposition* of the value in digits according to some base, and the *encoding scheme* of each digit in bits are the two dimensions of the space and are analyzed below.

(1) Attribute Value Decomposition Consider an attribute value v and a sequence of $(n-1)$ numbers $< b_{n-1}, b_{n-2}, \ldots, b_1 >$. Furthermore, define $b_n = \left\lceil \frac{C}{\prod_{i=1}^{n-1} b_i} \right\rceil$. Then v can be decomposed into a sequence of n digits $< v_n, v_{n-1}, \ldots, v_1 >$ as follows:

$$\begin{aligned} v &= V_1 \\ &= V_2 b_1 + v_1 \\ &= V_3(b_2 b_1) + v_2 b_1 + v_1 \\ &= V_4(b_3 b_2 b_1) + v_3(b_2 b_1) + v_2 b_1 + v_1 \\ &= \ldots\ldots \\ &= v_n \left(\prod_{j=1}^{n-1} b_j \right) + \ldots + v_i \left(\prod_{j=1}^{i-1} b_j \right) + \ldots + v_2 b_1 + v_1, \end{aligned}$$

where $v_i = V_i \; mod \; b_i$, $V_i = \left\lfloor \frac{V_{i-1}}{b_{i-1}} \right\rfloor$, $1 < i \leq n$, and each digit v_i is in the range $0 \leq v_i < b_i$.

Based on the above, each choice of n and sequence $< b_n, b_{n-1}, \ldots, b_1 >$ gives a different representation of attribute values, and therefore a different index. An index is *well-defined* if $b_i \geq 2$, $1 \leq i \leq n$. The sequence $< b_n, b_{n-1}, \ldots, b_1 >$ is the *base* of the index, which is in turn called a base-$< b_n, b_{n-1}, \ldots, b_1 >$ index. If $b_n = b_{n-1} = \ldots = b_1 \equiv b$, then the base is called *uniform* and the index is called base-b for short. The index consists of n *components*, i.e., one component per digit. Each component individually is now a collection of bitmaps, constituting essentially a base-b_j index.

Let n_i denote the number of bitmaps in the ith component of an index and $\{B_i^{n_i-1}, B_i^{n_i-2}, \ldots, B_i^0\}$ denote the collection of n_i bitmaps that form the ith component. Figure 3 shows a base-$< 3, 3 >$ Value-List index (based on the same 12-record relation in Figure 1). By decomposing a single-component index into a 2-component index, the number of bitmaps has been reduced from 9 to 6.

	$\pi_A(R)$		B_2^2	B_2^1	B_2^0	B_1^2	B_1^1	B_1^0
1	3	$\xrightarrow{1\times 3+0}$	0	1	0	0	0	1
2	2	$\xrightarrow{0\times 3+2}$	0	0	1	1	0	0
3	1	$\xrightarrow{0\times 3+1}$	0	0	1	0	1	0
4	2	$\xrightarrow{0\times 3+2}$	0	0	1	1	0	0
5	8	$\xrightarrow{2\times 3+2}$	1	0	0	1	0	0
6	2	$\xrightarrow{0\times 3+2}$	0	0	1	1	0	0
7	2	$\xrightarrow{0\times 3+2}$	0	0	1	1	0	0
8	0	$\xrightarrow{0\times 3+0}$	0	0	1	0	0	1
9	7	$\xrightarrow{2\times 3+1}$	1	0	0	0	1	0
10	5	$\xrightarrow{1\times 3+2}$	0	1	0	1	0	0
11	6	$\xrightarrow{2\times 3+0}$	1	0	0	0	0	1
12	4	$\xrightarrow{1\times 3+1}$	0	1	0	0	1	0

(a) (b)

Figure 3: Example of a 2-Component Value-List Index (a) Projection of indexed attribute values with duplicates preserved. (b) Base-$< 3, 3 >$ Value-List Index.

(2) Bitmap Encoding Scheme Consider the ith component of an index with attribute cardinality b_i. There are essentially two major schemes to directly encode the corresponding values v_i ($0 \leq v_i \leq b_i - 1$) in bits[1]:

- *Equality Encoding:* There are b_i bits, one for each possible value. The representation of value v_i has all bits set to 0, except for the bit corresponding to v_i, which is set to 1. Clearly, an equality-encoded component consists of b_i bitmaps.
- *Range Encoding:* There are b_i bits again, one for each possible value. The representation of value v_i has the v_i rightmost bits set to 0 and the remaining bits (starting from the one corresponding to v_i and to the left) set to 1[2]. Intuitively, each bitmap $B_i^{v_i}$ has 1 in all the records whose i^{th} component value is less than or equal to v_i. Since the bitmap $B_i^{b_i-1}$ has all bits set to 1, it does not need to be stored, so a range-encoded component consists of $(b_i - 1)$ bitmaps.

Figures 4(b) and (c) show the range-encoded indexes corresponding to the equality-encoded indexes in Figure 1 and Figure 3, respectively.

	$\pi_A(R)$	B^7	B^6	B^5	B^4	B^3	B^2	B^1	B^0
1	3	1	1	1	1	1	0	0	0
2	2	1	1	1	1	1	1	0	0
3	1	1	1	1	1	1	1	1	0
4	2	1	1	1	1	1	1	0	0
5	8	0	0	0	0	0	0	0	0
6	2	1	1	1	1	1	1	0	0
7	2	1	1	1	1	1	1	0	0
8	0	1	1	1	1	1	1	1	1
9	7	1	0	0	0	0	0	0	0
10	5	1	1	1	0	0	0	0	0
11	6	1	1	0	0	0	0	0	0
12	4	1	1	1	1	0	0	0	0

(a) (b)

B_2^1	B_2^0	B_1^1	B_1^0
1	0	1	1
1	1	0	0
1	1	1	0
1	1	0	0
0	0	0	0
1	1	0	0
1	1	0	0
1	1	1	1
0	0	1	0
1	0	0	0
0	0	1	1
1	0	1	0

(c)

Figure 4: Examples of Range-Encoded Bitmap Indexes. (a) Projection of indexed attribute values with duplicates preserved. (b) Single Component, Base-9 Range-Encoded Bitmap Index. (c) Base-$< 3, 3 >$ Range-Encoded Bitmap Index.

Figure 5 shows the two-dimensional design space of indexes for selection queries, with the two existing alternatives, namely, the Value-List index and the Bit-Sliced index [10], classified as shown. The Value-List index, which is the simplest index and is the most commonly implemented, has a single component and is equality-encoded. The Bit-Sliced index [10] has a uniform base and is range-encoded. A base-10 Bit-Sliced index has been used in MODEL 204 for evaluating range predicates [8]. The Bit-Sliced index is also used in Sybase IQ for evaluating range predicates and

[1] We have excluded the third "basic" encoding scheme (the binary encoding scheme) discussed in [13, 14], as it can be characterized in our framework as a base-2 decomposition with one of the two encodings we do present.

[2] Clearly, a symmetric scheme exists as well, where the roles of right and left are exchanged.

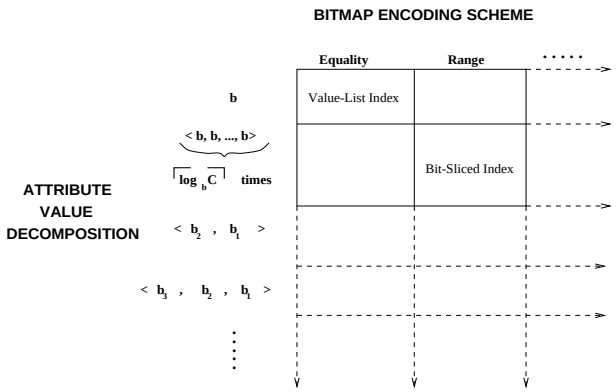


Figure 5: Design Space of Bitmap Indexes for Selection Queries.

performing aggregation [10]. In this paper, we consider all well-defined decompositions, and just the two encoding schemes presented above, which are used in practice.

3 An Improved Evaluation Algorithm for Range-Encoded Bitmap Indexes

An evaluation algorithm for selection queries based on range-encoded indexes has been proposed by O'Neil and Quass (Algorithm 4.3 in [10]), which we referred to as Algorithm `RangeEval`. In this section, we present an improved evaluation algorithm (Algorithm `RangeEval-Opt`) that reduces the number of bitmap operations by about 50% and requires one less bitmap scan for a range predicate evaluation. Given this, in the rest of the paper, the improved evaluation algorithm is used as the basis for the time analysis of range-encoded indexes. Both evaluation algorithms are shown in Figure 6.

Let $\mathcal{B}_0$ and $\mathcal{B}_1$ denote bitmaps with all bits set to 0 and 1, respectively. Also let $\wedge$, $\vee$, and $\oplus$ denote the logical AND, OR, and XOR operations, respectively, and $\overline{B}$ denote the complement of a bitmap B. Depending on the input predicate operator (op), Algorithm `RangeEval` evaluates each range predicate by computing two bitmaps: the B_{EQ} bitmap and either the B_{GT} or B_{LT} bitmap. For example, if the predicate is "$\leq$", then the result bitmap B_{LE} is obtained by computing the bitmaps B_{EQ} and B_{LT}; steps that involved B_{GT}, B_{GE}, or B_{NE} are not required. The final result bitmap that is returned is either B_{LT}, B_{LE}, B_{GT}, B_{GE}, B_{EQ}, or B_{NE}, corresponding to the predicate operator "$<$", "$\leq$", "$>$", "$\geq$", "$=$", or "$\neq$", respectively. As we show below, such an evaluation strategy not only costs more bitmap operations to incrementally build the two intermediate bitmaps, but also incurs more bitmap scans since the evaluation of the bitmap B_{EQ}, which corresponds to an equality predicate evaluation, is the most expensive predicate in terms of bitmap scans.

Algorithm `RangeEval-Opt` avoids the intermediate equality predicate evaluation by evaluating each range query in terms of only the "$\leq$" predicate based on the following three identities: (1) $A < v \equiv A \leq v - 1$, (2) $A > v \equiv \overline{A \leq v}$, (3) $A \geq v \equiv \overline{A \leq v - 1}$. Furthermore, the evaluation of the "$\leq$" predicate itself does not require an equality predicate evaluation. Consequently, Algorithm `RangeEval-Opt` requires the computation of only one bitmap B for any predicate evaluation, as opposed to two bitmaps (B_{EQ} and either B_{LT} or B_{GT}) required in Algorithm `RangeEval`.

Figure 7 compares the two algorithms for evaluating the predicate "$A \leq 864$" using a 3-component base-10 index. Clearly, Algorithm `RangeEval-Opt` is more efficient as it requires only 4 bitmap operations and 5 bitmap scans, compared to Algorithm `RangeEval`, which incurs 10 bitmap operations and 6 bitmap scans. Moreover, while efficient processing of Algorithm `BSRange-Opt` requires buffering for only one bitmap (the intermediate/final result), efficient processing of Algorithm `RangeEval` requires buffering for two bitmaps (for B_{EQ} and B_{GT}/B_{LT}).

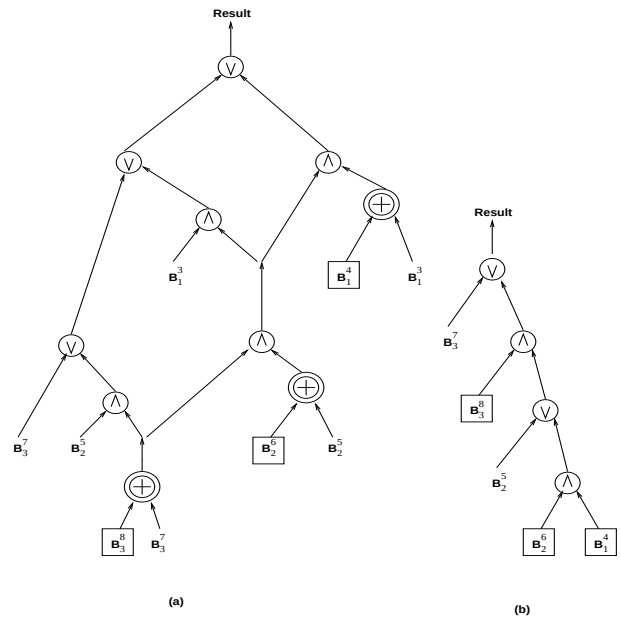


Figure 7: Comparison of Evaluation Strategies for Selection Predicate "$A \leq 864$" using a 3-Component, Base-10 Range-Encoded Bitmap Index. (a) Algorithm `RangeEval`. (b) Algorithm `RangeEval-Opt`.

3.1 Analytical Comparison of Evaluation Algorithms for Range-Encoded Bitmap Index

Evaluation Predicate	Number of Bitmap Operations					Num. of Scans
	AND	OR	NOT	XOR	Total	
`RangeEval`						
$A \geq c$	$2n$	$n+1$	n	n	$5n+1$	$2n$
$A \leq c$	$2n$	$n+1$	0	n	$4n+1$	$2n$
$A > c$	$2n$	n	n	n	$5n$	$2n$
$A < c$	$2n$	n	0	n	$4n$	$2n$
$A = c$	n	0	0	n	$2n$	$2n$
$A \neq c$	n	0	1	n	$2n+1$	$2n$
`RangeEval-Opt`						
$A \geq c$	n	n	1	0	$2n+1$	$2n-1$
$A \leq c$	n	n	1	0	$2n+1$	$2n-1$
$A > c$	n	n	0	0	$2n$	$2n-1$
$A < c$	n	n	0	0	$2n$	$2n-1$
$A = c$	n	0	0	n	$2n$	$2n$
$A \neq c$	n	0	1	n	$2n+1$	$2n$

Table 1: Worst Case Analysis of Evaluation Algorithms; n is the number of index components.

Table 1 shows the worst case analysis of the performance of the evaluation algorithms in terms of the number of bitmap operations and scans. For Algorithm `RangeEval`, since each range predicate evaluation entails an equality predicate evaluation (for B_{EQ} bitmap), the number of bitmap operations of a range predicate evaluation is at least twice that of the "$=$" predicate evaluation. By using a more direct evaluation strategy, Algorithm `RangeEval-Opt`

Evaluation Algorithms for Selection Queries Using Range-Encoded Bitmap Indexes.

Input: n is the number of components in the range-encoded index.
$< b_n, b_{n-1}, \ldots, b_1 >$ is the base of the index.
op is the predicate operator, $op \in \{<, >, \leq, \geq, =, \neq\}$.
v is the predicate value.
B_{nn} is a bitmap representing the set of records with non-null values for the indexed attribute.

Output: A bitmap representation of the set of records that satisfies the predicate "$A\ op\ v$".

Algorithm `RangeEval`

1) $B_{GT} = B_{LT} = \mathcal{B}_0$;
2) $B_{EQ} = B_{nn}$;
3) let $v = v_n v_{n-1} \ldots v_1$;
4) **for** i = n **downto** 1 **do**
5) **if** $(v_i > 0)$ **then**
6) $B_{LT} = B_{LT} \vee (B_{EQ} \wedge B_i^{v_i - 1})$;
7) **if** $(v_i < b_i - 1)$ **then**
8) $B_{GT} = B_{GT} \vee (B_{EQ} \wedge \overline{B_i^{v_i}})$;
9) $B_{EQ} = B_{EQ} \wedge (B_i^{v_i} \oplus B_i^{v_i - 1})$;
10) **else**
11) $B_{EQ} = B_{EQ} \wedge \overline{B_i^{b_i - 2}}$;
12) **else**
13) $B_{GT} = B_{GT} \vee (B_{EQ} \wedge \overline{B_i^0})$;
14) $B_{EQ} = B_{EQ} \wedge B_i^0$;
15) $B_{NE} = \overline{B_{EQ}} \wedge B_{nn}$;
16) $B_{LE} = B_{LT} \vee B_{EQ}$; $B_{GE} = B_{GT} \vee B_{EQ}$;
17) **return** B_{op};

Algorithm `RangeEval-Opt`

1) $B = \mathcal{B}_1$;
2) **if** $(op \in \{<, \geq\})$ **then** $v = v - 1$;
3) let $v = v_n v_{n-1} \ldots v_1$;
4) **if** $(op \in \{<, >, \leq, \geq\})$ **then**
5) **if** $(v_1 < b_1 - 1)$ **then** $B = B_1^{v_1}$;
6) **for** i = 2 **to** n **do**
7) **if** $(v_i \neq b_i - 1)$ **then** $B = B \wedge B_i^{v_i}$;
8) **if** $(v_i \neq 0)$ **then** $B = B \vee B_i^{v_i - 1}$;
9) **else**
10) **for** i = 1 **to** n **do**
11) **if** $(v_i = 0)$ **then** $B = B \wedge B_i^0$;
12) **else if** $(v_i = b_i - 1)$ **then** $B = B \wedge \overline{B_i^{b_i - 2}}$;
13) **else** $B = B \wedge (B_i^{v_i} \oplus B_i^{v_i - 1})$;
14) **if** $(op \in \{>, \geq, \neq\})$ **then**
15) **return** $\overline{B} \wedge B_{nn}$;
16) **else**
17) **return** $B \wedge B_{nn}$;

Figure 6: Algorithms `RangeEval` and `RangeEval-Opt`.

reduces the number of bitmap operations for range predicate evaluations by about half; the number of bitmap scans for range predicates is reduced by one. Both algorithms have the same cost for an equality predicate evaluation. For both evaluation algorithms, the worst case occurs when $0 < v_i < b_i - 1$, $1 \leq i \leq n$, which corresponds also to the most probable case.

3.2 Experimental Comparison of Evaluation Algorithms for Range-Encoded Bitmap Index

In this section, we compare the performance of the evaluation algorithms in terms of the average number of bitmap scans and the average number of bitmap operations. Indexes with uniform base are generated by varying the two main parameters: the attribute cardinality C and the base number b. We experimented with $C = 50, 100$, and 1000; for each value of C, the entire space of base-b range-encoded indexes are generated by varying b from 2 to C. For each index, we generated a total of $6C$ selection queries of the form "$A\ op\ c$" by varying the predicate $op \in \{<, \leq, >, \geq, =, \neq\}$ and the predicate constant $0 \leq c < C$. Each query is evaluated using Algorithm `RangeEval` and Algorithm `RangeEval-Opt`. For a given range-encoded index and an evaluation algorithm, the average number of bitmap scans (operations) is computed by taking the average of the number of bitmap scans (operations) over all $6C$ queries.

Figure 8 compares the performance of the two evaluation algorithms as a function of the base number with $C = 100$. The graphs clearly demonstrate that Algorithm `RangeEval-Opt` is more efficient than Algorithm `RangeEval`. Similar trends are obtained for other values of C.

4 Cost Model for Space-Time Tradeoff Analysis

Our cost model for the space-time tradeoff analysis uses the following two metrics:

- The space metric is in terms of the number of bitmaps stored.
- The time metric is in terms of the expected number of bitmap scans for a selection query evaluation, and is based on the assumption that the queries in the query space $\mathcal{Q}$ are uniformly distributed, where $\mathcal{Q} = \{A\ op\ v : op \in \{\leq, \geq, <, >, =, \neq\}, 0 \leq v < C\}$.

Note that our time metric incorporates only the I/O cost (i.e., number of bitmap scans) and does not include the CPU cost (i.e., number of bitmap operations) as the number of bitmap scans and the number of bitmap operations for a query evaluation are correlated. We denote the space and time metric of an index I by $Space(I)$ and $Time(I)$, respectively.

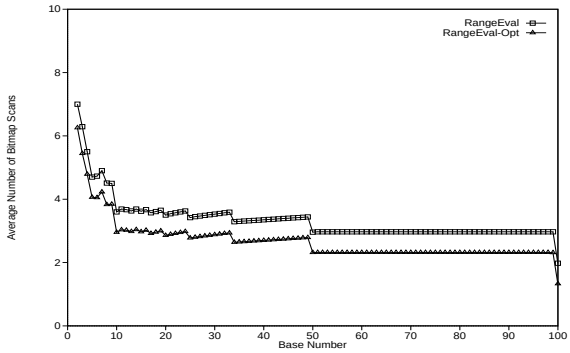
5 Comparison of Bitmap Encoding Schemes

This section compares the performance of the two basic encoding schemes, namely, equality encoding and range encoding, for selection queries. Our result shows that range encoding provides better space-time tradeoff than equality encoding in most cases. Let I denote an n-component index with base $< b_n, b_{n-1}, \ldots, b_1 >$.

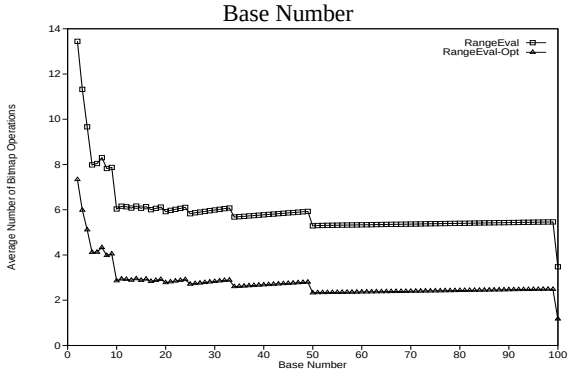
Based on the cost model, we have the following result.

Theorem 5.1 If I is equality-encoded, then

$$Space(I) = \sum_{i=1}^{n} s_i, \text{ where } s_i = \begin{cases} b_i & \text{if } b_i > 2, \\ 1 & otherwise. \end{cases} \quad (1)$$



(a) Average Number of Bitmap Scans as a Function of Base Number



(b) Average Number of Bitmap Operations as a Function of Base Number

Figure 8: `RangeEval` vs `RangeEval-Opt` for Uniform Base Range-Encoded Bitmap Indexes, $C = 100$.

$$Time(I) = \frac{1}{3}\sum_{i=1}^{n}(2t_i + 1), \text{ where} \quad (2)$$

$$t_i = \begin{cases} \frac{1}{b_i}\left(\left\lfloor\frac{b_i}{2}\right\rfloor^2 + (b_i - 1)\left(\left\lceil\frac{b_i}{2}\right\rceil - \frac{b_i}{2}\right)\right) & \text{if } b_i > 2, \\ 1 & otherwise. \end{cases}$$

If I is range-encoded, then

$$Space(I) = \sum_{i=1}^{n}(b_i - 1) \quad (3)$$

$$Time(I) = 2(n - \sum_{i=1}^{n}\frac{1}{b_i} + \frac{1}{3}(\frac{1}{b_1} - 1)) \quad (4)$$

The time analysis for range-encoded indexes is based on Algorithm `RangeEval-Opt`. The evaluation algorithm for equality-encoded indexes is not shown here due to space constraint; details can be found elsewhere [2].

A range-encoded index requires one or two bitmap scans per component for a selection predicate evaluation. On the other hand, an equality-encoded index requires only one bitmap scan per component for an equality predicate evaluation, but for a range predicate evaluation, the number of bitmap scans required per component is between two and half the number of bitmaps in that component. Figure 9 compares the space-time tradeoff of range- and equality-encoded indexes for $C = 20$, 100, and 1000. The results show that range-encoding in general provides better space-time tradeoff than equality-encoding in most cases.

Given the overall superiority of range-encoding, in the remainder of this paper, we focus on the space-time tradeoff of range-encoded indexes. Henceforth, we use the term index as an abbreviation for a range-encoded index.

6 Space-Optimal and Time-Optimal Bitmap Indexes

In this section, we identify both the space-optimal and time-optimal indexes (i.e., points (A) and (D) in Figure 2). We refer to the most space-efficient (respectively, time-efficient) index among all indexes with n components as the *n-component space-optimal index* (respectively, *n-component time-optimal index*). Note that the n-component space-optimal index might not be unique; for example, if $C = 1000$, then the base-$< 32, 32 >$ and base-$< 31, 33 >$ indexes are both 2-component space-optimal indexes.

Based on Equations (3) and (4), we have the following result.

Theorem 6.1 (1) The number of bitmaps in an n-component space-optimal index is given by $n(b-2) + r$, where $b = \lceil \sqrt[n]{C} \rceil$ and r is the smallest positive integer such that $b^r(b-1)^{n-r} \geq C$. The index with base $< \underbrace{b-1, \ldots, b-1}_{n-r}, \underbrace{b, \ldots, b}_{r} >$ is an n-component space-optimal index.

(2) The space-efficiency of space-optimal indexes is a non-decreasing function of the number of components.

(3) The base of the n-component time-optimal index is $< \underbrace{2, \ldots, 2}_{n-1}, \lceil \frac{C}{2^{n-1}} \rceil >$.

(4) The time-efficiency of time-optimal indexes is a non-increasing function of the number of components.

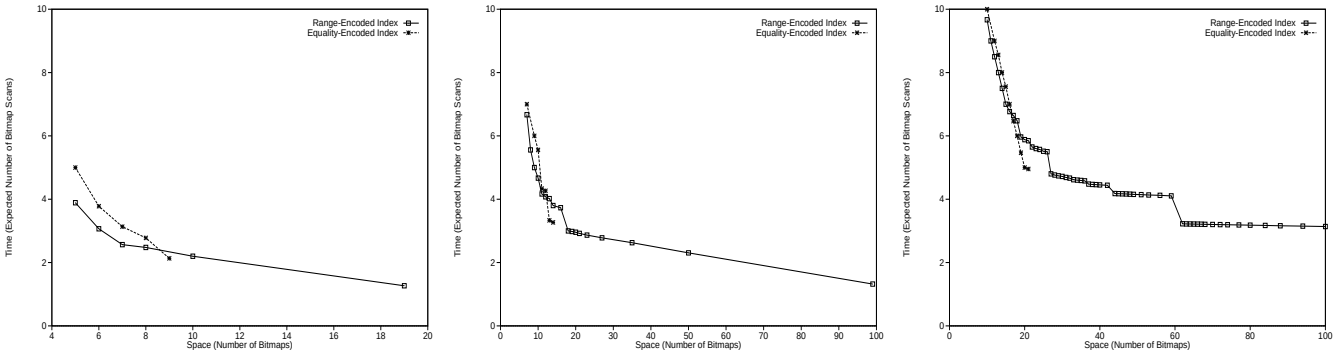
By Theorem 6.1, it follows that the space-optimal index corresponds to the index with the maximum number of components (i.e, $n = \lceil log_2(C) \rceil$ with each base number equal to 2), and the time-optimal index corresponds to the index with the minimum number of components (i.e, $n = 1$). These are probably immediate corollaries of information theory and/or number theory results as well, but they are easy enough to prove that doing so seemed easier than tracking down the existing literature.

7 Bitmap Index with Optimal Space-Time Tradeoff (Knee)

In this section, we characterize the knee of the space-time tradeoff graph, referred to as the *knee index* (point (C) in Figure 2), which corresponds to the index with the best space-time tradeoff. Note that our characterization is an approximate one as finding a precise analytical characterization is generally a difficult problem. However, it turns out that our approximate characterization has very good accuracy.

We first give a definition of the knee index that will serve as a basis for comparing with our approximate characterization. Let $\mathcal{S} = \{I_1, I_2, \ldots, I_p\}$ denote the set of optimal indexes[3] corresponding to the points in the space-time tradeoff graph such that $Space(I_j) < Space(I_{j+1}), 1 \leq j < p$. For $1 < j < p$, let LG_j and RG_j denote the gradients of the line segments formed by I_j

[3] The set of optimal indexes S is the maximal subset of all possible indexes S' such that for each $I \in S$, there does not exist another index in S' that is more efficient than I in both time and space.



(a) $C = 20$ (b) $C = 100$ (c) $C = 1000$

Figure 9: Comparison of Space-Time Tradeoff of Range- and Equality-Encoded Bitmap Indexes.

with I_{j-1} and I_{j+1}, respectively. LG_j and RG_j are defined as follows:

$$RG_j = \frac{Time(I_j) - Time(I_{j+1})}{Space(I_{j+1}) - Space(I_j)} \times F \quad \text{and}$$

$$LG_j = \frac{Time(I_{j-1}) - Time(I_j)}{Space(I_j) - Space(I_{j-1})} \times F$$

where $F = Space(I_p)/Time(I_1)$ is a normalizing factor. The index $I_k \in \{I_j \in \mathcal{S} : LG_j > 1, RG_j < 1\}$ with the maximum ratio LG_k/RG_k is the knee index.

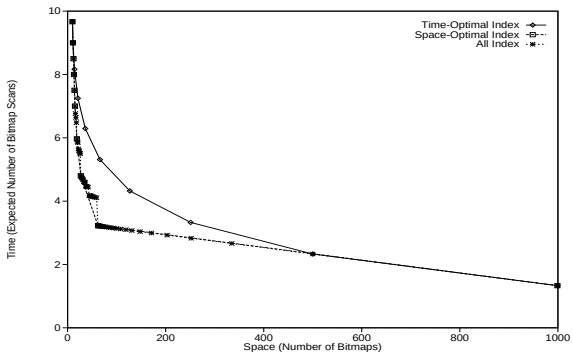


Figure 10: Space-Time Tradeoff of Bitmap Indexes, $C = 1000$.

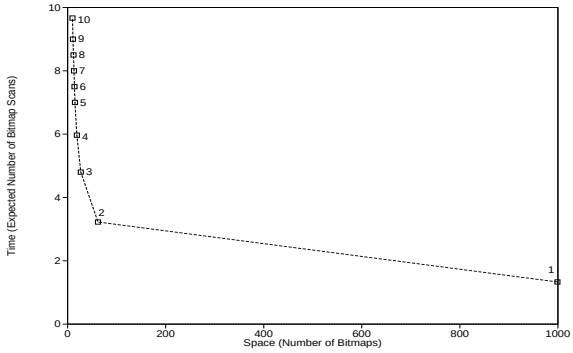


Figure 11: Space-Time Tradeoff of Space-Optimal Bitmap Indexes, $C = 1000$.

We now motivate our approximate characterization, which is based on the results of Theorem 6.1. Figure 10 compares the space-time tradeoff graphs for three classes of indexes: the class of space-optimal indexes, the class of time-optimal indexes, and the entire class of indexes, for $C = 1000$; similar results are obtained for other values of C. The graph for space-optimal (respectively, time-optimal) indexes consist of at most $\lceil log_2(C) \rceil$ points, where each point corresponds to an n-component space-optimal (respectively, time-optimal) index, $1 \leq n \leq \lceil log_2(C) \rceil$. Note that since the space-optimal index is generally not unique, each point in the space-optimal graph shown corresponds to the most time-efficient index among all equally space-efficient indexes with the same number of components. Figure 10 shows that the tradeoff graph for space-optimal indexes provides a good approximation to that for all indexes. In particular, the set of points on the graph for space-optimal indexes is a subset of the set of points on the graph for all indexes.

Figure 11 shows the same space-optimal tradeoff graph as in Figure 10 but with each point labelled with the number of components of the corresponding space-optimal index. We observe that the knee of the space-time tradeoff graph for the space-optimal index corresponds to a 2-component index, something that was consistent throughout our experimentation. Hence, we characterize the knee index as the most time-efficient 2-component space-optimal index, which is obtained from the following result.

Theorem 7.1 The base of the most time-efficient 2-component space-optimal index is given by $< b_2 - \delta, b_1 + \delta >$, where $b_1 = \lceil \sqrt{C} \rceil$, $b_2 = \lceil \frac{C}{b_1} \rceil$, and $\delta = \max\{0, \lfloor \frac{b_2 - b_1 + \sqrt{(b_2+b_1)^2 - 4C}}{2} \rfloor\}$.

We have compared the knee index based on our approximate characterization with that based on the definition for various values of attribute cardinality; the results show that both knee indexes match exactly for all the cases that we compared.

8 Time-Optimal Bitmap Index Under Space Constraint

In this section, we consider the following practical optimization problem (point (B) in Figure 2): Given a constraint on the available disk space to store an index, say at most M bitmaps (or equivalently, at most MN bits, where N is the number of records), determine the most time-efficient index. We first present an algorithm that finds the optimal solution, and then present a more efficient heuristic approach, which is near-optimal. Both algorithms are shown in Figure 12. In the following, let I_n denote an n-component index; and I_n^{space} and I_n^{time} denote the n-component space- and time-optimal indexes, respectively.

Algorithms to Find Time-Optimal Bitmap Index Under Space Constraint.

Input: M is the space constraint in terms of the maximum number of bitmaps.
C is the attribute cardinality.

Output: The time-optimal bitmap index with no more than M bitmaps.

Algorithm `TimeOptAlg`

1) let n be the smallest number such that $Space(I_n^{space}) \leq M$;
2) **if** $(Space(I_n^{time}) \leq M)$ **then return** I_n^{time};
3) let n' be the smallest number such that $n' \geq n$ and $Space(I_{n'}^{time}) \leq M$;
4) let $\mathcal{I} = \{I_k : n \leq k < n', Space(I_k) \leq M\} \cup \{I_{n'}^{time}\}$;
5) **return** $I' \in \mathcal{I}$ such that $Time(I') \leq Time(I)$, $\forall\, I \in \mathcal{I}$;

Algorithm `TimeOptHeur`

1) (n, I) = `FindSmallestN`(M,C);
2) **if** $(Space(I_n^{time}) \leq M)$ **then return** I_n^{time};
3) **return** `RefineIndex`(I);

Figure 12: Algorithms `TimeOptAlg` and `TimeOptHeur`.

8.1 Optimal Approach

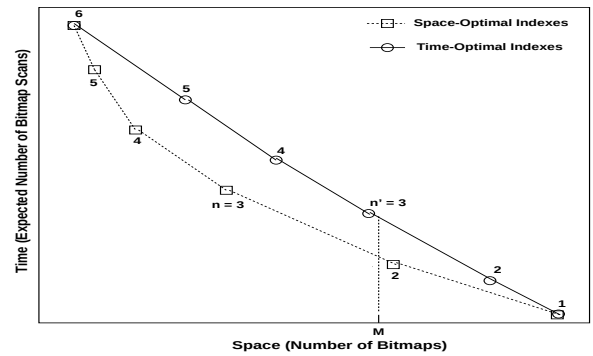
Algorithm `TimeOptAlg` finds the actual optimal solution. By result (2) of Theorem 6.1, the solution must have at least n components, where n is the smallest number of components such that $Space(I_n^{space}) \leq M$. If the n-component time-optimal index satisfies the space constraint (step 2), then by result (4) of Theorem 6.1, it is the solution; otherwise, by result (4) of Theorem 6.1, the solution index must have no more than n' components where $n' \geq n$ is the smallest number of components such that the n'-component time-optimal index satisfies the space constraint. Figure 13 shows two cases to illustrate the bounds on the number of components of the solution index; each point on the space-time tradeoff graphs shown is labelled with the number of components of the corresponding index.

The time complexity of the algorithm is determined by the size of the set of candidate indexes defined by $\mathcal{I}$ (step 4). This search space is large as we need to enumerate for each value of k, $n \leq k < n'$, all possible k-component indexes such that $\prod_{i=1}^{k} b_i \geq C$ and $\sum_{i=1}^{k}(b_i - 1) \leq M$. Figure 14 shows the size of $\mathcal{I}$ as a function of M for $C = 1000$.

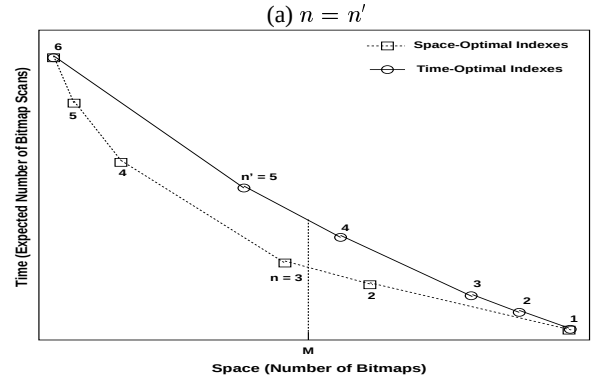
8.2 Heuristic Approach

To avoid the costly exhaustive search in the optimal approach (steps 4 and 5), we present in this section a heuristic approach (Algorithm TimeOptHeur in Figure 12) to find an approximate optimal solution. Our experimental results show that the heuristic approach is near-optimal with the optimal index being selected at least 97% of the time. The main idea is to first select a "seed" index that satisfies the space constraint, and then to improve the time-efficiency of the seed index by adjusting its base numbers. For the seed index, our approach uses an n-component index I with $Space(I) = M$, where n is the least number of components such that $Space(I_n^{space}) \leq M$. Both n and I are determined by Algorithm `FindSmallestN` (Figure 15). Note that if I_n^{time} satisfies the space constraint (step 2), then, as in step 2 of Algorithm `TimeOptAlg`, I_n^{time} is the optimal solution index. Algorithm `RefineIndex` (Figure 15) improves the time-efficiency of an index; its correctness is based on the following result.

Theorem 8.1 Let I_n be an n-component index with base $< b_n, b_{n-1}, \ldots, b_1 >$. Suppose there exists two base numbers b_p and b_q, $2 < b_p \leq b_q$, and some integer constant $0 < \delta \leq b_p - 2$, such that $\prod_{i=1}^{n} b'_i \geq C$, where $b'_i = \begin{cases} b_p - \delta & \text{if } i = p, \\ b_q + \delta & \text{if } i = q, \\ b_i & \text{otherwise.} \end{cases}$



(a) $n = n'$



(b) $n < n'$

Figure 13: Bounds on Number of Components of Solution Index in Algorithm `TimeOptAlg`.

Let I'_n be another n-component index with base $< b'_n, b'_{n-1}, \ldots, b'_1 >$ as defined above. Then $Time(I'_n) \leq Time(I_n)$.

Based on Theorem 8.1, Algorithm `RefineIndex` essentially tries to maximize the number of components with small base numbers; in particular, step 8 determines the largest possible δ value for adjusting a pair of base numbers. To illustrate Algorithm `RefineIndex`, consider a 3-component index I with base $< 21, 21, 22 >$ and $C = 1000$. In the first iteration, with $b_p = b_q = 21$, $\delta = 19$ and the base is refined to $< 2, 22, 40 >$. In the second iteration, with $b_p = 22$ and $b_q = 40$, $\delta = 12$ and the base is refined to $< 2, 10, 52 >$. After the final refinement step 3, the base of the refined index I' is $< 2, 10, 50 >$. By Equation (3), $Space(I) = 61$

Algorithm `FindSmallestN`**: Algorithm to Find the Bitmap Index with Least Number of Components Under Space Constraint.**

Input: M is the space constraint in terms of the maximum number of bitmaps.
C is the attribute cardinality.

Output: n, the smallest number of components such that $Space(I_n^{space}) \leq M$, and
An n-component index I_n with $Space(I_n) = M$.

1) $n = 0$; // n is the number of components
2) **repeat**
3) $n = n + 1$;
4) $b = \left\lfloor \frac{M+n}{n} \right\rfloor$;
5) $r = (M + n) \; mod \; n$; // r is the number of components with base $(b + 1)$
6) **until** $((b + 1)^r b^{n-r} \geq C)$;
7) let I_n be the n-component bitmap index with base $< \underbrace{b, \ldots, b}_{n-r}, \underbrace{b+1, \ldots, b+1}_{r} >$;
8) **return** (n, I_n);

Algorithm `RefineIndex`**: Algorithm to Improve the Time-Efficiency of a Bitmap Index.**

Input: I_n is a n-component bitmap index.
C is the attribute cardinality.

Output: A n-component bitmap index I'_n such that $Time(I'_n) \leq Time(I_n)$ and $Space(I'_n) \leq Space(I_n)$.

1) let $seqOfBase =< b_n, b_{n-1}, \ldots, b_1 >$ be the base of I_n;
2) $prodOfBase = \prod_{i=1}^{n} b_i$;
3) **for** $i = n$ **downto** 2 **do**
4) let b_p be the smallest base number from $seqOfBase$;
5) delete b_p from $seqOfBase$;
6) **if** $(b_p > 2)$ **then**
7) let b_q be the smallest base number from $seqOfBase$;
8) let $\delta = \left\lfloor \frac{b_p - b_q + \sqrt{(b_p+b_q)^2 - \frac{4 b_p b_q C}{prodOfBase}}}{2} \right\rfloor$;
9) **if** $(0 < \delta \leq b_p - 2)$ **then**
10) $prodOfBase = \frac{prodOfBase(b_p - \delta)(b_q + \delta)}{b_p b_q}$;
11) $b_p = b_p - \delta$;
12) $b_q = b_q + \delta$;
13) $b'_i = b_p$;
14) $b'_1 = \left\lceil \frac{C}{\prod_{j=2}^{n} b'_j} \right\rceil$;
15) **return** the n-component bitmap index with base $< b'_n, b'_{n-1}, \ldots, b'_1 >$;

Figure 15: Algorithms `FindSmallestN` and `RefineIndex`.

and $Space(I') = 59$. By Equation (4), $Time(I) = 5.08$ and $Time(I') = 4.10$.

The time complexities of Algorithms `FindSmallestN` and `RefineIndex` are $O(log_2(C))$ and $O(nlog_2(n))$, respectively, where n is at most $\lceil log_2(C) \rceil$. Therefore, the time complexity of Algorithm `TimeOptHeur` is $O(log_2(C) \; log_2(log_2(C)))$.

Attribute Cardinality, C	Percentage of Solutions that are Optimal	Max. Difference in Expected Number of Bitmap Scans
50	97.9	0.11
500	98.8	0.21
1000	98.8	0.25
2000	99.1	0.28

Table 2: Effectiveness of Heuristic Approach to Select Time-Optimal Bitmap Index under Space Constraint.

Table 2 shows the effectiveness of the heuristic approach for various values of attribute cardinality. The second column shows the percentage of solutions selected by the heuristic approach that are optimal for the attribute cardinality value indicated in the first column. For those cases where the optimal and heuristic approaches differ, the third column shows the maximum difference in the expected number of bitmap scans of their selected indexes. The result indicates that the proposed heuristic approach is near-optimal.

9 Effect of Bitmap Compression on Space-Time Tradeoff

This section examines the effect of bitmap compression on the space-time tradeoff of bitmap indexes.

9.1 Bitmap Index Storage Schemes

We first investigate different physical organizations for bitmap indexes to identify storage schemes that have good space-time trade-

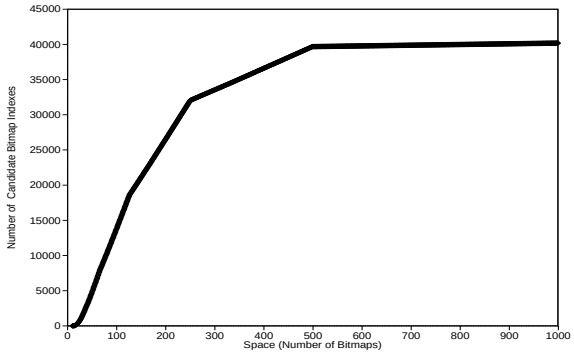


Figure 14: Size of Set of Candidate Bitmap Indexes, $\mathcal{I}$, as a Function of the Space Constraint M for $C = 1000$.

off. Consider an k-component bitmap index for a N-record relation, where the ith index component comprises of n_i bitmaps, $1 \leq i \leq k$. Let $n = \sum_{i=1}^{k} n_i$. The entire bitmap index is essentially a $(N \times n)$ bit-matrix, where the ith component is a $(N \times n_i)$ bit-matrix, and each bitmap is a $(N \times 1)$ bit-vector. Based on the above view of a bitmap index, we compare the following three storage schemes:

Bitmap-Level Storage (BS) Each bitmap is stored individually in a single bitmap file of N bits. Thus, the bitmap index is stored in n N-bit bitmap files.

Component-Level Storage (CS) Each index component is stored individually in a row-major order in a single bitmap file of $(N \times n_i)$ bits, $1 \leq i \leq k$. Thus, the bitmap index is stored in k bitmap files.

Index-Level Storage (IS) The entire index is stored in a row-major order in a single bitmap file of Nn bits.

We refer to a bitmap index that is organized using storage scheme X (without any compression) as a X-*index*, and refer to a X-index with all its bitmaps compressed as a cX-*index* (i.e., c for compressed). For a 1-component bitmap index, both a CS-index and an IS-index are equivalent. For a bitmap index with the maximum number of components; i.e., each component is of base 2, (1) both a BS-index and a CS-index are equivalent, and (2) an IS-index is a projection index[4].

In a CS-index, each row of a component bit-matrix has either a consecutive stream of 1 bits followed by a consecutive stream of 0 bits if the index is range-encoded, or exactly one bit set to 1 if the index is equality-encoded. In contrast, in a BS-index, the distribution of the bits in each bitmap is dependent on the distribution of the attribute values. Thus, we expect a CS-index to be more compressible than a BS-index.

9.2 Experimental Setup

Our experimental study uses two data sets extracted from the TPC-D Benchmark [4]: data set 1 is for small attribute cardinality, and data set 2 is for large attribute cardinality. Table 3 shows the key characteristics of our experimental data. To limit the number of experiments for each data set, our comparison of the space-time tradeoff is restricted to the class of space-optimal indexes (as a function of the number of index components, n) with n being varied from 1 to 6; this is motivated by our results in Section 7 which show that the space-time tradeoff of space-optimal indexes provides a good approximation to the space-time tradeoff of all indexes. The data compression code used is from the zlib library[5].

[4]A projection index [10] on an attribute A in a relation R is simply the projection of A on R with duplicates preserved and stored in RID-order.

	Data Set 1	**Data Set 2**
Relation	Lineitem	Order
Relation Cardinality	$6,001,215$	$1,500,000$
Attribute	Quantity	Order Date
Attribute Cardinality, C	50	$2,406$

Table 3: Characteristics of TPC-D Benchmark Data used in Experiments

Table 4 compares the compressibility of the three storage schemes for both data sets. For each n-component index I and each compressed storage scheme S, where $1 \leq n \leq 6$ and $S \in \{cBS, cCS, cIS\}$, we compute the percentage of the size of I under S to the size of I under BS. For both data sets, the results show that CS-indexes give the best compression.

Base of Index I	**Size of I under BS (in bytes)**	**Compressibility of Storage Scheme (%)**		
		cBS	**cCS**	**cIS**
$< 50 >$	$36,757,448$	77.6	27.2	27.2
$< 5, 10 >$	$9,751,976$	84.1	58.8	70.1
$< 2, 5, 5 >$	$6,751,368$	89.1	67.5	86.3
$< 2, 3, 3, 3 >$	$5,251,064$	93.2	82.9	99.2
$< 2, 2, 2, 2, 4 >$	$5,251,064$	94.0	93.3	98.7
$< 2, 2, 2, 2, 2, 2 >$	$4,500,912$	98.4	98.4	99.2

(a) Data Set 1

Base of Index I	**Size of I under BS (in bytes)**	**Compressibility of Storage Scheme (%)**		
		cBS	**cCS**	**cIS**
$< 2406 >$	$450,937,500$	76.2	2.2	2.2
$< 43, 56 >$	$18,187,500$	77.6	26.3	28.8
$< 11, 13, 17 >$	$7,125,000$	80.7	40.9	48.8
$< 5, 7, 7, 10 >$	$4,687,500$	84.2	60.5	76.8
$< 4, 5, 5, 5, 5 >$	$3,562,500$	87.7	67.2	87.6
$< 3, 3, 3, 3, 5, 6 >$	$3,187,500$	89.7	75.1	93.0

(b) Data Set 2

Table 4: Comparison of Compressibility of Different Storage Schemes.

In terms of query evaluation cost, a BS-index requires at most 2 bitmap file scans per index component. On the other hand, both a CS-index and an IS-index require all the bitmap files to be scanned; moreover, they also incur additional CPU overhead to extract the bits of each relevant bitmap from the component bitmap files (which are stored in row-major order). Thus, we expect cBS-indexes to be more time-efficient than cCS- and cIS-indexes; this is indeed supported by our experimental results. For example, consider the base-50 cBS-index; the average size of each compressed bitmap is $\frac{36,757,448 \times 0.776}{49} = 582,118$ bytes. Even when two bitmaps (an average of $1,164,236$ bytes) are scanned to evaluate an equality query, using the cBS-index is still significantly cheaper than using the cCS-index (or cIS-index), which requires $(36,757,448 \times 0.272) = 9,998,026$ bytes to be scanned.

Since cBS-indexes are the most time-efficient and cCS-indexes are the most space-efficient, we shall compare the space-time trade-

[5]The zlib library is written by Jean-loup Gailly and Mark Adler (http://www.cdrom.com/pub/infozip/zlib); the compression method is based on a LZ77 variant called deflation.

offs of only cBS- and cCS-indexes against BS-indexes for the rest of this section. Each bitmap index is used to evaluate a set of $2C$ selection queries $\mathcal{Q}$ (using Algorithm `RangeEval-Opt`), where $\mathcal{Q} = \{A\ op\ v : op \in \{\leq, =\}, 0 \leq v < C\}$. The predicate operator is restricted to "$\leq$" (for range queries) and "$=$" (for equality queries) to limit the number of queries.

The performance of each index is measured in terms of its space- and time-efficiency. The space metric is in terms of the total space of all its bitmaps (in MBytes), and the time metric is in terms of the average predicate evaluation time (in secs) which includes (1) the time to read the bitmaps, (2) the time for in-memory bitmap decompression (for compressed bitmaps), and (3) the time for bitmap operations. The average predicate evaluation time, T_{avg}, is computed as follows: $T_{avg} = \frac{1}{2C}\sum_{v=0}^{C-1}\left(T_v^{\leq} + T_v^{=}\right)$, where T_v^{op} denote the time to evaluate the predicate "$A\ op\ v$".

9.3 Experimental Results

This section presents experimental results for indexes under the storage schemes BS, cBS, and cCS for data set 1; results for data set 2 are not shown due to space limitation. Figure 16(a) shows the time-efficiency of the indexes as a function of the number of index components. Both BS- and cBS-indexes outperform cCS-indexes significantly; in particular, for cCS-indexes, over 70% of the total evaluation time is due to bitmap decompression. The decompression cost for cCS-indexes generally increases with the number of bitmap components since the number of bits to be extracted is an increasing function of the number of components. For cCS-indexes, since all the compressed bitmap files need to be scanned for a query evaluation, their I/O cost is a function of the total size of all compressed bitmap files, which generally decreases as the number of components increases. In contrast, for both BS- and cBS-indexes, since the number of bitmaps to be scanned increases with the number of components, their I/O cost is an increasing function of the number of components. Comparing BS- and cBS-indexes, while cBS-indexes incur a lower I/O cost, this is often offset by their bitmap decompression cost; our results show that both BS- and cBS-indexes are almost comparable in their time-efficiency.

Figure 16(b) compares the space-efficiency of the indexes as a function of the number of index components. There are two main observations. First, as we already know from Table 4, cCS-indexes are the most space-efficient. Second, the results show that the effectiveness of bitmap compression decreases as the number of components increases. This can be explained by the fact that the number of bitmaps is a decreasing function of the number of components (result (2) of Theorem 6.1); in particular, there is a significant space reduction when an index is decomposed from one to two components. Consequently, the gain of bitmap compression, with respect to space reduction, is only marginal once an index has been decomposed (i.e., $n \geq 2$). Thus, in terms of improving space efficiency, decomposing an i-component BS-index to an $(i+1)$-component BS-index could be more effective than compressing the i-component index.

Figure 16(c) compares the space-time tradeoff of the indexes. The result shows that BS- and cBS-indexes have comparable space-time tradeoff, which is better than that of cCS-indexes.

10 Effect of Buffering on Space-Time Tradeoff

This section considers the effect of bitmap buffering on the space-time tradeoff issues that we have addressed. As the typical size of buffer space is at least 1 GB in database systems running on SMP systems for large data warehouse applications [11], it is likely that a good number of bitmaps can remain memory-resident. By taking into account the amount of main-memory allocated for buffering bitmaps, more optimal indexes with better space-time tradeoff can be designed.

The unit of buffering that we consider here is the number of bitmaps. Consider an n-component index I with base $< b_n, b_{n-1}, \ldots, b_1 >$. Let m denote the number of bitmaps that can be buffered in main-memory, where $1 \leq m < Space(I)$. We denote a *m-bitmap buffer assignment* for I by $< f_n, f_{n-1}, \ldots, f_1 >$, where f_i is the number of bitmaps in the ith component of I that are buffered. A m-bitmap buffer assignment for I is *well-defined* if $0 \leq f_i < b_i,\ \forall\, 1 \leq i \leq n$, and $\sum_{i=1}^{n} f_i = m$.

In addition to the uniform query distribution assumption stated in Section 4, we further assume that the buffer hit rate for each referenced bitmap is uniformly distributed (i.e., the probability that a reference to any bitmap in the ith component is a buffer hit is given by $\frac{f_i}{b_i - 1}$). Then, the expected number of bitmaps scans for a selection query evaluation using an n-component index I with a m-bitmap buffer assignment is given by

$$Time(I) = 2(n - \sum_{i=1}^{n}\frac{1+f_i}{b_i} + \frac{1}{3}(\frac{1+f_1}{b_1} - 1)) \qquad (5)$$

We now present an optimal bitmap buffering policy for indexes. From result (3) of Theorem 6.1 and Theorem 8.1, we know that an index is more time-efficient if it has more components with smaller base numbers. Since buffering the bitmaps in an index component effectively reduces the base of that component, it follows that it is more beneficial (in terms of reducing the expected number of bitmap scans) to buffer a bitmap from a component with a smaller base number than from one with a larger base number. The following optimal bitmap buffering policy is based on a prioritization of the index components using their base numbers.

Theorem 10.1 Consider the following priority assignment to bitmap components:

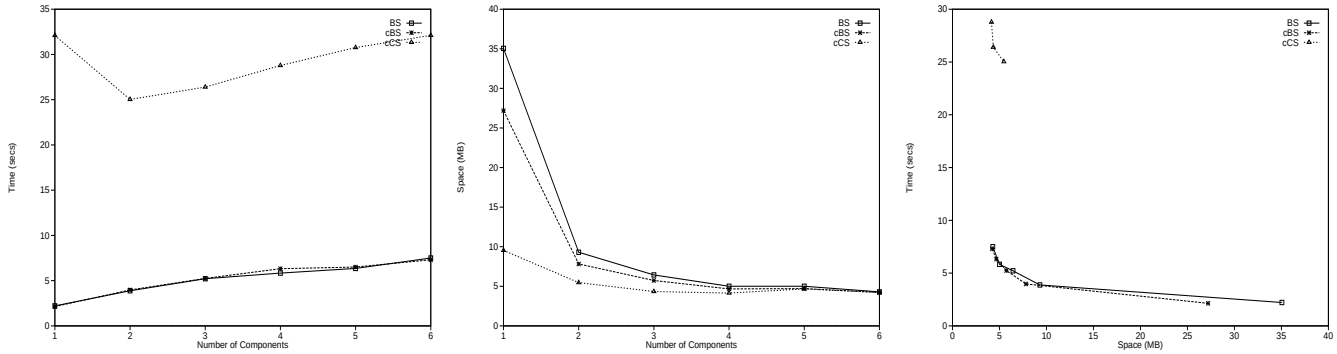
1. Partition the components of an index into two disjoint sets X and X' such that $X = \{1 < i \leq n : b_i \leq \frac{3}{2}b_1\}$ and $X' = \{1 \leq i \leq n : i \notin X\}$.
2. Components in X have higher priority than those in X'.
3. Within each set (X or X'), components with smaller base numbers have higher priority.

Based on the above priority assignment, a bitmap buffering policy that favors bitmaps from a higher priority component over those from a lower priority component is optimal.

Based on Equation (5) and the above optimal bitmap buffering policy, we have the following result.

Theorem 10.2 If $m > 0$, the time-optimal index is an m-component index with base $< \underbrace{2, \ldots, 2}_{m-1}, \left\lceil \frac{C}{2^{m-1}} \right\rceil >$.

Figure 17 compares the space-time tradeoff of indexes (based on the optimal bitmap buffering policy) as a function of the number of buffered bitmaps m for $C = 1000$. As expected, the space-time tradeoff improves as m increases. Based on our experimental observations, we have the following approximate characterization of the knee:



(a) Time vs Number of Components (b) Space vs Number of Components (c) Time vs Space

Figure 16: Comparison of Space-Time Tradeoff of Compressed Bitmap Indexes under Different Storage Schemes for Data Set 1.

The knee of the space-time tradeoff graph corresponds approximately to an $(m+2)$-component index with base $< \underbrace{2, \ldots, 2}_{m}, b_2 - \delta, b_1 + \delta >$, where $C' = \lceil \frac{C}{2^m} \rceil$,

$b_1 = \lceil \sqrt{C'} \rceil$, $b_2 = \left\lceil \frac{C'}{b_1} \right\rceil$, and

$\delta = \max\{0, \left\lfloor \frac{b_2 - b_1 + \sqrt{(b_2+b_1)^2 - 4C'}}{2} \right\rfloor\}$.

Note that the approximate knee characterization in Section 7 is a special case of the above result. Our experimental results show that the approximate knee characterization has very good accuracy.

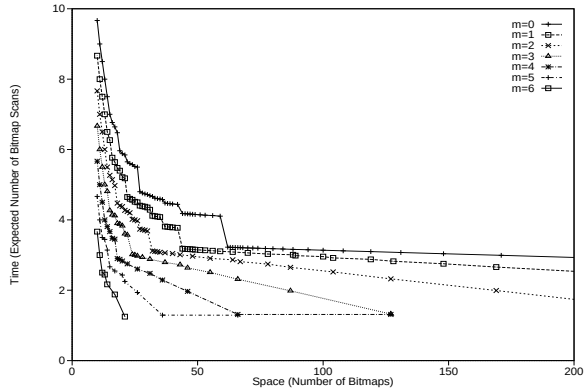


Figure 17: Space-Time Tradeoff as a Function of the Number of Buffered Bitmaps m, $C = 1000$.

11 Conclusions

In this paper, we have presented a general framework to study the design space of bitmap indexes for selection queries, and have examined several space-time tradeoff issues. To the best of our knowledge, this study represents a first set of guidelines for physical database design using bitmap indexes.

Our results indicate that range-encoded bitmap indexes offer, in most cases, better space-time tradeoff than equality-encoded bitmap indexes. Concentrating on these, we have identified the time-optimal index, the space-optimal index, the index with the optimal space-time tradeoff, and the time-optimal index under disk-space constraints. In addition, we have proposed an optimal bitmap buffering policy and examined its impact on the space-time tradeoff issues. All these results are based on an improved evaluation algorithm for selection queries that we have proposed for range-encoded bitmap indexes.

For future work, we plan to explore bitmap indexes for the more general class of selection queries that is based on the membership operator.

References

[1] AIPD Technical Publications. Sybase IQ Indexes. In *Sybase IQ Administration Guide*, Sybase IQ Release 11.2 Collection, chapter 5. Sybase Inc., March 1997. http://sybooks.sybase.com/cgi-bin/nph-dynaweb/siq11201/iq_admin/1.toc.

[2] C.Y. Chan and Y.E. Ioannidis. Bitmap Index Design and Evaluation. Computer Sciences Dept., University of Wisconsin-Madison, 1997. http://www.cs.wisc.edu/∼cychan/paper101.ps.

[3] S. Chaudhuri and U. Dayal. An Overview of Data Warehousing and OLAP Technology. *ACM SIGMOD Record*, 26(1):65–74, March 1997.

[4] Transaction Processing Performance Council, May 1995. http://www.tpc.org.

[5] H. Edelstein. Faster Data Warehouses. *Information Week*, pages 77–88, December 1995.

[6] C.D. French. "One Size Fits All" Database Architectures do not work for DSS. In *SIGMOD'95*, pages 449–450, San Jose, California, May 1995.

[7] G. Graefe. Query Evaluation Techniques for Large Databases. *Computing Surveys*, 25(2):73–170, 1993.

[8] P. O'Neil. Model 204 Architecture and Performance. In *Proceedings of the 2nd International Workshop on High Performance Transactions Systems*, pages 40–59, Asilomar, CA, 1987. Springer-Verlag. In Lecture Notes in Computer Science 359.

[9] P. O'Neil and G. Graefe. Multi-Table Joins Through Bitmapped Join Indices. *ACM SIGMOD Record*, pages 8–11, September 1995.

[10] P. O'Neil and D. Quass. Improved Query Performance with Variant Indexes. In *SIGMOD'97*, pages 38–49, Tucson, Arizona, May 1997.

[11] T-F. Tsuei, A.N. Packer, and K-T. Ko. Database Buffer Size Investigation for OLTP Workloads. In *SIGMOD'97*, pages 112–122, Tucson, Arizona, May 1997.

[12] J. Winchell. Rushmore's Bald Spot. *DBMS*, 4(10):58, September 1991.

[13] H.K.T. Wong, J.Z. Li, F. Olken, D. Rotem, and L. Wong. Bit Tranposition for Very Large Scientific and Statistical Databases. *Algorithmica*, 1(3):289–309, 1986.

[14] H.K.T. Wong, H-F. Liu, F. Olken, D. Rotem, and L. Wong. Bit Tranposed Files. In *VLDB'85*, pages 448–457, Stockholm, 1985.