An Efficient Bitmap Encoding Scheme for Selection Queries

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 indexes are useful in processing complex queries in decision support systems, and they have been implemented in several commercial database systems. A key design parameter for bitmap indexes is the *encoding scheme*, which determines the bits that are set to 1 in each bitmap in an index. While the relative performance of the two existing bitmap encoding schemes for simple selection queries of the form "$v_1 \leq A \leq v_2$" is known (specifically, one of the encoding schemes is better for processing equality queries; i.e., $v_1 = v_2$, while the other is better for processing range queries; i.e., $v_1 < v_2$), it remains an open question whether these two encoding schemes are indeed optimal for their respective query classes in the sense that there is no other encoding scheme with better space-time tradeoff. In this paper, we establish a number of optimality results for the existing encoding schemes; in particular, we prove that neither of the two known schemes is optimal for the class of two-sided range queries. We also propose a new encoding scheme and prove that it is optimal for that class. Finally, we present an experimental study comparing the performance of the new encoding scheme with that of the existing ones as well as four hybrid encoding schemes for both simple selection queries and the more general class of membership queries of the form "$A \in \{v_1, v_2, ..., v_k\}$". These results demonstrate that the new encoding scheme has an overall better space-time performance than existing schemes.

* Partially supported by the National Science Foundation under Grant IRI-9157368 (PYI Award) and the members of the Wisconsin database group industrial affiliates (www.cs.wisc.edu/~raghu/dbaffiliates.html).

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

SIGMOD '99 Philadelphia PA

1 Introduction

A promising approach to process complex queries in Decision Support Systems (DSS) is the use of bitmap indexing [O'N87, OG95, OQ97]. In a conventional B^+-tree index, each distinct attribute value v is associated with a list of RIDs (called a *RID-list*) of all the records associated with the attribute value v. The basic idea of a bitmap index is to replace the RID-list with a bit vector (bitmap), i.e., a bitmap index is essentially a collection of bitmaps. The size of each bitmap is equal to the cardinality of the indexed relation, and the i^{th} bit corresponds to the i^{th} record. In the simplest bitmap index design, the i^{th} bit of a bitmap associated with value v is set to 1 if and only if the i^{th} record has a value v for the indexed attribute.

Bitmap indexes have been implemented in several commercial DBMSs (IBM [Win99], Informix [Inf, O'N97], Oracle [Jak97], RedBrick [Ede95], Sybase [Ede95, Syb97]). A major advantage of bitmap indexes is that bitmap manipulations using bit-wise operators (AND , OR , XOR, NOT) are very efficiently supported by hardware. Furthermore, bitmap indexes are very space-efficient, especially for attributes with low cardinality.

A key design parameter for bitmap indexes is the *encoding scheme*, which determines the set of attribute values "represented" by each bitmap in an index, that is the attribute values that set the corresponding records' bits in a bitmap to 1. For example, in the simple design mentioned earlier, the encoding scheme is such that the bitmap associated with the value v represents v alone. Previous studies [WLO+85, WLO+86, OQ97, CI98b] have identified two basic bitmap encoding schemes: *Equality Encoding*, which is the one mentioned above and is efficient for equality queries (i.e., queries of the form "$A = v$"), and *Range Encoding*, which is efficient for one-sided range queries (i.e., queries of the form "$A \leq v$" or "$A \geq v$"). However, the space-time performance optimality[1] of either of these encoding

[1] Informally, an encoding scheme S is optimal for a query class Q if there is no other encoding scheme with strictly better space-

schemes remains an open issue; that is, it is not known whether or not there exists an encoding scheme with strictly better space-time performance than equality encoding for equality queries or range encoding for one-sided range queries (and more generally for the class of two-sided range queries of the form "$v_1 \leq A \leq v_2$"). In addition, to the best of our knowledge, no earlier work has examined the performance of bitmap indexes for the more general class of *membership queries* (i.e., queries of the form "$A \in \{v_1, v_2, \ldots, v_k\}$").

In this paper, we address these two issues and make the following contributions:

- We establish a number of optimality results for existing encoding schemes; in particular, we prove that neither of the two existing encoding schemes is optimal for the class of two-sided range queries.
- We propose a new encoding scheme, called *Interval-Encoding*, and prove that it is optimal for both one-sided and two-sided range queries.
- We introduce an efficient query evaluation framework for multi-component indexes that takes into account the buffer size to avoid rescanning from disk the same bitmaps.
- We present the results of an experimental study comparing the new encoding scheme with the existing ones as well as four hybrid encoding schemes for both simple selection queries and the more general class of membership queries of the form "$A \in \{v_1, v_2, \ldots, v_k\}$". All encoding schemes have been studied in both compressed and uncompressed forms. The results of this study show that the interval encoding scheme exhibits better space-time overall for various query classes.

We conclude this section with some preliminaries. Consider an attribute A of a relation R, where the attribute cardinality is C. For simplicity and without loss of generality, the domain of A is assumed to be a set of consecutive integers from 0 to $C-1$. Let B be an individual bitmap of a bitmap index on A. For notational convenience, we overload the symbol B so that it indicates both the bitmap itself (i.e., a sequence of 0's and 1's) and the set of attribute values in A that correspond to its bits that are set to 1. This allows us to use set operators and logical operators interchangeably. The logical operators AND, OR, and XOR are denoted by $\wedge$, $\vee$, and $\oplus$, respectively, while the complement of B is denoted by $\overline{B}$.

An **interval query** on attribute A is a query of the form "$x \leq A \leq y$" or "NOT ($x \leq A \leq y$)". An interval query is an **equality query** if $x = y$; it is a **one-sided range query** if $x = 0$ or $y = C-1$; and it is a **two-sided range query** if $0 < x < y < C-1$. A one-sided or two-sided range query is also called a **range query**. We denote the class of equality queries, one-sided range queries, two-sided range queries, and range queries by EQ, $1RQ$, $2RQ$ and RQ, respectively. We refer to a query that belongs to the query class Q, where $Q \in \{EQ, 1RQ, 2RQ, RQ\}$, as a *Q-query*. Queries of the form "$A \in \{v_1, v_2, \ldots, v_k\}$" are **membership queries**.

The rest of this paper is organized as follows. A review of related work is presented in Section 2. Section 3 presents optimality results for the existing encoding schemes. In Section 4, we introduce *interval encoding*, which is an optimal encoding scheme for the class of two-sided range queries. In Section 5, we consider the evaluation of membership queries and propose four hybrid encoding schemes. Section 6 describes query evaluation using multi-component bitmap indexes. Section 7 presents an experimental study comparing the various encoding schemes for evaluating both interval as well as membership queries. Finally, we summarize our results in Section 8. The proofs of theoretical results are given elsewhere [CI98a].

2 Previous Work

We first review the two existing bitmap encoding schemes, *equality encoding*, denoted by $\mathcal{E}$, and *range encoding*, denoted by $\mathcal{R}$. These schemes have been described in several papers under different names [WLO+85, WLO+86, OQ97, CI98b].

Equality encoding is the most fundamental and common bitmap encoding scheme. It consists of C bitmaps $\mathcal{E} = \{E^0, E^1, \ldots, E^{C-1}\}$, where each bitmap $E^v = \{v\}$[2]. (Recall the notational overload mentioned earlier, where a bitmap may be seen as the set of attribute values corresponding to its 1 bits.) For example, consider Figure 1(a) showing the projection on an attribute A with cardinality $C = 10$ of a 12-record relation. Figure 1(b) shows the equality-encoded bitmap index for the data in Figure 1(a), where each column represents an equality-encoded bitmap E^v associated with an attribute value v. Evaluation of interval queries using an equality-encoded bitmap index proceeds as in Equation (1):

$$\text{"}v_1 \leq A \leq v_2\text{"} = \begin{cases} \bigvee_{i=v_1}^{v_2} E^i & \text{if } v_2 - v_1 + 1 \leq \lfloor \frac{C}{2} \rfloor, \\ \overline{\bigvee_{i=0}^{v_1-1} E^i \vee \bigvee_{i=v_2+1}^{C-1} E^i} & \text{otherwise.} \end{cases} \quad (1)$$

time performance than S for Q.

[2]For the case when $C = 2$, since $E^1 = \overline{E^0}$, it suffices to store just E^0.

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

(a)

	E^9	E^8	E^7	E^6	E^5	E^4	E^3	E^2	E^1	E^0
1	0	0	0	0	0	0	1	0	0	0
2	0	0	0	0	0	0	0	1	0	0
3	0	0	0	0	0	0	0	0	1	0
4	0	0	0	0	0	0	0	1	0	0
5	0	1	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	1	0	0
7	1	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	1
9	0	0	1	0	0	0	0	0	0	0
10	0	0	0	0	1	0	0	0	0	0
11	0	0	0	1	0	0	0	0	0	0
12	0	0	0	0	0	1	0	0	0	0

(b)

	R^8	R^7	R^6	R^5	R^4	R^3	R^2	R^1	R^0
1	1	1	1	1	1	1	0	0	0
2	1	1	1	1	1	1	1	0	0
3	1	1	1	1	1	1	1	1	0
4	1	1	1	1	1	1	1	0	0
5	1	0	0	0	0	0	0	0	0
6	1	1	1	1	1	1	1	0	0
7	0	0	0	0	0	0	0	0	0
8	1	1	1	1	1	1	1	1	1
9	1	1	0	0	0	0	0	0	0
10	1	1	1	1	0	0	0	0	0
11	1	1	1	0	0	0	0	0	0
12	1	1	1	1	1	0	0	0	0

(c)

Figure 1: Example of Existing Bitmap Indexes with $C = 10$. (a) Projection of Indexed Attribute (with duplicates preserved). (b) Equality-Encoded Index. (c) Range-Encoded Index.

The range encoding scheme consists of $(C-1)$ bitmaps $\mathcal{R} = \{R^0, R^1, \ldots, R^{C-2}\}$, where each bitmap $R^v = [0, v]$. Figure 1(c) shows the range-encoded bitmap index for the data in Figure 1(a). Evaluation of interval queries using a range-encoded bitmap index proceeds as in Equation (2):

$$\text{“}v_1 \leq A \leq v_2\text{”} = \begin{cases} R^0 & \text{if } v_1 = v_2 = 0, \\ R^{v_1} \oplus R^{v_1-1} & \text{if } 0 < v_1 = v_2 < C-1, \\ \overline{R^{C-2}} & \text{if } v_1 = v_2 = C-1, \\ \overline{R^{v_1-1}} & \text{if } 0 < v_1 < C-1, v_2 = C-1, \\ R^{v_2} & \text{if } v_1 = 0, 0 \leq v_2 < C-1, \\ R^{v_2} \oplus R^{v_1-1} & \text{otherwise.} \end{cases} \quad (2)$$

The collection of bitmaps in a bitmap index essentially forms a two-dimensional bit matrix (e.g., Figure 1). The definition of a bitmap index emphasizes on a column-wise view of this bit matrix: a bitmap index is a collection of bitmaps, where each bitmap represents a subset of the attribute's domain values. However, by focusing on a bitmap index row-wise, we observe that each row is basically a representation of an attribute value using some number of bits encoded in some way. By varying the representation of the attribute values, different bitmap index designs can be obtained.

Accordingly, in earlier work [CI98b], we have proposed a two-dimensional framework to explore the design space of bitmap indexes for the class of equality and range queries. By varying the options in each dimension, different bitmap index designs are obtained with different space-time performance. The two orthogonal dimensions in our framework are (1) the bitmap encoding scheme, and (2) the bitmap index decomposition. The decomposition of a bitmap index is varied by selecting a number representation scheme for the attribute values. For example, consider an attribute with cardinality $C = 50$. A value of 35 can be represented as a single base-50 digit (i.e., $35 = 35_{50}$), or as two base-8 digits (i.e., $35 = 4_8 3_8$), and so on. In general, given a sequence of integers $< b_n, b_{n-1}, \ldots, b_1 >$, an attribute value v can be decomposed into a sequence of n digits $v = v_n v_{n-1} \ldots v_1$ as follows:

$$v = v_n \left(\prod_{j=1}^{n-1} b_j \right) + \ldots + v_i \left(\prod_{j=1}^{i-1} b_j \right) + \ldots + v_2 b_1 + v_1 \quad (3)$$

where each v_i is a base-b_i digit (i.e., $0 \leq v_i < b_i$), $b_n = \lceil \frac{C}{\prod_{i=1}^{n-1} b_i} \rceil$, and $b_i \geq 2$, $\forall\ 1 \leq i \leq n$. Each choice of n and sequence $< b_n, b_{n-1}, \ldots, b_1 >$ gives a different representation of attribute values and defines a different n-component index. We refer to an index formed by the sequence $< b_n, b_{n-1}, \ldots, b_1 >$ as a base-$< b_n, b_{n-1}, \ldots, b_1 >$ index. Figure 2 shows two base-$< 3, 4 >$ indexes based on the same data set as in Figure 1. We use the notation B_i^j to denote a bitmap in the i^{th} component associated with value j in an n-component bitmap index, where $1 \leq i \leq n$, $0 \leq j < b_i$, and b_i is the base of the i^{th} component. Note that the evaluation equations (1) and (2) are directly applicable on one-component indexes only (or equivalently on individual components). A generalization of them to multi-component indexes is discussed in Section 6.

Based on the above framework, designing a bitmap index is essentially an optimization problem of identifying a point in this two-dimensional space that exhibits “optimal” space-time performance. The only somewhat related piece of work that we are aware of is that of Wu and Buchmann [WB98], who studied a rather different optimization problem for *binary-encoded* bitmap indexes (i.e., in our framework terminology, equality-encoded indexes with the maximum number of components). In a binary-encoded bitmap index, each attribute value is represented in binary form (i.e., with $\lceil log_2(C) \rceil$ bits, where C is the attribute cardinality); so there are a total of $\lceil log_2(C) \rceil$ bitmaps in a binary-encoded index. Given a set of membership queries S,

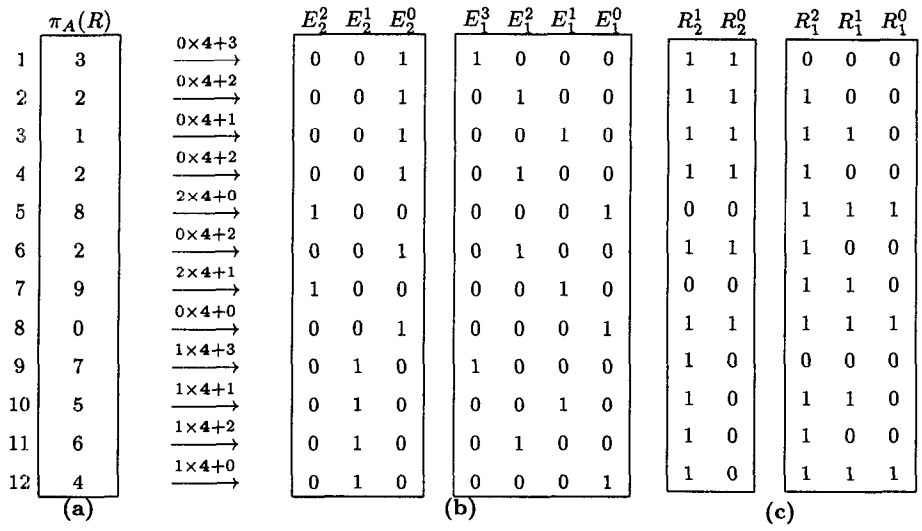


Figure 2: Example of Base-$< 3,4 >$ Indexes with $C = 10$ (a) Projection of Indexed Attribute (with duplicates preserved). (b) Equality-Encoded Index. (c) Range-Encoded Index.

the optimization problem that Wu and Buchmann addressed is to find an assignment of binary representations for the C attribute values so that the number of bitmap scans required to evaluate all the queries in S is minimized. The authors have identified a sufficient condition for this problem, but as pointed out by them, the optimal solution might not always exist; furthermore, the complexity of their solution is an exponential function of C and the number of queries in S.

3 Optimality Results for Existing Encoding Schemes

While it is known that equality encoding is better than range encoding for equality queries and vice-versa for range queries, it remains an open question whether or not there exist an encoding scheme that has better space-time performance than these. In this section, we examine the optimality of the two existing encoding schemes, $\mathcal{E}$ and $\mathcal{R}$, for the query classes EQ, 1RQ, 2RQ, and RQ. The discussion refers to 1-component indexes, or equivalently, to individual components of multi-component indexes, so that differences between bitmap index designs are essentially differences between their encoding schemes. The optimality definition could easily be extended to include arbitrary index designs.

Let $Time(S, C, Q)$ and $Space(S, C)$ denote the time- and space-cost of a bitmap encoding scheme S for an attribute with cardinality C and query class Q. The time-efficiency is in terms of the *expected* number of bitmap scans for evaluating a query in class Q, and the space-efficiency is in terms of the number of bitmaps stored. An encoding scheme $\mathcal{S}$ is **complete** if it captures all information necessary to evaluate any interval query, or equivalently, any query in class EQ. A bitmap encoding scheme $\mathcal{S}$ is said to be **optimal** with respect to C and Q if there exists no other complete bitmap encoding scheme $\mathcal{S}'$ such that

1. $Time(\mathcal{S}', C, Q) \leq Time(\mathcal{S}, C, Q)$ and
2. $Space(\mathcal{S}', C) \leq Space(\mathcal{S}, C)$ and
3. at least one of these inequalities is strict.

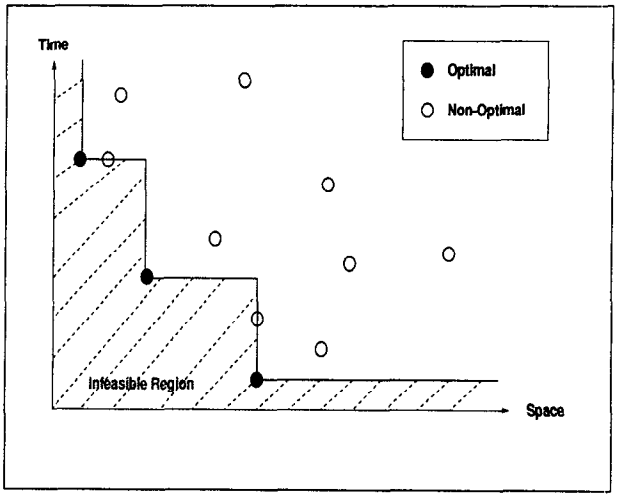


Figure 3: Space-Time Performance Field

Figure 3 illustrates the notion of optimality within a space-time performance field for a universe of 12 bitmap encoding schemes. Each point represents the space- and

time-cost of a distinct index (encoding scheme), with the optimal and non-optimal indexes indicated by black and white points, respectively. Note that there may be many optimal indexes, as they may be incomparable in terms of both time and space, i.e., for any pair of optimal points, one of them strictly dominates the other in terms of time and the opposite holds in terms of space.

Given the above definition of optimality, Theorem 3.1 states several results for the existing encoding schemes.

Theorem 3.1 The following statements hold:

1. Range encoding is optimal for EQ iff $C \leq 5$.
2. Range encoding is optimal for 1RQ for all C.
3. Range encoding is not optimal for 2RQ for any C.
4. Range encoding is optimal for RQ for all C[3].
5. Equality encoding is optimal for EQ for all C.
6. Equality encoding is not optimal for 1RQ, 2RQ, and RQ for any C.

4 Interval Encoding Scheme

An interesting result of Theorem 3.1 is that neither of the two existing encoding schemes is optimal for the class of 2-sided range queries. In this section, we present a new encoding scheme called *interval encoding*, denoted by $\mathcal{I}$, which is optimal for the query class 2RQ. The intuition for the new encoding scheme is based on range encoding. In range encoding, each bitmap $R^i = [0, i]$, and each 2RQ-query is evaluated by operating on an appropriate pair of bitmaps: $[x, y] = R^y \oplus R^{x-1}$. Figure 4(a) shows the set of values captured by each bitmap in a range-encoded bitmap for $C = 10$.

The interval encoding scheme consists of $\lceil \frac{C}{2} \rceil$ bitmaps $\mathcal{I} = \{I^0, I^1, \ldots, I^{\lceil \frac{C}{2} \rceil - 1}\}$, where each bitmap $I^j = [j, j + m]$, and $m = \lfloor \frac{C}{2} \rfloor - 1$.[4] Figure 5 shows the interval-encoded bitmap index for the data in Figure 1(a). Evaluation of all interval queries using a one-component interval-encoded bitmap index proceeds as in Equations (4) to (6) for equality, one-sided range, and two-sided range queries, respectively. (Again, evaluation using multi-component indexes is discussed in Section 6.) Note that for one-sided range queries where $v_1 > 0$ and $v_2 = C - 1$, they are simply evaluated in terms of Equation (5) by negating the result of the evaluation of "$A \leq v_1 - 1$". Overall, we have obtained an encoding scheme that has almost half the space of range encoding, as shown in Figure 4(b), while still guaranteeing at most a two-scan evaluation for any query.

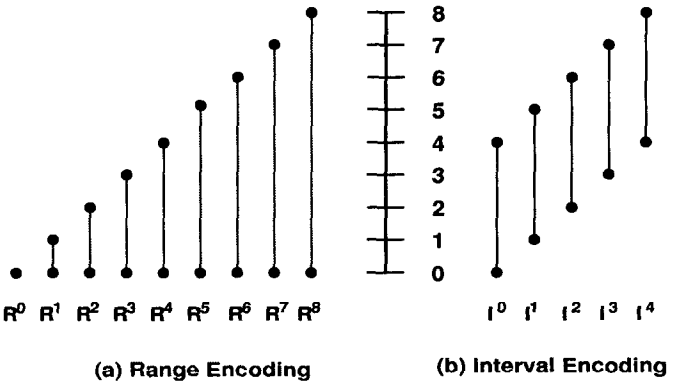


Figure 4: Range vs Interval Encoding, $C = 10$.

$\mathcal{I} = \{I^0, I^1, I^2, I^3, I^4\}$, where

I^0	=	R^4	=	$[0, 4]$
I^1	=	$R^5 \oplus R^0$	=	$[1, 5]$
I^2	=	$R^6 \oplus R^1$	=	$[2, 6]$
I^3	=	$R^7 \oplus R^2$	=	$[3, 7]$
I^4	=	$R^8 \oplus R^3$	=	$[4, 8]$

(a)

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

(b) (c)

Figure 5: Example of an Interval-Encoded Bitmap Index with $C = 10$. (a) Definition of Interval-Encoded Index for $C = 10$. (b) Projection of Indexed Attribute (with duplicates preserved). (c) Interval-Encoded Index for Data Set in (b).

[3]Note that $\mathcal{R}$ is optimal for the class of RQ queries although it is not optimal for its subclass of 2RQ queries. This may seem counterintuitive, but its quite plausible. For instance, imagine $\mathcal{R}$ and another hypothetical encoding $\mathcal{S}$ as black points in Figure 3 for RQ queries, with $\mathcal{R}$ having better (expected) time and $\mathcal{S}$ having better space. In the corresponding figure for 2RQ, the space for both encodings remains the same of course and the time for $\mathcal{R}$ increases (as 1RQ queries, which are its best, are removed); assuming the time for $\mathcal{S}$ decreases so that it becomes better than that for $\mathcal{R}$, then the latter is completely dominated by $\mathcal{S}$ and is no longer optimal.

[4]Another variant of the interval encoding scheme for the case when C is odd is discussed elsewhere [CI98a].

$$\text{“}A = v\text{”} = \begin{cases} I^0 & \text{if } v = 0,\ m = 0, \\ \overline{I^0} & \text{if } v = 1,\ C = 2, \\ I^1 & \text{if } v = 1,\ C = 3, \\ I^v \wedge \overline{I^{v+1}} & \text{if } v < m, \\ I^v \wedge I^0 & \text{if } v = m,\ m > 0, \\ I^{v-m} \wedge \overline{I^{v-m-1}} & \text{if } m < v < C-1,\ m > 0, \\ \overline{(I^{\lceil \frac{C}{2} \rceil - 1} \vee I^0)} & \text{if } v = C-1. \end{cases} \quad (4)$$

For $0 < v < C-1$,

$$\text{“}A \leq v\text{”} = \begin{cases} I^0 \wedge \overline{I^{v+1}} & \text{if } v < m, \\ I^0 & \text{if } v = m, \\ I^0 \vee I^{v-m} & \text{if } m < v < C-1. \end{cases} \quad (5)$$

For $0 < v_1 < v_2 < C-1$,

$$\text{“}v_1 \leq A \leq v_2\text{”} = \begin{cases} I^{v_1} \wedge \overline{I^{v_2+1}} & \text{if } v_2 < m, \\ I^{v_1} \wedge I^0 & \text{if } v_2 = m, \\ I^{v_1} \wedge I^{v_2-m} & \text{if } v_2 < v_1 + m,\ v_1 < n, \\ I^{v_1} & \text{if } v_2 = v_1 + m,\ v_1 < n, \\ I^{v_1} \vee I^{v_2-m} & \text{if } v_2 > v_1 + m,\ v_1 < m, \\ I^{v_1} \vee I^{v_1+1} & \text{if } v_2 = v_1 + m + 1,\ v_1 = m, \\ I^{v_2-m} \wedge \overline{I^{v_1-m-1}} & \text{if } v_1 \geq n. \end{cases} \quad (6)$$

4.1 Optimality of Interval Encoding Scheme

The following theorem states several optimality results for interval encoding.

Theorem 4.1 The following statements hold:

1. Interval-encoding is not optimal for EQ if $C \geq 14$.
2. Interval-encoding is optimal for 1RQ for all C.
3. Interval-encoding is optimal for 2RQ for all C.
4. Interval-encoding is optimal for RQ for all C.

Table 1 summarizes the results of Theorem 4.1 together with those of Theorem 3.1 for ease of comparison. A $\surd$ for (Q, S) means that encoding scheme S is optimal for query class Q; a $\times$ for (Q, S) means that encoding scheme S is not optimal for query class Q. An empty (Q, S) means that it is unknown whether or not the encoding scheme S is optimal for query class Q; this case is implicitly present in the table for $Q = EQ$, $S = \mathcal{I}$, and $C < 14$. Observe that interval encoding appears to be optimal for the widest possible class of queries.

Query Class	Encoding Scheme		
	$\mathcal{E}$	$\mathcal{R}$	$\mathcal{I}$
EQ	$\surd$	$\surd$ iff $C \leq 5$	$\times$ if $C \geq 14$
1RQ	$\times$	$\surd$	$\surd$
2RQ	$\times$	$\times$	$\surd$
RQ	$\times$	$\surd$	$\surd$

Table 1: Optimality of Existing and New Encoding Schemes.

4.2 Comparison of Basic Encoding Schemes

Among the three basic encoding schemes, interval encoding is the most space-efficient requiring about half the number of bitmaps of the other two basic schemes. Equality encoding is the most query-efficient for equality queries but the least query-efficient for range queries. Range encoding is the most query-efficient for one-sided range queries, and is equally query-efficient as interval encoding for equality queries as well as two-sided range queries.

For a new data record, the update-cost of an encoding scheme can be measured in terms of the number of bitmaps to be updated (to set the bits to 1). Equality encoding is the most update-efficient requiring only one bitmap update. Range encoding is the least update-efficient requiring 1, $\frac{C-1}{2}$, and $C-1$ bitmap updates, respectively, for the best, expected, and worst cases. Interval encoding falls in between requiring 1, $\frac{C}{4}$, and $\lfloor \frac{C}{2} \rfloor$ bitmap updates, respectively, for the best, expected, and worst cases. However, index updates in DSS are typically batched and performed for a set of updates rather than for a single update. In that case, both the particular batching performed and the precise distribution of the updated records on the disk blocks become important. Studying the effect of these parameters on the update-costs of the various encoding schemes is beyond the scope of this paper and is left as future work.

5 Hybrid Encoding Schemes for Membership Queries

We now consider encoding schemes targeted for the more general class of membership queries of the form “$A \in \{v_1, v_2, \ldots, v_k\}$”. A membership query is essentially a collection of equality and range queries; specifically, each membership query can be uniquely expressed as a disjunction of a minimal number of equality and range queries. For example, the membership query “$A \in \{6, 19, 20, 21, 22, 35\}$” can be rewritten as “$(A = 6) \vee (19 \leq A \leq 22) \vee (A = 35)$”. We refer to the equality and range queries in a membership query as its *constituent queries*. Clearly, encoding schemes for

interval queries can also be applied to evaluate membership queries. In this section, we propose four additional hybrid encoding schemes that offer different space-time tradeoffs. The first two hybrid schemes are based on the two known encoding schemes (i.e., equality and range encoding schemes), and the last two schemes are based on equality and interval encoding schemes.

5.1 Equality-Range Encoding Scheme

The first hybrid scheme, *equality-range encoding* (denoted by $\mathcal{ER}$), is simply a combination of the two known encoding schemes; i.e., $\mathcal{ER} = \mathcal{E} \cup \mathcal{R}$. The bitmaps R^0 and R^{C-2} need not be physically materialized since by definition, $R^0 = E^0$ and $R^{C-2} = \overline{E^{C-1}}$.

Equality and range constituent queries are evaluated using equality- and range-encoded bitmaps, respectively. Since equality encoding is the most time-efficient scheme for equality queries, and range encoding is the most time-efficient scheme for one-sided range queries, we expect this hybrid scheme to do really well time-wise for membership queries at the cost of about twice the space requirement of the basic schemes.

5.2 OREO Encoding Scheme

The second hybrid encoding scheme, *OREO encoding scheme*[5] (denoted by $\mathcal{O}$), offers a different approach to complement the strengths of the two basic schemes using the same amount of space as range encoding.

One obvious way to reduce the space requirement of $\mathcal{ER}$ is to have only half its number of equality- and range-encoded bitmaps; for example, for an attribute with cardinality C, we can have an encoding scheme that uses equality encoding for the first $\frac{C}{2}$ values and range encoding for the remaining $\frac{C}{2}$ values. However, such a design suffers from a similar performance problem as equality encoding for range query evaluation when the range query falls on the "wrong" values (first half of values in this example).

The OREO encoding scheme alleviates the above problem by essentially interleaving equality- and range-encoded bitmaps. For an attribute with cardinality C, *OREO* consists of $(C-1)$ bitmaps $\mathcal{O} = \{O^1, O^2, \ldots, O^{C-1}\}$, such that $O^{C-1} = \bigvee_{i \text{ is even}} E^i$; and for $1 \leq i < C-1$, $O^i = E^{i-1} \vee E^i$ if i is even; otherwise, $O^i = R^i$. Details on the design of OREO and its evaluation expressions are given elsewhere [CI98a].

5.3 Equality-Interval Encoding Scheme

The third hybrid encoding scheme, *equality-interval encoding* (denoted by $\mathcal{EI}$), is simply a combination of both the equality and interval encoding schemes; i.e., $\mathcal{EI} = \mathcal{E} \cup \mathcal{I}$. Note that $\mathcal{EI}$ reduces to $\mathcal{E}$ when $C \leq 3$.

[5] OREO = Oscillating Range and Equality Organization.

Equality and range queries are evaluated using equality- and interval-encoded bitmaps, respectively. In terms of its space- and time-efficiency, $\mathcal{EI}$ falls in between the first two hybrid schemes: it is more time-efficient than $\mathcal{O}$ and more space-efficient than $\mathcal{ER}$.

5.4 Variant of Equality-Interval Encoding Scheme

The final hybrid encoding scheme, denoted by $\mathcal{EI}^*$, is a variant of $\mathcal{EI}$ that requires about two-thirds of the space of $\mathcal{EI}$ without sacrificing its time-efficiency for many queries. Its design is based on the observation that bitmap $I^0 = [0, \lfloor \frac{C}{2} \rfloor - 1]$ is commonly accessed for query evaluation. By combining pairs of equality-encoded bitmaps E^x and E^y such that $x \leq \lfloor \frac{C}{2} \rfloor - 1 < y$, each equality query can be evaluated with one such combined bitmap and I^0. More precisely, $\mathcal{EI}^* = \mathcal{I} \cup \{P^1, P^2, \ldots, P^r\}$, where $r = \lceil \frac{C-4}{2} \rceil$ and $P^i = E^i \cup E^{i+m+1}$. Note that $\mathcal{EI}^*$ reduces to $\mathcal{I}$ when $C \leq 4$. Thus, $\mathcal{EI}^*$ requires only $(\lceil \frac{C}{2} \rceil + \lceil \frac{C-4}{2} \rceil)$ bitmaps, which is about two-thirds of the $(C + \lceil \frac{C}{2} \rceil)$ bitmaps required for $\mathcal{EI}$. The evaluation expressions using $\mathcal{EI}^*$ are given elsewhere [CI98a].

6 Query Processing with Multi-Component Indexes

So far, we have presented the various encoding schemes and their evaluations assuming one-component indexes. In this section, we explain how arbitrary membership queries (and effectively, how individual interval queries as well) are processed for the more general case of multi-component indexes.

Query processing with multi-component indexes consists of two phases: a *query rewrite phase* followed by a *query evaluation phase*. The first phase transforms the input query into a query evaluation graph, where each internal node in the graph represents a logical operator (AND, OR, NOT, XOR) and each leaf node represents a bitmap. The second phase optimizes the execution of the query evaluation graph by taking into account the allocated buffer space to minimize input bitmap rescans and intermediate result replacements. The details of each phase are explained in the following subsections.

We use n to denote the number of components in a bitmap index with base $< b_n, b_{n-1}, \ldots, b_1 >$, and $v_n v_{n-1} \ldots v_n$ to denote the digits of an attribute value v decomposed using the base of the index as given by Equation (3).

6.1 Query Rewrite Phase

This phase consists of the following 3 steps:

1. *Membership Query Rewrite*
   First, if the input query is a membership query, it is rewritten into a disjunction of a minimal number of

interval queries; an example of this straightforward rewrite was shown in Section 5.

2. *Interval Query Rewrite*
Next, each predicate constant in each interval query is decomposed into n digits using the base of the n-component index, and each interval query is then rewritten into a more detailed expression involving predicates at the digit-level. For example, consider the interval query "$A \leq 85$" and a base-$< 10, 10 >$ index. After decomposition of the predicate constant, the query becomes "$A_2A_1 \leq 8_{10}5_{10}$" and is further rewritten as "$(A_2 \leq 7) \vee [(A_2 = 8) \wedge (A_1 \leq 5)]$". This rewrite step is a function of both the query class (equality, one-sided range, two-sided range) as well as the index's encoding scheme; details of this step are explained in Section 6.2.

3. *Predicate-Level Rewrite*
Finally, based on the encoding scheme of the index, each predicate is rewritten into a bitmap-level evaluation expression. Such evaluation expressions correspond to the one-component evaluation expressions that we have already presented along with each encoding scheme. Continuing with the previous example, and assuming that the index bitmap is equality-encoded (Equation (1)), the final bitmap-level evaluation expression is "$\overline{(E_2^8 \vee E_2^9)} \vee [E_2^8 \wedge \overline{(E_1^6 \vee E_1^7 \vee E_1^8 \vee E_1^9)}]$".

6.2 Interval Query Rewrite

This section elaborates on the interval query rewrite step of the query rewrite phase; specifically, we explain with examples how an interval query is rewritten for the three interval query subclasses.

Equality Queries

An equality query is rewritten as a conjunction of n equality predicates, one per index component. For example, for a base-$< 10, 10, 10 >$ bitmap index, the query "$A = 357$" is rewritten as "$(A_3 = 3) \wedge (A_2 = 5) \wedge (A_1 = 7)$".

In general, if $EQ(i, j, v)$ denotes the evaluation of "$A_jA_{j-1} \ldots A_i = v_jv_{j-1} \ldots v_i$", where $1 \leq i \leq j \leq n$, then

$$EQ(i, j, v) = \bigwedge_{k=i}^{j} (A_k = v_k). \tag{7}$$

Hence, the evaluation of an equality query "$A = v$" is given by $EQ(1, n, v)$.

One-Sided Range Queries

There are two basic alternatives for rewriting a one-sided range query. For example, for a base-$< 10, 10 >$ bitmap index, the query "$A \leq 57$" can be rewritten as either "$(A_2 \leq 4) \vee [(A_2 \leq 5) \wedge (A_1 \leq 7)]$" or "$(A_2 \leq 4) \vee [(A_2 = 5) \wedge (A_1 \leq 7)]$". Both forms are equivalent, and the choice is made based on whether the index's encoding scheme is more efficient for range or equality queries: the first option is more efficient for range encoding, while the second option is more efficient for equality encoding.

In addition, to avoid unnecessary predicate evaluations, the least significant digits in the predicate constant are dropped if they all have maximal values. For example, for a base-$< 10, 10, 10 >$ index, the query "$A \leq 499$" will be simplified to "$A_3 \leq 4$" to avoid two unnecessary predicate evaluations for the two least significant digits. One-sided queries of the form "$A \geq v$" are evaluated as "$\overline{A \leq v - 1}$", $v > 0$.

In general, if $LE(k, v)$ and $GE(k, v)$ denote the evaluations "$A_kA_{k-1} \ldots A_1 \leq v_kv_{k-1} \ldots v_1$" and "$A_kA_{k-1} \ldots A_1 \geq v_kv_{k-1} \ldots v_1$", respectively, where $1 \leq k \leq n$, then

$$LE(k, v) = \begin{cases} (A_k \leq v_k - 1) \vee (\alpha_k \wedge LE(k-1, v)) & \text{if } k > 1,\ v_k > 0, \\ \alpha_k \wedge LE(k-1, v) & \text{if } k > 1,\ v_k = 0, \\ (A_k \leq v_k - 1) \vee LE(k-1, v) & \text{if } k > 1,\ v_k = b_k - 1, \\ A_k \leq v_k & \text{if } k = 1. \end{cases} \tag{8}$$

Hence, the evaluation of a one-sided range query "$A \leq v$" is given by $LE(n, v)$, while $GE(n, v) = \overline{LE(n, v-1)}$. In the above, α_k is either $(A_k = v_k)$ or $(A_k \leq v_k)$ depending on the bitmap encoding scheme. Furthermore, $LE(n, v) = LE(n', v)$ if $v_{n'} < b_{n'}$ and $v_i = b_i - 1$ for $1 \leq i < n'$.

Two-Sided Range Queries

In general, two-sided range queries are evaluated as a conjunction of two one-sided range queries. For example, for a base-$< 10, 10, 10, 10 >$ index, the query "$4254 \leq A \leq 8015$" is rewritten as "$(A \geq 4254) \wedge (A \leq 8015)$". To minimize bitmap operations, a prefix of common most significant digits are evaluated as equality queries. For example, the query "$4326 \leq A \leq 4377$" is rewritten as "$(A_4 = 4) \wedge (A_3 = 3) \wedge (26 \leq A_2A_1 \leq 77)$".

Again, whenever there is a choice, the rewrite step chooses predicates to match the strengths of the encoding of the available indexes. Continuing on with the previous example, for equality-encoded indexes, the evaluation expression will be further refined into "$(A_4 = 4) \wedge (A_3 = 3) \wedge \{(3 \leq A_2 \leq 6) \vee [(A_2 = 2) \wedge (A_1 \geq 6)] \vee [(A_2 = 7) \wedge (A_1 \leq 7)]\}$".

6.3 Query Evaluation Phase

In general, the bitmap evaluation expression generated by the rewrite phase for a membership query is a not a linear tree but an acyclic graph[6]. Optimizing the

[6]The evaluation graph simplifies to a linear tree when the query is an equality or a one-sided range query and the index is range- or interval-encoded.

query evaluation becomes a scheduling problem: for a given allocation of buffer space, find the best bitmap access and replacement schedule that minimizes both the number of re-scans for input bitmaps as well as re-scans for intermediate results that were replaced.

There are two simple query evaluation strategies that represent the two extreme points in the solution space with regards to their buffer space requirement. At one extreme, we have a *query-wise* evaluation that evaluates one constituent interval query at a time so that only one intermediate result is maintained. This approach requires the least amount of buffer space to run, but is also expected to incur the highest number of bitmap re-scans. At the other extreme, we have a *component-wise* evaluation that evaluates all the constituent interval queries one component at a time; that is, we first evaluate all the predicates for the first digit, and then proceed to evaluate all the predicates for the second digit, and so on. The number of intermediate results that needs to be maintained is $(n_1 + 2n_2)$, where n_1 is the number of equality and one-sided range queries, and n_2 is the number of two-sided range queries. This approach requires the largest amount of buffer space to run, but does not incur any bitmap/intermediate result re-scans, and therefore raises no scheduling problem.

For the purpose of our performance study (to be described in the next section), we use the component-wise evaluation strategy. We plan to look into efficient heuristics for the scheduling problem as part of future work.

7 Performance Study

This section presents experimental results comparing the space-time performance of the various bitmap encoding schemes for evaluating both interval as well as membership queries. The experiments were run on a 200 MHz Intel Pentium Pro processor with 64 MB main memory running Solaris X86 2.6. The indexes were created using the Unix file system on a 2.1 GB Quantum Fireball disk; the file system buffer was flushed before each query was run.

Data Sets: The experiments were conducted using synthetic data sets with over 6 million records. The data sets are characterized by two parameters: the attribute cardinality, denoted by C, and the attribute value distribution. We created indexes with C values of 50 and 200, and generated the attribute value distribution using the Zipf distribution with values of 0,1,2 and 3 for its skew parameter z. Note that skew increases with z, with $z = 0$ corresponding to the uniform distribution. The data sets were generated such that there was no correlation between the attribute values and their frequencies.

Queries: We used 8 different query sets for the experiments; each query set is characterized by two parameters: the total number of interval queries in a membership query, denoted by N_{int}, with $N_{int} = 1, 2,$ and 5; and the number of equality queries among these interval queries, denoted by N_{equ}, with $N_{equ} = 0, \lceil \frac{N_{int}}{2} \rceil$, and N_{int}. Ten queries were randomly generated for each query set.

Indexes: The bitmap indexes were generated by varying three parameters: the bitmap index encoding scheme ($\mathcal{E}$, $\mathcal{R}$, $\mathcal{I}$, $\mathcal{ER}$, $\mathcal{O}$, $\mathcal{EI}$, and $\mathcal{EI}^*$), the bitmap index decomposition, and whether or not the index is compressed. The bitmap compression algorithm used is a byte-aligned run-length encoding scheme proposed by Antoshenkov [Ant93] which is used in Oracle8 Database Server [Jak97].

The space-efficiency of an index is in terms of the disk space for storing all its bitmaps. The time-efficiency of an index for a query set is in terms of its average processing time over all 10 queries in the query set; the processing time includes both disk I/O time for reading bitmaps, as well as CPU time for bitmap operations (including decompression time for compressed bitmaps). As mentioned earlier, we used the component-wise evaluation strategy; a buffer pool size of 11 MB was adequate for our experiments. We present results only for $C = 50$ as the results for $C = 200$ were similar. Also, we do not present any results for hybrid encoding schemes, as they rarely offered a better index than non-hybrid ones (occasionally such an index had a slightly lower time at the expense of much higher space).

7.1 Space-Efficiency and Compressibility of Encoding Schemes

Figure 6 compares the space-efficiency and compressibility of the various encoding schemes for the case where $C = 50$ and $z = 1$. The comparison is in terms of three different space ratios (the precise definitions are given below) as a function of the number of index components, n. Among all the bitmap indexes with the same encoding and the same number of index components, the index that yields the "best" space ratio (i.e., smallest ratio value) is plotted on each graph.

Figure 6(a) compares the space-efficiency of the encoding schemes in terms of the ratio of the size of the uncompressed form of an n-component index to that of the uncompressed, one-component equality-encoded index; the latter is used as the base case as it corresponds to the simplest and most common bitmap index design. As expected, for uncompressed indexes, interval encoding is the most space-efficient and equality encoding is the least space-efficient.

Figure 6(b) compares the compressibility of the encoding schemes in terms of the ratio of the size of the compressed version of an n-component index to that of its uncompressed version. The graph shows that

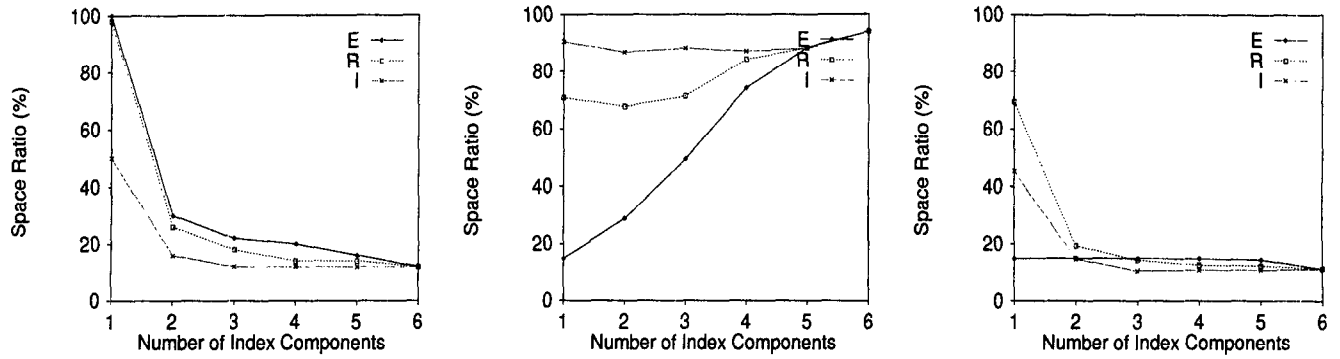


Figure 6: Space-Efficiency and Compressibility of Basic Encoding Schemes ($C = 50, z = 1$). The y-axis measures the ratio of the space of two indexes. (a) Uncompressed Index to Uncompressed One-Component Equality-Encoded Index. (b) Compressed Index to Uncompressed Index. (c) Compressed Index to Uncompressed One-Component Equality-Encoded Index.

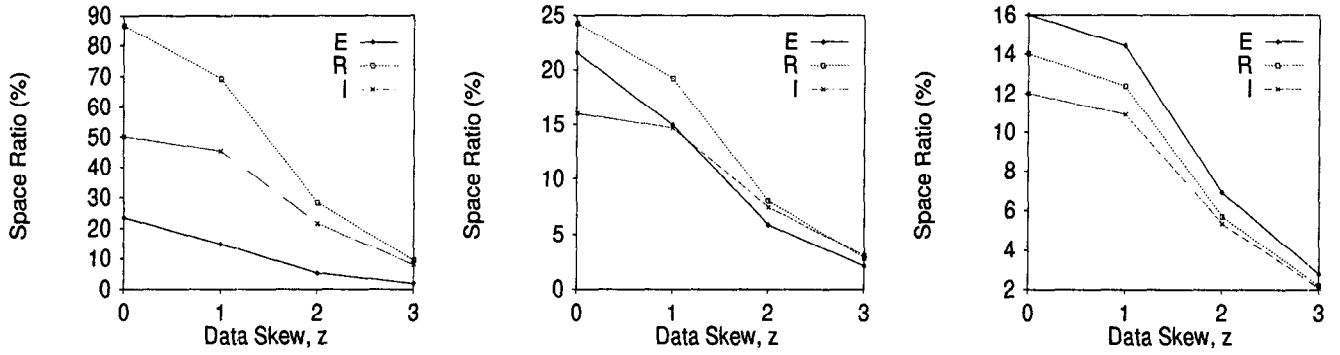


Figure 7: Effect of Data Skew on Space-Efficiency of Compressed Indexes ($C = 50$). Compares the Ratio of Space of a Compressed n-Component Index to that of the Uncompressed One-Component Equality-Encoded Index.

equality encoding gives the best compressibility, while interval encoding has the worst compressibility. This result, which is consistent throughout our experiments, is not surprising. Since each equality-encoded bitmap represents only a single attribute value, equality-encoded bitmaps are more "sparse" and are therefore more amenable to the run-length encoding form of compression. In contrast, each interval-encoded bitmap represents half the values in the attribute domain; so the bitmaps are less likely to have long runs of the same bit value. Thus, although interval encoding is the most space-efficient scheme in terms of the number of bitmaps, it is the least compressible scheme.

Figure 6(c) compares the effect of compression on the space-efficiency of the encoding schemes in terms of the ratio of the size of the compressed form of an n-component index to that of the uncompressed, one-component equality-encoded index. The results show that interval encoding is generally the most space-efficient encoding scheme for compressed indexes as well.

Figure 7 shows the effect of data skew on the space-efficiency of compressed n-component indexes, for $n = 1, 2$, and 5. As the data skew increases, the space-efficiency of compressed indexes improves for all encoding schemes, and the difference in space-efficiency among the encoding schemes also decreases.

7.2 Space-Time Tradeoff of Encoding Schemes

In this section, we compare the performance of the encoding schemes in terms of their space-time tradeoffs. For each basic encoding scheme S, we show only one of its uncompressed and compressed forms (labelled "S" and "Compressed S", respectively), depending on which one was in general dominant in terms of overall space-time performance.

Figures 8 compares the space-time tradeoff of the encoding schemes for $C = 50$ and $z = 1$; the graphs on the same row (column) have the same value for

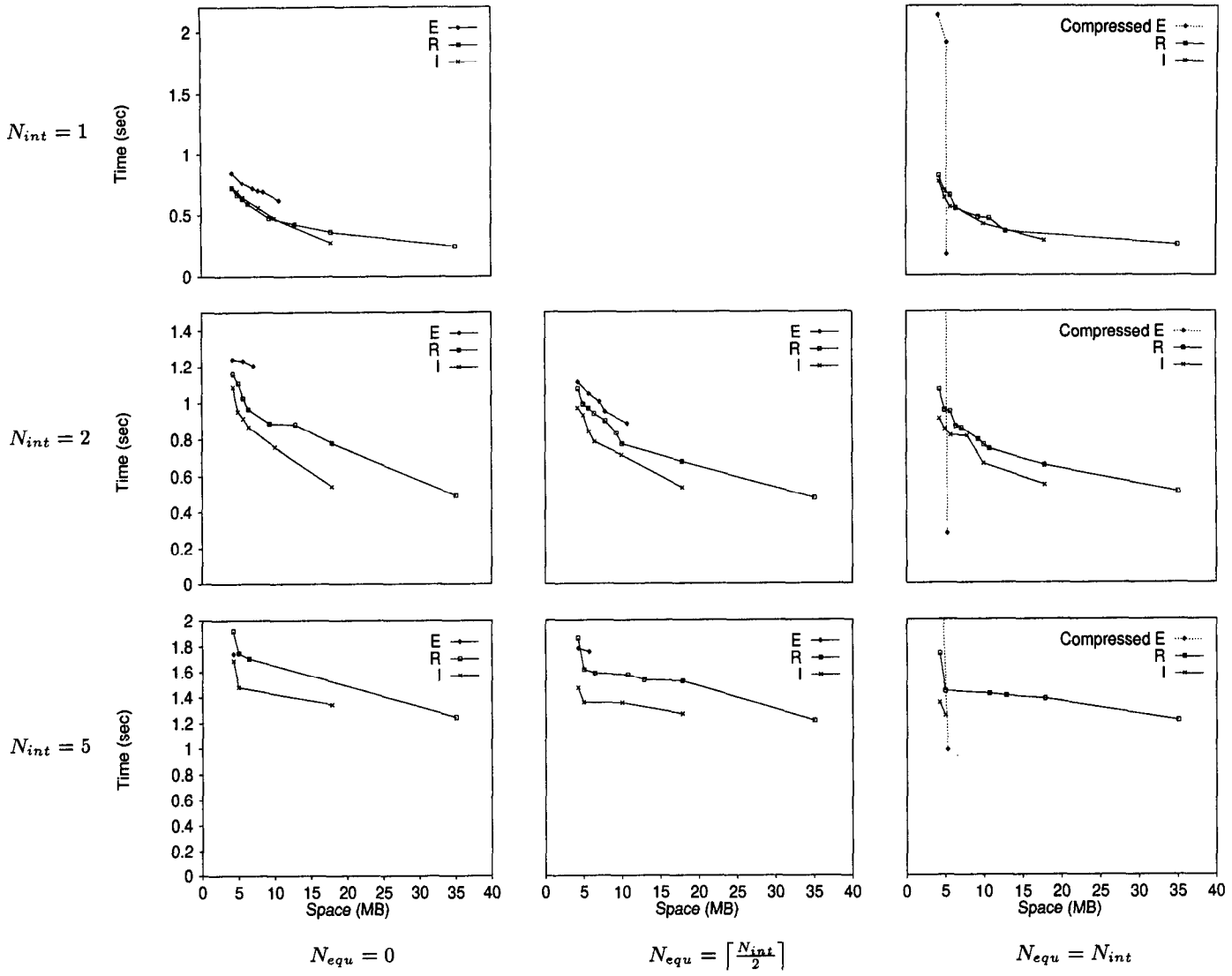


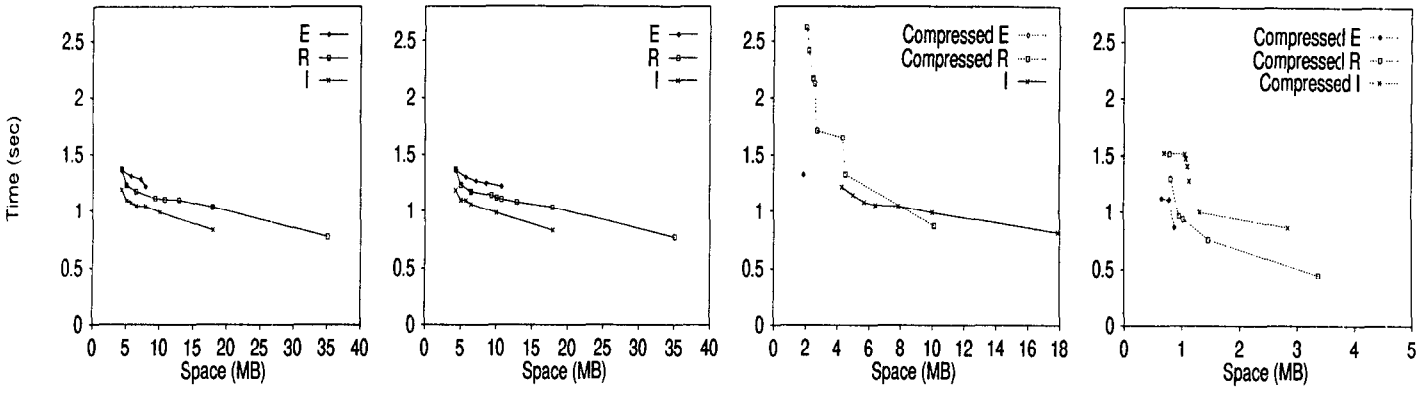
Figure 8: Comparison of Space-Time Tradeoff ($C = 50, z = 1$).

N_{int} (N_{equ}). The results show that interval encoding generally has the best space-time tradeoff except for cases where $N_{equ} = N_{int}$ in which equality encoding is the winner, in the sense that the most time-efficient equality-encoded index has rather low space requirements as well.

Figure 9 compares the effect of data skew z on the space-time tradeoff of the encoding schemes. The processing time shown is the averaged timing over all queries in all 8 query sets. The results show that for low-to-medium-skew data (graphs (a) and (b)), uncompressed indexes have better space-time performance than compressed ones and interval encoding is the overall winner, whereas for medium-to-high-skew data (graphs (c) and (d)), compressed indexes have better space-time performance than uncompressed ones. The benefit of using compression increases with data skew because the bitmaps become more compressible, and both the I/O cost as well as the decompression overhead also become lower.

8 Conclusions

Bitmaps indexes appear to be an effective and efficient structure to help with processing complex ad-hoc queries. Most earlier studies and commercial index implementations have focused on one-component indexes encoded using the equality or the range encoding schemes, typically in compressed form. In this paper, we have first demonstrated analytically that these two encodings are not in general optimal for interval queries. In pursuit of optimality, we have introduced and studied interval encoding, which has been proven optimal for all but equality selection queries, as well as four other hybrid encodings. To deal with general mem-



(a) $z = 0$. (b) $z = 1$. (c) $z = 2$. (d) $z = 3$.

Figure 9: Effect of Data Skew on Space-Time Tradeoff ($C = 50$).

bership queries, we have also devised a query processing algorithm for multi-component indexes (independent of encoding) that accesses each component once on behalf of all subqueries of a query, thus minimizing bitmap rescans from disk. Finally, we have conducted an experimental evaluation of all encoding schemes, in both compressed and uncompressed forms, which has revealed some interesting facts about their behavior: interval encoding has the overall best space-time performance, losing to equality encoding only on equality or equality-rich membership queries; for low-to-medium-skew data, uncompressed indexes have better space-time performance than compressed ones and interval encoding is the overall winner, whereas the situation is reversed for medium-to-high-skew data.

References

[Ant93] G. Antoshenkov. Byte Aligned Data Compression. U.S. Patent No: 142640, October 1993.

[CI98a] C.Y. Chan and Y.E. Ioannidis. An Efficient Bitmap Encoding Scheme for Selection Queries. Computer Sciences Department, University of Wisconsin-Madison, 1998. http://www.cs.wisc.edu/~cychan/interval.ps.

[CI98b] C.Y. Chan and Y.E. Ioannidis. Bitmap Index Design and Evaluation. In *Proceedings of the Intl. ACM SIGMOD Conference*, pages 355–366, Seattle, Washington, June 1998.

[Ede95] H. Edelstein. Faster Data Warehouses. *Information Week*, pages 77–88, December 1995.

[Inf] Informix Inc. Informix Decision Support Indexing for the Enterprise Data Warehouse. http://www.informix.com/informix/corpinfo/zines/whiteidx.htm.

[Jak97] H. Jakobsson. Bitmap Indexing in Oracle Data Warehousing. Database seminar at Stanford University. http://www-db.stanford.edu/dbseminar/Archive/Fall97/-slides/oracle/, October 1997.

[OG95] P. O'Neil and G. Graefe. Multi-Table Joins Through Bitmapped Join Indices. *ACM SIGMOD Record*, pages 8–11, September 1995.

[O'N87] 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.

[O'N97] P. O'Neil. Informix Indexing Support for Data Warehouses. *Database Programming and Design*, 10(2):38–43, February 1997.

[OQ97] P. O'Neil and D. Quass. Improved Query Performance with Variant Indexes. In *Proceedings of the Intl. ACM SIGMOD Conference*, pages 38–49, Tucson, Arizona, May 1997.

[Syb97] Sybase Inc. 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.

[WB98] M.C. Wu and A.P. Buchmann. Encoded Bitmap Indexing for Data Warehouses. In *Proceedings of the Intl. Conference on Data Engineering*, pages 220–230, Orlando, Florida, February 1998.

[Win99] R. Winter. Indexing Goes a New Direction. *Intelligent Enterprise*, 2(2):70–73, January 1999.

[WLO+85] H.K.T. Wong, H-F. Liu, F. Olken, D. Rotem, and L. Wong. Bit Transposed Files. In *Proceedings of the Intl. Conference on Very Large Data Bases*, pages 448–457, Stockholm, 1985.

[WLO+86] H.K.T. Wong, J.Z. Li, F. Olken, D. Rotem, and L. Wong. Bit Transposition for Very Large Scientific and Statistical Databases. *Algorithmica*, 1(3):289–309, 1986.