BITMAP INDEXES

E0 261

Jayant Haritsa
Computer Science and Automation
Indian Institute of Science

Low-Cardinality Domains

- e.g. in IISc: Gender, Program, Home state
- B-trees or hashing don’t work well for low-cardinality domains, which are common in decision-support environments
- Intersecting lists of RIDS for satisfying conjunction of predicates is an expensive operation

Bitmap Index

- Bitmap Index is defined on attribute A as a sequence of M bitmaps, where M is the number of distinct values of A, and record 'j' has a 1 in the bitmap corresponding to its value, and 0 otherwise – the length of each bitmap is equal to the relation's cardinality.

Bit-vector:
1 bit for each possible value.

Gender M F	SRno	name	gender	grade	Grade A^+ A B C D
10	112	Sachin	M	A+	10000
10	115	Rishabh	M	D	00001
01	119	Katrina	F	C	00010
10	113	Ashwin	M	D	00001

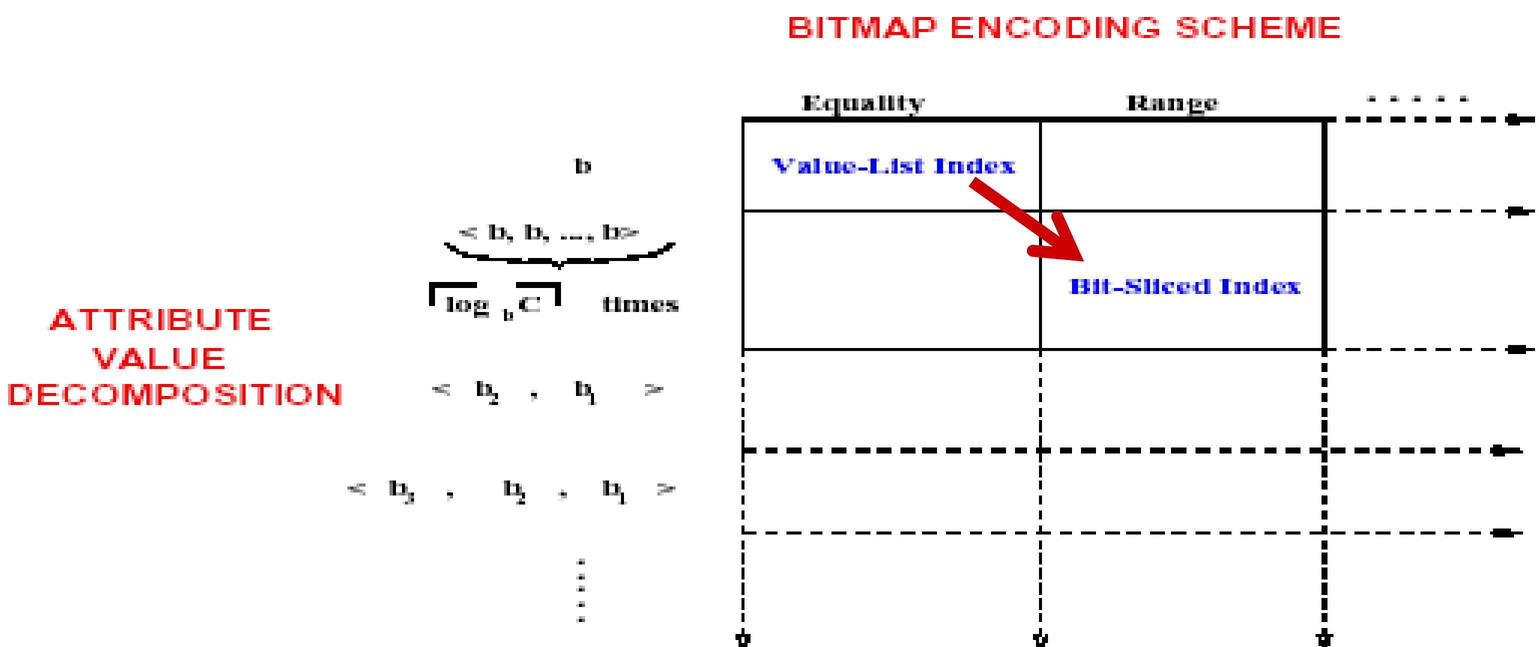
Query: Male students with A+ grade
derived by ANDing the Male and A+ bitvectors

Features

- well suited for low cardinality columns
- utilizes bitwise operations like AND, OR, XOR, NOT which are efficiently supported by hardware
- uses small amount of space
- performs efficiently with columns involving scalar functions (e.g.,COUNT)
- many variations to reduce space requirement and improve query performance

Design Space

- **Attribute Value Decomposition**
  - Determines structure of index - number and size of index component
- **Index Encoding Scheme**
  - Determines how index components are encoded



(1) Attribute Value Decomposition

- Given C, the attribute value cardinality, and a sequence of n numbers $<b_n, b_{n-1}, \ldots, b_1>$, such that $C \leq \pi b_i$, attribute value A is decomposed into n digits $A_n A_{n-1} \ldots A_1$ where A_i is a base-b_i digit
- Example C = 1000 and attribute value A = 256

$<b_n, b_{n-1}, \ldots, b_1>$	Decomposition of A
<1000>	256
<50,20>	12 (20) + 16
<32,32>	8 (32) + 0
<5,20,10>	1(20)(10) + 5(10) + 6

- Each $<b_n, b_{n-1}, \ldots, b_1>$ base of index defines an n-component index.

AVD Details

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

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

$$
\begin{aligned}
v &= 256 \quad \text{and index } <5,20,10> \\
&= 25\,(10) + 6 \\
&= 1\,(20)(10) + 5\,(10) + 6
\end{aligned}
$$

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

- If **$b_n = b_{n-1} = \ldots = b_1 = b$**, base is called *uniform* and the index is called *base-b.*

(2) Bitmap Encoding Schemes

- Two basic ways to encode a base $\mathbf{b_i}$ value $\mathbf{x}$ (i.e.,$\mathbf{0 \leq x < b_i}$) using b_i bits

Encoding Scheme	b_i-bit Representation for value x						
	$b_i - 1$	$\cdots$	$x+1$	x	$x-1$	$\cdots$	0
Equality	0	$\cdots$	0	1	0	$\cdots$	0
Range	1	$\cdots$	1	1	0	$\cdots$	0

- Equality Encoded Bitmap E_i^x = { records with $A_i = x$ }
- Range Encoded Bitmap R_i^x = { records with $A_i \leq x$ }
  - $R_i^{b_i - 1}$ is not materialized since all its bits are set to 1.

Equality Encoded Base <3,3> Index

	$\pi_A(R)$		B_2^2	B_2^1	B_2^0	B_1^2	B_1^1	B_1^0
1	3	$\xrightarrow{1\times3+0}$	0	1	0	0	0	1
2	2	$\xrightarrow{0\times3+2}$	0	0	1	1	0	0
3	1	$\xrightarrow{0\times3+1}$	0	0	1	0	1	0
4	2	$\xrightarrow{0\times3+2}$	0	0	1	1	0	0
5	8	$\xrightarrow{2\times3+2}$	1	0	0	1	0	0
6	2	$\xrightarrow{0\times3+2}$	0	0	1	1	0	0
7	2	$\xrightarrow{0\times3+2}$	0	0	1	1	0	0
8	0	$\xrightarrow{0\times3+0}$	0	0	1	0	0	1
9	7	$\xrightarrow{2\times3+1}$	1	0	0	0	1	0
10	5	$\xrightarrow{1\times3+2}$	0	1	0	1	0	0
11	6	$\xrightarrow{2\times3+0}$	1	0	0	0	0	1
12	4	$\xrightarrow{1\times3+1}$	0	1	0	0	1	0

Range Encoded Base <3,3> Index

	$\pi_A(R)$		B_2^1	B_2^0	B_1^1	B_1^0
1	3	$\xrightarrow{1\times3+0}$	1	0	1	1
2	2	$\xrightarrow{0\times3+2}$	1	1	0	0
3	1	$\xrightarrow{0\times3+1}$	1	1	1	0
4	2	$\xrightarrow{0\times3+2}$	1	1	0	0
5	8	$\xrightarrow{2\times3+2}$	0	0	0	0
6	2	$\xrightarrow{0\times3+2}$	1	1	0	0
7	2	$\xrightarrow{0\times3+2}$	1	1	0	0
8	0	$\xrightarrow{0\times3+0}$	1	1	1	1
9	7	$\xrightarrow{2\times3+1}$	0	0	1	0
10	5	$\xrightarrow{1\times3+2}$	1	0	0	0
11	6	$\xrightarrow{2\times3+0}$	0	0	1	1
12	4	$\xrightarrow{1\times3+1}$	1	0	1	0

Evaluation Algorithm for Selection Queries on Range-Encoded Bitmaps

- RangeEval [SIGMOD97]
  - Evaluates each range predicate by computing two bitmaps
    - The B_{EQ} bitmap
    - Either B_{GT} or B_{LT}

- RangeEval-Opt (this paper):
  - avoids the intermediate equality predicate B_{EQ} evaluation by evaluating each range predicate in term of only ≤ .
    - $A < v$ == $A \leq v-1$ $A > v$ == $!(A \leq v)$ $A \geq v$ == $!(A \leq v-1)$
  - Working with only one bitmap B which
    - reduces number bitmap operations by 50%
    - one less bitmap scan for range predicate evaluation

Algorithm: RangeEval

- $B_{LT} = B_{LT} \vee (B_{EQ} \wedge B_i^{v_i-1})$
- $B_{GT} = B_{GT} \vee (B_{EQ} \wedge \overline{B_i^{v_i}})$
- $B_{EQ} = B_{EQ} \wedge (B_i^{v_i} \oplus B_i^{v_i-1})$

Algorithm `RangeEval`

$B_{GT} = B_{LT} = B_0;$
$B_{EQ} = B_{nn};$
let $v = v_n v_{n-1} \ldots v_1;$
for i = n **downto** 1 **do**
    if $(v_i > 0)$ **then**
        $B_{LT} = B_{LT} \vee (B_{EQ} \wedge B_i^{v_i-1});$
        if $(v_i < b_i - 1)$ **then**
            $B_{GT} = B_{GT} \vee (B_{EQ} \wedge \overline{B_i^{v_i}});$
            $B_{EQ} = B_{EQ} \wedge (B_i^{v_i} \oplus B_i^{v_i-1});$
        else
            $B_{EQ} = B_{EQ} \wedge \overline{B_i^{b_i-2}};$
    else
        $B_{GT} = B_{GT} \vee (B_{EQ} \wedge \overline{B_i^0});$
        $B_{EQ} = B_{EQ} \wedge B_i^0;$
$B_{NE} = \overline{B_{EQ}} \wedge B_{nn};$
$B_{LE} = B_{LT} \vee B_{EQ};\ B_{GE} = B_{GT} \vee B_{EQ};$
return $B_{op};$

RangeEval: <3,3> Example

	A	V2	V1	B_2^1	B_2^0	B_1^1	B_1^0
1	3	1	0	1	0	1	1
2	2	0	2	1	1	0	0
3	1	0	1	1	1	1	0
4	7	2	1	0	0	1	0
5	8	2	2	0	0	0	0

$b_2 = 3,\ b_1 = 3$

Consider selection predicate $A \leq 5$

$<v_2,v_1> = <1,2>$
$= [1,0][0,0]$

$i = 2,\ v_2 = 1$

$B_{LT} \vee (B_{EQ} \wedge B_i^{v_i-1}) = B_{LT}$
$0 \vee (1 \wedge 0) = 0$
$0 \vee (1 \wedge 1) = 1$
$0 \vee (1 \wedge 1) = 1$
$0 \vee (1 \wedge 0) = 0$
$0 \vee (1 \wedge 0) = 0$

$B_{EQ} \wedge (B_i^{v_i} \oplus B_i^{v_i-1}) = B_{EQ}$
$1 \wedge 1 = 1$
$1 \wedge 0 = 0$
$1 \wedge 0 = 0$
$1 \wedge 0 = 0$
$1 \wedge 0 = 0$

RangeEval (Example continued)

	A	→	V2	V1	→	B_2^1	B_2^0	B_1^1	B_1^0
1	3		1	0		1	0	1	1
2	2		0	2		1	1	0	0
3	1		0	1		1	1	1	0
4	7		2	1		0	0	1	0
5	8		2	2		0	0	0	0

Consider selection predicate $A \leq 5$

$\langle v_2, v_1 \rangle = \langle 1,2 \rangle$
$= [1,0][0,0]$

$i = 1, \; v_1 = 2$

$B_{LT} \vee (B_{EQ} \wedge B_i^{v_i - 1}) = B_{LT}$	$B_{EQ} \wedge \overline{B_i^{b_i - 2}} = B_{EQ}$	$B_{LE} = B_{LT} \vee B_{EQ}$
$0 \vee (1 \wedge 1) = 1$	$1 \wedge 0 = 0$	1
$1 \vee (0 \wedge 0) = 1$	$0 \wedge 1 = 0$	1
$1 \vee (0 \wedge 1) = 1$	$0 \wedge 0 = 0$	1
$0 \vee (0 \wedge 1) = 0$	$0 \wedge 0 = 0$	0
$0 \vee (0 \wedge 0) = 0$	$0 \wedge 1 = 0$	0

RangeEval-Opt

$$LE(v,i) = \begin{cases} B_i^{v_i-1} \vee (B_i^{v_i} \wedge LE(v,i-1)) & \text{if } 1 < i \le n, \\ B_1^{v_1} & \text{if } i = 1. \end{cases}$$

$$EQ(v,i) = \begin{cases} (B_i^{v_i} \oplus B_i^{v_i-1}) \wedge EQ(v,i-1) & \text{if } 1 < i \le n, \\ B_1^{v_1} \oplus B_1^{v_1-1} & \text{if } i = 1. \end{cases}$$

Algorithm `RangeEval-Opt`

$B = B_1$;
if ($op \in \{<, \ge\}$) **then** $v = v - 1$;
let $v = v_n v_{n-1} \ldots v_1$;
if ($op \in \{<, >, \le, \ge\}$) **then**
    if ($v_1 < b_1 - 1$) **then** $B = B_1^{v_1}$;
    for i = 2 **to** n **do**
        if ($v_i \ne b_i - 1$) **then** $B = B \wedge B_i^{v_i}$;
        if ($v_i \ne 0$) **then** $B = B \vee B_i^{v_i-1}$;
else
    for i = 1 **to** n **do**
        if ($v_i = 0$) **then** $B = B \wedge B_i^0$;
        else if ($v_i = b_i - 1$) **then** $B = B \wedge \overline{B_i^{b_i-2}}$;
        else $B = B \wedge (B_i^{v_i} \oplus B_i^{v_i-1})$;
if ($op \in \{>, \ge, \ne\}$) **then**
    return $\overline{B} \wedge B_{nn}$;
else
    return $B \wedge B_{nn}$;

RangeEval-Opt (Example)

	A	V2	V1	B_2^1	B_2^0	B_1^1	B_1^0
1	3	1	0	1	0	1	1
2	2	0	2	1	1	0	0
3	1	0	1	1	1	1	0
4	7	2	1	0	0	1	0
5	8	2	2	0	0	0	0

$b_2 = 3,\ b_1 = 3$

Consider selection predicate $A \leq 5$

$<v_2, v_1> = <1,2>$
$= [1,0][0,0]$

$i = 2,\ v_2 = 1$

$B_i^{v_i - 1} \vee (B_i^{v_i} \wedge B) = B$

$0 \vee (1 \wedge 1) = 1$

$1 \vee (1 \wedge 1) = 1$

$1 \vee (1 \wedge 1) = 1$

$0 \vee (0 \wedge 1) = 0$

$0 \vee (0 \wedge 1) = 0$

Worst-case Analysis

Evaluation Algorithm	Evaluation Predicate	Number of Bitmap Operations					Number of Bitmap 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	$2n+1$	$2n$

For RangeEval Algorithm

$$B_{LT} = B_{LT} \vee (B_{EQ} \wedge B_i^{v_i-1})$$

$$B_{GT} = B_{GT} \vee (B_{EQ} \wedge \overline{B_i^{v_i}})$$

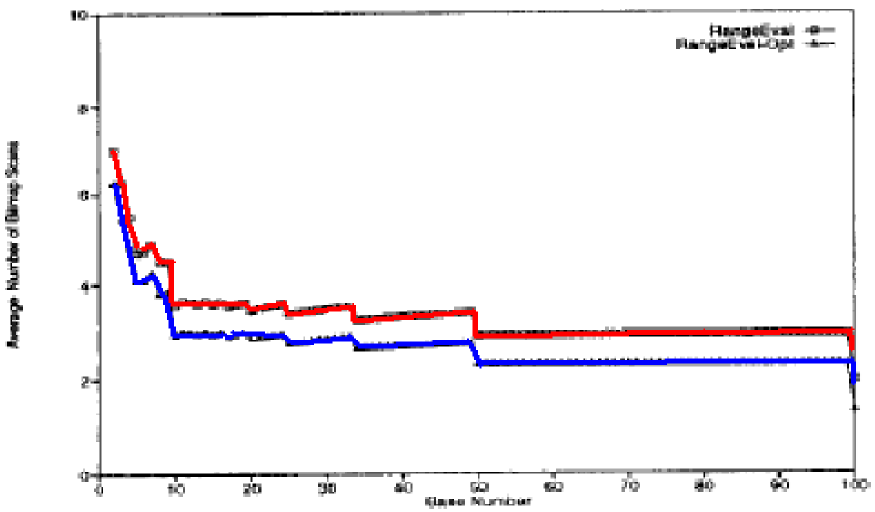
$$B_{EQ} = B_{EQ} \wedge (B_i^{v_i} \oplus B_i^{v_i-1})$$

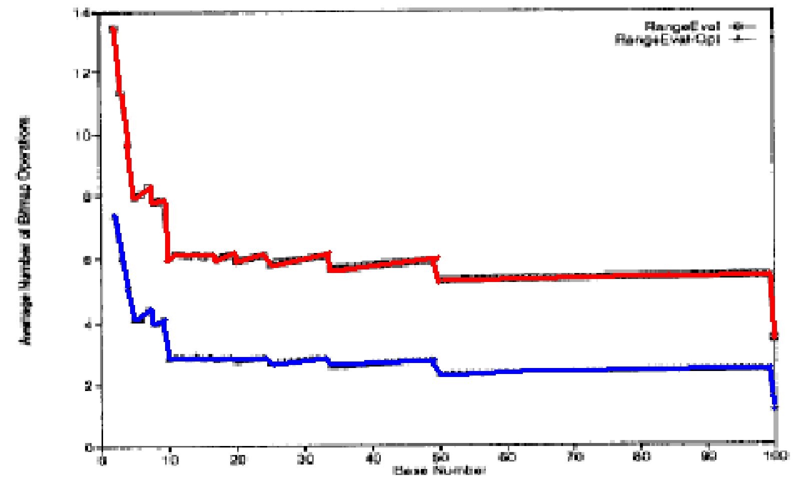
For RangeEval- Opt Algorithm

$$LE(v,i) = \begin{cases} B_i^{v_i-1} \vee (B_i^{v_i} \wedge LE(v,i-1)) & \text{if } 1 < i \leq n, \\ B_1^{v_1} & \text{if } i = 1. \end{cases}$$

$$EQ(v,i) = \begin{cases} (B_i^{v_i} \oplus B_i^{v_i-1}) \wedge EQ(v,i-1) & \text{if } 1 < i \leq n, \\ B_1^{v_1} \oplus B_1^{v_1-1} & \text{if } i = 1. \end{cases}$$

Experimental comparison





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

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

Space/Time Cost Models

- Space(I)
  - number of bitmaps stored
- Time(I)
  - Expected number of bitmap scans for a selection query
    - queries in the query space Q are uniformly distributed, where $Q = \{ A \text{ op } v : \text{op} \in \{\leq, \geq, <, >, =, \neq\}, 0 \leq v < C \}$
  - I/O and CPU are correlated, hence ignore CPU

Comparison of Encoding Schemes for Selection Queries

Theorem 5.1

For equality encoded Bitmap Index

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

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

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

For range encoded Bitmap Index

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

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

[with Range-Eval-Opt]

Proof of Time for Range Encoded Index with Range-Eval-Opt

Computed expected number of scans for **r (range predicate)** and **e (equality predicate)**

Range

$i = 1 \qquad E_{r,1} = \left(1 - \frac{1}{b_1}\right)$ Line 5 of REO algo

$\mathrm{i} = 2 \rightarrow n \qquad E_{r,i} = 1 * \Pr(scan = 1) + 2 * \Pr(scan = 2)$

Line 7 and 8 of REO algo

$$= 1\left(\frac{2}{b_i}\right) + 2\left(1 - \frac{2}{b_i}\right)$$

$$= 2\left(1 - \frac{1}{\mathrm{b_i}}\right)$$

$$\therefore E_r = \sum_{i=1}^{n} E_{r,i} = 2\left(n - \sum_{i=1}^{\mathrm{n}} \frac{1}{b_i}\right) - \left(1 - \frac{1}{b_1}\right)$$

Proof (contd)

Equality

$$E_{e,i} = 1 * \frac{2}{b_i} + 2(1 - \frac{2}{b_i}) = 2\left(1 - \frac{1}{b_i}\right)$$

Lines 11-13 of REO algo

$$\therefore E_e = \sum_{i=1}^{n} E_{e,i} = 2\left(n - \sum_{i=1}^{n} \frac{1}{b_i}\right)$$

$$=, \neq \quad <, \leq, >, \geq$$

$$\downarrow \qquad \downarrow$$

$$\text{Total Expected } \# = \frac{2}{6} E_e + \frac{4}{6} E_r$$

$$= 2\left[n - \sum_{i=1}^{n} \frac{1}{b_i} + \frac{1}{3}\left(\frac{1}{b_1} - 1\right)\right]$$

Space Optimal Indexes

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 $b^r (b-1)^{n-r} \geq C$.

   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.

Proof for Theorem 6.1

- **Lemma 1** For any n-component range-encoded bitmap index I_n with attribute cardinality C, there exists another n-component range-encoded bitmap index I'_n with attribute cardinality C such that
  - Space(I_n) = Space(I'_n)
  - the difference between any two base numbers of I'_n is at most one, and
  - the product of all the base numbers of I'_n is ≥ that of I_n

- **Proof:**
  - Let $<b_n, b_{n-1}, \ldots, b_1>$ be the base of I_n.
  - Let $b_{max} = \max\{b_i\}$, $1 \le i \le n$ $\quad b_{min} = \min\{b_i\}$, $1 \le i \le n$
  - Consider the non-trivial case where $b_{max} - b_{min} \ge 2$.
  - Let $b_p = b_{max}$ and $b_q = b_{min}$ for some $1 \le p, q \le n$.
  - Define another n-component bitmap index I'_n with base $<b'_n, b'_{n-1}, \ldots, b'_1>$, where

$$b'_i = \begin{cases} b_p - 1 & \text{if } i = p, \\ b_q + 1 & \text{if } i = q, \\ b_i & \text{otherwise.} \end{cases}$$

  - Clearly, Space(I_n) = Space(I'_n) and $b'_p b'_q = (b_p - 1)(b_q + 1) = b_p b_q + (b_p - b_q) - 1$ and $(b_p - b_q) \ge 2$, leading to $\pi\, b'_i > \pi\, b_i$
  - By a finite number of applications of the above base number adjustment to I_n, we will obtain an n-component bitmap index I'_n such that all the three properties hold.

Proof for Theorem 6.1 (contd)

Main Proof of (1):

For any n-component range encoded bitmap index with attribute cardinality C, its maximum base number must be at least equal to b , where $b = \lceil \sqrt[n]{C} \rceil$ because

$$(b-1)^n < C \le (b)^n$$

Also by Lemma 1 it follows that $< \underbrace{b-1, \ldots, b-1}_{n-r}, \underbrace{b, \ldots, b}_{r} >$

is an n-component space optimal index with attribute cardinality C.

Number of bitmaps = (n-r) (b-2) + r (b-1)

= n(b-2) + r

Make r as small as possible, subject to $b^r (b-1)^{n-r} \ge C$.

Proof for Theorem 6.1 (contd)

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

Proof:

Let $< b_n , b_{n-1} ,\ldots , b_1 >$ be the base of an n-component space-optimal bitmap index I_n with attribute cardinality C, where $n < \lceil log_2(C) \rceil$

$n < \lceil log_2(C) \rceil \Rightarrow \exists\ 1 \leq p \leq n$ such that $b_p > 2$

Define an (n+1)-component bitmap index I_{n+1} with attribute cardinality C and base $< b'_{n+1}, b'_n ,\ldots , b'_1 >$ where

$$b'_i = \begin{cases} 2 & \text{if } i = n+1, \\ \left\lceil \frac{b_p}{2} \right\rceil & \text{if } i = p, \\ b_i & \text{otherwise.} \end{cases}$$

I_{n+1} is well-defined since $\prod_{i=1}^{n+1} b'_i \geq C$

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

$Space(I_n) - Space(I_{n+1}) = b_p - 1 - \left\lceil \frac{b_p}{2} \right\rceil = \left\lfloor \frac{b_p}{2} \right\rfloor - 1 \geq 0$ (since $b_p > 2$).

It follows that an (n+1)-component space-optimal bitmap index is at least as space-efficient as an n-component space-optimal bitmap index.

Time Optimal Indexes

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

Proof :

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

Time(I_n) is minimized when $b_1 \geq b_i$, $1 < i \leq n$, and the base numbers are as small as possible (subject to the constraint $\prod_{i=1}^{n} b_i \geq C$).

Thus it follows that Time(I_n) is minimized when $b_i = 2$, $1 < i \leq n$.

leading to $b_1 = \lceil \frac{C}{2^{n-1}} \rceil$

Time Optimal Indexes

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

Proof:

By the equation $Time(I) = 2(n - \sum_{i=1}^{n} \frac{1}{b_i} + \frac{1}{3}(\frac{1}{b_1} - 1))$ and result (3)

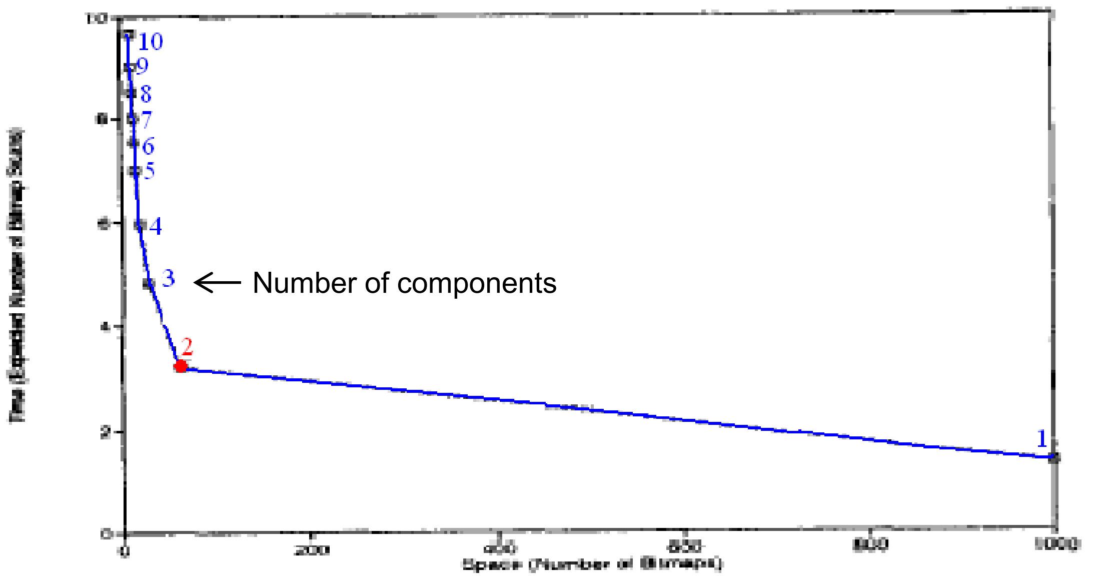
- $Time(I_n^{time}) = n + \frac{1}{3}(1 - \frac{4}{b_{n,1}})$ where $b_{n,1} = \left\lceil \frac{C}{2^{n}-1} \right\rceil$.
- So $Time(I_{n+1}^{time}) - Time(I_n^{time}) = 1 + \frac{4}{3 b_{n,1} b_{n+1,1}}(b_{n+1,1} - b_{n,1})$
- Since $b_{n,1} \geq b_{n+1,1} \geq 2 \Rightarrow Time(I_{n+1}^{time}) \geq Time(I_n^{time})$

Summary

- Space-optimal index is index with
  $n = \lceil log_2(C) \rceil$ and all b = 2
- Time-optimal index is index with
  n = 1 and b = C

Bitmap Index with optimal Space-time Tradeoff (Knee)

- *knee index*
  - corresponds to the index with the best space-time tradeoff.
  - approximate knee of all index with knee of space-optimal indexes (Fig10)



Space-time tradeoff for space optimal indexes

  - the knee of the space-time tradeoff graph for space-optimal indexes usually corresponds to a 2-component index

Theorem 7.1

- The base of the most time-efficient 2-component space-optimal index is given by < b_2 - δ, b_1 + δ >, where $b_1 = \lceil \sqrt{C} \rceil$ $b_2 = \lceil \frac{C}{b_1} \rceil$ and

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

Proof :

Let I be a 2- component index with base <b_2,b_1> as per above values.

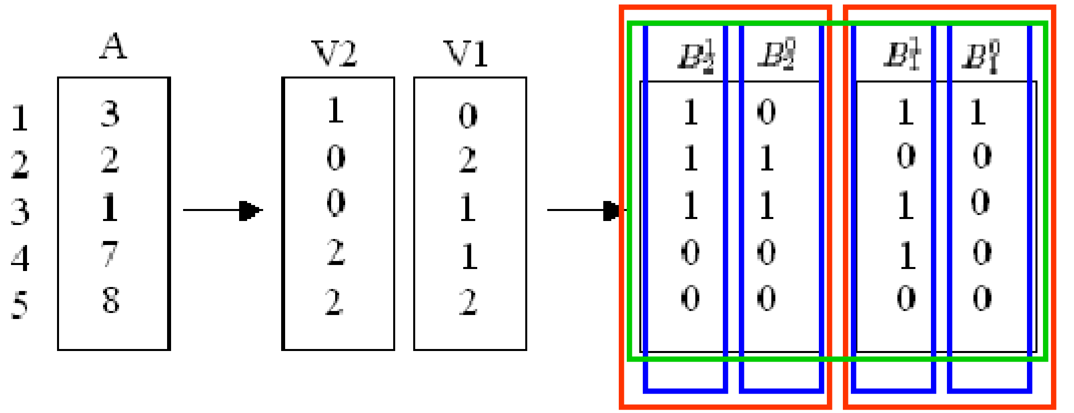
By Theorem 6.1, I is a 2-component space-optimal bitmap index.

Time(I) is minimized if $b_1 \geq b_2$ because $Time(I) = 2(n - \sum_{i=1}^{n} \frac{1}{b_i} + \frac{1}{3}(\frac{1}{b_1} - 1))$

Also, by Theorem 8.1, Time (I) is minimized when b_2 is minimized $\Rightarrow$ δ is largest non-negative integer value such that $(b_2 - \delta)*(b_1 + \delta) \geq C$

IMPLEMENTATION

Bitmap Index 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 * 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.

Experimental Setup

- Dataset 1 :
  - Small attribute cardinality

- Dataset 2 :
  - Large attribute cardinality

The data compression code used is from library zlib library

Compression Comparison

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

Performance Comparison

- cCS-indexes are most space-efficient

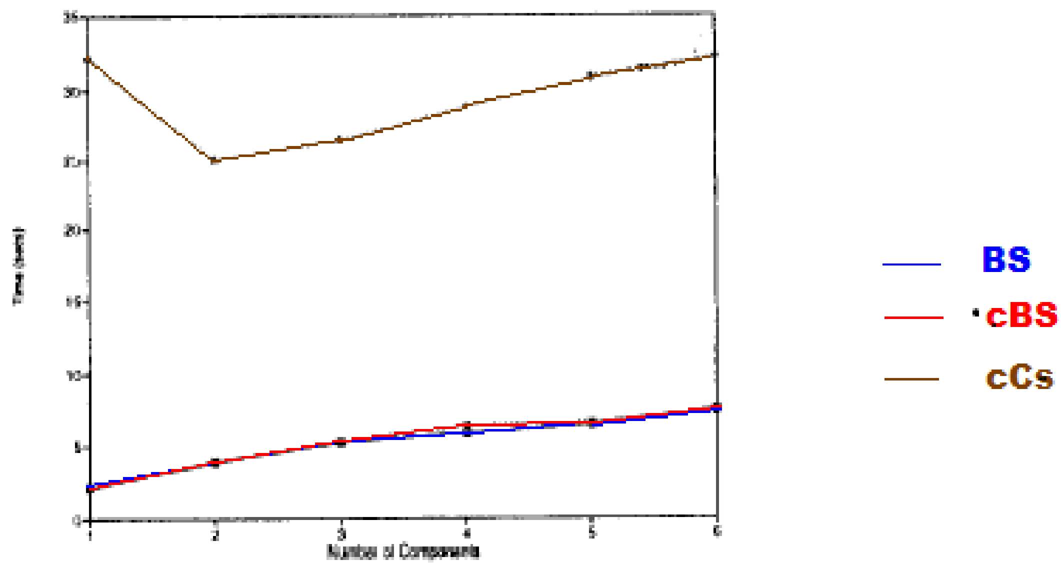
- cBS-indexes are most time-efficient since no additional baggage has to be carried/decompressed

Efficiency Metrics

- Space Metric
  - Total space of all its bitmaps (in MBytes)

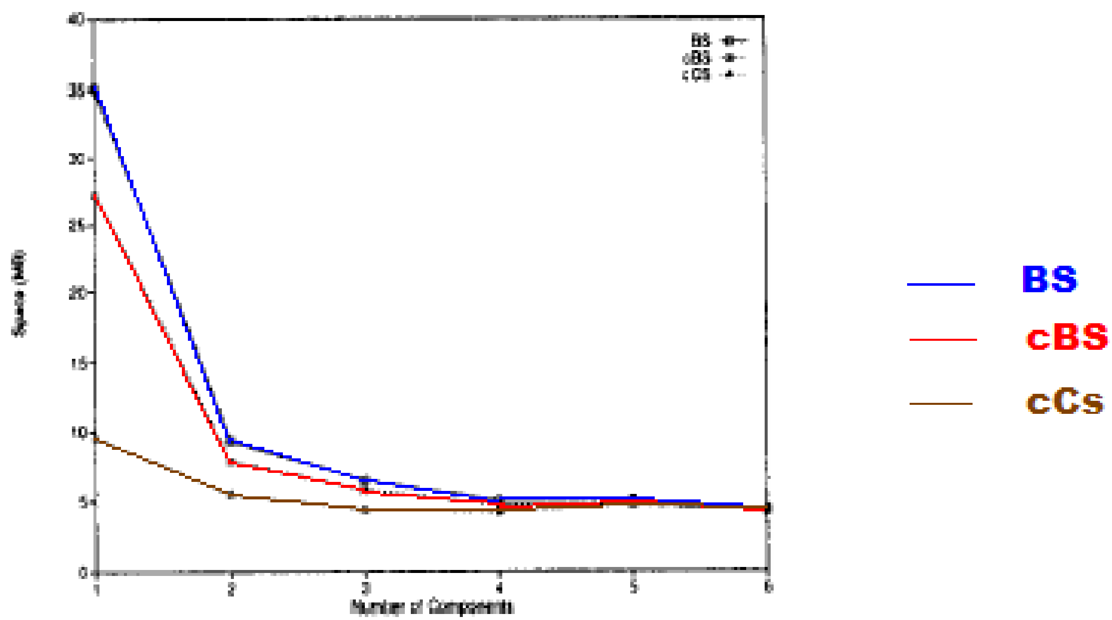
- Time Metric
  - Average predicate evaluation time (in seconds)
    - The time to read the bitmaps
    - The time for in-memory bitmap decompression (for compressed bitmap)
    - The time for bitmap operations

Experimental Results (a)



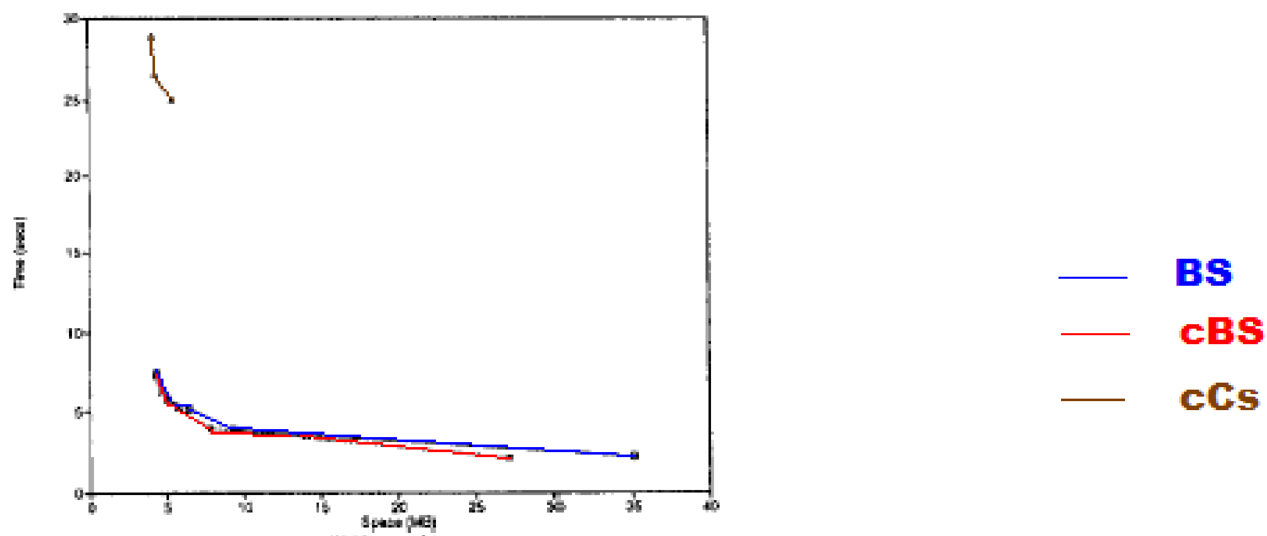
Time Vs Number of components

Experimental Results (b)



Space Vs Number of components

Experimental Results (c)



Time Vs Space

Conclusions

- This paper gives the general framework to study the design space of bitmap indexes for selection queries and examines several space-time tradeoff issues.

- Range-encoded bitmap indexes offer, in most cases, better space-time tradeoff than equality-encoded bitmap indexes.

- cBS-indexes are most time efficient and cCS-indexes are most space efficient. But BS, cBS-indexes have comparable space time tradeoff which is better than that of cCS

END BITMAP INDEXES

E0 261