一个基于位向量存储结构的统计与科学数据库管理系统原型

李建中

(黑龙江大学，哈尔滨 150080)

A PROTOTYPE SYSTEM FOR STATISTICAL/SCIENTIFIC DATA MANAGEMENT BASED ON BIT VECTOR STORAGE STRUCTURE

Li Jianzhong

(Heilongjiang University, Harbin 150080)

ABSTRACT

A prototype system for statistical/scientific data management based on bit vector storage structure, called VSSDB, is presented in this paper. The physical organization, query processing method and the time complexity model of the query processing are discussed. A new compression method for bit vectors is proposed also.

摘 要

本文介绍了一个基于位向量存储结构的统计与科学数据库管理系统原型(称为 VSSDB),提出了适于位向量文件的数据压缩方法，讨论了基于位向量文件的统计与科学数据库的物理组织、查询处理与优化和查询处理的时间复杂性模型. 我们在 VAX780 计算机上实现了这个原型系统.

§1. 引 言

统计与科学数据库 (SSDB) 具有很多独特的性质 [2,3]. 尽管目前已经存在很多先进的存取方法， SSDB 的主要存储结构仍然是顺序文件，其原因是现有的先进方法与 SSDB

1990 年 4 月 5 日收到， 1990 年 8 月 3 日定稿. 作者 **李建中**， 现任黑龙江大学教授. 目前主要从事数据库理论与技术方面的研究工作.

的特点不相容. 首先，由于 SSDB 的属性值域一般都很小，使用索引文件等方法只能把数据库划分为很少几个大的子集，效率提高很小，却要增加大量开销. 其次， SSDB 一般都是静态的，从而多数存取方法中开销很大的动态机构成为无益的浪费. 第三， SSDB 的数据重复性很大，需要数据压缩. 第四， SSDB 的统计查询操作仅涉及数据库中很少几个（常常是一个）属性的值，通常的以记录为单位的存储方法不能有效地支持这类查询操作. 这种方法把每个记录的所有属性值邻接地存储在一起，使得每个查询操作必须存取有关记录的所有属性值，降低了 SSDB 查询操作的效率. 按属性纵向存储数据的存储结构就是为了克服这种缺点而出现的 [4,5]. 这种方法按属性划分数据库，每个属性对应一个单属性文件. 每个单属性文件仅存储对应属性的值. 在处理查询操作中，这种方法可避免存取无关属性值，极大地提高了效率.

本文讨论的位向量文件是单属性文件的极端形式 [1]. 这种方法按位纵向划分数据库，每一个二进制位对应一个位向量文件，位向量文件除了具有单属性文件的优点以外，还有另外三个优点：第一，对每个属性采用合适的数据编码，查询操作只需存取相关属性的部分信息；第二，位向量文件具有更高的数据压缩率；第三，查询操作可以被转换为位向量上的布尔表达式，便于快速计算，易于硬件实现. 在 SSDB 应用中，可以用位向量文件存储描述属性 [2]，用单属性文件存储统计属性 [2]. 本文以 VSSDB 为例，讨论适于位向量文件的数据压缩方法、基于位向量文件的 SSDB 的物理组织、查询处理与优化和查询处理的时间复杂性模型. 为了减少篇幅，我们省略了全部定理证明，感兴趣的读者可查阅 [6].

§ 2. 位向量存储结构

用位向量文件存储一个给定的数据库需要三个步骤：第一，为每个属性选定一种编码方法，并对记录值进行编码；第二，按位纵向划分数据库，用一组位向量文件存储给定的数据库；第三，压缩位向量文件. 下面详细讨论这三个步骤.

1. 编码方法

编码方法是位向量存储结构的关键，直接影响查询处理的效率. 这里讨论三种编码方法 [1].

(1) 二元编码. 设 A 是一个具有 M 个可能值的属性. A 的任意值的二元编码是一个 $\log_2(M)$ 位的二进制数，而且若 A 的值 $x_1 < x_2$，则其对应的编码 $e_1 < e_2$.

(2) K-N 编码. 设 A 是一个具有 M 个可能值的属性. A 的任意值可以用一个 N 位的 0 与 1 串编码，满足： a). N 位 0 与 1 串有 K 位为 1; b). $M \le \binom{N}{K}$(N 个元素中任取 K 个元素的组和数); c). 若 A 的值 $x_1 < x_2$, x_1 与 x_2 的编码为 e_1 与 e_2，则当 e_1 与 e_2 被解释为二进制数时， $e_1 < e_2$. 例如，设属性 A 的值域由十个值组成，用 2-5 编码可编码这十个值为 00011 、 00101 、 00110 、 01001 、 01010 、 01100 、 10001 、 10010 、 10100 、 11000.

(3) 一元编码. 设 A 是一个可取 M 个不同值的属性，其值为 $x_1 < x_2 < \cdots < x_M$. 对于 $1 \le i \le M$, x_i 的一元编码是一个 M 位的 0 与 1 串，其中最右 i 位为 1. 例如， A 有十个值，则其编码按顺序为 00···001 、 00···011 、 00···111 、 ··· 、 01···111 、 11···111.

2. 数据库的按位纵向划分

每个属性的编码方法选定后，即可对记录值进行编码. 编码后的每个记录都是一个 n 位 0 与 1 串. 数据库的按位纵向划分是把编码后的数据库划分为 n 个位向量. 第 $i(1 \le i \le n)$ 个位向量包含所有记录值的第 i 位上的 0 或 1. 例如， DB 是一个具有属性 A_1, A_2, A_3 的数据库，其内容如图 (a) 所示.

A1	A2	A3
x1	y1	z1
x2	y2	z2
x3	y3	z3

图 (a)

A1	A2	A3
0001	00	0011
0111	10	0110
0011	01	0101

图 (b)

b0	b1	b2	b3	b4	b5	b6	b7	b8	b9
0	0	0	1	0	0	0	0	1	1
0	1	1	1	1	0	0	1	1	0
0	0	1	1	0	1	0	1	0	1

图 (c)

如果 A1 、 A2 、 A3 分别使用一元、二元、 2-4 编码方法，且 x1 、 x2 、 x3 编码为 0001 、 0111 、 0011, y1 、 y2 、 y3 编码为 00 、 10 、 01, z1 、 z2 、 z3 编码为 0011 、 0110 、 0101, 则编码后的 DB 如图 (b) 所示. 按位划分后，其相应的十个位向量如图 (c) 所示.

3. 位向量的数据压缩

我们针对位向量的特点设计了一种压缩方法，称为 CD 方法. 设 B 是一个位向量，其中包含 M 个全 0 或全 1 段 (简称同值段). 经由 CD 压缩后， B 成为 <Count, Data>, 其中 Count 称为计数文件， Data 称为数据文件. Data 文件由 M 位组成，每一位对应于 B 中的一个同值段. 如果 B 的第 i 同值段为全 1(或全 0), 则 Data 的第 i 位为 1(或 0). Count 文件包含 M 个数据项. 第 i 个数据项 C_i 对应于 B 的第 i 个同值段，其值是 B 中从第 1 同值段到第 i 同值段的总位数.

例如，设 B=0 0 0 0 1 1 1 1 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0. B 被压缩后成为：

Data: 0 1 0 1 0 1 0,

Count: $C_1 = 4$, $C_2 = 8$, $C_3 = 10$, $C_4 = 14$, $C_5 = 18$, $C_6 = 21$, $C_7 = 26$.

这种方法可以在对数时间内完成一次随机查找. 一般地，查找 B 的第 p 位数值的算法如下：

1. 搜索 Count 文件，找到一个 i, 使得 $C_{i-1} < p \le C_i$;

2. Data 的第 i 位值即为 B 的第 p 位值.

§ 3. 查询处理

在 VSSDB 中，查询模型是布尔表达式 $\bigvee_{i=1}^{M}\left(\bigwedge_{j=1}^{m_i} EXP_{ij}\right)$, 其中， EXP_{ij} 具有形式 $A\theta v, \theta \in \{=, \neq, <, \le, >, \ge\}$, A 是变量， v 是常量. 记录 R 满足 EXP_{ij} 是说 R 在属性 A 上的值与 v 满足关系 θ. 查询处理由四步完成： 1. 将查询中的属性值翻译成相应的编码； 2. 根据有关属性的编码方法，把译码后的查询变换为位向量的布尔表达式，使结果位向量中 "1" 的序号为欲查记录的序号； 3. 位向量布尔表达式的求值； 4. 根据第 3 步的结果产生查询结果. 第 1 、 4 步较简单，本文仅讨论第 2 、 3 步.

1. 查询变换

给定查询 $Q = \bigvee_{i=1}^{M}\left(\bigwedge_{j=1}^{m_i} EXP_{ij}\right)$, 如果每个 EXP_{ij} 都已变换为位向量布尔表达式 B_{ij},

则 Q 的变换结果为 $Q=\bigvee_{i=1}^{M}\left(\bigwedge_{j=1}^{m_i}B_{ij}\right)$. 我们只须讨论 EXP_{ij} 的变换. 以下分别对不同的编码方法给出 EXP_{ij} 的变换算法. 我们用符号 "→" 表示 " 被变换为 ", "–" 表示布尔非运算. 位向量的 ∧ 、 ∨ 、 – 运算结果仍为位向量, 其第 K 位是参加运算的位向量的第 K 位的 ∧ 、 ∨ 、 – 运算结果.

(1) 二元编码. 设属性 A 可取 a 个不同值, 则 A 有 $L=log_2(a)$ 个位向量, 设其从左到右为 $b_1,\cdots,b_L$. 令 $v=e_1,\cdots e_L$ 是属性 A 的一个值, 其中 $e_{k_1},\cdots,e_{k_m}$ 为 1 (m ≤ L), 其余位为 0, $k_1<\cdots<k_m$.

定理 1: EXP_{ij} 可按下述规则变换为位向量布尔表达式, 其结果位向量中 "1" 的序号为满足 EXP_{ij} 的记录序号: (1) A=v → $B_{=}(v)=\bigwedge_{q=1}^{L}X_q$, 其中, 如果 $e_q=1$, $X_q=b_q$, 如果 $e_q=0$, $X_q=\overline{b_q}$; (2) A > v → $B_{>}(v)=\bigvee_{p=0}^{m}\left(\left(\bigwedge_{q=1}^{p}b_{k_q}\right)\wedge\left(\bigvee_{q=k_q+1}^{k_{p+1}-1}b_q\right)\right)$, 其中, $k_0=0$, $k_{m+1}-1=L$, 而且当 x < y 时, $\bigwedge_{q=y}^{x}b_q=(1,\cdots,1)$, $\bigvee_{q=y}^{x}b_q=(0,\cdots,0)$; (3) A < v → $B_{<}(v)=\overline{B_{=}(v)}\wedge\overline{B_{>}(v)}$; (4) A ≠ v → $\overline{B_{=}(v)}$; (5) A ≤ v → $\overline{B_{>}(v)}$; (6) A ≥ v → $\overline{B_{<}(v)}$.

(2) K–N 编码. 如果使用 K–N 编码, 则属性 A 有 N 个位向量, 令其从左到右为 $b_1,\cdots,b_N$. 设 $v=e_1\cdots e_N$ 是属性 A 的值, $e_{l_1},\cdots,e_{l_k}$ 为 1, 其余位为 $0, l_1<\cdots<l_k$.

定理 2: EXP_{ij} 可按如下规则变换为位向量布尔表达式, 其结果位向量中 1 的序号为满足 EXP_{ij} 的记录序号: (1) A=v → $B'_{=}(v)=\bigwedge_{q=1}^{k}b_{l_q}$; (2) A > v → $B'_{>}$ (v) $=\bigvee_{p=0}^{k}\left(\left(\bigwedge_{q=1}^{p}b_{l_q}\right)\wedge\left(\bigvee_{q=l_p+1}^{l_{p+1}-1}b_q\right)\right)$, 其中, $l_0=0$, $l_{k+1}-1$ =N, 而且当 x < y 时, $\bigwedge_{q=y}^{x}b_q$ $=(1,\cdots,1)$, $\bigvee_{q=y}^{x}b_q=(0,\cdots,0)$; (3) A ≠ v → $\overline{B'_{=}(v)}$; (4) A < v → $B'_{<}(v)=\overline{B'_{=}(v)}\wedge\overline{B'_{>}(v)}$; (5) A ≤ v → $\overline{B'_{>}(v)}$; (6) A ≥ v → $\overline{B'_{<}(v)}$.

(3) 一元编码. 如果属性 A 使用一元编码, 且可取 a 个不同值, 则 A 有 a 个位向量, 令其从左到右为 $b_1,b_2,\cdots,b_a$. 设 $v=e_1e_2\cdots e_a$ 是 A 的值, 其中 $e_1=e_2=\cdots=e_1=0$, $e_{l+1}=e_{l+2}=\cdots=e_a=1$.

定理 3: EXP_{ij} 可按如下规则变换为位向量布尔表达式, 其结果位向量中 1 的序号为满足 EXP_{ij} 的记录的序号: (1) A=v → $\overline{b_1}\wedge b_{l+1}$; (2) A > v → b_1; (3) A < v → $\overline{b_{l+1}}$; (4) A ≠ v → $b_l\vee\overline{b_{l+1}}$; (5) A ≤ v → $\overline{b_l}$; (6) A ≥ v → b_{l+1}.

2. 位向量布尔表达式的求值

(1) 主算法. 主算法计算位向量布尔表达式 $B=\bigvee_{i=1}^{M}\left(\bigwedge_{j=1}^{m_i}B_{ij}\right)$, 其中 B_{ij} 为定理 1 、定理 2 、定理 3 的结果. B_{ij} 的算法将在下节讨论. 主算法要调用这个算法. 主算法的时间复杂性模型与计算顺序有关. 给定计算 $\bigvee_{i=1}^{M}$ 和 $\bigwedge_{j=1}^{m_i}$ 的顺序 $O_1=(i_1,i_2,\cdots,i_M)$ 和 $O_2=\{(j_{i1},j_{i2},\cdots,j_{im_i})|i=i_1,\cdots,i_M\}$, B 的求值算法如下 (命名为 MAIN):

```
1. A=(0, ···, 0), B=(1, ···, 1);
2. FOR  p=1  TO  M  DO
3.     FOR  q=1  TO  m_{i_p}  DO
4.         C=B_{i_p j_{i_p q}};
5.         B=B ∧ C;
```

6. A=A ∨ B;
7. B=(1, ···, 1);
8. END.

MAIN 的第 4 步调用 $B_{i_pj_{i_pq}}$ 的算法. 记 V [i, j] 为位向量 V 的从位置 i 到 j 的子段. 考虑到 ∨ 和 ∧ 的性质以及第 6 步的 ∨ 运算, 第 5 步仅需对满足以下条件的 B 子段 B [i, j] 和对应的 C 子段 C [i, j] 做 ∧ 运算: B [i, j] 为全 1, 而且 A [i, j] 为全 0. 这样可减少 C 的读入和 B 的写出. 同理, 第 6 步仅须对 A 的全 0 子段 A [i, j] 和对应的 B [i, j] 做 ∨ 运算. 巧妙地选择 O_1 和 O_2, 可加速 A 中 1 比率和 B 中 0 比率的上升, 减少 MAIN 的 I/O 时间复杂性. 我们将在查询优化中讨论 O_1 和 O_2 的选择问题.

(2) 时间复杂性模型. 我们仅讨论 I/O 时间复杂性, 即 MAIN 要求读写的信息位数. 从 MAIN 的定义容易看到, MAIN 的 I/O 时间复杂性由三部分组成, 即

$$COST=\sum_{p=1}^{M}\sum_{q=1}^{m_{i_p}}COST(B_{i_pj_{i_pq}},O_1,O_2,O)+\sum_{p=1}^{M}COST(C_{i_p},O_1,O_2)+COST(A,O_1),$$

其中 $COST(C_{i_p},O_1,O_2)$ 是在顺序 O_1 和 O_2 下计算 $C_{i_p}=\bigwedge_{q=1}^{m_{i_p}}B_{i_pj_{i_pq}}$ 的 I/O 时间复杂性; $COST(A,O_1)$ 是在顺序 O_1 下计算 $A=\bigvee_{p=1}^{M}C_{i_p}$ 的 I/O 时间复杂性; $COST(B_{i_pj_{i_pq}},O_1,O_2,O)$ 是在顺序 O_1 和 O_2 下, 按 O 规定的顺序计算 $B_{i_pj_{i_pq}}$ 的 I/O 时间复杂性, 下节讨论.

由位向量的定义可知, 参加运算的位向量的长度皆相等, 设其为 n. 在以下的讨论中, 令 $R_0=0$, 对于 $p\geq 1$, $R_p=\prod_{q=1}^{m_{i_p}}f_{i_pj_{i_pq}}$, 其中 f_{ij} 是 B_{ij} 中 1 的比率, 即若 B_{ij} 中有 w 位1 则 $f_{ij}=w/n$. 以下设 B_{ij} 中 1 的位置与 $B_{i'j'}$ 中1的位置是相互独立的(i≠ i' 或 j ≠ j'). 下面的结论是显然的.

结论 1: $B_{ij}\wedge B_{i'j'}$ 中 1 的比率是 $f_{ij}\times f_{i'j'}$, $\overline{B_{ij}}$ 中 1 的比率 (即 B_{ij} 中 0 的比率) 是 $1-f_{ij}$, $B_{ij}\vee B_{i'j'}$ 中 1 的比率是 $1-(1-f_{ij})\times(1-f_{I'f'})$.

定理 4:设 O_1, O_2 是为算法 MAIN 选择的运算顺序. 我们有:

(1) $COST(A,O_1)=O\Big(\sum_{p=1}^{M}n\times\prod_{l=1}^{p}(1-R_{l-1})\Big)$.

(2) $COST(C_{i_p},O_1,O_2)=O\Big(\sum_{q=1}^{m_{i_p}}n\times\Big(\prod_{l=1}^{p}(1-R_{l-1})\times\Big(\prod_{x=1}^{q}f_{i_pj_{i_px-1}}\Big)\Big)\Big)$, 其中 $f_{i_pj_{i_p0}}=1$.

f_{ij} 与编码方法有关. 下边按不同编码方法给出 f_{ij} 的算法. 以下设 s_p 为第 p 个位向量中 1 的比率.

(1) 二元编码. 若 B_{ij} 对应于 A=v, 则由定理 1 的 (1) 式和结论 1 可知, $f_{ij}=\prod_{k=1}^{L}x_k$(记作 $f_=$), 其中, x_k 是 X_k 的 1 比率, 可由 A 的位向量直接求得, X_k 和 L 同定理 2 的 (1) 式中的 X_k 和 L. 若 B_{ij} 对应于 A>v, 则由定理 1(2) 式和结论 1 可知 $f_{ij}=1-\prod_{p=0}^{m}\Big(1-\Big(\prod_{q=1}^{p}s_{k_q}\Big)\times\Big(1-\prod_{q=k_p+1}^{k_{p+1}-1}(1-s_q)\Big)\Big)$, 其中, m 和 k 同定理 1 (2) 式中的 m 和

k. 记上式为 $f_>$. 若 B_{ij} 为 A <v 、 A ≠ v 、 A ≥ v 、 A ≤ v, 则由定理 1 的 (3) 到 (6) 式和结论 1, 有 $f_{ij}=(1-f_=)\times(1-f_>)=f_<$, $f_{ij}=1-f_=$, $f_{ij}=1-f_<$, $f_{ij}=1-f_>$.

(2) K–N 编码. 类似二元编码的情形.

(3) 一元编码. 当 B_{ij} 对应于 A=v 、 A ≠ v 、 A < v 、 A ≤ v 、 A > v 、 A ≥ v 时, 由定理 3 和结论 1, 有 $f_{ij}=(1-s_l)\times s_{l+1}$, $f_{ij}=1-(1-s_l)\times s_{l+1}$, $f_{ij}=1-s_{l+1}$, $f_{ij}=1-s_l$, $f_{ij}=s_l$, $f_{ij}=s_{l+1}$.

3. $B_{i_pj_{i_pq}}$ 的计算与 COST ($B_{i_pj_{i_pq}}$, O_1, O_2, O)

本节仅给出 $B_{i_pj_{i_pq}}$ 对应于二元编码的 A > v 时的算法和 I/O 时间复杂性模型. 其他情况类似. 为了叙述方便, 设 $v=e_1\cdots e_L$, e_{k_1} 、 $\cdots$ 、 e_{k_m} 为 1, 其余为 0, 而且 $k_m<L$, $k_1>1$. 定理 1 的 (2) 式可知, 此时 $B_{i_pj_{i_pq}}=\bigvee_{x=0}^{m}\left(\left(\bigwedge_{y=1}^{x}b_{k_y}\right)\wedge\left(\bigvee_{y=k_x+1}^{k_{x+1}-1}b_y\right)\right)$, 其中, $k_0=0$, $k_{m+1}-1=L$. 给定一个计算 $B_{i_pj_{i_pq}}$ 的顺序 $O=(O'_1,O'_2,O'_3)$, 其中 $O'_1=(x_1,\cdots,x_m)$ 是计算 $\bigvee_{x=0}^{m}$ 的顺序, $x_1,\cdots,x_m$ 是 $0,\cdots,$ m 的排列; $O'_2=\{(y_1,\cdots,y_{x_l})\}$ 是在 O'_1 下计算 $\bigwedge_{y=1}^{x_l}$ 的顺序 ($0\le l\le m$), $y_1,\cdots,y_{x_l}$ 是 $1,\cdots,x_l$ 的一个排列; $O'_3=\{(y'_1,\cdots,y'_{k_{x_l+1}-k_{x_l}-1})\}$ 是在 O'_1 下计算 $\bigvee_{y=k_{x_l}+1}^{k_{x_l+1}-1}$ 的顺序, $y'_1,\cdots,y'_{k_{x_l+1}-k_{x_l}-1}$ 是 $k_{x_l}+1,\cdots,k_{x_l+1}-1$ 的一个排列.

计算 $B_{i_pj_{i_pq}}$ 的算法 SUBA 如下:

(1) Z=C=(0, ···, 0), D=(1, ···, 1);
(2) FOR I=1 TO m DO
(3) IF $x_l=0$ THEN
(4) 对于 z 循环计算 C=C ∨$b_{y'_z}$ $(1\le z\le k_{x_l+1}-k_{x_l}-1)$;
(5) ELSE
(6) 对于 z 循环计算 D=D∧ $b_{k_{y_z}}$ $(1\le z\le x_l)$;
(7) 对于 z 循环计算 C=C ∨ $b_{y'_z}$ $(1\le z\le k_{x_l+1}-k_{x_l}-1)$;
(8) Z=Z ∨ (D ∧ C);
(9) C=(0, ···, 0), D=(1, ···, 1);
(10) END.

类似于 MAIN, 为了减少 I/O 时间, SUBA 的第 (4) 、 (6) 、 (7) 、 (8) 步仅对 C 、 $b_{y'_z}$ 、 D 、 $b_{k_{y_z}}$ 、 Z 的某些子段做 ∨ 或 ∧ 运算. 由于 SUBA 是 MAIN 的子算法, 这四步的实现也要考虑 MAIN 的影响. 在第 (4) 或 (7) 步中, 参加 ∨ 运算的 C 的子段 C [i, j] 和 $b_{y'_z}$ 的子段 $b_{y'_z}$ [i, j] 的选择应使得 i 和 j 同时满足: a). C [i, j] 为全 0; b). Z [i, j] 为全 0; c). MAIN 中的 A 的对应部分 A [i, j] 为全 0; d). MAIN 中的 B 的对应部分 B [i, j] 为全 1. 第 (6) 步和第 (8) 步也做类似的考虑.

现在讨论 SUBA 的 I/O 时间复杂性. 以下令 s_p 是第 p 个位向量中 1 的比率, $s_{y_0}=0$, $s_{k_{y_0}}=1$, $f_0=1$, $f_1=\prod_{z=1}^{p}(1-R_{z-1})\prod_{x=1}^{q}f_{i_pj_{i_px-1}}$, $f_{i_pj_{i_p0}}=1$, $f_l=\prod_{u=1}^{l}(1-f'_u)$ $(2\le l\le m-1)$, 其中, $f'_l=\prod_{z=1}^{x_l}s_{k_{y_z}}\times\left(1-\prod_{z=1}^{k_{x_l+1}-k_{x_l}-1}(1-s_{y'_z})\right)$, R_l 同前.

定理 5: SUBA 在顺序 O, O_1, O_2 下的 I/O 时间复杂性为: $COST(B_{i_pj_{i_pq}}, O, O_1, O_2)$

$= O\left(\sum_{l=1}^{m}\left(\sum_{w=1}^{k_{x_l+1}-k_{x_l}-1} nf_1f_{l-1}\prod_{z=1}^{w}(1-s_{y_{x-1}})+\sum_{w=1}^{x_l} nf_1f_{l-1}\prod_{z=1}^{w}s_{k_{y_{x-1}}}+nf_1f_{l-1}\right)\right)$. 其中 $O\left(\sum_{w=1}^{k_{x_l+1}-k_{x_l}-1} nf_1f_{l-1}\prod_{z=1}^{w}(1-s_{y_{x-1}})\right)$、$O\left(\sum_{w=1}^{x_l} nf_1f_{l-1}\prod_{z=1}^{w}s_{k_{y_{x-1}}}\right)$ 和 $O(nf_1f_{l-1})$ 分别是在 SUBA 的第 l 次循环中，第 (4) 或 (7) 步、第 (6) 步和第 (8) 步的 I/O 时间复杂性模型. .

§ 4. 查询优化

查询优化的目标是适当地选择顺序 O, O_1, O_2, 使得上节讨论的 COST 最小化. 我们采用启发式算法，利用优化计算顺序的性质来选择实际计算顺序，使 COST 最小化或接近最小化.

1. 合取式与析取式的优化计算顺序. 以下设 f_i 是位向量 B_i 中 1 的比率，B_i 与 $B_j(i \neq j)$ 中 1 的位置是相互独立的，$COST_1(O)$ 是按顺序 O 计算 $B=\bigwedge_{i=1}^{m}B_i$ 的 I/O 时间复杂性，$COST_2(O)$ 是按顺序 O 计算 $B'=\bigvee_{i=1}^{m}B_i$ 的 I/O 时间复杂性. 我们使用循环的方法计算 B(或 B′), 在每次循环中仅读取与 B(或 B′) 中每个全 1(或全 0) 子段 B [k, l] (或 B′[k, l]) 相对应的 $B_i(1 \leq i \leq m)$ 中的子段 $B_i[k,l]$.

引理 1: 设 $O_1=(i_1,\cdots,i_{j-1},i_j,i_{j+1},\cdots,i_m)$, $O_2=(i_1,\cdots,i_{j-1},i_{j+1},i_j,\cdots,i_m)$. 如果 $COST_1(O_1) \neq COST_1(O_2)$, 而且 $f_{i_j} < f_{i_{j+1}}$, 则 $COST_1(O_1) < COST_1(O_2)$.

证明： 显然，$COST_1(O_1)=X+nf_{i_l}\cdots f_{i_{j-1}}f_{i_j}+Y$, $COST_1(O_2)=X+nf_{i_1}\cdots f_{i_{j-1}}f_{i_{j+1}}+Y$, 其中，$X=n+nf_{i_1}+\cdots+nf_{i_1}\cdots f_{i_{j-1}}$, $Y=nf_{i_1}\cdots f_{i_{j-1}}f_{i_j}f_{i_{j+1}}+\cdots+nf_{i_1}\cdots f_{i_m}$. $COST_1(O_1)-COST_1(O_2)=nf_{i_1}\cdots f_{i_{j-1}}(f_{i_j}-f_{i_{j+1}})$. 从而，当 $f_{i_j} < f_{i_{j+1}}$ 时，$COST_1(O_1) < COST_1(O_2)$. 证毕.

引理 2: O_1 与 O_2 同引理 1. 如果 $COST_2(O_1) \neq COST_2(O_2)$, 则当 $f_{i_j} > f_{i_{j+1}}$ 时, $COST_2(O_1) < COST_2(O_2)$.

证明： 类似于引理 1 的证明.

引理 1 说明，如果 $O=(i_1,i_2,\cdots,i_m)$ 最小化 $COST_1(O)$, 则 O 必使 $f_{i_1} < f_{i_2} < \cdots < f_{i_m}$, 既具有最小 1 比率的位向量最先参加合取运算. 引理 2 则说明，如果 O 最小化 $COST_2(O)$, 则 O 必需满足 $f_{i_1} > f_{i_2} > \cdots > f_{i_m}$, 既具有最大 1 比率的位向量最先参加析取运算.

2. 优化算法. 由引理 1 和引理 2, 我们有如下的启发式查询优化算法：

(1) 根据 2 的第 (2) 节中的算法计算每个 B_{ij} 的 1 比率 $f_{ij}(1 \leq i \leq m, 1 \leq j \leq m)$;

(2) 计算 $C_i=\bigwedge_{j=1}^{m_i}B_{ij}$ 的 1 比率 $f_i=\prod_{j=1}^{m_i}f_{ij}(1 \leq i \leq m)$.

(3) 选择 $O_1=(i_1,i_2,\cdots,i_m)$ 使得 $f_{i_1} > f_{i_2} > \cdots > f_{i_m}$.

(4) 对于每个 $i_p \in O_1$, 选择 $O_2=(k_1,k_2,\cdots,k_{m_{i_p}})$ 使得 $f_{i_pk_1} < f_{i_pk_2} < \cdots < f_{i_pk_{m_{i_p}}}$;

(5) 对于每个 $i_p \in O_1$, $q \in O_2$, 为 $B_{i_pj_{i_pq}}$ 中每组形为 $\bigvee_{t=1}^{x}B_t$(或 $\bigwedge_{t=1}^{x}B_t$) 的布尔算式选择顺序 $O_{12}=(t_1,t_2,\cdots,t_x)$ 使得 $f_{t_1} > f_{t_2} > \cdots > f_{t_x}$ (或 $f_{t_1} < f_{t_2} < \cdots < f_{t_x}$).

§ 5. 原型系统的实现与实验结果

我们在 VAX/VMS 环境下用 C 语言实现了一个基于位向量存储结构的 SSDB 管理系统原型 VSSDB. 我们用位向量文件存储描述属性，用单属性文件存储统计属性. 系统分为两级：物理级由编码器、位划分器、数据压缩程序包（压缩、反压缩、向前映象等）和位向量布尔运算操作组成；逻辑级由用户接口、译码器、查询变换器、查询优化器、布尔表达式求解器和查询结果产生器构成.

我们用一个具有 110000 个记录的癌病信息库对原型系统进行了性能实验，并与 DEC 公司的关系数据库系统 DATATRIEVE 进行了比较. 这个信息库的每个记录包括病人的性别、发病年代、年龄、癌部位、癌细胞类型等信息. 我们在 VSSDB 中装入了 110000 个记录，在 DATATRIEVE 中仅装入了 83729 个记录. 在 VSSDB 中，性别、发病年代、年龄、癌细胞类型四个属性使用位向量文件结构，其他描述属性采用了以记录为单位的顺序文件结构，统计属性使用单属性文件结构.

整个数据库在 VSSDB 中仅占 6974 页（每页 512 字节），在 DATATRIEVE 中占 8100 页. 用位向量文件存储上述四个属性，仅用 1332 页. 在 DATATRIEVE 中对这四个属性建立索引，需要 10134 页. 显然，VSSDB 在节省存储空间方面高于传统的方法.

为了比较 VSSDB 和 DATATRIEVE 的效率，我们分别用 VSSDB 和 DATATRIEVE 对十个查询进行了处理，比较了两者的查询处理时间. 图 1 给出了这十个查询在两个系统中的处理时间的比较，横坐标为查询编号，纵坐标为查询处理时间. 图 1 说明 VSSDB 的查询处理效率远高于 DATATRIEVE.

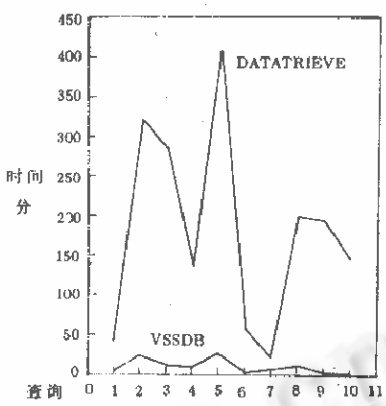

图 1 VSSDB 与 DATATRIEVE 的查询时间比较

参考文献

[1] H.K.T. Wong, J.Z. Li（李建中）, F.Olken, D.Rotem and L. Wong, "Bit Transposition for Very Large Scientific and Statistical Databases", Algorithmica, Vol. 1, No. 2, 1986, 289–309.

[2] 李建中，"统计和科学数据库系统的要求 — 与面向事务处理数据库系统的比较"，计算机科学，第 1 期，1990, 65–70.

[3] A. Shoshani and H.K.T. Wong, "Statistical and Scientific Database Issues", IEEE Transactions on Software Engineering, Vol. SE–11, No. 10, 1985.

[4] D.S.Batory, Index Encoding: a Compression Technique for Very Large Statistical Databases, Proc. of the 2nd Workshop on Statistical Database Management, 1983, 306–314.

[5] M.Turner, R. Hammond and F. Cotton, A DBMS for Large Statistical Databases, Proc. of Int. Conf. on Very Large Data Bases, 1979, 319–327.

[6] 李建中，"统计与科学数据库的位向量存储结构"，中国科学院管理、决策与信息系统开放研究实验室技术报告，YB89–0002.