Estimation of average length of search on random zero-one matrices

By A. Békéssy

The real content of this short paper is simply a theorem about zero-one matrices. In order to enlighten the background however, reference is made to a certain method of data retrieval.

Let there be given a zero-one matrix of size $m \times n$ such that all of its rows are different from each other. Let us suppose that the rows of this matrix constitute a primary key to a certain file of records stored in a computer. Therefore, the rows of the matrix will be called "names". Our problem is to find the location of any particular name (and the record associated with it) quickly, whenever wanted. The most rapid search-algorithms performing this job, e.g. "binary search" [1] are based on comparisons of the names by their magnitudes and if one complete comparison is counted one decision step then the average number of decision steps to be done for finding any name comes close to the lowest theoretically possible information limit, this latter being $\log_2 m$ if all names are looked for with equal frequencies. A complete comparison of two names, however, requires a considerable amount of time on some computers, so other procedures, though less effective in terms of decision steps, might come into consideration, too, if an elementary decision step is less time consuming.

The simplest looking search strategy would consist of decision steps to be performed column-by-column: given the name to be found the first column of the name-matrix is inspected first. If it consisted of zeros (or ones) only then we pass over to the second column immediately. If not then one decision is counted and the subset of those names is selected whose first column bit was identical to that of the name to be looked for. The second column is then inspected in the same way but restricted to the subset of names selected before, and so on, until the name is completely identified. For finding each name the steps to be made are completely determined by the structure of the name matrix and can be represented by a "search-tree" (Table 1, Fig. 1). The numbers in the nodes show the column no. of the bit the decision should be made on.

The strategy described above would not come into consideration at all should it be done in "run-time" i.e. when the names are looked for repeatedly and be found as quickly as possible. But assumed the file does not change often there might have been ample time for constructing the corresponding search-tree or, more precisely, an equivalent "search-table" [2] when the file was generated.

The search-table (Table 2) is a list of two pointers. The first column indicated shows the relative location address of that line only; it does not belong to its content. The real first column field is the serial number of the bit to look at, and according to whether it proves to be zero or one the first, resp. the second pointer should be followed by the search-algorithm working in run-time. Zero in the first column would indicate that the search has its end there and the fields belonging to this line would contain the record or a single pointer to that record, for instance.

Names	Column no. 1	2	3	4	5
1.	1	0	1	0	0
2.	1	0	0	0	1
3.	0	1	0	1	0
4.	1	0	0	1	1
5.	1	0	1	1	1
6.	1	0	0	1	0

Table 1. Name-matrix

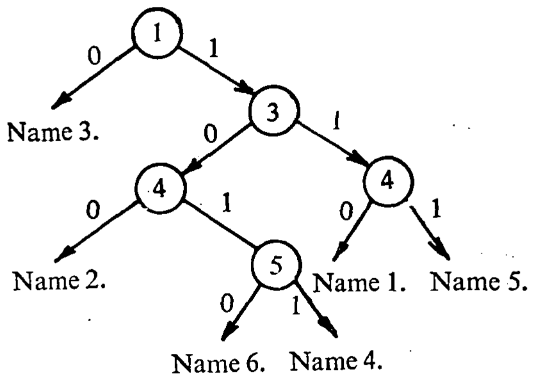

Figure 1
Search-tree to the matrix of Table 1.

A serious objection against the simple strategy described above is that it might, in some cases, result in a highly unbalanced search-tree. For the worst matrices the average number of the necessary decisions is as high as $(m+1)/2$ about. But matrices of ill behaviour, i.e. *matrices with highly unbalanced search-trees are rare.* This is the meaning of the theorem shown below.

Remark. It is possible, in practice, to make the algorithm a bit more flexible: let the decision in turn to be performed on the column in which the zeros and ones

Location	Col. no.	Pointer 1.	Pointer 2.
L+1	1	L+2	L+3
L+2	0	Pointer to record 3.	
L+3	3	L+4	L+9
L+4	4	L+5	L+6
L+5	0	Pointer to record 2.	
L+6	5	L+7	L+8
L+7	0	Pointer to record 6.	
L+8	0	Pointer to record 4.	
L+9	4	L+10	L+11
L+10	0	Pointer to record 1.	
L+11	0	Pointer to record 5.	

Table 2. Search-table to the matrix of Table 1.

Name	Number of Decisions
1.	3
2.	3
3.	1
4.	4
5.	3
6.	4
	18

Ave. number of steps: 18:6=3
Table 3.
Number of decisions to be done according to Table 2.

are distributed most evenly for that particular subset of names that was selected in the previous step. This will help in a lot of cases where the first approach would result in a highly unbalanced tree.

Now we prove the following

Theorem. Let all zero-one matrices of size $m\times n$ with all rows different be considered and supposed to be equiprobable. Let $E(M_{m,n})$ be the arithmetic mean of decisions to be made in order to find each row of matrix $M_{m,n}$ according to the simple strategy described above. Let $\mathscr{E}_{mn}$ be the expectation of the averages $E(M_{m,n})$. Then for all $m\leqq 2^{n-1}+1$

$$\mathscr{E}_{mn} < 1+\left(1+\frac{1}{2}+\ldots+\frac{1}{m-1}\right)/\ln 2 \tag{1}$$

or, because of $1+\frac{1}{2}+\ldots+\frac{1}{m-1}=\ln m+\gamma+0\left(\frac{1}{m}\right)$ (where $\gamma=0.577\ldots$ Euler's constant),

$$\mathscr{E}_{mn} < 1.833\ldots+\log_2 m+O\left(\frac{1}{m}\right), \quad (m, n\to\infty, \; m<2^{n-1}+1)^* \tag{2}$$

Proof. Let $N(m, n, d_1, d_2, \ldots, d_m)$ be the number of matrices $M_{m,n}$ such that d_i decisions have to be made for finding the i-th row ($i=1, 2, \ldots, n$). Then $E(M_{m,n})= =\sum_i d_i/m$ and

$$\mathscr{E}_{mn} = \frac{1}{N}\sum_{d_1, d_2, \ldots, d_m} E(M_{m,n})N(m, n, d_1, d_2, \ldots, d_m)$$

where $N=\binom{2^n}{m}m!$ is the number of all matrices $M_{m,n}$. The latter expression can be simplified, because of symmetry in the variables d_i, to

$$\mathscr{E}_{mn} = \frac{1}{N}\sum_d d\cdot N(m, n, d) \tag{3}$$

where $N(m, n, d)$ is the number of matrices $M_{m,n}$ such that there are d decisions needed for selecting the *first* row. For this number $N(m, n, d)$ the recursion

$$N(m, n+1, d) = 2N(m, n, d)+2\sum_{j=1}^{m-1}\binom{m-1}{j-1}N(j, n, d-1)\cdot\binom{2^n}{m-j}(m-j)! \tag{4}$$

holds. The first term gives account on the matrices the first column of which consists of zeros (or ones) only. The j-th term under the summation is the number of matrices

* It is thought that the condition $m\leqq 2^{n-1}+1$ is, in fact, not necessary. Also the constant might perhaps be improved to $-0.5+\gamma/\ln 2=0.33\ldots$.

with j zeros in the first column while the first-row-first-column bit is zero, as well. Matrices of j ones in the first column when the first-row-first-column bit is one, are of the same number; therefore the sum should be multiplied by two. The boundary conditions

$$N(m,1,d)=\begin{cases}2 & \text{if } m=1,\ d=0,\\ 2 & \text{if } m=2,\ d=1,\\ 0 & \text{otherwise};\end{cases}$$
$$N(m,n,0)=\begin{cases}2^n & \text{if } m=1,\\ 0 & \text{otherwise}\end{cases} \tag{5}$$

complete the recursion (4).

Introducing the function

$$H(x,n,y)=\sum_{d=0}^{\infty}\sum_{m=1}^{\infty} y^d \frac{x^{m-1}}{(m-1)!} N(m,n,d) \tag{6}$$

we obtain

$$H(x,1,y)=1+xy,$$
$$H(x,n+1,y)=2\cdot\{1+y[(1+x)^{2^n}-1]\}\cdot H(x,n,y)$$

from (4) and (5), with solution

$$H(x,n,y)\equiv 2^n\cdot\prod_{p=0}^{n-1}\{1+y[(1+x)^{2^p}-1]\}. \tag{7}$$

Since by (3) and the definition (6) of H

$$\left.\frac{\partial H}{\partial y}\right|_{y=1}=\sum_m \binom{2^n}{m} m x^{m-1}\mathscr{E}_{mn}$$

it follows easily

$$\mathscr{E}_{mn}=n-\sum_{p=0}^{n-1}\binom{2^n-2^p}{m-1}\Big/\binom{2^n-1}{m-1} \tag{8}$$

or, under the restriction $m\leqq 2^{n-1}+1$

$$\mathscr{E}_{mn}=\sum_{p=0}^{n}\left[1-\prod_{j=1}^{m-1}\left(1-\frac{2^{-p}}{1-j2^{-n}}\right)\right]. \tag{9}$$

Again, for $m\leqq 2^{n-1}+1$

$$\mathscr{E}_{mn}<1+\sum_{p=1}^{n}\left[1-\prod_{j=1}^{m-1}\left(1-\frac{2^{-p}}{1-j2^{-n}}\right)\right]<$$
$$<1+\int_1^{\infty}\left[1-\left(1-\frac{2^{-x}}{1-(m-1)2^{-n}}\right)^{m-1}\right]dx \tag{10}$$

giving the end-result.

Remark 1. For $n \to \infty$, $m = \text{const}$.

$$\mathscr{E}_{mn} \to \mathscr{E}_{m\infty} = \sum_{p=1}^{\infty} \left(1-(1-2^{-p})^{m-1}\right).$$

Could it be proved $\mathscr{E}_{mn} \leqq \mathscr{E}_{m\infty}$ for all n, a better constant would be achieved in the inequality (2).

Remark 2. The problem dealt with here resembles strongly that of calculating the average hight of random trees [3]. Instead of looking, however, for an appropriate link between the two problems the straightforward method presented here seemed to be simpler.

Abstract

The average efficiency of a simple search algorithm defined on random zero-one matrices is estimated.

COMPUTER AND AUTOMATION INSTITUTE
HUNGARIAN ACADEMY OF SCIENCES
P. O. BOX 63
BUDAPEST, HUNGARY
H—1502

References

[1] MARTIN, J., *Data-base organization,* Prentice-Hall, 1975, pp. 254—406.
[2] KNUTH, D. E., *The art of computer programming,* Vol. I, Addison-Wesley, 1968, pp. 315— 16.
[3] RÉNYI, A., G. SZEKERES, On the height of trees, *J. Austral Math. Soc.,* v. 7, 1969, pp. 497—507.

(Received Jan. 24, 1979)