On B-tree Indices for Skewed Distributions

Christos Faloutsos†

University of Maryland
College Park, MD

H. V. Jagadish

AT&T Bell Laboratories
Murray Hill, NJ

ABSTRACT

It is often the case that the set of values over which a B-Tree is constructed has a skewed distribution. We present a geometric growth technique to manage postings records in such cases, and show that the performance of such a technique is better than that of a straightforward fixed length postings list: It guarantees 1 disk access on searching, and it takes a fraction of the space that its competitor requires (55% to 66%, in our experiments).

1. INTRODUCTION

Often, an index is constructed over an attribute that takes on discrete values and has the same value in multiple records. For instance, in a typical employee database, attributes of this sort include Job Title, Age, Work Location, and perhaps even Salary. A straightforward way to construct such an index is to build a B-Tree on the *distinct* attribute values, and then to have a pointer to a *postings list* at the leaf of the tree. Each postings list is a set of *postings pointers* (which could be record locators or object identifiers or tuple IDs), one for each record that has the specified value for that attribute.

In this paper we study how to organize such postings lists. We show that how these lists are organized can significantly affect performance, in terms of times for look-up and for insertion, and storage cost.

These costs are a function of the distribution of attribute values. Our thesis is that traditional approaches are reasonable when this distribution is uniform, or near uniform. However, there are many situations where this distribution is skewed [Christodoulakis84a]. For instance, each of the attributes mentioned in the first paragraph above is likely to be distributed in a highly non-uniform fashion. Most companies would have many people with a few common titles ("Member of Technical Staff", "Sales Associate", "Secretary", etc.) but only one person with a title of "President". Similarly with the other attributes listed above.

The straightforward way of managing the postings lists is to use fixed size *postings records*, chained together for long lists. We present a "doubling" scheme that has significantly better performance than fixed-size postings records when the distribution of attribute values is skewed.

In Section 2, we present experimental evidence to show that significantly skewed distributions do arise in many different situations. In Section 3, we describe the different techniques to organize a postings list. In Section 4, we present some analytical results comparing the different approaches, and in Section 5 we provide simplified formulas for Zipf distributions. In Section 6 we present some experimental evidence to back up the analysis. We make some concluding remarks in Section 7.

† Also member of the U.M.I.A.C.S. (University of Maryland Institute for Advanced Computer Studies). This research was sponsored partially by the National Science Foundation under the grants IRI-8719458 and IRI-8958546, by a Department of Commerce Joint Statistical Agreement JSA-91-9, by a donation by EMPRESS Software Inc. and by a donation by Thinking Machines Inc..

Proceedings of the 18th VLDB Conference Vancouver, British Columbia, Canada 1992.

2. MOTIVATION

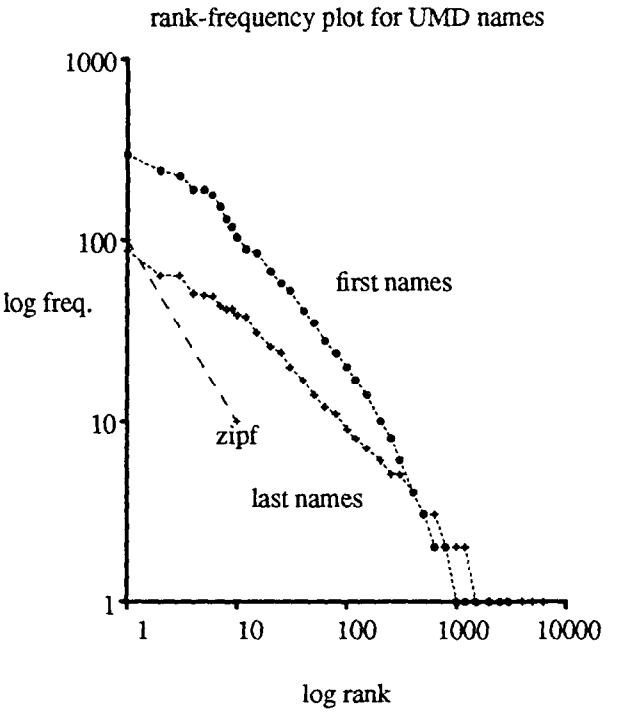
Skewed distributions of attribute values appear frequently in database systems [Christodoulakis84a], and they are the norm in text retrieval systems [Zipf49a]. However, due to the simplicity that the uniformity assumption results in, few analyses have considered skewed distributions. Non-uniformities have been considered in estimating the intermediate results of joins [Ioannidis91a], as well as in a parallel hash-join algorithm proposed in [Wolf91a]. In this paper we examine the effects of non-uniformity on a secondary index, organized as a B-tree. With minor or no changes, the proposed methods and analysis can be used for a hashed secondary index.

We conducted measurements on several different databases to determine whether there indeed were attributes that occurred with skewed distributions, as we would expect intuitively. Some results of interest are presented below.

The first database contained N=11,657 records from the on-line telephone directory at University of Maryland. Each record contained the last and first name of an employee, the telephone number, the address etc. The frequency-rank distributions of the names are skewed, as shown in Graph 2.1. Notice that they are close to straight lines, with slopes -0.65 and ≈ -0.8 for the last and first names, respectively. As intuitively expected, the distribution of first names is more skewed than the distribution of last names. In Section 5 we shall see that these are cases of *generalized Zipf* distributions.

Similar trends were observed on the names from the telephone directory at Bell Laboratories. The database had ≈37,000 records. The slope for the last names was -0.63 (very close to the one of the previous graph!), while the slope for the first names was ≈-0.9. The results are shown in Graph 2.2.

Finally, Graph 2.3 gives the log-log rank-frequency plot for words in two collections of the Associated Press News-wire articles. This collection was used in [Faloutsos92a]. The large collection contained 10,075 documents, from 40 random days in 1989. Each article was 300 to 600 words long; the total size of the database was a little over 30Mb. The small collection was a sample of 10% of the articles in the large collection. The straight line corresponds to an ideal Zipf distribution (see Section 5). The plot illustrates that, for large text databases, the distribution of words in natural language can be closely approximated with a Zipf distribution, regardless of the size of the database.



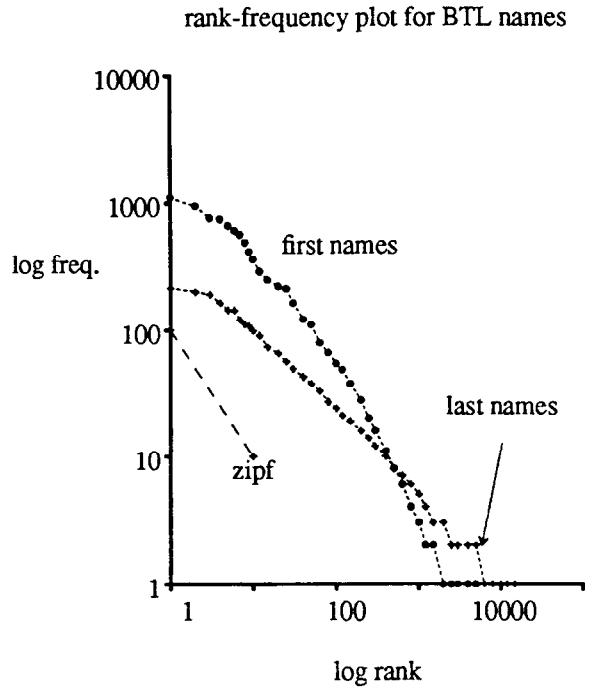
Graph 2.1: Rank-Frequency plot of the names in the UMD telephone directory. Last names are marked with "+", first names with circles.

3. ORGANIZING POSTINGS RECORDS

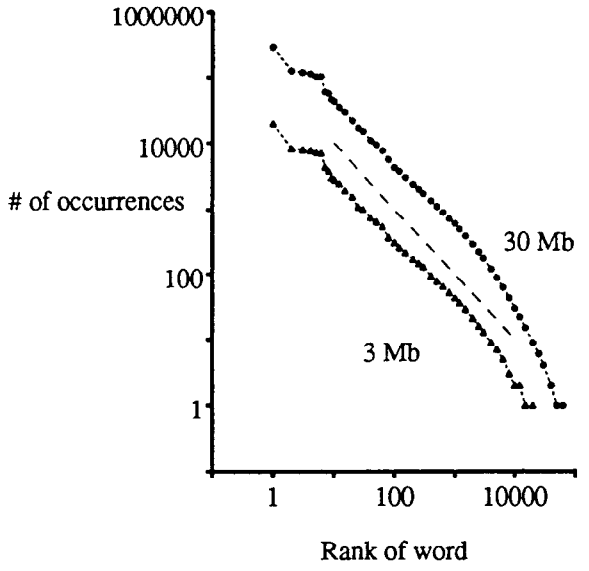
Several different schemes are introduced below. We measure the space overhead using the postings pointer as the unit (4 bytes, in our experiments). We also assume that we can force two pages to be consecutive on the disk, for example, by writing on the raw device, under UNIX™.

We are concerned with three measures of cost:

i) Storage Overhead. In addition to the space occupied by the postings pointers, there are three more reasons for wasting space. One is a constant charge per postings record used, in terms of a pointer to the next postings record, and possibly some additional identifying information. The second reason is internal fragmentation, that is, the empty slots left when the number of postings does not completely fill a postings record. The third reason is external fragmentation, when a postings record is returned to the free store. For the rest of the paper, the term "fragmentation" will denote "external fragmentation", unless explicitly mentioned otherwise.



Graph 2.2: Rank-frequency plot of the names in the Bell Labs directory. Last names are marked with "+", first names with circles.



Graph 2.3: Rank-frequency plot of the words in Associated Press Newswire articles - 30Mb and 3Mb databases, marked with circles and triangles, respectively. The dashed line corresponds to an ideal Zipf distribution.

ii) Time to Access. Access is always to the entire postings list. For a large enough database it is unlikely that successive postings records will be in the same disk block. In fact it is unlikely even that they will be in successive disk blocks. Therefore the access time is roughly the cost of reading as many disk blocks at random as there are postings records.

iii) Time to Update. As before, this cost is measured by the number of disk accesses, either reads or writes. On insertions, we have to traverse the chain of postings records, to locate the correct one, bring it in, and write it back; if an overflow occurs, we have to do some additional disk accesses, depending on the method. On deletion, we have to do the reverse, possibly facing an underflow. Deletions pose the extra problem of "holes" - we propose to fill in the holes, either by contracting the whole postings list, or by copying the last entry of the list into the hole.

Next, we present some alternative designs for the organization of the postings lists.

3.1. Fixed Size Posting Records ("FCHAIN")

The standard design is to keep fixed size postings records, chained in a linked list on overflows. This method will be referred to as "FCHAIN", for fixed-size postings records, chained together. Figure 3.1 illustrates the approach, assuming that each postings record can hold $b=2$ record identifiers and 1 pointer for chaining.

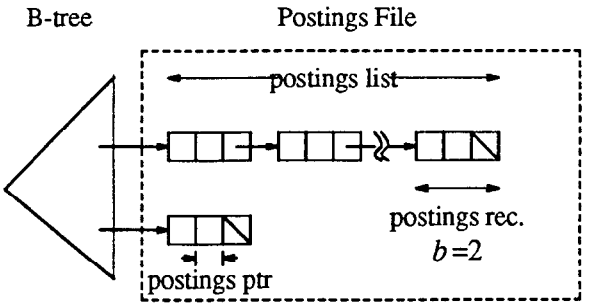


Figure 3.1: Fixed Size Postings Records ("FCHAIN"), with b=2 postings per record (plus the overflow pointer.

The smaller the size of a postings record, the greater the chance of an overflow, and the greater the number of postings records required for any given attribute value, resulting in a greater time to

access. On the other hand, the larger the size of a postings record, the more the space wasted, as a result of having empty postings pointer slots in the last postings record. Of course, there is a fixed storage overhead per postings record, in terms of the pointer to the next record in the postings list, so too small a size for the postings record also wastes space.

3.2. Adaptive Size Postings Records ("CONTIGUOUS")

The idea is to have larger postings records for attribute values that occur frequently, and smaller ones for the others. If this information were known a priori, it could be used to assign a postings record size for each value, statically, based upon the expected frequency of occurrence. However, a more typical situation is one in which we do not know how any particular postings list will grow. Therefore, the size has to be made to adapt dynamically. We propose that, on overflow, a new, larger postings record is allocated, and that the contents of the old postings record are copied consecutively in the new one. For this reason, we call this method "CONTIGUOUS". Figure 3.2 illustrates the method.

We propose that the size of a postings record be multiplied by a "growth factor" g, every time that there is an overflow. Thanks to the contiguity, the access cost is guaranteed to be a single look-up. Insertion cost is generally low, unless a copy has to be done The smaller g is, the lower the storage overhead, but the higher the expected insertion cost.

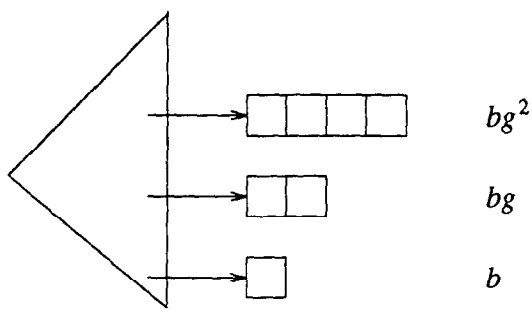


Figure 3.2: Adaptive Size Postings Records ("CONTIGUOUS"). Initial size $b=1$, growth factor $g=2$.

3.3. Adaptive Size with No Copying ("ACHAIN")

The major drawback of the adaptive scheme proposed above is the potentially large amount of copying that may have to be done upon an insertion. This work can be avoided if we adaptively grow the size of postings records as before, but do not have any copying allowed. That is to say, the first postings record is of a small size b, the next record, when needed, is h times bigger than first one, the next is h times bigger again, and so on. In such a case, the insertion cost is back down to a low fixed cost as for fixed size records. Each access requires reading more than one record. The choice of h value is a tradeoff between number of records read at access time, and storage space wasted due to empty slots in the last postings record. We call this method "ACHAIN" since it consists of chains of postings records, whose sizes grow adaptively. Figure 3.3 illustrates the method.

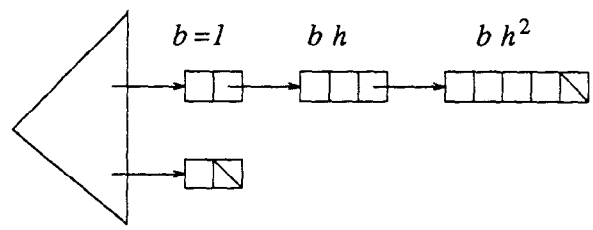


Figure 3.3: Adaptive size without copying ("ACHAIN"). Initial size $b=1$, growth factor $h=2$.

3.4. Hybrid Adaptive, with Copying ("HYBRID")

A compromise between ACHAIN and CONTIGUOUS is a method that copies only after K records have overflowed. That is, the maximum length of a chain of postings records is K. Overflows that result in chains shorter than K are handled according to ACHAIN; overflows that would result in a chain longer than K are handled by copying all the postings pointer in a large, contiguous place. This postings record will be the first of a chain that will grow according to the ACHAIN method, with growth factor h, until the new chain contains K postings records. In that case, all the postings pointers will be copied on a contiguous place again, and so on. Figure 3.4 illustrates the method, with initial size $b=1$, growth factors $h=g=2$ and $K=3$ records in the longest chain.

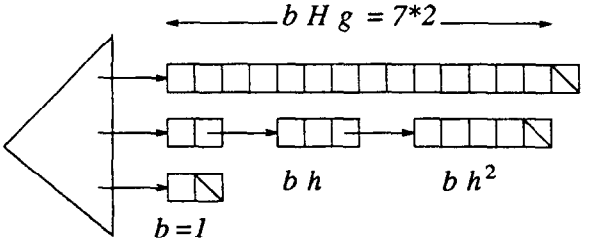


Figure 3.4: HYBRID method, for b=1, h=g=2, K=3. After the third record overflows (middle list) everything is copied in a record g times larger (top list).

Thus the cost of access is limited to a fixed constant K, and the cost of copying during insertion is paid only infrequently. There are two parameters now, in addition to K: a parameter h determines the size of each new postings record within a set of K, and a parameter g determines the size of a new postings record into which the postings pointers in each K records are to be copied.

Let there be b postings pointers in the very first postings record. Then there are bh postings in the second record, bh^2 in the third record, and so on, for the first K records, which in total store $b*(h^K-1)/(h-1)$ pointers ($h>1$). Let us call this $b*H$, where

$$H \equiv \sum_{i=0}^{K-1} h^i = \frac{h^K-1}{h-1} \quad \text{if } h>1 \qquad (3.1)$$

or

$$H \equiv K \quad \text{if } h=1$$

Then the $K+1^{th}$ postings record can have $b\,H\,g$ postings, enough to hold the $b\,H$ entries of the first K records, plus some room for growth (this is why we multiply by g). Of course, the first K records will be returned to the free store for reuse. The $K+2^{th}$ record can have $bHgh$, the $K+3^{th}$ can have $bHgh^2$ and so on so that the second set of K records can have a total of bH^2g postings. The $2K+1^{th}$ postings record then has room for bH^2g^2 postings, and the process carries on.

The HYBRID scheme offers the greatest flexibility, and subsumes each of the schemes presented above:

- For $K \to \infty$ and h=1, it gives the FCHAIN scheme with g immaterial.
- For $K \to \infty$ and some $h>1$, it gives the ACHAIN scheme, with g once again immaterial.
- For K=1 with some $g>1$, it gives the CONTIGUOUS scheme, with h immaterial.

4. ANALYSIS

Since the HYBRID scheme subsumes all the other schemes, it is the only one we need to analyze in terms of the different measures of cost listed in Section 3. In the next Section we study the choice of parameter values that will minimize these cost measures. In our analysis we assume only insertions (archival environment). Including deletions in the analysis is a topic for future research. The FCHAIN and CONTIGUOUS methods will be examined in more detail, because the former is the simplest method, while the latter is the fastest on search. Table 4.1 gives a list of symbols and their definitions.

Symbol	Definition
$p(x)$	Probability a postings list has length x
$P(x)$	prob. a list has length $\leq x$
M	size of longest postings list (in postings ptrs)
V	vocabulary = cardinality = number of distinct attribute values
N	number of records indexed = number of non-distinct attribute values
b	size of initial block (in postings ptrs)
g	growth factor of postings records on copy
h	growth factor of postings records when no copy
K	maximum number of postings records in a chain
v	overhead per post. record (in postings ptrs - typically: 1)
$R(b,g,h,K)$	avg. response time (in disk accesses)
$I(b,g,h,K)$	avg. insertion time (in disk accesses)
$S(b,g,h,K)$	exp. space required (in postings ptrs)
$Frag(b,g,h,K)$	space wasted due to External fragmentation
p	exponent in the generalized Zipf distribution (Section 5)

Table 4.1: Symbols and definitions.

Before we proceed with the analysis, we define two fundamental probability distributions and prove a lemma.

Definition 1. For a given state of the relation, we define as $p(x)$ the probability that a randomly selected value (among the V distinct values of the indexed attribute) occurs x times in the relation.

$p(x) \equiv$ Prob { a value appears x times }

Definition 2. Let $P(x)$ be the cumulative probability, that is

$$P(x) \equiv \sum_{i=0}^{x} p(i)$$

This is the probability that a randomly chosen attribute value occurs x or fewer times. Clearly

$$P(0)=0, \quad P(M)=1$$

where M is the length of the longest postings list.

Lemma 1 The total storage occupied by a postings list after i overflows is

$$b(Hg)^{i \text{ div } K} \frac{h^{(i+1) \bmod K} - 1}{h-1}$$

Proof: Since we have i overflows, we have gone through copying $i \text{ div } K$ times, and we currently have $(i+1) \bmod K$ postings records (i *div* k is the quotient when i is divided by k, and i *mod* k is the remainder, e.g., 7 div 3 = 2 and 7 mod 3 = 1). From eq. (3.1) we have that the last record has $b(Hg)^{i \text{ div } K} h^{i \bmod K}$ postings for all integer $i \geq 0$. Adding the lengths of the previous records proves the lemma.

Here we give the final cost formulas for the HYBRID method in general. The proofs are given in a technical report [Faloutsos92b].

The total space (measured in postings pointers) is given by:

$$\begin{aligned} S(b,g,h,K) = V\Big[& vR(b,g,h,K) \\ & + bP(b) + \sum_{i=1}^{\infty}(bH^i g^i (P(bH^i g^i) - P(bH^i g^{i-1}))) \\ & + \sum_{j=2}^{K}\Big[bH^i g^i (\sum_{l=0}^{j-1} h^l) \\ & (\sum_{i=0}^{\infty}(P(bH^i g^i (\sum_{l=0}^{j-1} h^l)) - P(bH^i g^i (\sum_{l=0}^{j-2} h^l))))\Big]\Big] \\ & + Frag(b,g,h,K) \end{aligned} \quad (4.1)$$

where $Frag()$ measures the external fragmentation:

$$\begin{aligned} & Frag(b,g,h,K) = \\ & \sum_{i=0}^{\infty}\Big(\frac{bH^{i+1}g^{i+1}p(bH^{i+1}g^i)}{p(bH^{i+1}g^{i-1}) - Hgp(bH^{i+1}g^i)} + \\ & \sum_{j=1}^{K-1} \frac{bH^{i+1}g^i(\sum_{l=0}^{j} h^l)p(bH^{i+1}g^i)}{(\sum_{l=0}^{j-1} h^l)p(bH^i g^i \sum_{l=0}^{j-1} h^l) - Hp(bH^{i+1}g^i)}\Big) \end{aligned} \quad (4.2)$$

Insertion cost, in disk accesses:

$$I(b,g,h,K) = 2 + \sum_{i=1}^{\infty} KbH^i g^{i-1} p(bH^i g^{i-1}) \quad (4.3)$$

Retrieval cost, in disk accesses:

$$\begin{aligned} R(b,g,h,K) = & P(b) \\ & + \sum_{i=1}^{\infty}(P(bH^i g^i) - P(bH^i g^{i-1})) \\ & + \sum_{j=2}^{K} j \sum_{i=0}^{\infty}\Big[P(bH^i g^i (\sum_{l=0}^{j-1} h^l)) - \\ & P(bH^i g^i (\sum_{l=0}^{j-2} h^l))\Big] \end{aligned} \quad (4.4)$$

For the FCHAIN and CONTIGUOUS methods, the above formulae are simplified as follows:

FCHAIN ($K=\infty$, $h=1$):

$$S_{FCHAIN} = V(b+v)R_{FCHAIN} \quad (4.5)$$

$$I_{FCHAIN} = 2 \quad (4.6)$$

$$R_{FCHAIN} = P(b) + \sum_{j=2}^{\infty} j[P(bj) - P(b(j-1))] \quad (4.7)$$

CONTIGUOUS ($K=1$):

$$\begin{aligned} S_{CONTIGUOUS} = & V\Big[v + bP(b) + \\ & \sum_{i=1}^{\infty}(bg^i(P(bg^i) - P(bg^{i-1})))\Big] + \\ & \sum_{i=0}^{\infty} \frac{bg^{i+1}p(bg^{i+1})}{p(bg^{i-1}) - gp(bg^i)} \end{aligned} \quad (4.8)$$

$$I_{CONTIGUOUS} = 2 + \sum_{i=1}^{\infty} bg^{i-1}p(bg^{i-1}) \quad (4.9)$$

$$R_{CONTIGUOUS} = 1 \quad (4.10)$$

5. SPECIAL CASE: ZIPF DISTRIBUTION

The formulas derived up to now hold for *any* frequency distribution $p()$. Thus, these formulas could be used to predict the performance of the

corresponding method, when the distribution $p()$ is known or when it can be estimated, for example, through sampling. In order to get a better intuition about the behavior of the methods as their parameters vary, in this Section we simplify the derived formulas even more, assuming a Zipfian distribution. The main goal is to arrive at arithmetic examples that will highlight the weak and strong points of each method. Moreover, the simpler formulas will allow optimal choice of the design parameters.

For relations, we use the terms "attribute value", "cardinality of the attribute"; for documents, the equivalent terms are "word" and "vocabulary". We use the latter set of terms, for brevity. Under a Zipfian distribution we have

$$\text{frequency}(r)=C/r \tag{5.1}$$

where r is the rank of the word, sorted in decreasing frequency order, and frequency(r) is the occurrence frequency of the r-th most frequent word. C is a normalizing constant ($C=V=$ vocabulary size). Clearly, the rank-frequency plot of a Zipfian distribution in a log-log graph is a straight line with slope -1. The *generalized* Zipf distribution obeys the equation

$$\text{frequency}(r)=C/r^p \tag{5.2}$$

In this case, the log-log rank-frequency plot has slope $-p$. As mentioned in Section 2, the distributions of the first names and the last names in our experiments can be approximated by generalized Zipfian distributions with p=0.63-0.65 for the last names and p=0.8-0.9 for the first names. For Zipfian distributions, notice that

$$V \approx M$$

and the total number of records is

$$N \approx V \ln V$$

In a Zipfian distribution, the probability of a vocabulary word occurring x times or more is $1/x$. The probability of a vocabulary word occurring $x-1$ times or less is then

$$1-\frac{1}{x}=\frac{x-1}{x}$$

Therefore,

$$P(x)=\frac{x}{x+1}$$

and

$$p(x)=P(x)-P(x-1)=\frac{1}{x(x+1)}$$

For x 1, we shall approximate this with

$$p(x)=1/x^2 \tag{5.3}$$

We also approximate $P(x)$ with

$$P(x)=1-1/x \tag{5.4}$$

We use eqs (5.3) and (5.4) to simplify the cost formulas for the two methods of interest, FCHAIN and CONTIGUOUS. Eqs. (4.5) to (4.10) become as follows:

$$R_{FCHAIN}=(1-\frac{1}{b}+\sum_{j=2}^{J}\frac{1}{b(j-1)}) \approx 1-\frac{1}{b}+\frac{\ln J}{b}$$

and, since $J=\lceil M/b \rceil$ we finally obtain eq. (5.5).

$$R_{FCHAIN}=\approx 1-\frac{1}{b}+\frac{\ln(M/b)}{b} \tag{5.5}$$

$$S_{FCHAIN}=V(b+v)R_{FCHAIN} \tag{5.6}$$

$$I_{FCHAIN}=2 \tag{5.7}$$

The respective formulae for the CONTIGUOUS method are eqs. (5.8-9). The approximation in eq. (5.9) holds since $I \equiv \left\lceil \ln_g(M/b) \right\rceil$ and $M \, b$.

$$S_{CONTIGUOUS}=V(v+b-1+I(g-1))+\frac{gM-b}{(g-1)^2} \tag{5.8}$$

$$I_{CONTIGUOUS}=2+\sum_{i=1}^{I}\frac{1}{bg^{i-1}} \approx 2+\frac{g}{b(g-1)} \tag{5.9}$$

Within the family of schemes that we have introduced, it appears that copying every time is indeed the best, since the additional cost at insertion time is not too high, and the savings at look-up time can be considerably more than that. We compare the scheme CONTIGUOUS to the traditional FCHAIN scheme in the following, using the Zipf distribution. CONTIGUOUS requires one disk access on search; the cost of insertion decreases monotonically with g. Similarly, FCHAIN has a constant cost for insertion, and the search time decreases monotonically with b. Thus, the only non-trivial cost to optimize is the space overhead. The details follow:

Optimizing the space for FCHAIN: From eqs. (5.6) and (5.5) we have

$$S_{FCHAIN}=V(b+v)(1-\frac{1}{b}+\sum_{j=2}^{J}\frac{1}{b(j-1)})$$

with $J=M/b$. To optimize it, we have to solve

$$\frac{\partial S_{FCHAIN}}{\partial b}=0$$

Numerical techniques may be required. Arithmetic examples, with $v=1$, show that

$b=2$ is optimal for a vocabulary $V=10$;

$b=3$ is optimal for V in the range from 100 to 10^3;

$b=4$ is optimal for V in the range from 10^4 to 10^7.

Moreover, the function is rather flat at the minimal point. Thus, if

$$b=3 \quad \text{or} \quad b=4$$

the space of FCHAIN will be optimal or close to optimal for any practical application.

Optimizing the space for CONTIGUOUS: From eq. (5.8) we have

$$S_{CONTIGUOUS} = V(v+b-1+I(g-1)) + \frac{gM-b}{(g-1)^2}$$

and, after substituting $I=\ln_g(M/b)$ we have

$$S_{CONTIGUOUS} = V\left[v+b-1+(g-1)\ln(M/b)/\ln(g)\right] + \frac{gM-b}{(g-1)^2} \qquad (5.10)$$

The optimal values should fulfill the set of equations

$$\frac{\partial}{\partial g}S_{CONTIGUOUS}=0 \qquad (5.11)$$

$$\frac{\partial}{\partial b}S_{CONTIGUOUS}=0 \qquad (5.12)$$

Eq. (5.11) gives

$$V\left[\frac{\ln(M/b)}{\ln g}-\frac{(g-1)\ln(M/b)}{g(\ln g)^2}\right]-\frac{M(g+1)-2b}{(g-1)^3}=0$$

The second term is the derivative of the external fragmentation, and is always negative for $g>1$ and $b<M/2$. This means that the external fragmentation decreases with g, which agrees with our intuition. The first term is the derivative of the internal fragmentation. Algebraic manipulation shows that it has the same sign as

$$1-\frac{(g-1)}{g\ln g}$$

which is positive for $g>1$. As such, to reduce the internal fragmentation we should choose g to be as small as possible. Since the external fragmentation is small compared to the external one (see, e.g., Graph 6.2), the optimal value of g should be small. Since $g \geq 1.5$ (otherwise, there is no guarantee that the new record will be larger than the old record, on overflow), g should be close to 1.5.

From eq. (5.12) we have:

$$\frac{\partial S_{CONTIGUOUS}}{\partial b}=V\left[1-\frac{(g-1)}{b\ln(g)}\right]-\frac{1}{(g-1)^2}=0$$

or, after solving for b:

$$b=\frac{g-1}{(1-\frac{1}{V(g-1)})\ln g}$$

If $V(g-1)1$, then we have

$$b \approx \frac{g-1}{\ln g}$$

For g in the range (1.5, 2), b should be close to 1. This is intuitively expected: Since several values appear once only, it is wasteful to have a large initial size b for postings records.

Next, we present the results of arithmetic examples, indicating the optimal values of b and g for several values of V (recall that $M=V$, for Zipfian distributions).

V	opt. b	opt. g
10	2	2.0
100	1	1.7
10^3	1	1.6
10^4	1	1.54
10^5	1	1.5

Thus, the conclusion is that good choices for the parameters are:

$$b=1 \quad \text{and} \quad g \approx 1.5\text{-}2$$

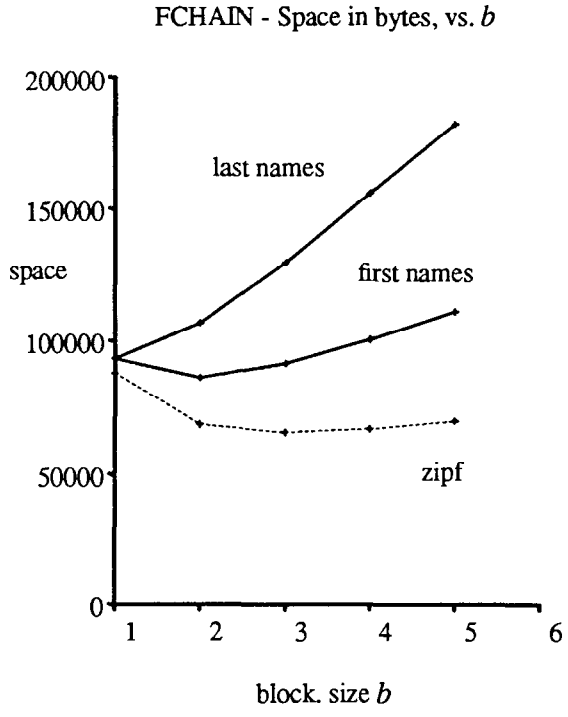
6. EXPERIMENTS

We implemented the FCHAIN and the CONTIGUOUS method under a B-tree, in "C" and UNIX. The size of each pointer was 4 bytes. We performed experiments with the FCHAIN method and several values of b, and with the CONTIGUOUS method with several values of the growth factor g including $g=1.618$ (the "golden ratio").

We used the telephone catalogue at the University of Maryland which is available on-line. It contains 11,657 entries, with 86Kb of last names and 80Kb of first names (middle names, "Jr." etc were ignored). The distribution of names was plotted in Graph 2.1. There were $V_{last}=7,148$ distinct last names, with $M_{last}=88$ appearances for the most

common one. Similarly, there were V_{first}=3,269 distinct first names, with M_{first}=288 appearances for the most common one.

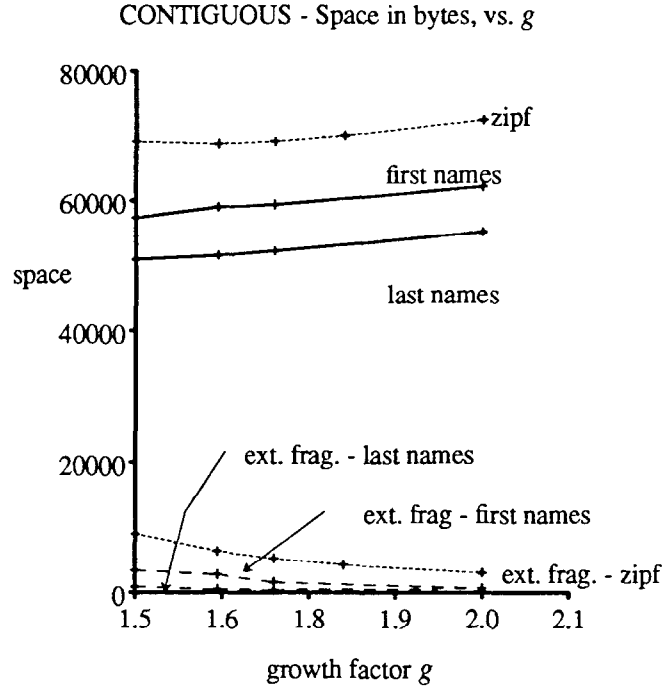
We used eqs. (5.5-5.9) to predict the performance in a fictitious database with the same number of records N, but with a Zipfian distribution (p=1). We used V=M=1500, since $V \ln V$=10,960≈N. In the upcoming graphs, the labels "last names" and "first names" mark the curves for the corresponding data sets; the label "zipf" marks the curve for theoretically derived values, assuming Zipf distribution. For the latter, the curve is plotted with a dotted line.



Graph 6.1 : Space of the postings file, as a function of b (FCHAIN method).

Graph 6.1 shows the space occupied by the postings file, as a function of b for the FCHAIN method, for the last names, and for the first names. Notice that

- the Zipf curve is minimized for b=3, as discussed before.
- the smaller the slope p of the generalized Zipf distribution, the smaller the optimal value of b: A small slope p means that many values appear once only, which means that b=1 is a good choice. For this reason, the curve for the first names (p=0.8) was optimized for b=2, while the curve for the last names (p=0.65) was optimized for b=1.
- the curve for the Zipf distribution is rather flat around the minimal value of b, suggesting that a small error in the choice of b will waste little space.

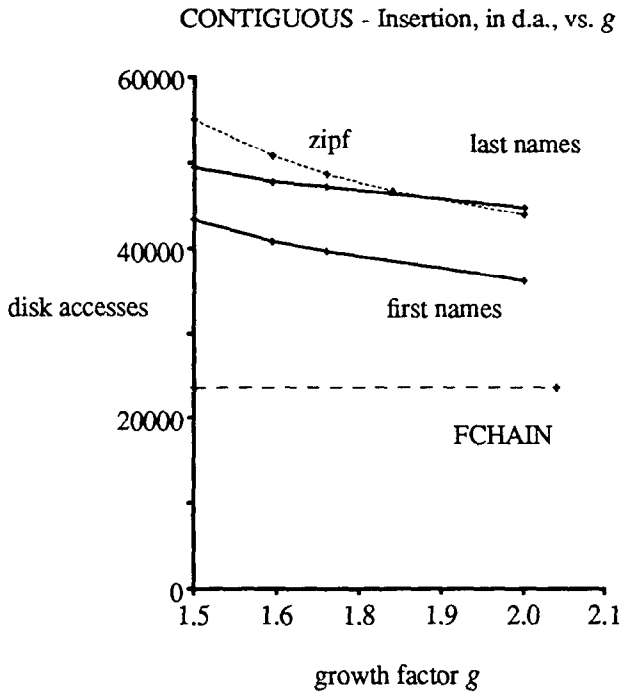


Graph 6.2: Space of the postings file, as a function of g (CONTIGUOUS method). b=1. Dashed lines shows external fragmentation for real data; dotted line for the fictitious Zipf distribution.

Graph 6.2 shows the space of the postings file as a function of g for the CONTIGUOUS method, for the last and first names. The space wasted on external fragmentation is shown with dashed lines. The main observations are:

- the space lost to external fragmentation is negligible for the real data: >6% for the first names, >1% for the last names. For the fictitious Zipf distribution, it is small (>13%).
- therefore, the total space increases with increasing g, according to eq. (5.8).
- The Zipf distribution needs more the space; the space occupied increases with the slope p. This is expected, because the uniform distribution (p=0, i.e., every value occurs once) will have no internal fragmentation; the more we deviate from that, the worse the internal fragmentation. A large growth factor g will amplify the problem, as confirmed from the graph.

- for the actual data, the space requirements are significantly smaller than FCHAIN (51Kb vs. 93Kb for the last names, or 55%; 57Kb vs. 86Kb for the first names, or 66%)

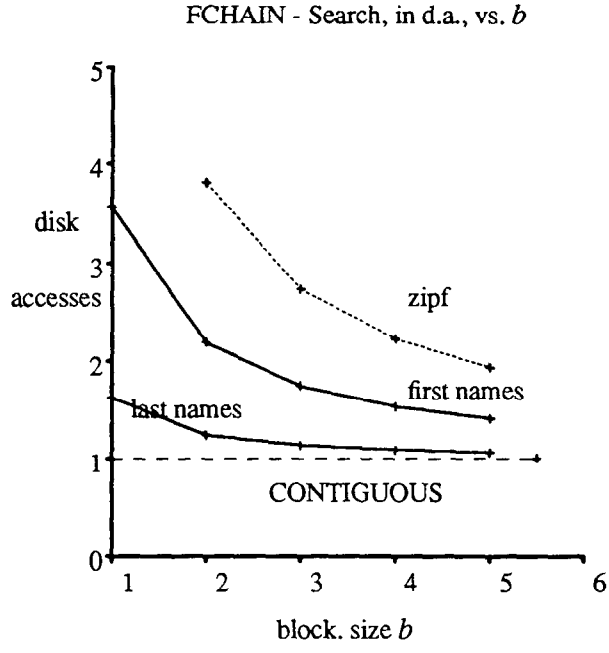


Graph 6.3 : Insertion cost (total # of disk accesses) as a function of g (CONTIGUOUS method). The dashed line shows the (constant) cost of FCHAIN.

Graph 6.3 plots the insertion time (number of random disk accesses - sum of reads and writes to insert all N keys) for the CONTIGUOUS method as a function of g. For FCHAIN, the cost is constant, as expected theoretically. Its numerical value is $2*N$=22,314 disk accesses. Graph 6.3 shows that the insertion cost decreases with the growth factor g, as expected by eq. (5.9).

However, the number of (logical) disk accesses in Graph 6.3 should only be treated as a qualitative measure of the actual time. Logical disk accesses do not necessarily translate to physical disk accesses; the translation depends on several implementation details, which are difficult to include in our formulas. For example, the buffering that UNIX performs, the specific handling of the updates on the heads of the free lists of the postings records etc.

For a better comparison between the two methods, we timed them while they were building each index. With the same, light load on a Sparc IPC, the speed of CONTIGUOUS is ≈75% of the speed of FCHAIN.



Graph 6.4 : Search time (avg. # of disk accesses) for the postings file, as a function of b (FCHAIN method).

Graph 6.4 plots the search time (number of random disk accesses for the postings file), for the FCHAIN method. For the CONTIGUOUS method the answer is always: 1; the dashed line corresponds to this performance, and is plotted for comparison purposes. The main observation is that the Zipf distribution suffers from the worst performance, exactly because it has longer lists than the other two, smoother distributions.

7. CONCLUSIONS

We have proposed the idea of adapting the size of the postings records, to accommodate skewed distributions. The proposed methods, and especially the CONTIGUOUS version, achieve low space overhead and excellent response time, compared to the simple FCHAIN method.

Our methods mainly focus on B-tree secondary indices for relations, both for dynamic and archival environments. All these methods can be used for B-tree indices on text as well, where skewed distributions are almost universal. For text, these methods could possibly be used either by themselves, or as building blocks within a hybrid method [Faloutsos92a], which could utilize signature files as well [Faloutsos90a, Sacks-

Davis87a].

In addition, the adaptive growth idea can be used in any situation that we have to maintain lists of highly varying sizes, where insertions (and possibly deletions) are allowed. For example, if the indexed attribute has a skewed value distribution, even a hashed secondary index will have a skewed distribution of its postings list (if the hash table is large enough to result in few collisions). Another example is the case of joins, where it is often beneficial to build an index on the joining attribute, on the fly.

The main observations and practical guidelines of this paper are the following:

- skewed distributions of attribute values appear often
- the FCHAIN method is straightforward and gives low insertion cost, at the expense of space and search time. When using FCHAIN, a small value of b
$$b = 3 \quad \text{or} \quad b = 4$$
will minimize the space under Zipfian distributions. Under smoother distributions (generalized Zipf, with $p < 1$), even smaller values of b are optimal.
- the CONTIGUOUS method achieves the fastest possible search time (1 disk access) per query, and low space overhead ($\approx 2/3$ of the FCHAIN, according to our experiments). The external fragmentation is negligible, as it was expected by the analysis. The insertion speed was experimentally found to be $\approx 75\%$ of the insertion speed of the FCHAIN method. For skewed distributions, the recommended values are
$$b = 1 \quad \text{and} \quad g \approx 1.5\text{-}2$$

The contributions of this paper are:

- The proposal of the CONTIGUOUS method, as well as a whole family of methods, whose general case is the HYBRID method.
- The derivation of formulas (eqs. (4.1-4)) that calculate the performance (space/insertion/search) for each method, for *any* given distribution of the attribute values.
- Guidelines on how to choose good values for the design parameters b and g, for Zipf distributions. Although real distributions deviate from the ideal Zipf distribution, the guidelines give a good initial estimate for b and g.
- The experimental comparison of the CONTIGUOUS method versus the FCHAIN method.

Future research could involve:

- analysis in case that the first n pointers are stored within the B-tree leaves; optimal choice of n, in this case.
- analysis for generalized Zipfian distributions
- analysis in the presence of deletions.
- fine-tuning of the implementation of CONTIGUOUS to reduce its insertion cost. Promising approaches include the caching of some of the postings lists, or the batching of the insertions.

Acknowledgment.

The authors would like to thank Frank Andrasco for his help with the implementation of the CONTIGUOUS method.

References

Christodoulakis84a.
Christodoulakis, S., "Implication of Certain Assumptions in Data Base Performance Evaluation," *ACM TODS*, June 1984.

Faloutsos92b.
Faloutsos, Christos and H.V. Jagadish, "On B-tree Indices for Skewed Distributions," Technical Report, Univ. of Maryland, 1992.

Faloutsos90a.
Faloutsos, C., "Signature-Based Text Retrieval Methods: A Survey," *IEEE Data Engineering*, vol. 13, no. 1, pp. 25-32, March 1990.

Faloutsos92a.
Faloutsos, C. and H.V. Jagadish, "Hybrid Index Organizations for Text Databases," *EDBT '92*, Vienna, Austria, March 23-27, 1992. Also available as UMIACS-TR-91-33 and CS-TR-2621

Ioannidis91a.
Ioannidis, Yannis E. and Stavros Christodoulakis, "On the Propagation of Errors in the Size of Join Results," *Proc. of ACM SIGMOD*, pp. 268-277, Denver, Colorado, May 29-31, 1991.

Sacks-Davis87a.
Sacks-Davis, R., A. Kent, and K. Ramamohanarao, "Multikey Access Methods Based on Superimposed Coding Techniques," *ACM Trans. on Database Systems (TODS)*, vol. 12, no. 4, pp. 655-696, Dec. 1987.

Wolf91a.
Wolf, Joel L., Daniel M. Dias, Philip S. Yu, and John Turek, "An Effective Algorithm for Parallelizing Hash Joins in the Presence of Data Skew," *Proc. IEEE Conf. on Data Engineering*, pp. 200-209, Kobe, Japan, April 8-12, 1991.

Zipf49a.
Zipf, G.K., *Human Behavior and Principle of Least Effort: An Introduction to Human Ecology*, Addison Wesley, Cambridge, Massachusetts, 1949.