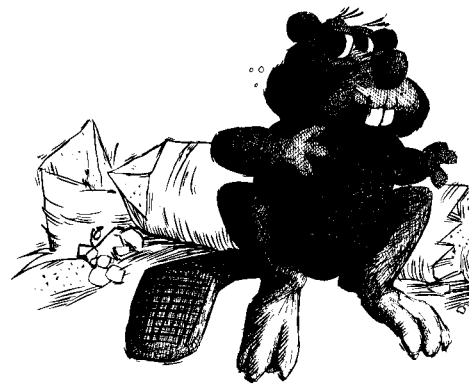
UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT



Analysis of B^+-trees with Partial Expansions

Ricardo A. Baeza-Yates
Per-Åke Larson

Research Report
CS-87-04

February, 1987

Analysis of B$^+$-trees with Partial Expansions

Ricardo A. Baeza-Yates
Per-Åke Larson

Data Structuring Group
Department of Computer Science
University of Waterloo
Waterloo, Ontario
Canada N2L 3G1
CS-87-04

ABSTRACT

In a recent paper, Lomet proposed a new version of B$^+$-trees. The idea is to gradually increase the size of an overflowing bucket, up to some maximum size, instead of immediately splitting it. In this paper we analyze the behaviour of the new file structure under random insertions, focusing on the expected storage utilization and the expected cost of insertions. The model can be used for studying both the asymptotic and dynamic behaviour. Disk space management is more difficult than for standard B$^+$-trees. We investigate two simple space management schemes specifically designed for handling buckets of two different sizes. It is found that an overall storage utilization of 81% can be achieved.

1. Introduction

The B$^+$-tree is one of the most widely used file organizations. In a B$^+$-tree all data records are stored at the lowest level (the *bucket level*), and the upper levels merely serve as an index to the data buckets [4]. All buckets are of the same size. File growth is handled by bucket splitting, that is, when a bucket overflows, an additional bucket is allocated and half of the records from the overflowing bucket are moved to the new bucket. The same type of splitting is applied to index nodes.

In a recent paper [7], Lomet proposed the use of *elastic buckets*. The idea is simply to increase the size of an overflowing bucket instead of splitting it. Each expansion step is called a *partial expansion* and a bucket is expanded until it reaches some predetermined maximum size. When a bucket of maximum size overflows, it is split into two buckets (of minimum size) in the same way as for a standard B$^+$-tree. This process of a bucket gradually growing from its minimum size until it splits, is called a *full expansion*. A full expansion increases the number of buckets by one. The idea of partial expansions has previously been applied to Linear Hashing

This work was supported by the Natural Sciences and Enginnering Research Council of Canada under Grant No. A2460 and by the University of Chile. Electronic mail: {rabaeza,palarson}%watdaisy@waterloo.csnet

[7].

Let r be the number of expansion steps required to grow a bucket from its minimum size up to and including the final split. For simplicity we assume that records are of fixed length and measure the bucket size (capacity) in number of records. Let the r different bucket sizes be $s_0, s_1, \ldots, s_{r-1}$, where $s_0 < s_1 < \cdots < s_{r-1}$. We call this the bucket *growth sequence*. In principle, any strictly increasing sequence is a valid growth sequence, provided that $2s_0 \geq s_{r-1}+1$. However, due to hardware and software limitations, the following growth sequence seems most practical [6]. Let a *page* be the smallest unit of transfer between disk and main memory, and a *page block* some fixed number of consecutive pages. Assume that the capacity of a page block is b records. For simplicity, we assume that b is integer. Then choose the growth sequence $rb, (r+1)b, \cdots, (2r-1)b$. In other words, the minimum bucket size is rb records and each partial expansion increases the bucket size by one page block, up to the maximum bucket size of $(2r-1)b$.

Example: For page blocks of size 5 and 3 partial expansions, the growth sequence is 15, 20, 25. When inserting the 26th record into a bucket, it splits into two buckets of size 15, each one containing 13 records. The minimum storage utilization for the various bucket sizes is $13/15 = 0.866...$, $16/20 = 0.8$, $21/25 = 0.84$.

In this paper we analyze the expected performance of B^+-trees with elastic buckets. The analysis is based on a model for standard B^+-trees developed by Nakamura and Mizoguchi [8]. The model is an example of *fringe analysis*, which is a technique first introduced by Yao [9] in 1978 and formalized by Eisenbarth et el. [5]. In our case, the fringe consists of only the bucket level and the index part of the file is ignored. Lomet [7] developed an approximate model which enabled him to compute the asymptotic storage utilization and bucket distribution for large buckets. Our model applies to any bucket size and can be used to study both dynamic and asymptotic behaviour.

The rest of the paper is organized as follows. The basic model is developed in Section 2, along with formulas for computing the performance measures considered. Section 2 also contains some numerical results illustrating the convergence to the steady state. In Section 3 an asymptotic model is derived and some numerical results are given. In Section 4 we study the problem of disk space management for the proposed scheme.

2. Basic Model

Consider a file that contains n records. We assume that the keys of the n records are a random sample from some underlying key distribution. Similarly, the key of a new record to be inserted is assumed to be drawn at random from the key distribution. Under this assumption, the new key is equally likely to hit any one of the $n+1$ intervals between two successive keys.

A bucket that contains j records "owns" j key intervals. (The first or last bucket has one interval more, but for large n the effect of this is insignificant and hence ignored in the subsequent analysis.) An interval that belongs to a bucket that contains j records will be called a j-interval. Let $p_j(n)$ denote the probability that an interval is a j-interval. Note that the total number of j-intervals is $(n+1)p_j(n)$.

As explained in the previous section, we assume that there are buckets of r different sizes, $rb, (r+1)b, \cdots, (2r-1)b$. Let m denote the maximum number of records per bucket, that is, $m = (2r-1)b$. The minimum number of records per bucket is then $\lfloor (m+1)/2 \rfloor$. The amount of space allocated to a bucket is uniquely determined by the number of records it contains. A

bucket of size $s_0=rb$ contains from $\lfloor(m+1)/2\rfloor$ to rb records and a bucket of size $s_i=(r+i)b$, $i>0$, contains from $(r+i-1)b+1$ to $(r+i)b$ records.

Assume, for the moment, that m is odd. The probability of a random insertion hitting a j-interval is $p_j(n)$. When a record hits a j-interval, the number of records in the corresponding bucket increases to $j+1$, provided that $j<m$. If the bucket is full ($j=m$), it splits into two buckets, each one containing $(m+1)/2$ records. However, we are more interested in the effect on the number of different j-intervals. For $j<m$, an insertion into a j-interval decreases the number of j-intervals by j and increases the number of $(j+1)$-intervals by $j+1$. For $j=m$, the number of m-intervals decreases by m and the number of $((m+1)/2)$-intervals increases by $m+1$. This gives us the following set of recurrence relations:

$$(n+1)p_j(n) = np_j(n-1) - jp_j(n-1) + (m+1)p_m(n-1), \qquad \text{for } j=(m+1)/2,$$

$$(n+1)p_j(n) = np_j(n-1) - jp_j(n-1) + jp_{j-1}(n-1), \qquad \text{for } (m+1)/2+1 \le j \le m$$

which can be rewritten in the form

$$p_j(n) = p_j(n-1)+\frac{1}{n+1}\Big[-(j+1)p_j(n-1) + (m+1)p_m(n-1)\Big] \qquad \text{for } j=(m+1)/2$$

$$p_j(n) = p_j(n-1)+\frac{1}{n+1}\Big[-(j+1)p_j(n-1) + jp_j(n-1)\Big] \qquad \text{for } (m+1)/2+1 \le j \le m$$

Following [5], we write the recurrence in matrix form:

$$\vec{p}(n) = \left(I+\frac{1}{n+1}H\right)\vec{p}(n-1) \tag{1}$$

where I is the identity matrix, H is called the transition matrix and $\vec{p}(n)$ is the probability vector $p_{\lfloor(m+1)/2\rfloor}(n)$, $p_{\lfloor(m+1)/2\rfloor+1}(n)$, $\cdots$, $p_m(n)$. When m is odd, the transition matrix is

$$H = \begin{bmatrix} -((m+1)/2+1) & 0 & . & . & . & 0 & m+1 \\ (m+1)/2+1 & -((m+1)/2+2) & 0 & . & . & . & 0 \\ 0 & (m+1)/2+2 & -((m+1)/2+3) & 0 & . & . & . \\ . & 0 & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & 0 & m-1 & -m & 0 \\ 0 & . & . & . & 0 & m & -(m+1) \end{bmatrix}$$

When m is even, the transition matrix is

$$H = \begin{bmatrix} -(m/2+1) & 0 & . & . & . & 0 & m/2 \\ m/2+1 & -(m/2+2) & 0 & . & . & 0 & m/2+1 \\ 0 & m/2+2 & -(m/2+3) & 0 & . & . & 0 \\ . & 0 & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & 0 & m-1 & -m & 0 \\ 0 & . & . & . & 0 & m & -(m+1) \end{bmatrix}$$

Given an initial state, we can use equation (1) to compute the effect of a sequence of random insertions. From $\vec{p}(n)$ we can then compute not only the storage utilization but several other performance measures. The expected number of buckets which contain j records is

$$A_j(n) = \frac{p_j(n)(n+1)}{j}$$

The expected number of buckets of size $s_i=(r+i)b$ per record is

$$Nb_i = \sum_j \frac{A_j(n)}{n+1},$$

where j ranges from $\lfloor (m+1)/2 \rfloor$ to rb for $i=0$ (the minimum bucket size), and from $(r+i-1)b+1$ to $(r+i)b$ for $i=1, 2, \cdots, r-1$. The expected total number of buckets per record is

$$NB = \sum_{i=0}^{r-1} Nb_i$$

The expected number of records in buckets of size s_i per record is

$$iNr_i = \sum_j \frac{jA_j(n)}{n+1} = \sum_j p_j(n)$$

where j ranges over the same values as in the definition of Nb_i.

The expected storage utilization in buckets of size s_i is

$$u_i = \mathrm{E}\left(\frac{\textit{number of records in buckets of size } s_i}{\textit{space used by buckets of size } s_i} \right),$$

that is, the expected value of the quotient of two random variables. We estimate this by

$$u_i = \frac{Nr_i}{s_i\, Nb_i}$$

For finite n this is not entirely correct because, in general, $\mathrm{E}(x/y) \neq \mathrm{E}(x)/\mathrm{E}(y)$. However, it can be shown that the above estimate is asymptotically correct. The details of showing this are somewhat involved, but in essence it follows directly from the law of large numbers. In the same way, we estimate the expected fraction of buckets of size s_i as

$$f_i = \frac{Nb_i}{NB}$$

which is also an asymptotically correct estimate. The amount of space actually in use is n, the number of records in the file, and the expected total amount of space allocated is

$$S = (n+1)\sum_{i=0}^{r-1} (r+i)bNb_i$$

The expected storage utilization is defined as

$$U = nE\left\{\frac{1}{total\ space}\right\}$$

which we estimate by

$$U = \frac{n}{E(\ total\ space\ )} = \frac{1}{S}(1 + \frac{1}{n})$$

This is also an asymptotically correct estimate. For finite n we can find upper and lower bounds on U by using the Kantorovich inequality [3, p.314]

$$1 \leq E(x)E(1/x) \leq 1 + \frac{(x_{\max} - x_{\min})^2}{4x_{\min}x_{\max}}$$

with $x_{\min} = n$ and $x_{\max} = n/U_{\min}$, where $U_{\min}$ is the minimum possible storage utilization. This gives the bounds

$$\frac{1}{S}(1 + \frac{1}{n}) \leq U \leq \left(1 + \frac{(1-U_{\min})^2}{4U_{\min}}\right)\frac{1}{S}(1 + \frac{1}{n})$$

The final performance measure to be considered is the cost of insertions. The total cost of an insertion depends on the height of the index, which in turn depends on how the index is organized. Following Lomet [7], we consider only accesses to the bucket level and the bottom level of the index. The fanout of an index is typically sufficiently high that the amount of activity in the index above the bottom level is insignificant. It is also assumed that the bookkeeping necessary for allocating and freeing disk space is done entirely in main memory. Under these assumptions, insertion of a new record requires a minimum of two disk accesses (one read and one write). If the insertion causes a partial expansion, three accesses are needed: read the old bucket, write the expanded bucket, update and write the affected index page. A split requires four accesses: read the old page, write the old page, write the new page, update and write the affected index page. Note that we assume that the index page affected is already in main memory as a result of traversing the index. The expected number of accesses for an insertion is then

$$\begin{aligned}\bar{I}_{acc} &= 2 + Prob\{\ partial\ expansion\ \} + 2\ Prob\{\ split\ \} \\ &= 2 + \sum_{i=0}^{r-2} p_{(r+i)b}(n) + 2p_{2(r-1)b}(n)\end{aligned}$$

To obtain the "true" total cost of an insertion, the number of accesses required to traverse the index would have to be added.

Figures 1a and 1b show the development of the overall storage utilization for 1, 2, and 3 partial expansions. In figure 1a the smallest bucket size is 12 and in figure 1b it is 60. In the initial state, all buckets were of the smallest size and completely filled. Figure 2 shows the expected cost of an insertion and figures 3a and 3b the expected fraction of buckets of different sizes.

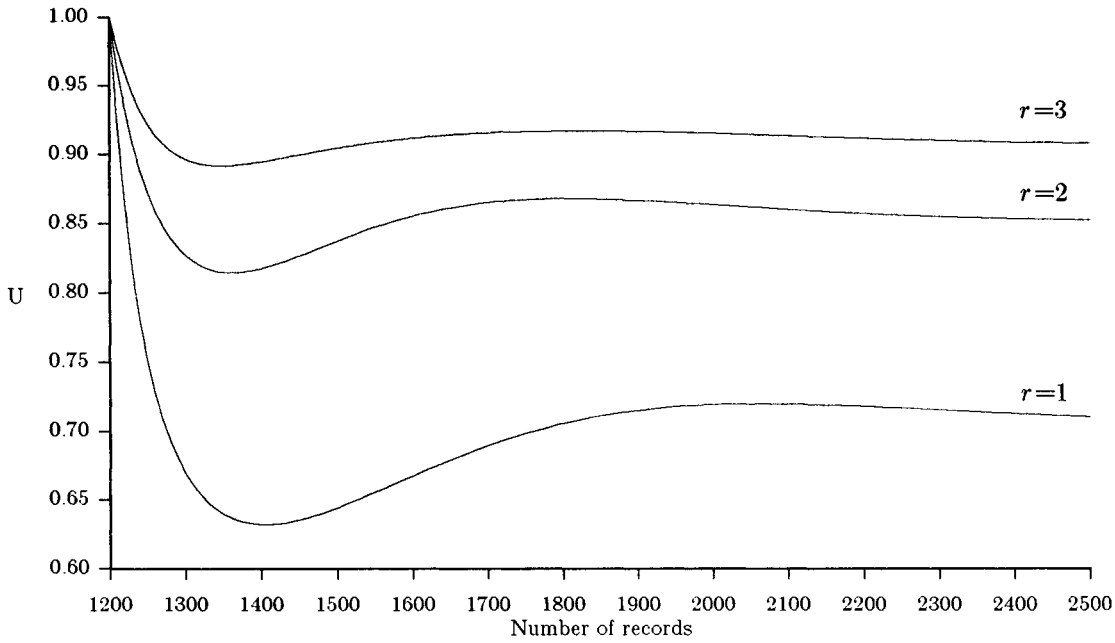


Figure 1a: Expected storage utilization after initial loading for smallest bucket size 12

It is apparent from figures 1 to 3 that initially filling the buckets completely is not a good strategy. As records are inserted the overall storage utilization drops very rapidly and then slowly converges with considerable oscillation. The oscillations increase with the page size. Partial expansions improve the storage utilization, as expected, and also reduce the oscillations. The variation in the storage utilization is accompanied by a similar variation in the expected insertion cost.

Figures 3a and 3b reveal the cause of the varying insertion cost. When two partial expansions are used, all buckets of smallest size are quickly expanded. These larger buckets then slowly fill up and start splitting. This alternation between an excess of buckets of the smallest size and an excess of larger buckets eventually dies out, but very slowly. The pattern is similar for three partial expansions, with the difference that there is alternation between three different bucket sizes.

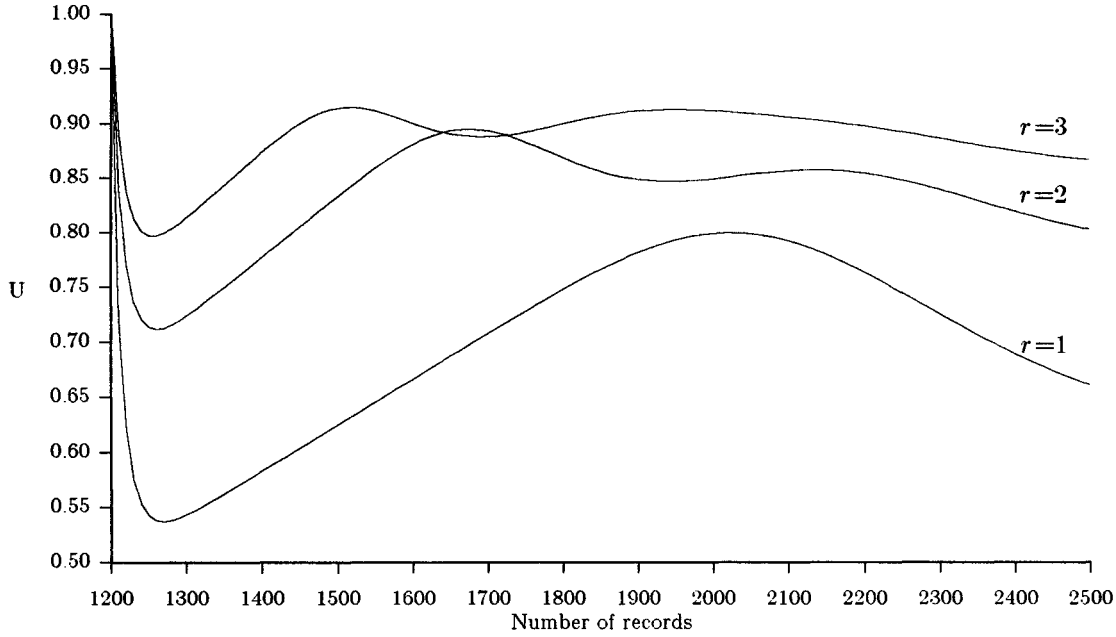


Figure 1b: Expected storage utilization after initial loading for smallest bucket size 60

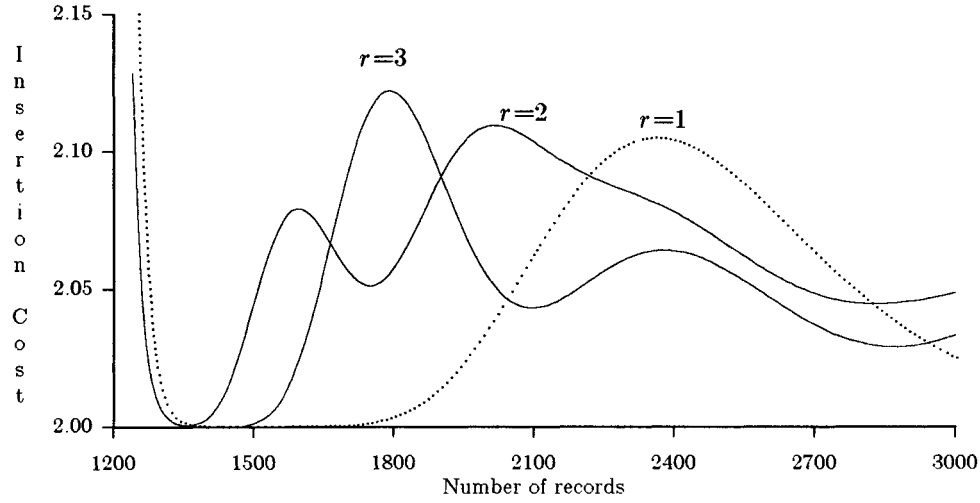


Figure 2: Insertion cost after initial loading for smallest bucket size 60

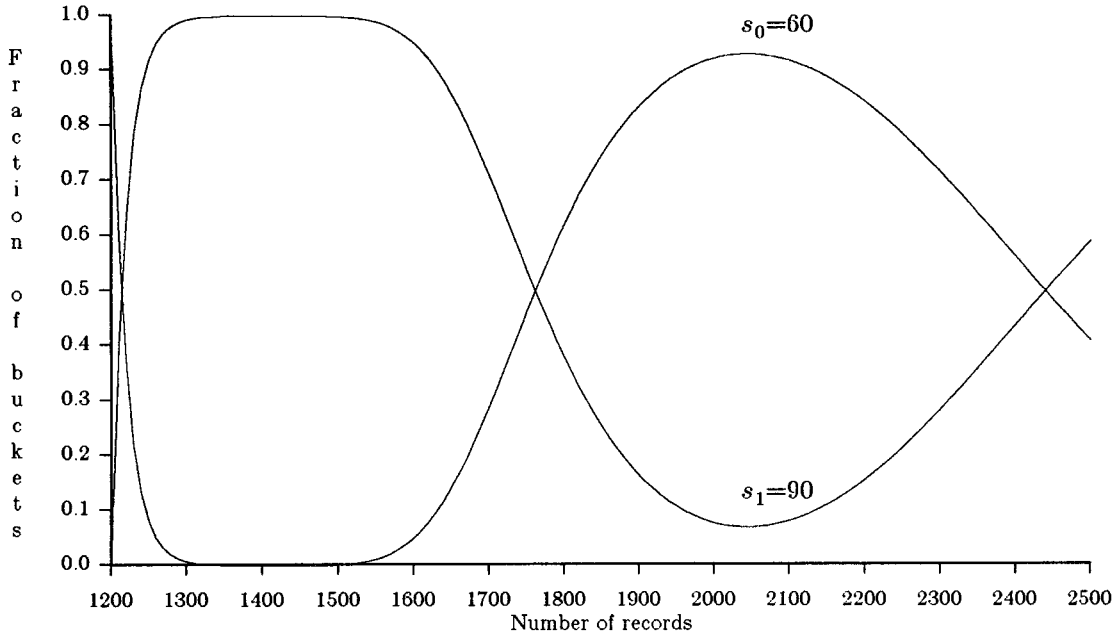


Figure 3a: Expected fraction of buckets of each size after initial loading for 2 partial expansions

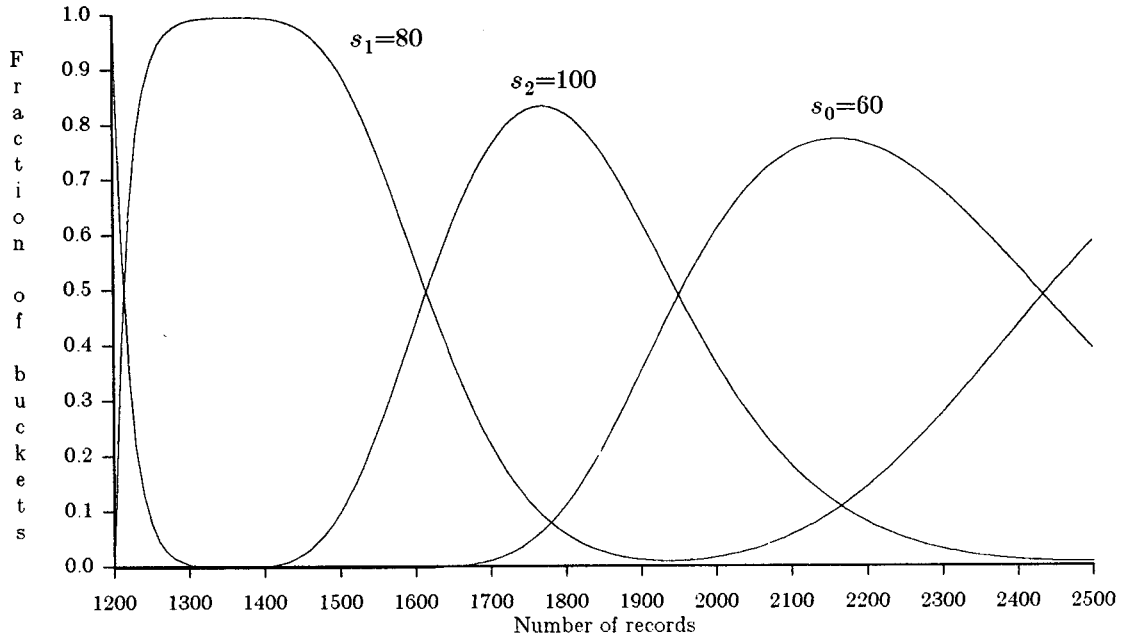


Figure 3b: Expected fraction of buckets of each size after initial loading for 3 partial expansions

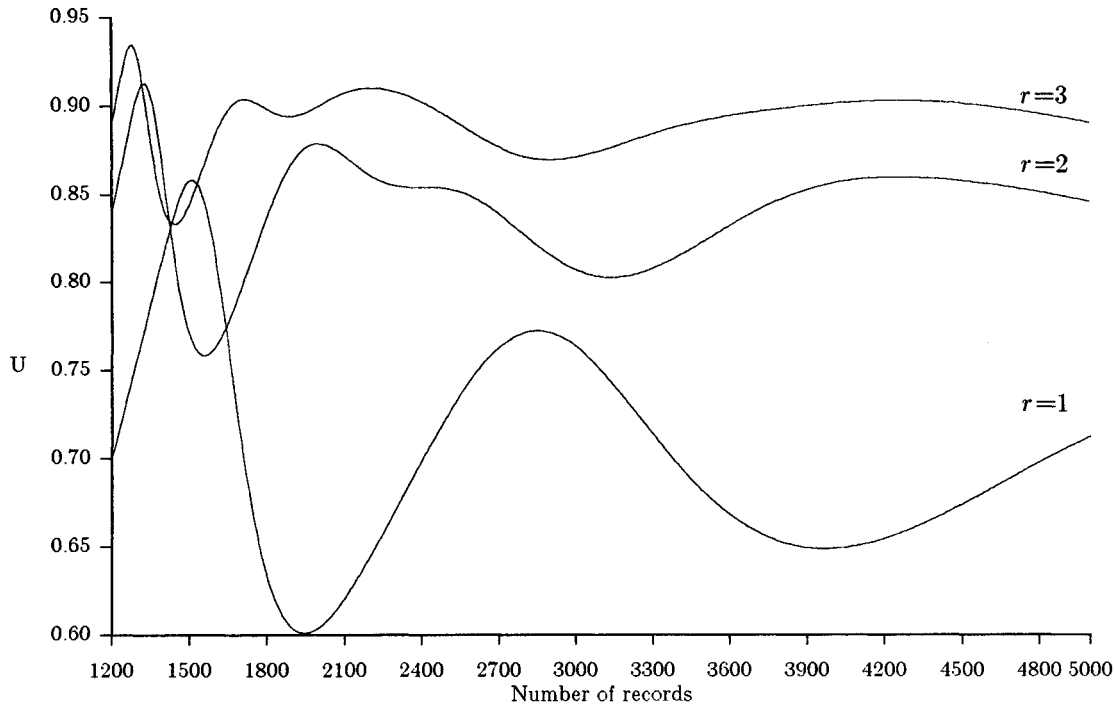


Figure 4: Expected storage utilization after initial loading to steady utilization using only buckets of smallest size (smallest bucket size 60)

Figures 4 and 5 show the development of the overall storage utilization starting from a different initial state. In figure 4 the file initially consisted of only smallest buckets, but these were filled to approximately the steady state utilization. The variation decreases but it is still significant. Figure 5 shows the development when starting with an empty file. It is rather surprising how long the oscillations persist even in this case.

What conclusions can we draw from these results? It is clearly not advisable to "overload" a B^+-tree, with or without elastic buckets, during initial loading. As new records are inserted, the overall storage utilization will drop fairly rapidly and oscillate around the steady-state level quite long. Oscillations cause increased activity in allocating and freeing disk space. As we will see in the next section, this affects the overall storage utilization.

We have performed rather extensive simulation experiments to test the accuracy of the model. Figure 6 shows the results from two series of simulation experiments. The solid lines show the 95% confidence interval for the the average storage utilization as obtained from 100 simulated file loadings. The dotted line shows the theoretical values. The results are in excellent agreement. Other experiments gave similar results. One other observation from the experiments is worther mentioning. If we want an initial load of 100%, it is better to use two different bucket sizes, for example, in the case of partial expansions, use the same number of buckets of each size.

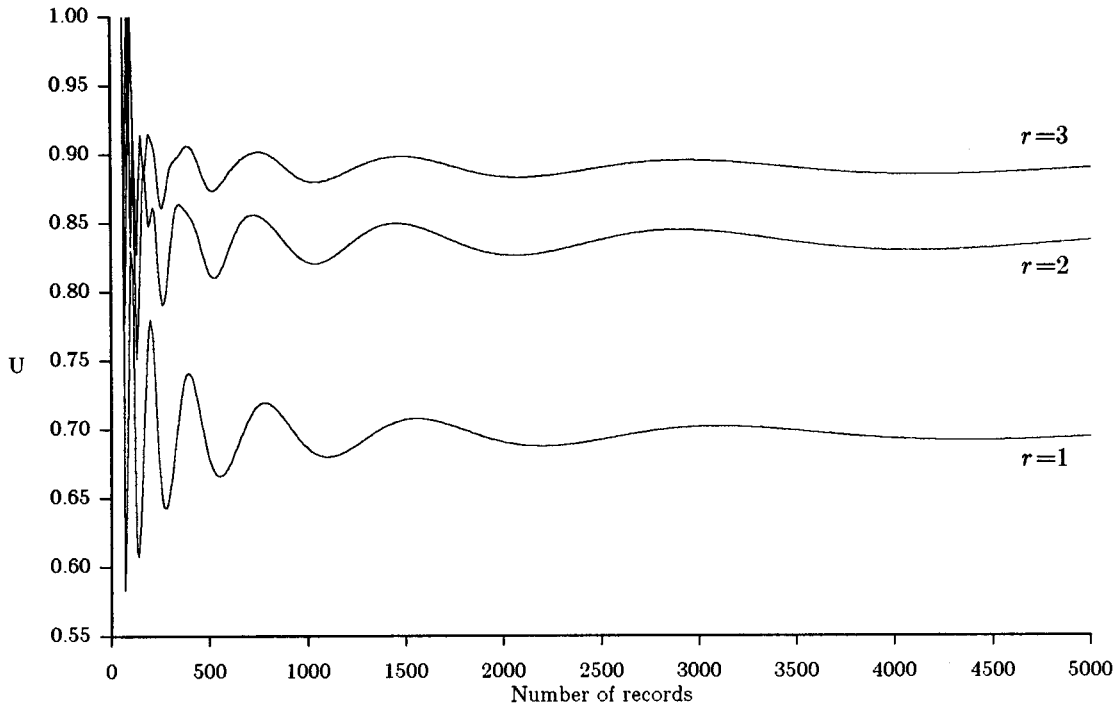


Figure 5: Expected storage utilization starting from an empty file (smallest bucket size 60)

3. Asymptotic Analysis

When $n \rightarrow \infty$, recurrence (1) converges to the solution of the following linear system of equations [5, p.135] (with an error term of order n^{λ_2} with $\mathrm{Re}(\lambda_2) < 0$):

$$H\vec{p}(\infty) = 0, \qquad \sum_j p_j(\infty) = 1.$$

Let $p_j = p_j(\infty)$ denote the steady-state probabilities. The system is easily solved because of the simple structure of H. When m is odd, the solution is

$$p_j = \frac{1}{(j+1)(H_{(m+1)} - H_{(m+1)/2})}, \qquad \text{for } j=(m+1)/2,\ldots,m.$$

where $H_m = \sum_{i=1}^{m} 1/i$.

When m is even, the solution is

$$p_j = \frac{1}{(j+1)(H_m - H_{m/2})}, \qquad \text{for } j=m/2+1,\ldots,m$$

$$p_{m/2} = \frac{m}{(m+2)(m+1)(H_m - H_{m/2})}$$

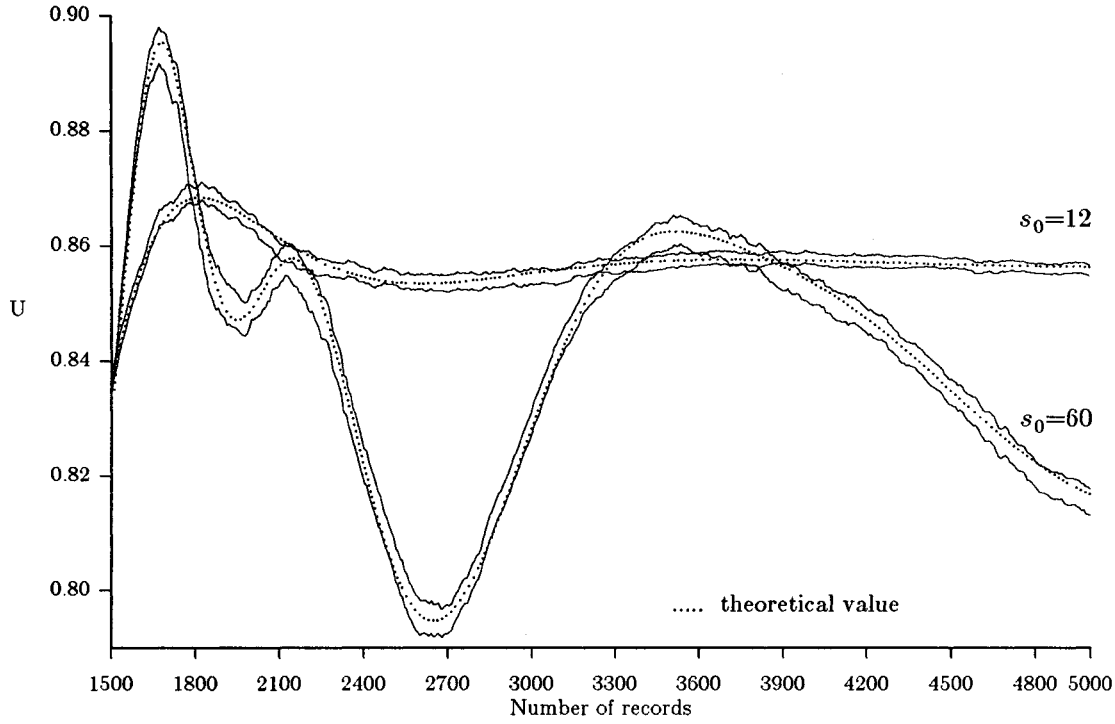


Figure 6: 95% confidence interval for the average storage utilization for smallest bucket sizes 12 and 60

The following formulas are valid for all m, unless explicitly separated for m odd and m even. They are obtained from the corresponding formulas in the previous section by using the steady-state probabilities. Define

$$\alpha(m) = \begin{cases} (H_{(m+1)} - H_{(m+1)/2})^{-1} & m \text{ odd} \\ (H_m - H_{m/2})^{-1} & m \text{ even} \end{cases}$$

The expected number of buckets of size $s_i = (r+i)b$ per record is

$$Nb_i = \begin{cases} \dfrac{(b+1)\alpha(m)}{(rb+1)(m+1)} & i=0 \\ \dfrac{b\alpha(m)}{((r+i)b+1)((r+i-1)b+1)} & i=1,2,\cdots,r-1 \end{cases}$$

and the total expected number of buckets per record is

$$NB = \frac{\alpha(m)}{(2r-1)b+1}.$$

Hence, the expected fraction of buckets of size $s_i = (r+i)b$ is

$$f_i = \begin{cases} \dfrac{b+1}{rb+1} & i=0 \\ \dfrac{b((2r-1)b+1)}{((r+i)b+1)((r+i-1)b+1)} & i=1,2,\cdots,r-1 \end{cases}$$

The expected fraction of records residing in buckets of size $s_i = (r+i)b$ is

$$Nr_i = \begin{cases} \alpha(m)(H_{rb+1} - H_{(m+1)/2}) & i=0,\ m \text{ odd} \\ \alpha(m)(H_{rb+1} - H_{m/2} - \dfrac{1}{m+1}) & i=0,\ m \text{ even} \\ \alpha(m)(H_{(r+i)b+1} - H_{(r+i-1)b+1}) & i=1,2,\cdots,r-1 \end{cases}$$

and the expected storage utilization in the buckets of size $s_i = (r+i)b$ is

$$u_i = \begin{cases} \dfrac{(rb+1)(m+1)}{rb(b+1)}(H_{rb+1} - H_{(m+1)/2}) & i=0,\ m \text{ odd} \\ \dfrac{(rb+1)(m+1)}{rb(b+1)}(H_{rb+1} - H_{m/2} - \dfrac{1}{m+1}) & i=0,\ m \text{ even} \\ \dfrac{((r+i)b+1)((r+i-1)b+1)}{b^2(r+i)}(H_{(r+i)b+1} - H_{(r+i-1)b+1}) & i=1,2,\cdots,r-1 \end{cases}$$

For large n, the expected storage utilization ($U_{\min} = r/(r+1)$, in the first partial expansion) is bounded by

$$\frac{\alpha(m)^{-1}}{\Psi(2r+1/b) - \Psi(r+1/b)} \leq U \leq (1 + \frac{1}{4r(r+1)})\frac{\alpha(m)^{-1}}{\Psi(2r+1/b) - \Psi(r+1/b)}$$

Asymptotically, it is equal to the lower bound. The function $\Psi(z)$ is defined as [1, p.248] $\Psi(z) = \dfrac{d}{dz}\ln(\Gamma(z))$. When z is an integer $\Psi(z) = H_{z-1}$. The difference between two values of Ψ is most easily computed using the formula

$$\Psi(m+x) - \Psi(n+x) = \sum_{i=n}^{m-1} \frac{1}{i+z} \qquad m>n,\ 0 \leq x < 1$$

for integer n and m.

The probability that an insertion causes a split equals p_m, that is,

$$Prob\{split\} = \frac{\alpha(m)}{m+1}$$

The probability that an insertion causes a bucket to expand from size s_{j-1} to s_j equals p_{j-1}, that is,

$$Prob\{j\text{th } partial\ expansion\} = \frac{1}{(r+j-1)b+1}\alpha(m)$$

and hence the probability that an insertion triggers a partial expansion is

$$Prob\{\ partial\ expansion\ \} = (\Psi(2r-1+1/b) - \Psi(r+1/b))\frac{\alpha(m)}{b}$$

Therefore the expected number of accesses for an insertion is

$$\bar{I}_{acc} = 2 + \frac{\alpha(m)}{b}(\Psi(2r+1/b) - \Psi(r+1+1/b) + \frac{1}{2r-1+1/b})$$

The expected number of page blocks per record is

$$\bar{P} = \sum_{i=0}^{r-1} (r+i)Nb_i = \frac{\alpha(m)}{b}(\Psi(2r+1/b) - \Psi(r+1/b)) .$$

When $b \rightarrow \infty$ we get the following formulas

$$f_i = \begin{cases} \dfrac{1}{r} & i=0 \\ \dfrac{2r-1}{(r+i)(r+i-1)} & i>0 \end{cases}$$

$$u_i = \begin{cases} (2r-1)(\ln(r) - \ln(r-1/2)) & i=0 \\ (r+i-1)(\ln(r+i) - \ln(r+i-1)) & i>0 \end{cases}$$

$$U = \frac{\ln 2}{H_{2r-1} - H_{r-1}}$$

Table 1 and figure 7 show the asymptotic storage utilization for different page block sizes and number of partial expansions. The storage utilization decreases when the size of page blocks is increased. Figure 7 shows more clearly the effect of increasing the number of partial expansions. The additional gain in storage utilization decreases rapidly, and in practice it hardly seems worthwhile to go beyond three partial expansions.

Figure 8 plots the expected cost of an insertion as a function of the page block size. Increasing the page block size dramatically reduces the cost of insertions because the overhead of bucket expansions and splits is distributed over more records. The figure also seems to indicate that increasing the number of partial expansions also reduces the cost. This should be taken with a grain of salt, however, because buckets are not of the same size. If page blocks are of size 5, for example, then the buckets are of size 5 when using one partial expansion but of size 10 and 15 when using two partial expansions. The reduced insertion cost is largely due to the larger bucket size.

b	Partial expansions (r) 1	2	3	4
2	.75	.8993	.9360	.9523
4	.7292	.8685	.9118	.9335
8	.7138	.8509	.8987	.9233
12	.7076	.8447	.8942	.9198
20	.7022	.8397	.8905	.9169
40	.6978	.8358	.8877	.9148
60	.6963	.8344	.8868	.9140
∞	.6931	.8318	.8849	.9126

Table 1: Asymptotic storage utilization for 1, 2, 3 and 4 partial expansions

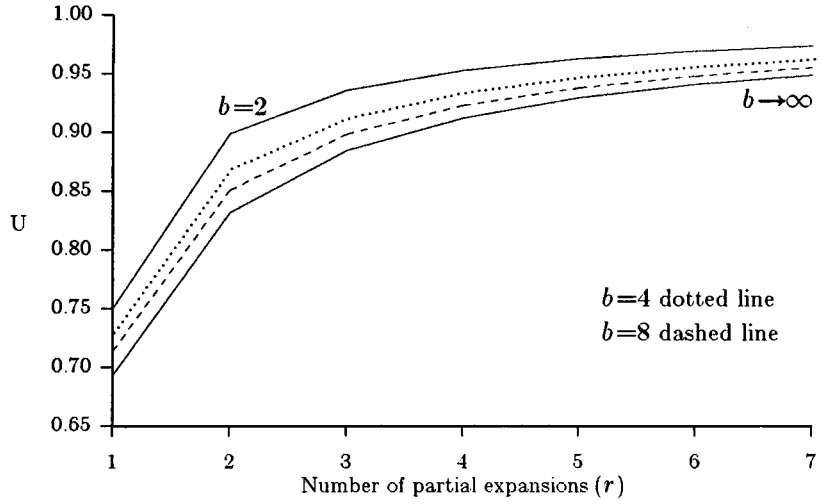


Figure 7: Asymptotic storage utilization for different page block sizes as a function of the number of partial expansions (smallest bucket size rb)

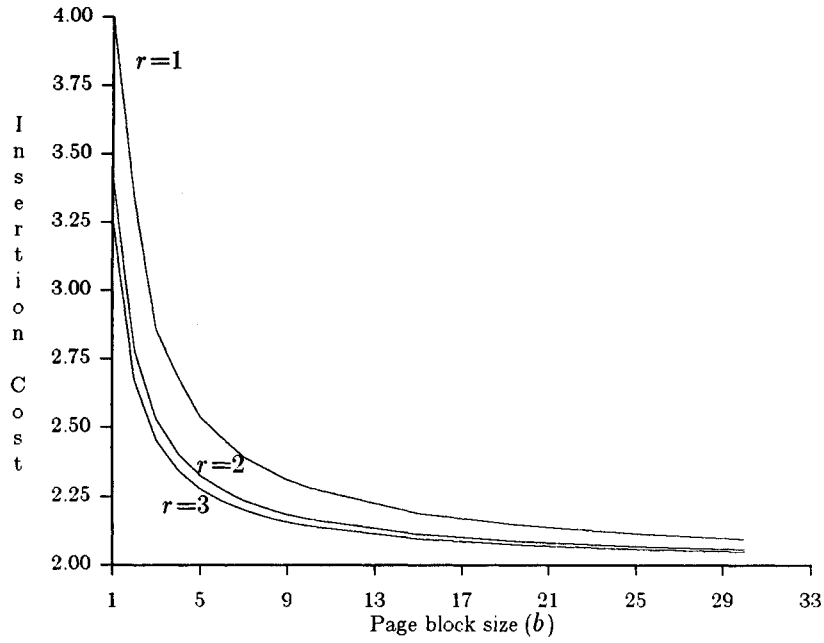


Figure 8: $\bar{I}_{acc}$ as a function of the page block size for
1, 2 and 3 partial expansions (smallest bucket size = rb)

Table 2 list the asymptotic storage utilization for one to eight partial expansions and $b \rightarrow \infty$ They agree with the results obtained by Lomet [7]. Table 3 shows the expected fraction of buckets of different size and the corresponding storage utilization for each bucket size for the case $b \rightarrow \infty$

r	1	2	3	4	5	6	7	8
Utilization	.6931	.8318	.8849	.9126	.9296	.9411	.9493	.9556

Table 2: Asymptotic storage utilization for $b \rightarrow \infty$

r	f_0	f_1	f_2	f_3	f_4	f_5	u_0	u_1	u_2	u_3	u_4	u_5
1	1						0.69					
2	0.50	0.50					0.86	0.81				
3	0.33	0.42	0.25				0.91	0.86	0.89			
4	0.25	0.35	0.23	0.17			0.93	0.89	0.91	0.92		
5	0.20	0.30	0.21	0.16	0.13		0.94	0.91	0.92	0.93	0.94	
6	0.17	0.26	0.20	0.15	0.12	0.10	0.95	0.92	0.93	0.94	0.95	0.95

Table 3: Expected fraction of and storage utilization within buckets of differents sizes for $b \rightarrow \infty$

The accuracy of the asymptotic model was also tested by simulation. Table 4 summarize the results from two simulation experiments and compare them with the corresponding theoretical results. Each experiment consisted of 100 file loadings. The number of partial expansions was two in both experiments. For a smallest bucket size of 12, all the theoretical results are within the 95% confidence interval. For a smallest bucket size of 60, and 150000 records, the theoretical values are slightly outside the 95% confidence interval. However, for 120000 records they were all within the confidence interval. This indicates that there are small, but noticeable oscillations even for a file over 150000 records.

Measure	Smallest bucket size 12		Smallest bucket size 60	
	Simulation (20000 records, 100 runs)	Theoretical	Simulation (150000 records, 100 runs)	Theoretical
U	0.85678 ± 0.00043	0.85696	0.83762 ± 0.00051	0.83706
u_0	0.87820 ± 0.00062	0.87783	0.86582 ± 0.00059	0.86645
u_1	0.84019 ± 0.00064	0.84073	0.81752 ± 0.00082	0.81681
NB	0.079004 ± 0.000085	0.079010	0.016027 ± 0.000020	0.01598
f_0	0.5377 ± 0.0026	0.5385	0.5168 ± 0.0032	0.5082
f_1	0.4623 ± 0.0026	0.4615	0.4832 ± 0.0020	0.4918

Table 4: Simulation results for large files (95% confidence interval)

4. Disk Space Management

With two partial expansions the minimum storage utilization is 67% and the expected value 83% (for large buckets). This is slightly higher then the expected value of 81% predicted for B*-trees [2]. In a B*-tree we attempt to move records to an adjacent bucket instead of immediately splitting an overflowing bucket. When this is not possible, two buckets are expanded into three. B*-trees also provide a minimum storage utilization of 67%. Using partial expansions appears to be a better solution than B*-trees, because is not necessary to read and write adjacent buckets. It is easier to implement and the insertion costs are lower. However, this conclusion may be premature. Disk space management is more difficult because we have to handle buckets of different sizes. We have investigated two simple space management schemes specifically designed for handling buckets of two different sizes. Results from simulation experiments are reported in this section.

The first method divides the total file space into chunks of five page blocks. The second method divides it into chunks of six page blocks. When more space is needed for the file, an additional chunk is requested from the operating system. A chunk stores either one, two or three buckets. Denoting the relative bucket sizes by 2 and 3, the following classification of the states of a chunk is sufficient for our purpose:

Size 5: empty, 2+0, 0+2, 3+0, 0+3, 2+0+2, full

Size 6: empty, 2+0, 0+2+0, 0+2, 3+0, 0+3, 2+2+0, 2+0+2, 0+2+2, full

The notation 2+0 means that a bucket of size 2 occupies the two leftmost page blocks and the rest are free. 2+0+2 means that there is free space in the middle of the chunk. An in-core table is used for keeping track of the state of each chunk of the file. We must be able to distinguish

between the states listed above. Hence the first scheme requires 3-bit entries and the second scheme 4-bit entries. From this table we can find out not only whether a chunk has room for a new bucket, but also the exact size and location of the free space within the chunk. We can therefore write out a new bucket directly without first having to read in the chunk. Even for a large file, the bit table is small enough to be kept in main memory.

Space may now be wasted because of internal and external fragmentation. Internal fragmentation refers to the space wasted within buckets. External fragmentation refers to the space wasted because chunks are not completely filled. The results in the previous sections took into account only internal fragmentation. We are now interested in the storage utilization taking into account also the space wasted due to external fragmentation. To distinguish between the two, we will refer to them as *internal storage utilization* and *external storage utilization*, respectively.

The scheme with chunks of size 5 allocates space as follows. When space is needed for a bucket of size 2, a chunk already containing a bucket of size 2 is used only if there are no chunks containing a bucket of size 3. That is, we try to avoid wasting 1/5 of a chunk. When space is needed for a bucket of size 3, we first try to place it in an existing chunk. Only when there is not sufficient space in any existing chunk, a new chunk is requested.

We can find an upper bound on the average external storage utilization for the first scheme in the steady state as follows. In the best case, most of the buckets will be completely filled. However, in the steady state there is an excess of buckets of size 2. The highest storage utilization is achieved if the excess buckets are stored in chunks in state 2+0+2. In each such chunk 1/5 of the space is wasted. Hence, an upper bound for the average external utilization is given by

$$U_e \leq (\frac{2}{5}u_0 + \frac{3}{5}u_1)(1-(f_0-f_1)) + \frac{4}{5}u_0(f_0-f_1)$$

$$= \frac{4}{5}u_0\, f_0 + \frac{6}{5}u_1 f_1$$

The scheme with chunks of size 6 searches for chunks with free space in a certain order. The order used was

Size 2: 0+2+0; 2+2+0 or 2+0+2 or 0+2+2; 2+0+0 or 0+0+2; 0+3 or 3+0

Size 3: 0+3 or 3+0; 0+0+2 or 2+0+0

That is, when space is needed for a bucket of size 2, we first try to find a chunk in state 0+2+0. If no such chunks exist, we try to find a chunk in one of the states 2+2+0, 2+0+2, or 0+2+2, and so on. In this case, the theoretical upper bound is equal to the internal storage utilization. In the best case all buckets of size 2 will be in 2+2+2 chunks and all buckets of size 3 in 3+3 chunks.

Figure 9 shows the development of the average external storage utilization for the two schemes for smallest bucket size 60. The two solid lines represent averages from 100 simulated file loadings. The dotted line represents the theoretically expected storage utilization for a standard B^+-tree with bucket size 60. Initially the file consisted of completely filled smallest buckets. This initial state was deliberately chosen to see how the schemes perform when there is heavy expansion and splitting activity, and when there is a significant excess of buckets of one type.

It is apparent from figure 9 that the second scheme performs significantly better than the first scheme when the bucket distribution is highly skewed. The performance of the first scheme is simply too erratic and unreliable. The lowest storage utilization for the first scheme occurs when most of the buckets are of size 3. Many chunks will then contain just one bucket of size 3 because there are not enough buckets of size 2 to fill the "holes". For the second scheme, the lowest storage utilization occurs when there has been an excess of bucket of size 3 and they start splitting. When there is an excess of buckets of size 3, there will be many buckets in state 3+3. When these buckets start splitting, a significant fraction of buckets in state 2+3 (or 3+2) will be created, wasting 1/6 of a chunk. This is further exacerbated by the low internal storage utilization in the newly created buckets of size 2.

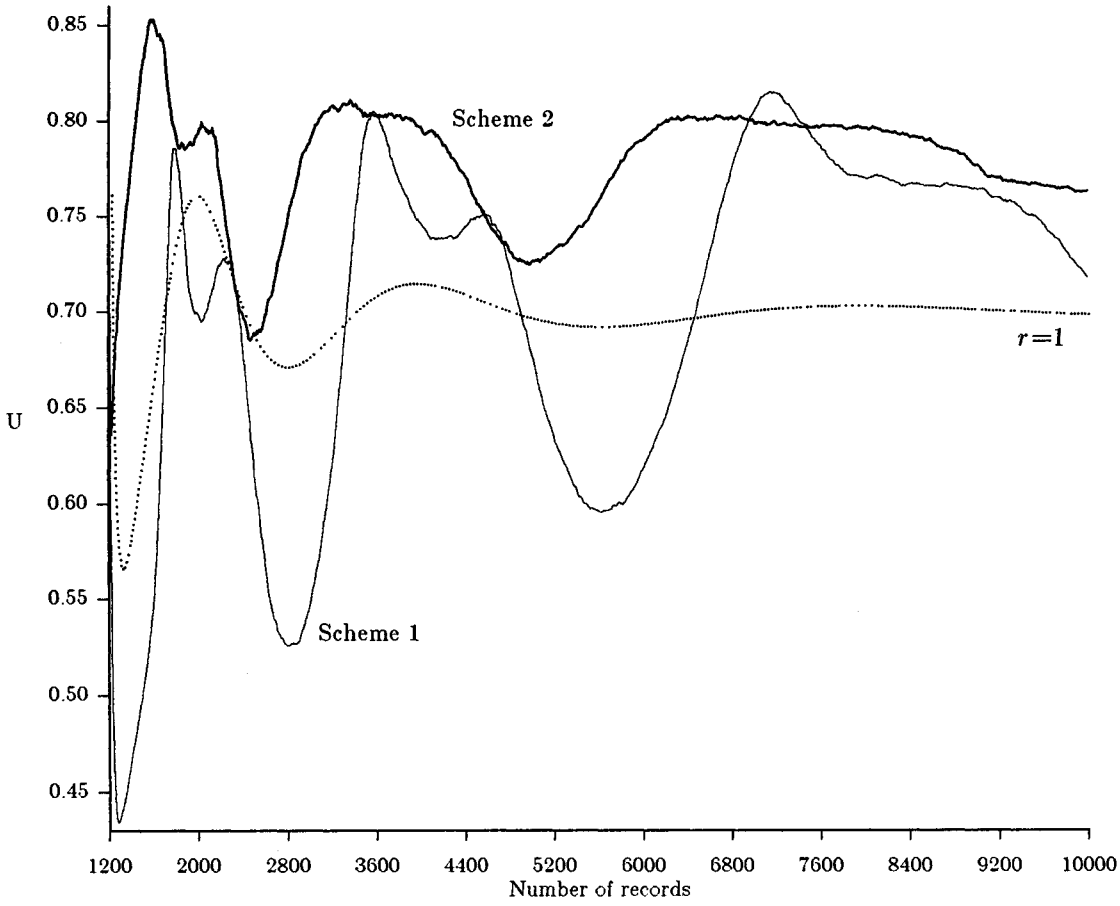


Figure 9: Average external storage utilization for smallest bucket size 60

Table 5 shows simulation results and theoretical upper bounds for large files and two different smallest bucket sizes. Note that the upper bound cannot be achieved without relocation of buckets. In the steady state there is still expansion and splitting activity going on. When a bucket is expanded or split, a "hole" may be created which then will persist for some time until it is eventually filled. At any given point in time, a few such "holes" may exist. To fill a "hole" immediately would require relocation of some existing bucket.

Smallest bucket size (100 runs)	Simulation		Upper bound	
	Scheme 1	Scheme 2	Scheme 1	Scheme 2
12 (20000 records)	0.8377 ± 0.0011	0.82800 ± 0.00071	0.84377	0.85696
60 (150000 records)	0.8232 ± 0.0014	0.81020 ± 0.00056	0.83431	0.83706

Table 5: External storage utilization for large files (95% confidence interval)

As seen from table 5, the first scheme gives a higher external storage utilization in the steady state. However, the difference is small (less than 1 percentage point) and the second scheme is more reliable during transient phases. Scheme two also handles clustered or ordered insertions better than scheme one. The external storage utilization is over 80%, which is more than 10 percentage points higher than the storage utilization of a standard B^+-tree. The results are close to the upper bound which indicates that there is little to be gained by bucket relocation.

We also simulated two variants of the basic schemes presented above. The first one was a variant of the scheme with chunks of size 5 which never stored two buckets of size 2 in the same chunk. The idea is to avoid wasting 1/5 of a chunk. Both its steady state storage utilization and its performance during transient phases were worse than for the basic scheme. The second variant was a simplified version of the scheme using chunks of size 6. This version did not attempt to use the chunks in any particular order. The first chunk with enough free space to store the new bucket was used. Its performance was slightly worse than the basic scheme, but only marginally so.

5. Conclusion

The following points summarize the main results

- Compared with standard B^+-trees [8], the storage utilization of B^+-trees with partial expansions is higher. Using bucket of two different sizes we can increase the storage utilization by more than 10 percentage points.
- During dynamic phases the storage utilization varies less than in standard B^+-trees, but takes longer to stabilize. This variation increases with the bucket size.
- The asymptotic storage utilization depends on the bucket size. It is slightly higher for small buckets.
- The expected insertion cost does not differ significantly from the insertion cost for standard B^+-trees.
- Among the space management schemes investigated, the basic scheme using chunks of size 6 performs best. Its dynamic behaviour is fairly stable and asymptotically the storage utilization is reduced by as little as 2-3 percentage points due to external fragmentation.
- Based on the storage utilization alone it is hardly worthwhile to go beyond four partial expansions. Taking into account other factors as well, it seems that two partial expansions is the best choice in practice - an overall storage utilization of approximately 81% can be achieved, the dynamic behaviour is relatively good, and storage management is easy.

- Compared with B*-trees, a B^+-tree with two partial expansions is easier to implement, the storage utilization is about the same, but insertion cost are significantly lower [2]. Compared with other known overflow handling techniques, this method appears to have the lower insertion cost.

References

[1] Abramowitz, M. and Stegun, I., *Handbook of Mathematical Functions.* Dover Publications, New York, 1972.

[2] Baeza-Yates, R., The Expected Behaviour of B^+-trees, Department of Computer Science, University of Chile, Santiago, Chile, 1985. Also available as Technical Report CS-86-68, Dept. of Computer Science, University of Waterloo, 1986.

[3] Clausing, A., Kantorovich-Type Inequalities, *The American Mathematical Monthly* **89**, 5 (1982), 314-330.

[4] Comer, D., The Ubiquitous B-Tree, *Computing Surveys* **11**, 2 (1979), 121-137.

[5] Eisenbarth, B., Ziviani, N., Gonnet, G., Mehlhorn, K. and Wood, D., The Theory of Fringe Analysis and Its Application to 2-3 Trees and B-Trees, *Information and Control* **55**, 1-3 (1982), 125-174.

[6] Larson, P-Å. Linear Hashing with Partial Expansions, Proc. 6th Int. Conf. on Very Large Data Bases, (Montreal, Canada, 1980), 224-232.

[7] Lomet, D., Partial Expansions for File Organizations with an Index, *ACM Trans. on Database Systems* (to appear). Also available as Technical Report TR-86-06, Wang Institute, 1986.

[8] Nakamura, T. and Mizoguchi, T., An Analysis of Storage Utilization Factor in Block Split Data Structuring Scheme; Proc. 4th Int. Conf. on Very Large Data Bases, (Berlin, 1978), 489-495.

[9] Yao, A., On Random 2-3 Trees, *Acta Informatica* **9** (1978), 159-170.