UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT

Performance Analysis of a Single-File Version of Linear Hashing

Per-Ake Larson

CS-83-28

November, 1983

Performance analysis of a single-file version of linear hashing *

Per-Ake Larson

Data Structuring Group
Department of Computer Science
University of Waterloo
Waterloo, Ontario N2L 3G1
Canada

Technical Report CS-83-28

ABSTRACT

A performance analysis of a new variant of linear hashing with partial expansions is presented. In the new variant the overflow area is combined with the prime storage area in the same file. The performance measures considered are: expected length of successful and unsuccessful searches and cost of insertions. The new method uses several overflow chains per page. Increasing the number of chains significantly improves the retrieval performance.

1. Introduction

Linear hashing with partial expansions is a file organization primarily designed for files which grow and shrink dynamically [1]. It can accommodate any number of insertions and deletions, retaining good storage utilization and retrieval performance without periodic reorganization. It is a generalization of linear virtual hashing developed by Litwin [5]. We will use the term "linear hashing" as a generic term covering both methods.

Linear hashing was originally designed as a two-area technique, that is, in addition to the prime storage area there is a separate storage area for overflow records. This results in having two files that grow and shrink according to the number of records stored. A new version of linear hashing was presented in [3]. In the new version the overflow area is combined with the

* This work was supported by Natural Sciences and Engineering Research Council of Canada, Grant No. A2460.

prime storage area in the same file. Hence, there is only one file that grows and shrinks dynamically. By using several overflow chains per bucket, instead of just one as originally suggested [1,5], less overflow storage is required and the retrieval performance is improved.

This paper presents an analysis of the expected performance of the new version of linear hashing. The details of the new scheme have been presented elsewhere [3], but a brief outline is given in section two. In section three the overflow space requirements are analysed and the necessary formulas for modelling the expansion rate are derived. The next two sections are devoted to analysing the expected retrieval performance and the costs of insertions and deletions. The expected performance depends on the way overflow records are allocated to overflow pages. The analysis in sections four and five makes some very pessimistic assumptions, and therefore a more optimistic analysis is carried out in section six. Numerical results for different page sizes and storage utilization have been assembled into an appendix.

Linear hashing was originally developed by Litwin [5] and generalized to linear hashing with partial expansions by Larson [1]. Spiral storage, developed by Martin [6], is based on a different idea but achieves the same goal as linear hashing. The performance of linear hashing with partial expansions was analysed by Larson [2]. Two modifications to the original scheme have been presented by Ramamohanarao and Lloyd [8]. The first one simplifies the expansion process, and the second one is a very efficient, loopless address computation algorithm. The simplifications do not come without extra cost: additional buffer space is required when the file is expanded. Regnier [9] also designed a loopless address computation algorithm. It is more complicated than the one by Ramamohanarao and Lloyd, but does not require additional buffer space. Mullin [7] has presented a version without a separate overflow area. Chaining is used, but overflow records are stored in the prime storage area.

2. An outline of the new version

Linear hashing is a technique for incrementally expanding (and contracting) the primary storage area of a hash file. The basic step consists of expanding the file by one page and relocating some of the records from one or a few existing pages to the new page. This is done in a systematic manner in order to ensure that all relocated records can be retrieved without having to access any of the old pages.

The expansion process of linear hashing with two partial expansions is illustrated in fig. 1. We start from a file consisting of $2N$ pages, logically subdivided into N pairs of pages (groups of size two), where the pairs are $(j, N + j)$, $j = 0, 1, \ldots, N - 1$. When the file is to be expanded, this is done by first expanding group 0, then group 1, etc. by one page, see fig. 1(a). When expanding group j, approximately 1/3 of the records from page j and $N + j$ are moved to a new page $2N + j$. When the last pair, $(N - 1, 2N - 1)$, has been expanded the file has increased from $2N$ to $3N$ pages, and the first partial expansion is completed. Then the second

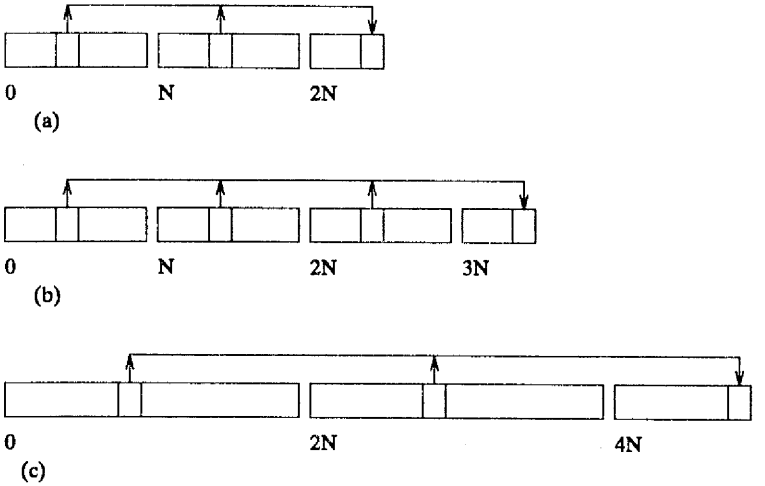


Fig. 1: Illustrating linear hashing with two partial expansions.

partial expansion starts, but now groups of three pages (j, $N + j$, $2N + j$), $j = 0,1,\ldots, N - 1$, are expanded to four pages, see fig. 1(b). When the second partial expansion has been completed the file size has doubled from $2N$ to $4N$. A doubling of the file size is called a full expansion. Now the situation is the same as the original one, the only difference being that the file size has doubled. The next partial expansion reverts to expanding groups of size two, (j, $2N + j$), $j = 0,1,\ldots, 2N - 1$, see fig. 1(c). The one after that will expand groups of size three, etc.

The above idea can easily be extended to an arbitrary number, n_0, of partial expansions per full expansion. Linear virtual hashing has $n_0 = 1$. To implement linear hashing with partial expansions an address computation algorithm is required. It must be able to compute the current home page of a record at any time. An address computation algorithm for arbitrary n_0 is given in [1]. A simpler algorithm for the case $n_0 = 1$ can be found in [5].

The scheme outlined above gives a method for expanding or contracting the file by one page at a time. In addition we also need rules for determining *when* to trigger an expansion or a contraction. Because a set of such rules controls the expansion and contraction rate, it is called a control function. There are many possible control functions, see [1,5,8], but we will here consider only the rule of constant storage utilization. According to this rule, the file is expanded whenever insertion of a record causes the storage utilization to rise above some threshold α, $0 < \alpha < 1$, selected by the user. This rule is optimal in a certain sense [1].

As linear hashing was originally designed, it was assumed that overflow records are handled by separate chaining using fairly large overflow pages. In other words, overflow records are stored by linking one or more overflow pages from a separate storage area to an overflowing primary page. Each overflowing primary page has its own, completely separate, chain of overflow pages. The size of overflow pages is one of the design parameters.

This method for handling overflow records has certain drawbacks, however. For a discussion, see [3]. Therefore, two modifications to this overflow handling scheme were proposed in [3]: to use not one, but several overflow chains per primary page, and to combine the overflow storage area and the primary storage area in the same file.

When several overflow chains per primary page are used, and an overflow record occurs, it is placed somewhere in an overflow page and linked into one of the chains emanating from its home page. The chain is selected by a hashing function. The function depends only on the key of the record, and distributes the overflow records uniformly between the overflow chains. To retrieve a record, the records in the home page are first checked, and then the records on the appropriate overflow chain. The chains are assumed to be separate (non-coalescing) with a resolution of one record, that is, the pointer points to the next *record* on the chain. In this way overflow records on completely different chains (from the same or different home pages) may share the same overflow page without being mixed up. The strategy for assigning overflow records to overflow pages will affect the performance. This question is discussed further in section six.

The idea for combining the primary storage area and the overflow storage area in the same file is very simple: every *rth* page in the file is designated as an overflow page, $r \geq 2$. Assuming that the page addresses start from 0, the overflow pages have addresses $r - 1$, $2r - 1$, $3r - 1$,.... Because the overflow area cannot grow independently of the primary area, a situation where all the overflow pages are full may occur. In such a situation we simply expand the file. This will normally decrease the number of overflow records, thereby freeing some overflow space. In any case, after expanding the file by at most $r - 1$ pages, a new overflow page will eventually be created.

3. Expansion rate and overflow space requirements

During a partial expansion a file using linear hashing consists essentially of two traditional hash files with different load factors. As records are inserted, the file will be expanded and the size of the two subfiles will change. The expansion rate depends not only on the number of records inserted, but also on the control function used. We will in this section derive formulas relating the total file size and the subfile size to the number of records stored.

First we have to introduce some notation. To facilitate comparisons we will, to the extent possible, use the same notation as in [2]. Let b, $b \geq 1$, denote the page size in number of records, and α, $0 < \alpha < 1$, the required storage utilization. The overflow storage factor f is defined as the fraction of

the total storage space reserved for overflow records. When every *rth* page is an overflow page the overflow storage factor is $f = 1/r$. The number of partial expansions per full expansion is denoted by n_0. The analysis in this and subsequent sections is asymptotic. The behavior of a large file is well approximated by an asymptotic model.

Consider first a traditional hash file where the records are uniformly distributed over the pages of the file. Denote the expected number of records hashing to a page by y. The load on the file is then characterized by the load factor z, defined as $z = y/b$. The probability that k records will hash to a page, given the load factor z, has a Poisson distribution:

$$P(k, z) = e^{-zb} (zb)^k / k! \quad , \quad k = 0,1,2,\ldots$$

The expected number of overflow records per bucket can be computed as

$$t(z) = \sum_{k=b+1}^{\infty} (k - b)P(k, z)$$

$$= b(z - 1) + \sum_{k=0}^{b} (b - k)P(k, z).$$

Consider a file where a partial expansion, increasing the size of each group from n to $n + 1$ buckets, $n_0 \le n < 2n_0$, has been partially completed. The expansion factor q is defined as $q = (n + 1)/n$. A fraction x, $0 \le x \le 1$, of the groups is assumed to have been expanded. Assume that the load factor of a bucket belonging to a group that has not yet been expanded by z, $z \ge 0$. Then the load factor of a bucket belonging to an expanded group is $zn/(n + 1) = z/q$. Note that the expected number of records per group is nzb, regardless of whether the group has been expanded or not. In this situation the expected number of overflow records per bucket is

$$T(x, z) = \frac{x(n+1)t(z/q) + (1-x)n\, t(z)}{x(n+1) + (1-x)n}$$

$$= \frac{xqt(z/q) + (1-x)t(z)}{1 + x(q-1)}$$

The formula is derived as follows. An expanded group is expected to have $(n + 1)t(z/q)$, and an unexpanded group $nt(z)$ overflow records. The fraction of expanded groups is x and the fraction of unexpanded groups hence $1 - x$. The average number of overflow records per group is then $x(n + 1)t(z/q) + (1 - x)nt(z)$. The (average) number of buckets per group is $x(n + 1) + (1 - x)n$. Dividing the number of overflow records per group by the number of buckets per group gives the above formula.

In the above formula there are two free variables: the fraction of expanded groups x, and the load factor z. However, the fraction of expanded groups depends on the load factor where the relationship is defined by the control function. We will assume that the following, modified rule of

constant storage utilization is used:

a) when there is sufficient overflow space, the file is expanded whenever the overall storage utilization rises above α,

b) if a record cannot be stored due to lack of free overflow space, the file is (prematurely) expanded (even though this results in a storage utilization less than α).

At any point of an expansion the expansion rate is determined by one of these two rules. If rule a) applies we say that the file is *load-controlled,* and if rule b) applies we say that is is *overflow-controlled.*

Let us first study case a), that is, we assume that the overflow storage factor f is large enough so that the expansion rate is completely load-controlled during the current partial expansion. In a situation when a fraction x of the groups has been expanded, the number of records per group is znb while the amount of space per group (including overflow pages) is $(x(n + 1) + (1 - x)n)b/(1 - f)$. The value of x must then be such that the following equality holds

$$znb = \alpha\,(x(n + 1) + (1 - x)n)b/(1 - f).$$

Solving for x gives us the following relation between x and the load factor:

$$x = g_a(z) = (z(1 - f)/\alpha - 1)/(q - 1).$$

We will need this function frequently in the sequel. It will be denoted by $g_a(z)$ and its inverse by $g_a^{-1}(x)$. In particular there are two values of g_a^{-1} that are of interest: $g_a^{-1}(0)$ and $g_a^{-1}(1)$. The first value is the load factor required at the beginning of the expansion and the second value is the load factor at the end of an expansion. They are easily found to be $g_a^{-1}(0) = \alpha/(1 - f)$ and $g_a^{-1}(1) = \alpha q/(1 - f)$.

Let us now analyse the situation when the expansion rate is governed by lack of overflow space. When a fraction x of the groups has been expanded, the expected number of overflow records per group is $x(n + 1)t(z/q) + (1 - x)nt(z)$, see above. On the other hand, the amount of overflow space per group is $(x(n + 1) + (1 - x)n)bf/(1 - f)$. It can be shown (by a similar argument as in [2]), that in the asymptotic case the overflow requirements per group will exactly match the available overflow storage space per group. (This is only approximately true for a finite file.) Consequently the value of x must be such that the following equation is satisfied:

$$x(n + 1)t(z/q) + (1 - x)nt(z) = (x(n + 1) + (1 - x)n)bf/(1 - f)$$

Again, solving for x gives us the following function

$$x = g_b(z) = \frac{d - t(z)}{qt(z/q) - t(z) - (q - 1)d} \qquad (1)$$

where $d = bf/(1 - f)$. This function will be denoted by $g_b(z)$. The restriction of g_b to $0 \le g_b(z) \le 1$ is invertible and the inverse function will be

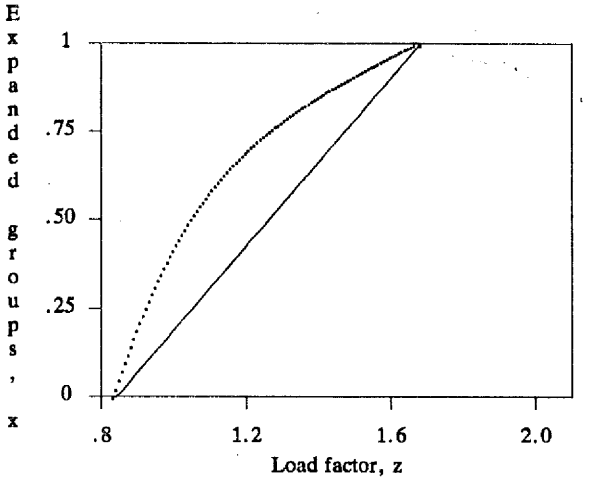
denoted by $g_b^{-1}(x)$, $0 \leq x \leq 1$. The inverse function is not available in explicit form, but it is easily computed numerically. We will in this case also need the two values $g_b^{-1}(0)$ and $g_b^{-1}(1)$, but this time they have to be computed numerically. It is easily proved from eq. (1) that $g_b^{-1} = qg_b^{-1}(0)$, so in fact we only have to compute $g_b^{-1}(0)$.

At any point the size of the file (fraction of expanded groups) will be determined by the more restrictive one of rules a) and b). Hence the combined control function is modelled by

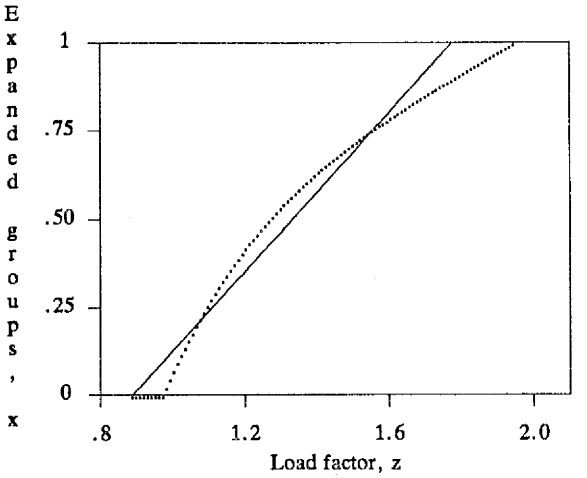
$$g(z) = \max(g_a(z), g_b(z)), \quad z_0 \leq z \leq gz_0$$

$$z_0 = \min(g_a^{-1}(0), g_b^{-1}(0))$$

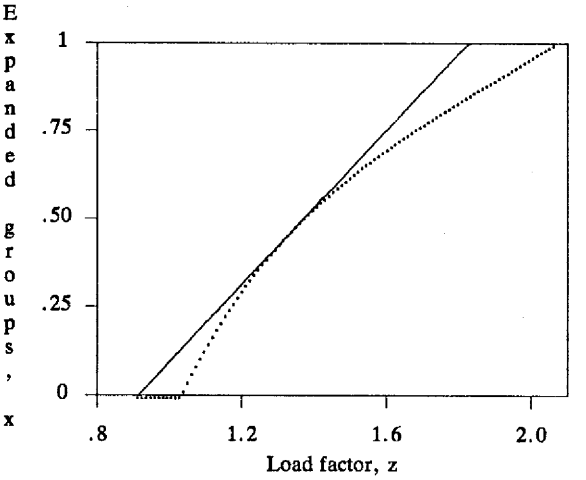
The behaviour of the two functions $g_a(z)$ and $g_b(z)$ is illustrated in Fig. 2. The fraction of expanded groups, x, is plotted as a function of the load factor for non-expanded groups. The parameters are: bucket size $b = 10$, storage utilization $\alpha = 0.80$ and group size $n = 1$. In Fig. 2(a) the overflow storage factor is $f = 1/20 = 0.05$ in Fig. 2(b) $f = 1/10 = 0.10$ and in Fig. 2(b) $f = 1/8 = 0.125$. The straight line is $g_a(z)$ and the dotted line is $g_b(z)$. Fig. 2(a) shows a situation where the expansion rate is fully overflow-controlled; the target storage utilization of $\alpha = 0.80$ can never be achieved due to insufficient space for overflow records. In Fig. 2(b) a mixed situation is illustrated. At the beginning and the end of the expansion the expansion rate is load-controlled. In the middle there is not enough overflow space and the expansion rate is overflow-controlled. In Fig. 2(c) there is sufficient space for overflow records throughout the expansion.



(a) f=0.05



(b) f=0.10



(c) f=0.125

Fig. 2: Fraction of expanded groups as a function of the load factor ($b = 10$, $\alpha = 0.8$).

We can now easily compute the utilization of the available overflow storage space at any point of a partial expansion with expansion factor q. Given that the load factor of a page belonging to an unexpanded group is z, $z_0 \leq z \leq qz_0$, a fraction $x = g(z)$ of the groups will have been expanded. The expected number of overflow records per group is $x(n + 1)t(z/q) + (1 - x)nt(z)$ and the available overflow space per group is $(x(n + 1) + (1 - x)n)bf/(1 - f)$, see above. Dividing and inserting $x = g(z)$ then gives the following formula for the utilization of the available overflow space

$$V(z, g(z)) = \frac{g(z)q\ t(z/q) + (1 - g(z))t(z)}{(1 + g(z)(q - 1))bf/(1 - f)}$$

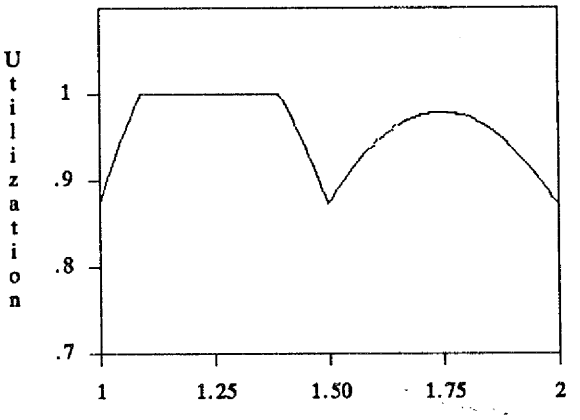


Fig. 3: Utilization of available overflow storage space
($b = 10$, $\alpha = 0.85$, $m = 5$, $f = 0.1$, $n_0 = 2$).

In fig. 3 the utilization of the available overflow storage space is illustrated for an example file. During the first partial expansion lack of overflow space occurs and for a while the expansion rate is overflow-controlled. How this affects the insertion costs is shown in section five. During the second partial expansion there is sufficient overflow space available and the expansion rate is load-controlled throughout.

In practice one would probably want to set the overflow storage factor high enough so that the risk of running out of overflow storage during an expansion is minimal. This is clearly not required, however. The only effect of running out of overflow storage space is that the file will be prematurely expanded and the storage utilization will drop below the desired level. On the other hand, an unnecessarily high overflow storage factor reduces the

fraction of the total storage space available for primary pages. This in turn increases the load on the primary pages, creating more overflow records and slowing down retrieval. In effect, part of the available storage space is not efficiently utilized, thereby increasing the pressure on the remaining area. Some guidelines for setting the overflow storage factor are needed. To this end we will analyse the following problem: given the page size (b), desired storage utilization (α) and number of partial expansions (n_0), what is the lowest possible overflow storage factor (f) such that we are not expected to run out of overflow space at any point of an expansion?

When the expansion rate is load-controlled the expected number of overflow records per page at point z, $z_0 \le z \le qz_0$ of an expansion is

$$\frac{g_a(z)qt(z/q) + (1 - g_a(z))t(z)}{1 + g_a(z)(q - 1)}.$$

The available overflow storage per page is $fb/(1 - f)$. To not run out of overflow space at any point of an expansion, the maximum requirements must be less than or equal to the available storage. Hence the lowest overflow storage factor for a partial expansion with expansion factor q is given by the solution to the equation

$$\frac{fb}{1 - f} = \max_{z_0 \le z \le z_1} \frac{g_a(z)qt(z/q) + (1 - g_a(z))t(z)}{1 + g_a(z)(q - 1)}, \tag{2}$$

where $g_a(z) = (z(1 - f)/\alpha - 1)/(q - 1)$, $z_0 = \alpha/(1 - f)$ and $z_1 = qz_0$. This equation can easily be solved numerically for given parameters b, α and q. Among the n_0 partial expansions in a full expansion, the highest overflow requirements occur during the first partial expansion (most uneven load), and therefore it is sufficient to solve the above equation only for $q = (n_0 + 1)/n_0$.

The lowest overflow storage factor for different parameters combinations are listed in Table 1 in the appendix. In practice we must distribute the overflow pages evenly over the file, allocating every *rth* page as an overflow page. The best (highest) value for r is then given by $r = \lfloor 1/f \rfloor$ where f is found in Table 1 or by solving eq. (2). For our example file ($b = 10$, $\alpha = 0.85$, $n_0 = 2$) the minimum overflow storage factor is $f = 0.1128$ which gives $r = \lfloor 1/0.1128 \rfloor = 8$.

4. Retrieval performance

In this section we study the retrieval performance of the new scheme. Formulas for computing the expected length of successful and unsuccessful searches at any point of an expansion, as well as the average over a full expansion, are derived.

The search lengths, and the insertion costs in the next section, are derived under a very pessimistic assumption: all overflow records from the same home page are assumed to reside on different overflow pages. This means that fetching the next record on an overflow chain always requires one read access. This is overly pessimistic; in practice one would strive to place records from the same chain on the same page. A more optimistic analysis is

done in section six.

We begin by deriving two basic formulas. Let $u(z)$ denote the expected length of an unsuccessful search for a traditional hash file, when m overflow chains per page are used and the load factor is z. This can be computed as

$$u(z) = 1 + \frac{1}{m} \sum_{k=b+1}^{\infty} (k - b)P(k, z)$$

$$= 1 + \frac{1}{m} \left(b(z - 1) + \sum_{k=0}^{b} (b - k)P(k, z) \right) \tag{3}$$

The infinite sum is the expected number of overflow records per bucket. Dividing this by m gives the expected length of a chain, because the overflow records are uniformly distributed over the m chains emanating from a primary page.

Let $s(z)$ denote the expected length of a successful search in the corresponding situation. We will derive it using the same approach as in [4]. When the file is loaded the load factor gradually rises from 0 to z. Consider a short load factor interval $(t, t + dt)$, $0 \le t \le z$. When inserting a record during $(t, t + dt)$ the probability of hitting a full page is

$$P_b(t) = 1 - \sum_{k=0}^{b-1} P(k, t).$$

The expected length of an overflow chain for a full page (as opposed to any page) is $(u(t) - 1)/P_b(t)$, because all overflow records belong to full pages, and the fraction of full pages is $P_b(t)$. The new record is effectively added to the end of the chain and will require one access more to retrieve. The expected number of accesses to retrieve a record inserted during $(t, t + dt)$ is then $1 + P_b(t)(1 + (u(t) - 1)/P_b(t))$. It is assumed that each record is equally likely to be retrieved, regardless of when it was inserted. The expected length of a successful search can then be computed as

$$s(z) = \frac{1}{z} \int_0^z \left\{ 1 + P_b(t) \left(1 + \frac{u(t) - 1}{P_b(t)} \right) \right\} dt$$

$$= 1 + \frac{1}{z} \int_0^z P_b(t)dt + \frac{1}{z} \int_0^z (u(t) - 1)dt$$

The integrals can actually be solved, and after some manipulation we obtain

$$s(z) = 2 + \frac{1}{zb} \sum_{k=0}^{b-1} P(k, z)(b - k)\left(1 - \frac{b - k + 1}{2m}\right) \tag{4}$$

$$- \frac{1}{z} + \frac{bz}{2m} - \frac{b}{m} + \frac{b + 1}{2mz}$$

We can now compute the expected search lengths at an arbitrary point of an expansion with expansion factor $q = (n + 1)/n$. When the load factor of a page belonging to a nonexpanded group is z, $z_0 \le z \le qz_0$, a fraction

$x = g(z)$ of the groups will have been expanded. Note that the expected number of records per group is nzb regardless of whether the group has been expanded or not. The expected number of accesses for an unsuccessful search in this situation can be computed as

$$U(z, g(z)) = g(z)u(z/q) + (1 - g(z))u(z).$$

The formula can be understood as follows. The probability that an unsuccessful search will hit a certain group is the same for all groups. The probability of hitting an expanded group is $g(z)$ and that of hitting a nonexpanded group is $1 - g(z)$. An unsuccessful search starting from a page belonging to an expanded group requires $u(z/q)$ accesses on average. If the search starts from a page belonging to a nonexpanded group the average is $u(z)$. By a similar reasoning the expected number of accesses for a successful search is found to be

$$S(z, g(z)) = g(z)s(z/q) + (1 - g(z))s(z).$$

The final task in this section is to derive formulas for computing the average search lengths over a full expansion. A full expansion, increasing the size of each group from n_0 to $2n_0$, consists of n_0 partial expansions. The *ith* expansion increases the group size from $n_0 + i - 1$, $i = 1,2,\ldots, n_0$, and has a expansion factor of $q_i = (n_0 + i)/(n_0 + i - 1)$. The average search lengths can be computed as

$$\bar{\bar{U}} = \frac{1}{n_0} \sum_{i=1}^{n_0} \frac{1}{(q_i - 1)z_0} \int_{z_0}^{q_i z_0} U_i(z, g(z))dz \tag{5}$$

$$\bar{\bar{S}} = \frac{1}{n_0} \sum_{i=1}^{n_0} \frac{1}{(q_i - 1)z_0} \int_{z_0}^{q_i z_0} S_i(z, g(z))dz \tag{6}$$

The notation $U_i(z, g(z))$ means the function $S(z, g(z))$ with the value q_i for the parameter q, and correspondingly for S_i. The *ith* term in each sum represents the average for the *ith* partial expansion within a full expansion. Each partial expansion is given the same weight in the sum, because they are all of the same duration whether measured in number of records added or number of pages added.

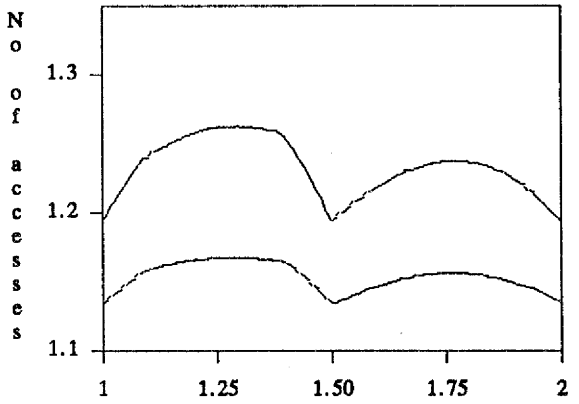


Fig. 4: Expected number of accesses for a successful (lower) and unsuccesful (upper) searches
($b = 10$, $\alpha = 0.85$, $m = 5$, $f = 0.1$, $n_0 = 2$).

The development of the expected search lengths for our example file over a full expansion is shown in Fig. 4. For a successful search the expected number of accesses ranges from 1.154 to 1.167 with an average of 1.154. For unsuccessful searches the range is 1.194 to 1.263 with an average of 1.233.

5. Insertion costs

The total cost of inserting a record consists of two components: the cost of inserting the record and the additional cost of expanding the file when necessary. The cost caused by file expansions depends on the expansion rate and the cost of actually carrying out the expansion.

Let us first derive the cost of inserting a record into a traditional hash file with load factor z. The first part of an insertion is an unsuccessful search with a cost of $u(z)$. If we hit a non-full page, the record is added to the home page and the page is rewritten. The probability of this occurring is $1 - P_b(z) = \sum_{k=0}^{b-1} P(k, z)$. If we hit a full (or over-full) page three accesses are required: one for reading a non-full overflow page, one for rewriting it, and one for updating the pointer from the preceding record and rewriting that page. This requires buffer space for two pages. In accordance with our pessimistic assumption we have assumed that all overflow records reside on different pages. We have also assumed that locating a non-full overflow page does not involve any external accesses, see [3]. Addition of these three terms results in

$$a(z) = u(z) + 1 - P_b(z) + 3P_b(z)$$

$$= 1 + u(z) + 2P_b(z)$$

The expected cost of inserting a record during a partial expansion and the average over a full expansion can be derived in exactly the same way as for an unsuccessful search. Note that the following formulas do not include any expansion costs:

$$A(z, g(z)) = g(z)a(z/q) + (1 - q(z))a(z)$$

$$\bar{A} = \frac{1}{n_0} \sum_{i=1}^{n_0} \frac{1}{(q_i - 1)z_0} \int_{z_0}^{q_i z_0} A_i(z, g(z))dz$$

An insertion will occasionally trigger an expansion of the file. The expected expansion cost per inserted record is the product of two factors: the expected number of access required to carry out an expansion and the expansion activity, expressed in number of expansions per inserted record. They both depend on the current state of the file, that is, load factor and expansion factor.

The function $(q - 1)x = g(z)/n$, $z_0 \le z \le qz_0$, represents the growth of the file (except for overflow pages) since the beginning of the current partial expansion. The rate of growth of point z measures the expansion activity at that point. The rate of growth as a function of the expected number of records per bucket, not the load factor, is needed, so we set $zb = y$. This gives us the function $g(y/b)/n$. The expected number of expansion operations, or non-overflow pages added, caused by inserting a record when the load factor is $z = y/b$, can then be computed as

$$F(z) = \frac{d(g(y/b)/n)}{dy} = \frac{g'(y/b)}{nb} = \frac{g'(z)}{nb}$$

where $g'(z)$ is the derivative of g with respect to z. By differentiation of g_a and g_b we obtain

$$g'(z) = \begin{cases} \dfrac{1 - f}{\alpha(q - 1)} & \text{for } g_a(z) > g_b(z) \\ \dfrac{(1 - g_b(z))t'(z) + t'(z/q)}{c + t(z) - q\,t(z/q)} & \text{for } g_a(z) < g_b(z) \end{cases}$$

where $c = \dfrac{(q - 1)fb}{1 - f}$ and $t'(x) = b(1 - P_b(x))$.

The expected number of accesses to expand a group from n to $n + 1$ pages is the sum of the following components:

a) reading n pages plus all associated overflow records: $n(1 + t(z))$

b) rewriting the n pages and associated overflow pages: $n(1 + t(z))$. We assume that the overflow records present after the expansion are placed in the "old" overflow pages.

c) writing the new page: 1

d) writing a new overflow page. The cost of this per expansion is $f/(1-f)$

Note that we in c) and d) have assumed that a new page can be written directly without reading it first. Adding up gives us

$$2n(1 + t(z)) + 1 + \frac{f}{1-f} = n\left(2 + 2t(z) + \frac{q-1}{1-f}\right)$$

Multiplying this by $F(z)$ then gives us the expected expansion cost per insertion:

$$E(z, g(z)) = \frac{g'(z)}{b}\left(2 + 2t(z) + \frac{q-1}{1-f}\right)$$

The average over a full expansion is computed in the usual way

$$\bar{E} = \frac{1}{n_0}\sum_{i=1}^{n_0} \frac{1}{(q_i - 1)z_0} \int_{z_0}^{q_i z_0} E_i(z_1 g(z))dz$$

The total cost of an insertion is the sum of A and E, that is, $W(z, g(z)) = A(z, g(z)) + E(z, g(z))$, $\bar{W} = \bar{A} + \bar{E}$.

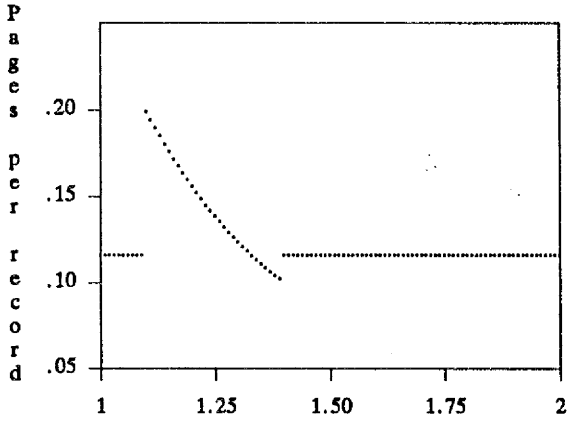


Fig. 5: Expected number of (non-overflow) pages added per record inserted ($b = 10$, $\alpha = 0.85$, $m = 5$, $f = 0.1$, $n_0 = 2$).

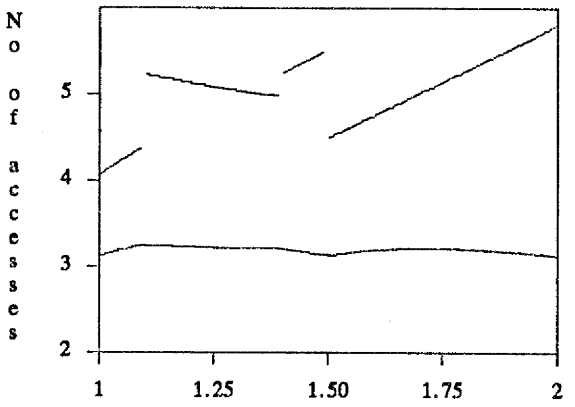


Fig. 6: Expected number of accesses for insertion of a record ($b = 10$, $\alpha = 0.85$, $m = 5$, $= 0.1$, $n_0 = 2$).

Fig. 5 shows the expansion rate (number of non-overflow pages added per record inserted) for our example file. When the file has been expanded to 1.09 times its original size, lack of overflow storage occurs. The expansion rate is now overflow-controlled. It suddenly increases from 0.118 to 0.203 pages per record and then slowly decreases. When the file has reached 1.40 times its original size there is again sufficient overflow storage and from that point on the expansion rate is storage-controlled.

Fig. 6 shows the cost of inserting a record. The lower line does not include the cost of expanding the file. The switch from storage-controlled to overflow-controlled expansion causes a higher expansion rate and hence higher insertion costs. The average over a full expansion is 5.080 accesses, of which 1.880 accesses are due to file expansions. The minimum is 4.088 and the maximum 5.812.

6. Optimistic analysis

So far we have assumed that overflow records on the same overflow chain all reside on different pages. This is overly pessimistic. One can envision an insertion scheme that strives to keep overflow records from the same home page stored on one or a few pages. This would improve the retrieval speed. The net effect on the total cost of insertions is unclear. On one hand, fewer accesses would be required for scanning down an overflow chain during insertion and for collecting overflow records during file expansions. On the other hand, it may be necessary to rearrange records during an insertion in order to store a new overflow record on the same page as existing overflow records (from the same home page). This requires additional accesses. However, to achieve optimal retrieval performance it is sufficient

to keep all records on the same overflow chain stored on one page. This may be easier than keeping all overflow records from the same *home page* stored on one page.

Designing an efficient clustering scheme is an open problem. Nevertheless, we can still find a lower bound on the expected retrieval performance achievable by any such scheme. Whatever the scheme is, fetching an overflow record, or checking all the records on an overflow chain will always require at least one additional access. We will compute the expected search lengths under this assumption.

A formula for the expected length of a successful search is easily found by letting $m \to \infty$ in eq. (4). Increasing the number of chains does not affect the number of overflow records. It merely decreases the length of each chain to the point where any overflow record can be retrieved in one extra access. Hence we have

$$s'(z) = 2 - \frac{1}{z} + \frac{1}{zb} \sum_{k=0}^{b-1} P(k, z)(b - k)$$

The formula for unsuccessful search cannot be derived in the same way. When $m \to \infty$ the expected length in eq (3) approaches one because the probability of hitting an empty chain converges to one. A more direct approach is called for. Assuming that retrieval of all the records on an overflow chain requires only one extra access we obtain the following formula

$$u'(z) = \sum_{k=0}^{b} P(k, z) + 2 \sum_{k=b+1}^{\infty} P(k, z) - \sum_{k=b+1}^{\infty} P(k, z)(1 - \frac{1}{m})^{k-b}$$

If we hit a page to which at most b records have hashed only one access is required; this is the first term. If we hit a page to which more than b records have hashed two accesses are required; this is the second term. However, if the overflow chain of interest is empty only one access is required; this is the third term. The above formula can be simplified to

$$u'(z) = 2 - h_b(zb) - (\frac{m}{m-1})^b e^{\frac{-zb}{m}} (1 - h_b(zb(m - 1)/m))$$

where $h_b(x) = e^{-x} \sum_{k=0}^{b} x^k/k!$.

By inserting $s'(z)$ and $u'(z)$ in eqs. (5) and (6) we get lower bounds on the corresponding expected search lengths. The difference between the expected search lengths under the pessimistic assumption and the corresponding lower bounds gives an indication of how much there is to be gained by a clustering scheme. For our example file we have the following differences between the averages

	Successful search	Unsuccessful search
Pessimistic	1.154	1.233
Optimistic	1.113	1.168
Difference	0.041	0.065

For this particular parameter configuration the difference is less than 0.1 accesses. This is due to the fact that the overflow chains are expected to be short. With a page size of 10 the number of overflow records per page is quite low. If the page size is increased the number of overflow records *per page* increases, but the total number of overflow records decreases. This means that we can allocate less overflow space, thus decreasing the pressure on the primary pages and decreasing the number of overflow records. If we, for our example file, increase the page size from 10 to 40 records we can decrease the overflow storage factor from 0.1 to 0.05. In this case we obtain

	Successful search	Unsuccessful search
Pessimistic	1.080	1.362
Optimistic	1.043	1.185
Difference	0.037	0.177

Further numerical results are given in the appendix, Tables 2 - 9. It seems that the potential for improving the retrieval speed by clustering is quite limited. Some simple, low-cost scheme may give an overall gain, but more elaborate schemes are likely to be too costly compared with the gain. Clustering may also reduce the cost of expanding the file, but to what extent has not been analysed.

7. Discussion

The simplest way of speeding up retrieval is to increase the number of overflow chains. The cost is modest: additional pointer space on each page. The effect is most dramatic on the length of unsuccessful searches, but it also significantly speeds up both successful searches and insertions. This is illustrated by the results in Table 1 and 2 below.

m	Successful search	Unsuccessful search	Insertion
1	1.314	2.165	6.012
2	1.214	1.583	5.429
3	1.180	1.388	5.235
4	1.164	1.291	5.138
5	1.154	1.233	5.080
10	1.133	1.117	4.963

Table 1: Effect of increasing the number of overflow chains per page ($b = 10$, $\alpha = 0.85$, $f = 0.1$, $n_0 = 2$).

m	Successful search	Unsuccessful search	Insertion
1	1.229	2.811	5.318
2	1.136	1.906	4.413
3	1.105	1.604	4.411
4	1.090	1.453	3.960
5	1.080	1.362	3.869
10	1.062	1.181	3.688

Table 2: Effect of increasing the number of overflow chains per page ($b = 40$, $\alpha = 0.85$, $f = 0.05$, $n_0 = 2$).

As discussed above the performance could be further improved by striving to cluster overflow records from the same home page on the same overflow page. The potential for improving the retrieval performance by clustering is quite limited, however, but it may reduce the cost of expanding the file.

A fraction of the total file space must be reserved for overflow records. This fraction should be as low as possible to reduce the number of overflow records. However, if it is set lower than a certain limit the target storage utilization cannot be achieved. This limit depends on the page size and the target storage utilization, see Table 1 in the appendix.

References

1. Larson, P.-A. Linear hashing with partial expansions. In Proc. 6th Conf. Very Large Data Bases (Montreal, Canada), ACM, New York, 1980, pp. 224-232.

2. Larson, P.-A. Performance analysis of linear hashing with partial

expansions. ACM Trans. Database Syst., 7, 4 (1982), pp. 566-587.

3. Larson, P.-A. A single-file version of linear hashing with partial expansions. In Proc. 8th Conf. Very Large Data Bases (Mexico City, Mexico), VLDB Endowment, California, 1982, pp. 300-309.
4. Larson, P.-A. Analysis of hashing with chaining in the prime area. J. of Algorithms (to appear).
5. Litwin, W. Linear hashing: A new tool for files and tables addressing. In Proc. 6th Conf. Very Large Data Bases (Montreal, Canada), ACM, New York, 1980, pp. 212-223.
6. Martin, G.N.N. Spiral storage: Incrementally augmentable hash addressed storage. Theory of Computation Rep. 27, Univ. of Warwick, England, 1979.
7. Mullin, J.R. Tightly controlled linear hashing without separate overflow storage. BIT, 21, 4 (1981), pp. 389-400.
8. Ramamohanarao K. and Lloyd J.K. Dynamic hashing schemes. The Computer J., 25, 4 (1982), pp. 478-485.
9. Regnier, M. Linear hashing with groups of reorganization: An algorithm for files without history. In Sheuermann P. (ed.): Improving Database Usability and Responsiveness, Academic Press, New York, 1982, pp. 257-272.

Appendix: Numerical results for page size 5, 10, 20 and 40 records

N_0	Page size	Storage utilization					
		0.70	0.75	0.80	0.85	0.90	0.95
1	5	0.0980	0.1293	0.1688	0.2198	0.2885	0.3919
	10	0.0661	0.0927	0.1275	0.1737	0.2375	0.3354
	20	0.0506	0.0751	0.1077	0.1513	0.2115	0.3031
	40	0.0444	0.0687	0.1012	0.1441	0.2022	0.2883
2	5	0.0719	0.0972	0.1300	0.1734	0.2336	0.3276
	10	0.0359	0.0535	0.0781	0.1128	0.1638	0.2480
	20	0.0176	0.0298	0.0483	0.0762	0.1193	0.1938
	40	0.0093	0.0184	0.0334	0.0573	0.0957	0.1632
3	5	0.0655	0.0891	0.1200	0.1611	0.2186	0.3090
	10	0.0289	0.0442	0.0658	0.0968	0.1433	0.2215
	20	0.0112	0.0201	0.0342	0.0566	0.0928	0.1584
	40	0.0040	0.0089	0.0181	0.0344	0.0631	0.1186

Table 1: Lowest overflow storage factor for which lack of overflow storage is not expected to occur.

N_0	α	f	Succ. search		Unsucc. search		Insertion	
			avg.	max.	avg.	max.	avg.	max.
1	0.70	1/10	1.153	1.176	1.125	1.154	4.498	5.096
	0.75	1/7	1.202	1.228	1.179	1.214	4.805	5.484
	0.80	1/5	1.272	1.301	1.260	1.305	5.199	5.962
	0.85	1/4	1.350	1.383	1.359	1.413	5.596	6.423
	0.90	1/3	1.484	1.519	1.543	1.608	6.176	7.062
	0.95	1/2	1.830	1.872	2.081	2.165	7.316	8.200
2	0.70	1/13	1.113	1.123	1.079	1.091	5.253	6.035
	0.75	1/10	1.145	1.157	1.109	1.124	5.493	6.357
	0.80	1/7	1.194	1.208	1.158	1.177	5.859	6.834
	0.85	1/5	1.263	1.278	1.235	1.259	6.361	7.469
	0.90	1/4	1.342	1.359	1.331	1.359	6.885	8.115
	0.95	1/3	1.431	1.449	1.450	1.482	7.429	8.765
3	0.70	1/15	1.104	1.109	1.070	1.076	6.225	7.316
	0.75	1/11	1.135	1.142	1.098	1.106	6.461	7.633
	0.80	1/8	1.177	1.184	1.139	1.148	6.809	8.088
	0.85	1/6	1.231	1.240	1.196	1.209	7.273	9.683
	0.90	1/4	1.332	1.342	1.316	1.332	8.101	9.722
	0.95	1/3	1.466	1.477	1.494	1.513	9.070	10.910

Table 2: Expected search lengths and total insertion costs for $b = 5$, $m = 5$ (pessimistic analysis).

N_0	α	f	Succ. search		Unsucc. search		Insertion	
			avg.	max.	avg.	max	avg.	max
1	0.70	1/15	1.102	1.134	1.156	1.214	3.805	4.286
	0.75	1/10	1.148	1.186	1.235	1.310	4.185	4.743
	0.80	1/7	1.212	1.258	1.351	1.446	4.653	5.297
	0.85	1/5	1.309	1.362	1.532	1.644	5.246	5.982
	0.90	1/4	1.425	1.483	1.754	1.880	5.838	6.639
	0.95	1/2	2.115	2.195	3.133	3.295	8.214	8.943
2	0.70	1/27	1.053	1.065	1.069	1.091	3.847	4.314
	0.75	1/18	1.078	1.094	1.108	1.137	4.155	4.725
	0.80	1/12	1.115	1.135	1.168	1.205	4.578	5.276
	0.85	1/8	1.172	1.196	1.266	1.313	5.161	6.019
	0.90	1/6	1.247	1.274	1.400	1.456	5.820	6.833
	0.95	1/4	1.396	1.427	1.683	1.749	6.893	8.105
3	0.70	1/34	1.045	1.051	1.056	1.066	4.249	4.846
	0.75	1/23	1.066	1.074	1.087	1.101	4.571	5.228
	0.80	1/15	1.097	1.108	1.136	1.155	4.952	5.772
	0.85	1/10	1.144	1.156	1.213	1.237	5.529	6.514
	0.90	1/6	1.232	1.247	1.369	1.401	6.503	7.738
	0.95	1/4	1.379	1.396	1.647	1.684	7.816	9.337

Table 3: Expected search lengths and total insertion costs for $b = 10$, $m = 5$ (pessimistic analysis).

N_0	α	f	Succ. search		Unsucc. search		Insertion	
			avg.	max.	avg.	max.	avg.	max.
1	0.70	1/19	1.080	1.124	1.217	1.343	3.452	3.863
	0.75	1/13	1.126	1.184	1.341	1.507	3.885	4.295
	0.80	1/9	1.194	1.267	1.527	1.734	4.437	4.837
	0.85	1/6	1.312	1.402	1.842	2.089	5.221	5.725
	0.90	1/4	1.533	1.640	2.422	2.693	6.383	7.017
	0.95	1/3	1.867	1.990	3.268	3.539	7.737	8.386
2	0.70	1/56	1.023	1.036	1.056	1.092	3.002	3.281
	0.75	1/33	1.040	1.059	1.101	1.156	3.307	3.692
	0.80	1/20	1.068	1.095	1.178	1.255	3.753	4.242
	0.85	1/13	1.113	1.148	1.304	1.403	4.361	4.993
	0.90	1/8	1.198	1.242	1.539	1.663	5.281	6.135
	0.95	1/5	1.365	1.418	2.003	2.142	6.644	7.736
3	0.70	1/89	1.015	1.021	1.036	1.053	3.102	3.386
	0.75	1/49	1.028	1.037	1.069	1.094	3.364	3.732
	0.80	1/29	1.049	1.062	1.125	1.162	3.765	4.259
	0.85	1/17	1.085	1.102	1.223	1.274	4.375	5.047
	0.90	1/10	1.151	1.174	1.407	1.473	5.313	6.234
	0.95	1/6	1.283	1.311	1.778	1.855	6.763	8.001

Table 4: Expected search lengths and total insertion costs for $b = 20$, $m = 5$ (pessimistic analysis).

N_0	α	f	Succ. search		Unsucc. search		Insertion	
			avg.	max.	avg.	max.	avg.	max.
1	0.70	1/22	1.078	1.143	1.336	1.613	3.346	3.962
	0.75	1/14	1.137	1.231	1.571	1.947	3.913	4.558
	0.80	1/9	1.238	1.367	1.949	2.428	4.706	5.311
	0.85	1/6	1.416	1.583	2.574	3.133	5.843	6.324
	0.90	1/4	1.774	1.982	3.746	4.314	7.657	8.118
	0.95	1/3	2.350	2.592	5.472	5.985	9.919	10.568
2	0.70	1/107	1.009	1.022	1.043	1.101	2.526	2.817
	0.75	1/54	1.021	1.043	1.096	1.198	2.802	3.208
	0.80	1/29	1.044	1.080	1.200	1.360	3.260	3.744
	0.85	1/17	1.088	1.141	1.394	1.620	3.968	4.475
	0.90	1/10	1.178	1.252	1.776	2.061	5.087	5.735
	0.95	1/6	1.377	1.472	2.564	2.867	6.849	7.792
3	0.70	1/250	1.004	1.008	1.018	1.039	2.494	2.599
	0.75	1/112	1.010	1.019	1.046	1.086	2.676	2.894
	0.80	1/55	1.023	1.038	1.105	1.175	3.020	3.366
	0.85	1/29	1.049	1.073	1.225	1.333	3.608	4.081
	0.90	1/15	1.106	1.141	1.483	1.632	4.627	5.303
	0.95	1/8	1.244	1.292	2.070	2.245	6.388	7.441

Table 5: Expected search lengths and total insertion cost for $b = 40$, $m = 5$ (pessimistic analysis).

N_0	α	f	Successful		Unsuccessful	
			avg.	max.	avg.	max.
1	0.70	1/10	1.125	1.141	1.100	1.120
	0.75	1/7	1.161	1.178	1.138	1.161
	0.80	1/5	1.208	1.225	1.192	1.219
	0.85	1/4	1.258	1.275	1.253	1.282
	0.90	1/3	1.334	1.349	1.354	1.382
	0.95	1/2	1.490	1.497	1.578	1.596
2	0.70	1/13	1.096	1.104	1.067	1.076
	0.75	1/10	1.122	1.131	1.091	1.102
	0.80	1/7	1.159	1.168	1.128	1.141
	0.85	1/5	1.208	1.217	1.183	1.197
	0.90	1/4	1.261	1.269	1.248	1.262
	0.95	1/3	1.342	1.349	1.357	1.370
3	0.70	1/15	1.090	1.094	1.060	1.065
	0.75	1/11	1.115	1.119	1.083	1.088
	0.80	1/8	1.147	1.152	1.114	1.121
	0.85	1/6	1.187	1.192	1.157	1.165
	0.90	1/4	1.257	1.261	1.240	1.248
	0.95	1/3	1.339	1.343	1.350	1.357

Table 6: Expected search lengths for $b = 5$, $m = 5$ (pessimistic analysis).

N_0	α	f	Successful		Unsuccessful	
			avg.	max.	avg.	max.
1	0.70	1/15	1.075	1.095	1.110	1.146
	0.75	1/10	1.104	1.126	1.158	1.201
	0.80	1/7	1.142	1.166	1.222	1.270
	0.85	1/5	1.193	1.218	1.311	1.361
	0.90	1/4	1.249	1.271	1.406	1.453
	0.95	1/2	1.480	1.486	1.768	1.798
2	0.70	1/27	1.043	1.051	1.055	1.070
	0.75	1/18	1.062	1.072	1.084	1.102
	0.80	1/12	1.088	1.099	1.126	1.148
	0.85	1/8	1.126	1.138	1.189	1.214
	0.90	1/6	1.171	1.184	1.269	1.293
	0.95	1/4	1.253	1.262	1.415	1.434
3	0.70	1/34	1.037	1.041	1.046	1.053
	0.75	1/23	1.053	1.058	1.070	1.078
	0.80	1/15	1.076	1.083	1.105	1.117
	0.85	1/10	1.108	1.115	1.158	1.171
	0.90	1/6	1.165	1.172	1.256	1.270
	0.95	1/4	1.248	1.253	1.406	1.416

Table 7: Expected search lengths for $b = 10$, $m = 5$ (optimistic analysis).

N_0	α	f	Successful		Unsuccessful	
			avg.	max.	avg.	max.
1	0.70	1/19	1.049	1.073	1.126	1.191
	0.75	1/13	1.072	1.101	1.183	1.261
	0.80	1/9	1.104	1.136	1.259	1.346
	0.85	1/6	1.152	1.186	1.369	1.459
	0.90	1/4	1.229	1.258	1.529	1.611
	0.95	1/3	1.322	1.341	1.697	1.765
2	0.70	1/56	1.017	1.025	1.041	1.064
	0.75	1/33	1.028	1.040	1.071	1.103
	0.80	1/20	1.046	1.061	1.118	1.158
	0.85	1/13	1.073	1.090	1.189	1.234
	0.90	1/8	1.118	1.136	1.306	1.350
	0.95	1/5	1.194	1.208	1.492	1.522
3	0.70	1/89	1.012	1.016	1.028	1.039
	0.75	1/49	1.021	1.027	1.051	1.067
	0.80	1/29	1.035	1.043	1.088	1.109
	0.85	1/17	1.058	1.067	1.149	1.174
	0.90	1/10	1.097	1.107	1.252	1.278
	0.95	1/6	1.164	1.172	1.425	1.443

Table 8: Expected search lengths for $b = 20$, $m = 5$ (optimistic analysis).

N_0	α	f	Successful		Unsuccessful	
			avg.	max.	avg.	max.
1	0.70	1/22	1.037	1.064	1.145	1.254
	0.75	1/14	1.058	1.093	1.217	1.350
	0.80	1/9	1.090	1.131	1.313	1.463
	0.85	1/6	1.138	1.181	1.441	1.596
	0.90	1/4	1.218	1.253	1.626	1.763
	0.95	1/3	1.315	1.335	1.803	1.924
2	0.70	1/107	1.006	1.013	1.027	1.060
	0.75	1/54	1.013	1.025	1.057	1.109
	0.80	1/29	1.025	1.042	1.110	1.181
	0.85	1/17	1.047	1.068	1.198	1.280
	0.90	1/10	1.085	1.108	1.343	1.422
	0.95	1/6	1.155	1.174	1.567	1.622
3	0.70	1/250	1.003	1.006	1.013	1.025
	0.75	1/112	1.007	1.012	1.031	1.054
	0.80	1/55	1.015	1.023	1.066	1.101
	0.85	1/29	1.029	1.041	1.131	1.177
	0.90	1/15	1.058	1.072	1.252	1.301
	0.95	1/8	1.117	1.129	1.470	1.502

Table 9: Expected search lengths for $b = 40$, $m = 5$ (optimistic analysis).