External Robin Hood Hashing

by

Pedro Celis

Computer Science Department
Indiana University
Bloomington, Indiana 47405

TECHNICAL REPORT N0. 246

External Robin Hood Hashing

By

Pedro Celis

March, 1988

External Robin Hood Hashing*

Pedro Celis

Computer Science Department,

Indiana University,

Bloomington, Indiana 47405

March 28, 1988

Abstract

We present some new and practical algorithms based on hashing with open addressing for implementing the operations of insert, delete and search in a file. Our method uses a small amount of internal storage to significantly reduce the number of disk operations required to maintain and search the file. The amount of memory used depends on several factors but is typically very small; about 4 bits per bucket or less than a bit per record are sufficient in most practical situations. We also present a detailed analysis on the behavior of our scheme and the results of some rather extensive simulations. The method requires only a constant number of external probes to perform successful and unsuccessful searches regardless of the size of the table, even if the table is completely full. The algorithms are very simple to code.

*Part of this work done while the author was at the University of Waterloo.

1 Introduction

Hashing is a well known and effective technique for organizing large tables in main memory or files in external storage. Numerous methods for handling overflow records have been proposed, which can be divided into three categories: perfect hashing, chaining and open addressing. The later alternative is the basic topic of this paper.

In open addressing the record key is used to generate a *probe sequence* : a list of bucket numbers to be tried in a search. In this technique the usual approach to insert a record is a "first come first serve" arrangement: when a record is to be inserted, the buckets specified by its probe sequence are examined until a nonfull bucket is found and the record is inserted there. This means that once inserted, a record is never moved. Robin Hood Hashing [5] is a reorganization scheme in the sense that records previously inserted into the file may be moved when a new insertion occurs. A number of reorganization schemes have been proposed for internally stored tables (i.e. bucket size equal to 1) [3,5,8,13,14,16]; all of which achieve a constant average search time for successful searches even for a full table. In what follows we briefly discuss some previously suggested methods.

Optimal, Binary Tree and Minmax Hashing [8,13,14,16] are schemes that have been proposed for internal tables. The tables they create are very good even if full; successful searches require constant time (between 1.82 and 2.13 probes) and unsuccessful searches require $O(\ln n)$ time. However, these methods are mainly of theoretical interest since the cost of creating the tables is very high both in time and memory. Brent's scheme [3] is a more practical method that requires 2.49 probes for successful search but $O(\sqrt{n})$ for unsuccessful searches. While this method could be extended for external files, it is not obvious how to use internal memory to reduce the number of external probes.

Since access to external storage is a fairly expensive operation, an important goal in external hashing schemes is to reduce the number of external probes required to insert and retrieve a record. To achieve this goal a reasonable approach is to use a small amount of internal memory.

Some Perfect Hashing schemes (see [12] and the references that appear there) have been proposed for external files. They guarantee retrievals with one external probe. However, they need a considerable amount of internal memory to implement the hashing function, and a considerable amount of time to achieve high load factors. Also, subsequent insertions may require a complete reorganization of the file.

Signature Hashing with fixed length separators [9] is an efficient technique that also guarantees retrievals in one external probe. The amount of internal memory required by this technique is comparable to that used by the method proposed in this paper. A disadvantage of signature hashing is that an insertion into the file may fail even if there is space available. The probability of this occurring is significant when the load factor is high or when a large number of deletions and insertions have been performed (regardless of the load factor).

In this paper we introduce a new hashing scheme for files called External Robin Hood Hashing. Our method uses a small amount of internal memory (about 4 bits per bucket in most practical cases) to reduce the number of external probes required during insertion and retrieval of records. The cost of retrieval depends on the load factor and the bucket size. Typical numbers are between 1.3 and 1.6 external accesses. Insertions are rejected only when the table is completely full. If the content of internal memory is lost (due to hardware failure for example), it can easily be recovered in one pass over the file. Most other methods require an amount of work proportional to recreating the file. Furthermore, our approach uses a very simple mechanism for supporting deletions and subsequent insertions that does not appear to cause significant performance degradation.

2 The Robin Hood Approach to Hashing

We already mentioned that the traditional hashing scheme works on a "first come first serve" arrangement: if b is the number of records per bucket, the first b records to hit a bucket are inserted there and the rest must be rehashed. The Robin Hood insertion heuristic consists of leaving in the bucket not the first b records to hash there but the b records that are in the largest probe positions. Hence the method gives preference to those records that have traveled the longest. The name of the scheme comes from the fact that we are taking from the more fortunate (those that have probed few buckets) to give to the less fortunate. This modification does not reduce the average wealth (probe position) of an individual but has the effect of dramatically reducing its variance.

The application of this heuristic to the insertion algorithm was first proposed in [5,6] for the case bucket size 1. There are several differences between the work presented there and in this paper:

— Our analysis is for an arbitrary but fix bucket size.

— Our main concern here is external probes and not running time; an analysis of the number of external probes required to search and insert is presented.

— The search algorithms presented in [5,6] are completely different from the one we present here: they require either $\Theta(\ln n)$ or no additional memory, perform successful searches in less than 2.6 *internal* probes, and unsuccessful searches in $\Theta(\ln n)$ probes. The search algorithm presented here requires $O(n)$ additional memory and less than two *external* probes for both successful and unsuccessful searches.

To reduce the number of external probes required to insert and search a record, we propose to have a table in main memory that contains for each bucket, the value of the smallest probe position among the records stored in that bucket. Let this table be called $bmin[1..n]$. Initially this array contains only zeros and all buckets in the file are filled with empty indicators.

When inserting, if a record probes bucket j on its i^{th} choice then:

— if $i \leq bmin[j]$, the record goes to its next choice without attempting an external probe;

— if $i > bmin[j]$, the bucket is retrieved. If there is an entry marked either *empty* or *deleted* the record is inserted there and the insertion procedure stops. Otherwise the record with the smallest probe position is replaced by the incoming record, the displaced record goes to its next choice (probe position $bmin[j] + 1$) and the value of $bmin[j]$ is adjusted if necessary.

When searching, if a record probes bucket j on its i^{th} choice then:

— if $i < bmin[j]$, the record is not stored in this bucket. The search continues at probe position $i + 1$ without attempting an external probe.

— if $i = bmin[j]$, the bucket is retrieved. If the record is not in the bucket the search continues at probe position $i + 1$.

— if $i > bmin[j]$ the bucket is retrieved. If the record is not in this bucket then the record is not in the file.

If the bucket size is 1, the search can be declared unsuccessful if $i > bmin[j]$ without retrieving the bucket. Hence exactly one external probe will be saved. If besides storing in main memory the minimum probe position we also store the maximum, a fraction of a probe will be saved for unsuccessful searches if $b > 1$. Successful searches and insertions would not benefit from this extra information.

To delete an element, the record is searched and marked as deleted but left in the bucket, and the entry of *bmin* corresponding to the bucket is left untouched. The only reason for marking the record as deleted and leaving it in the bucket instead of marking its entry as empty, is to be able to recover the contents of the array *bmin* in case they are lost.

From this description of the algorithms, it is clear that for each bucket the corresponding value of *bmin* can only increase. Therefore, if a large number of deletions and subsequent insertions are performed, the values stored in the table *bmin* can grow without bound requiring an arbitrarily large number of bits for their representation. This also implies that the number of internal operations that must be performed to insert or retrieve a record increase without bound.

These two problems can be overcome due to the fact that the gap between the smallest and the largest value in the array *bmin* is bounded. In section 4 we prove that for a full Robin Hood hash table where no deletions have occurred, the expected value of the largest entry in the array *bmin* (and hence the difference between the smallest and the largest) is $O\left(\ln \frac{n}{(b-1)!}\right)$. Unfortunately, we have not been able to analyze the effect that deletions and subsequent insertions have on the size of the gap between the smallest and largest values in the array *bmin*. However extensive simulations strongly indicate that the size of the gap remains bounded and is never greater than that of a full table in which no deletions have occurred.

What we suggest is to make each entry in the array *bmin*, $d = O(\ln\ln(n/(b-1)!))$ bits long (where d is 4 in most practical cases); and if the smallest probe position among the records stored in the bucket j is k, make $bmin[j] = k \bmod 2^d$. In order to reconstruct the value of k from the information stored in $bmin[j]$ we need to keep track of the smallest entry in the array *bmin*. Let the variable *smallest* denote the value of the smallest entry in the array *bmin*. Then k satisfies the expression $k = smallest + (bmin - smallest) \bmod 2^d$.

The second problem we mentioned was that the number of internal probes required to insert and retrieve increase without bound. The use of the variable *smallest* also solves this problem. When searching we should start the search at probe position *smallest* instead of 1 since the record cannot be stored before that probe position. Similarly we should start the insertion procedure at probe position $smallest + 1$.

A new problem that arises is that the value of the variable *smallest* can increase without a limit requiring therefore an arbitrarily large number of bits. To solve this problem we will assume that our hash function is such that for every key, the bucket probed at probe position k is the same as the one probed at position $k + n$. That is the case for double hashing. Then the variable *smallest* will contain the value of the smallest probe position among the records in the file $\bmod 2^d n$ (mod any common multiple would do). The minimum is taken among all records, including the ones marked as deleted.

The only remaining problem is how to keep track of the value of the variable *smallest*. The most efficient way is to have counters of how many bucket minimums are at each probe position. Only 2^d different counters are required. Figure 6 presents the pseudocode for the insertion and search algorithms we have just described. The function $fix(t)$ simply returns $smallest + (t - smallest) \bmod 2^d$.

3 Analysis of Insertions and Retrievals

In this section we present the analysis of some of the performance measures of External Robin Hood Hashing. The analysis in this section studies the performance of our scheme when no deletions have been made. We already mentioned that our simulations indicate that the performance of an nonfull table with load factor β where deletions have been made is similar to the performance of an nonfull table with load factor α ($\beta < \alpha < 1$) with no deletions. The analysis is asymptotic in nature, meaning that to study the performance of a file with m records in n buckets of capacity b we let $n, m \rightarrow \infty$ keeping $\alpha = \frac{m}{nb}$ constant and then analyze the performance of this infinite file. For the analysis we will assume that the model for the hashing function is Random Probing. Random Probing is equivalent to Uniform Hashing for infinite nonfull tables [11]. In practice, double hashing is a very convenient choice for our scheme and its performance is indistinguishable from the other two models. The simulations presented throughout this section use double hashing. To study the performance of our scheme we will introduce a probability model based on balls and urns. Throught this section we will compare the results of our analysis to results obtained through simulations.

3.1 The urn model

Consider the following urn model that corresponds to inserting all m records into a file with n buckets simultaneously: m balls are to be dropped among n urns. Each of these balls is given a label of 1. After the balls have been dropped, for each urn that contains more than b balls, b of them are selected according to some criterion and the rest are marked. All balls are left in the urn. For each marked ball with label 1, we create a new unmarked ball with label 2 and drop it into a random urn. After all the new balls have been distributed, we check the urns and for each urn that contains more than b unmarked balls, we select b balls among those with the highest labels and mark the rest. We then create an additional unmarked ball with label $i+1$ for each newly marked ball with label i and drop it into a random urn. We continue in this fashion until no new ball is marked.

In this model, urns represent buckets, balls represent probes into the file, ball labels represent probe positions or try numbers, and marking a ball represents rejecting a record from the bucket. An unmarked ball with label i represents a record placed in its i-th probe position.

Notice that the total number of unmarked balls with a label greater than or equal to i (number of records at or after their i^{th} probe position) is equal to the total number of balls, either marked or unmarked, with label i (number of records that probed their i^{th} position).

We define the following random variables:

— Let $\mathbf{N}$ represent the total number of balls dropped when the final assignment was reached.

— Let $\mathbf{X}_i$ be the total number of balls with label i.

— Let $\mathbf{N}_i$ be the total number of balls labeled i or less. Hence $\mathbf{N}_{i+1} = \mathbf{N}_i + \mathbf{X}_{i+1}$.

— Let $\mathbf{Z}_i$ be the total number of unmarked balls with label i or less. Then $\mathbf{X}_{i+1} = m - \mathbf{Z}_i$, and we know that $\mathbf{N}_1 = \mathbf{X}_1 = \mathbf{Z}_\infty = m$.

Each ball is labeled before it is dropped. Therefore the urn selected by a ball with label i, is independent of the urn selected by any other ball. However, the total number of balls with label i is not independent of the urns selected by other balls.

3.2 Probability distribution of psl

Let **psl** be the random variable that denotes the probe position where a randomly chosen record is stored in a Robin Hood hash table. In this subsection we derive the asymptotic probability distribution of the random variable **psl**. We do this by establishing a recurrence relation for $\mathrm{E}[\mathbf{N}_i/m]$.

Larson [11] has proved for α-full tables ($\alpha < 1$), that if we let m and n go to infinity, keeping $\alpha = \frac{m}{bn}$ constant, then the distribution of the number of balls that hit an arbitrary but fixed urn is poisson with parameter $\lambda = \lim_{n\to\infty} \mathrm{E}[\mathbf{N}/n]$. Following this we define $\lambda_i = \lim_{n\to\infty} \mathrm{E}[\mathbf{N}_i/n]$ and $t_i = \lim_{n\to\infty} \mathrm{E}[\mathbf{Z}_i/n]$. λ_i is the average number of balls with label less than or equal to i that hit an urn, and t_i is the average number of unmarked balls with label i or less in an urn.

Let $q_i(x,s)$ denote the probability that an urn is hit by x balls with label less than or equal to i and s balls with label higher than i. Then $q_i(x,s)$ is the product of a poisson and a binomial distribution as follows:

$$q_i(x,s) = \frac{e^{-\lambda}\lambda^{x+s}}{(x+s)!}\binom{x+s}{s}\left(\frac{\lambda_i}{\lambda}\right)^x\left(\frac{\lambda-\lambda_i}{\lambda}\right)^s = \frac{e^{-(\lambda-\lambda_i)}(\lambda-\lambda_i)^s}{s!}\cdot\frac{e^{-\lambda_i}\lambda_i^x}{x!}$$

The poisson factor gives the probability that an urn receives $x+s$ balls. $(\lambda-\lambda_i)/\lambda$ is the probability that a ball is labeled higher than i. This probability is independent of the labels of other balls in the same urn, since all balls are labeled before being dropped. Hence, the probability that s out of $x+s$ balls have a label higher than i has a binomial distribution with parameter $(\lambda-\lambda_i)/\lambda$. Notice that $q_i(x,s)$ is then equal to the product of two independent poisson distributions with parameters $\lambda-\lambda_i$ and λ_i.

The expected number of unmarked balls with label less than or equal to i is simply

$$\begin{aligned} t_i &= \sum_{s=0}^{b-1}\left[\sum_{x=0}^{b-s} x q_i(x,s) + \sum_{x=b-s+1}^{\infty}(b-s)q_i(x,s)\right] \\ &= e^{-\lambda}\sum_{s=0}^{b-1}\frac{(\lambda-\lambda_i)^s}{s!}\Big[\lambda_i \mathrm{e}_{b-s-1}(\lambda_i) + (b-s)\left(e^{\lambda_i} - \mathrm{e}_{b-s}(\lambda_i)\right)\Big] \end{aligned}$$

where $\mathrm{e}_j(y) = \sum_{i=0}^{j}\frac{y^i}{i!}$ is the truncated exponential function.

From the equations $\mathbf{N}_{i+1} = \mathbf{N}_i + \mathbf{X}_{i+1}$ and $\mathbf{X}_{i+1} = m - \mathbf{Z}_i$ we can write $\frac{\mathbf{N}_{i+1}}{n} = \frac{\mathbf{N}_i}{n} + \frac{m}{n} - \frac{\mathbf{Z}_i}{n}$

and taking expected values and the limit as $n \to \infty$ we get $\lambda_{i+1} = \lambda_i + \alpha\, b - t_i$, which together with $\lambda_1 = \alpha\, b$ gives us a recurrence relation for computing λ_i.

Define $p_i(\alpha, b)$ to be the probability that a record probes at least i locations before being placed in the table. Then, $p_i(\alpha, b) = \Pr\{\mathbf{psl} \geq i\}$ and satisfies

$$p_{i+1}(\alpha, b) = \lim_{m \to \infty} \frac{\mathrm{E}[\mathbf{X}_{i+1}]}{m}$$

and from the relation $\mathbf{X}_{i+1} = m - \mathbf{Z}_i$,

$$p_{i+1}(\alpha, b) = 1 - \lim_{n \to \infty} \frac{\mathrm{E}[\mathbf{Z}_i/n]}{\alpha\, b} = 1 - \frac{t_i}{\alpha\, b}$$

which together with $p_1(\alpha, b) = 1$ and the recurrence for λ_i, allows us to compute the probability distribution of **psl**. Figure 1 shows graphically the effect that the bucket size and load factor have on the expected value and variance of **psl**. In Tables 1 and 2 we compare the result of our analysis to results obtained through simulations. The number of urns in the file is n, the bucket size is b and the number of records in the file is $m = \alpha n\, b$. Whether or not the theoretically predicted value lies within the 95% confidence interval is indicated with a $\surd$ or $\times$ in the tables summarizing the results.

3.3 Probability distribution of bmin

In this subsection we derive the probability distribution of the random variable **bmin**, the minimum probe position among the records in an arbitrary but fixed bucket. Let $r_i(\alpha, b)$ be the probability that $\mathbf{bmin} = i$. The probability that $\mathbf{bmin} = 0$ is simply the probability that the bucket is nonfull, that is, that $b - 1$ or less records hashed into the bucket. It follows that

$$r_0(\alpha, b) = \sum_{x=0}^{b-1} \frac{e^{-\lambda}\lambda^x}{x!} = e^{-\lambda} \mathrm{e}_{b-1}(\lambda)$$

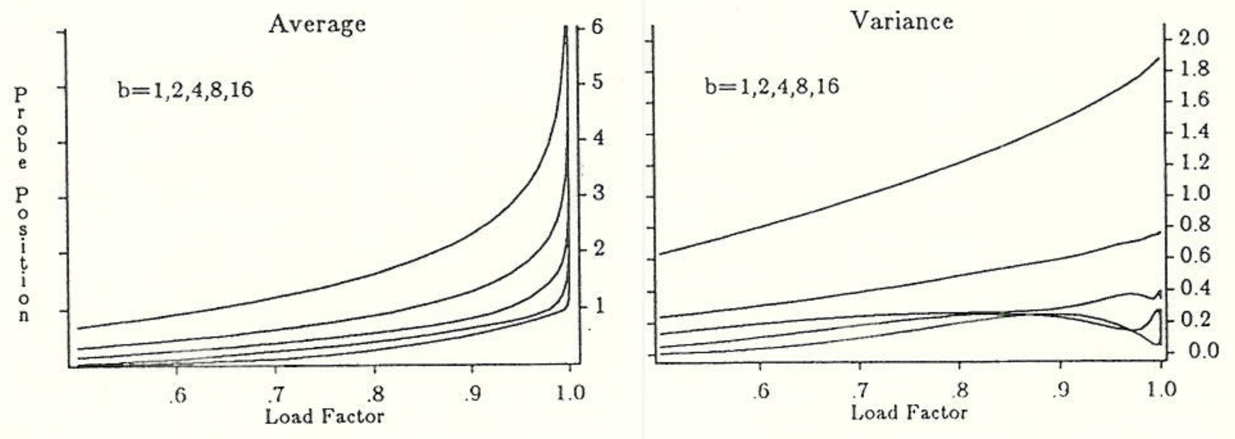

Figure 1: Expected value and variance of **psl**

The probability that **bmin** $\leq i$ is the probability of getting $b-1$ or less balls with label greater than i. Using the distribution $q(x,s)$ defined in the previous subsection, we can write this as

$$\Pr\{\mathbf{bmin} \leq i\} = \sum_{s=0}^{\infty}\sum_{x=0}^{b-1} q(x,s) = e^{-(\lambda-\lambda_i)}e_{b-1}(\lambda-\lambda_i).$$

It follows that

$$r_i(\alpha,b) = \Pr\{\mathbf{bmin} \leq i\} - \Pr\{\mathbf{bmin} \leq i-1\}$$

$$= e^{-(\lambda-\lambda_i)}e_{b-1}(\lambda-\lambda_i) - e^{-(\lambda-\lambda_{i-1})}e_{b-1}(\lambda-\lambda_{i-1}).$$

Figure 2 shows the effect that the load factor and the bucket size have on the expected value and variance of **bmin**, and Tables 3 and 4 compare our analysis to simulation results.

3.4 Cost of insertion

The cost of inserting a record measured in number of external probes is equal to one plus the number of stored records that are displaced or moved from their location. To study the expected

	$n = 1021$		$n = 4093$		$n = 16273$	
b	*predicted*	*simulation*	*predicted*	*simulation*	*predicted*	*simulation*
1	3.1371	$3.1278\pm0.0169\surd$	3.1519	$3.1446\pm0.0091\surd$	3.1530	$3.1540\pm0.0042\surd$
2	2.1602	$2.1545\pm0.0076\surd$	2.1639	$2.1594\pm0.0041\times$	2.1645	$2.1649\pm0.0019\surd$
4	1.6392	$1.6372\pm0.0042\surd$	1.6403	$1.6397\pm0.0020\surd$	1.6404	$1.6399\pm0.0010\surd$
8	1.3538	$1.3534\pm0.0020\surd$	1.3540	$1.3548\pm0.0010\surd$	1.3540	$1.3534\pm0.0005\times$
16	1.1938	$1.1940\pm0.0012\surd$	1.1938	$1.1935\pm0.0005\surd$	1.1938	$1.1938\pm0.0003\surd$

Table 1: E[**psl**] ($\alpha \approx 0.95$)

	$n = 1021$		$n = 4093$		$n = 16273$	
b	*predicted*	*simulation*	*predicted*	*simulation*	*predicted*	*simulation*
1	1.2239	$1.2158\pm0.0100\surd$	1.2293	$1.2246\pm0.0050\surd$	1.2297	$1.2299\pm0.0026\surd$
2	0.5550	$0.5532\pm0.0032\surd$	0.5559	$0.5555\pm0.0016\surd$	0.5561	$0.5564\pm0.0007\surd$
4	0.3117	$0.3104\pm0.0014\surd$	0.3119	$0.3119\pm0.0006\surd$	0.3119	$0.3117\pm0.0003\surd$
8	0.2293	$0.2289\pm0.0006\surd$	0.2293	$0.2295\pm0.0003\surd$	0.2294	$0.2292\pm0.0002\surd$
16	0.1562	$0.1563\pm0.0007\surd$	0.1562	$0.1560\pm0.0003\surd$	0.1563	$0.1562\pm0.0002\surd$

Table 2: V[**psl**] ($\alpha \approx 0.95$)

	$n = 1021$		$n = 4093$		$n = 16273$	
b	*predicted*	*simulation*	*predicted*	*simulation*	*predicted*	*simulation*
1	2.9773	$2.9685\pm0.0160\surd$	2.9940	$2.9871\pm0.0087\surd$	2.9953	$2.9962\pm0.0040\surd$
2	1.7071	$1.7022\pm0.0069\surd$	1.7113	$1.7076\pm0.0038\surd$	1.7121	$1.7122\pm0.0018\surd$
4	1.0953	$1.0933\pm0.0039\surd$	1.0967	$1.0958\pm0.0019\surd$	1.0968	$1.0964\pm0.0009\surd$
8	0.8111	$0.8114\pm0.0011\surd$	0.8112	$0.8116\pm0.0006\surd$	0.8113	$0.8111\pm0.0003\surd$
16	0.7246	$0.7248\pm0.0012\surd$	0.7246	$0.7242\pm0.0006\surd$	0.7247	$0.7247\pm0.0003\surd$

Table 3: E[**bmin**] ($\alpha \approx 0.95$)

	$n = 1021$		$n = 4093$		$n = 16273$	
b	*predicted*	*simulation*	*predicted*	*simulation*	*predicted*	*simulation*
1	1.6373	$1.6275\pm0.0132\surd$	1.6404	$1.6339\pm0.0069\surd$	1.6406	$1.6411\pm0.0034\surd$
2	0.6675	$0.6643\pm0.0049\surd$	0.6682	$0.6651\pm0.0025\times$	0.6683	$0.6685\pm0.0012\surd$
4	0.3503	$0.3477\pm0.0028\surd$	0.3505	$0.3496\pm0.0014\surd$	0.3506	$0.3503\pm0.0006\surd$
8	0.1662	$0.1658\pm0.0007\surd$	0.1661	$0.1662\pm0.0004\surd$	0.1661	$0.1661\pm0.0002\surd$
16	0.1995	$0.1994\pm0.0005\surd$	0.1995	$0.1997\pm0.0002\surd$	0.1995	$0.1995\pm0.0001\surd$

Table 4: V[**bmin**] ($\alpha \approx 0.95$)

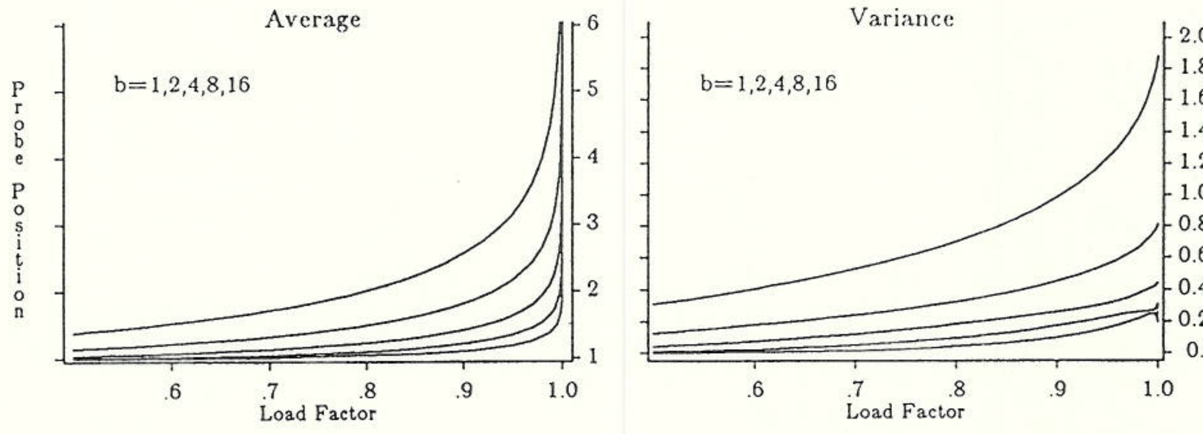

Figure 2: Expected value and variance of **bmin**

value of the number of records moved, we need to model the insertion of a record when the load factor is α, and increasing it by $\delta\alpha$.

The model based on a Markov chain is defined as follows: Let the state of the Markov chain be the probe position of the record we are trying to insert (which is not necessarily the new record in the table). If the bucket being probed has a value of **bmin** which is less than the state, the new state becomes **bmin** $+ 1$, otherwise the state is increased by one. The one step transition probability matrix corresponding to this Markov chain is the following infinite matrix

$$\begin{array}{c|ccccccccc}
 & 0 & 1 & 2 & 3 & 4 & \cdots & i & i+1 & \cdots \\
\hline
0 & 1 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & \cdots \\
1 & r_0 & 0 & 1-r_0 & 0 & 0 & \cdots & 0 & 0 & \cdots \\
2 & r_0 & 0 & r_1 & 1-r_0-r_1 & 0 & \cdots & 0 & 0 & \cdots \\
3 & r_0 & 0 & r_1 & r_2 & 1-r_0-r_1-r_2 & \cdots & 0 & 0 & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots \\
i & r_0 & 0 & r_1 & r_2 & r_3 & \cdots & r_{i-1} & 1-\sum_{j=0}^{i-1} r_j & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots
\end{array}$$

and the initial state is 1. It is clear from this that as long as $r_0(\alpha, b) > 0$, all states in the chain are transient, except state 0 which is absorbing.

Even though some of the buckets will have their value for **bmin** changed during the insertion process, the transition probability matrix, does not change since only a finite (out of an infinite) number of **bmin**'s can change and the effect on the distribution $r_i(\alpha, b)$ is zero.

The total number of external probes required for an insertion is then equal to the total number of times the Markov chain goes from a state numbered i to a state numbered less than or equal to i until state 0 is reached. Or

$$\begin{aligned} \text{E[insert cost]} &= 1 + \sum_{i=1}^{\infty} \text{E[times state } i \text{ is reached from a state} \geq i] \\ &= 1 + \sum_{i=1}^{\infty} r_i(\alpha, b)\text{E[times state} \geq i \text{ is reached]}. \end{aligned}$$

Hence we need only to determine for every i, the value of E[times state $\geq i$ is reached]. To determine this, we will modify our transition probability matrix to obtain a finite matrix by lumping together all states with label larger than i to obtain

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ \vdots \\ i \\ > i \end{array} \begin{array}{c} \begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & \cdots & i & > i \end{array} \\ \left(\begin{array}{cccccccc} 1 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 \\ r_0 & 0 & 1 - r_0 & 0 & 0 & \ldots & 0 & 0 \\ r_0 & 0 & r_1 & 1 - r_0 - r_1 & 0 & \ldots & 0 & 0 \\ r_0 & 0 & r_1 & r_2 & 1 - r_0 - r_1 - r_2 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ r_0 & 0 & r_1 & r_2 & r_3 & \ldots & r_{i-1} & 1 - \sum_{j=0}^{i-1} r_j \\ r_0 & 0 & r_1 & r_2 & r_3 & \ldots & r_{i-1} & 1 - \sum_{j=0}^{i-1} r_j \end{array} \right) \end{array}.$$

Let $Q^{(i)}$ be the substochastic matrix obtained by removing the first 2 rows and columns from this matrix and let $M^{(i)} = \left(I - Q^{(i)}\right)^{-1}$. Then, it is a well known result (see for example chapter 5 of [2]) that the (j,k) element of a matrix M constructed in this fashion is the average number of times the k^{th} state was reached before absorption, if the process was started at state j. Let the elements of the matrix $M^{(j)}$ be denoted by $\mu_{k,l}^{(j)}$. It follows that

$$\text{E[times a state} \geq i \text{ is reached]} = (1 - r_0(\alpha, b))\left(\mu_{2,i}^{(i)} + \mu_{2,>i}^{(i)}\right)$$

In practice it is not necessary to compute $M^{(i)}$ for every i of interest since $\mu_{2,i}^{(i)} = \mu_{2,i}^{(j)} \quad \forall j > i$. Hence a value of j large enough so that $1 - (r_0(\alpha, b) + \cdots + r_j(\alpha, b))$ is negligible can be chosen and then the first row of $M^{(j)}$ will give us all the information that we need.

In computing the vector $\mu_{2,i}^{(j)}$ if $r_0(\alpha, b)$ is not close to 0, it may prove to be more convenient to use the identity

$$(1, 0, \ldots, 0)M^{(j)} = (1, 0, \ldots, 0)\left(I + Q^{(j)} + \left[Q^{(j)}\right]^2 + \cdots\right)$$

we can then use Horner's rule and rewrite this as

$$= (1, 0, \ldots, 0)\left(I + Q^{(j)}\left(I + Q^{(j)}(I + \cdots)\right)\right).$$

The evaluation of this expression can be done using only two vectors. Define initially $v = (1, 0, \ldots, 0)$ and $\mu_2 = v$. Then repeat the operations $v = vQ^{(j)}$ and $\mu_2 = \mu_2 + v$ until the contribution of an additional term would be negligible. Since $Q^{(j)}$ is small (i.e. $||Q^{(j)}|| < 1$), $\left[Q^{(j)}\right]^k$ should converge rapidly to 0. Also since $Q^{(j)}$ is very simple it is not necessary to store it explicitly. After ℓ iterations, the sum of the elments of v is $(1 - r_0)^{\ell}$, so the number of iterations required to reach a specified tolerance ϵ is $\ln(\epsilon)/\ln(1 - r_0)$. Hence, if r_0 is close to 0 such that $j \ll \ln(\epsilon)/\ln(1 - r_0)$, it is better to work with the decomposition of $Q^{(j)}$.

To obtain the average insertion cost up to load factor β, we can integrate numerically from $\alpha = 0$ to β the insertion cost at load factor α and divide the integral by β. Figure 3 shows the cost

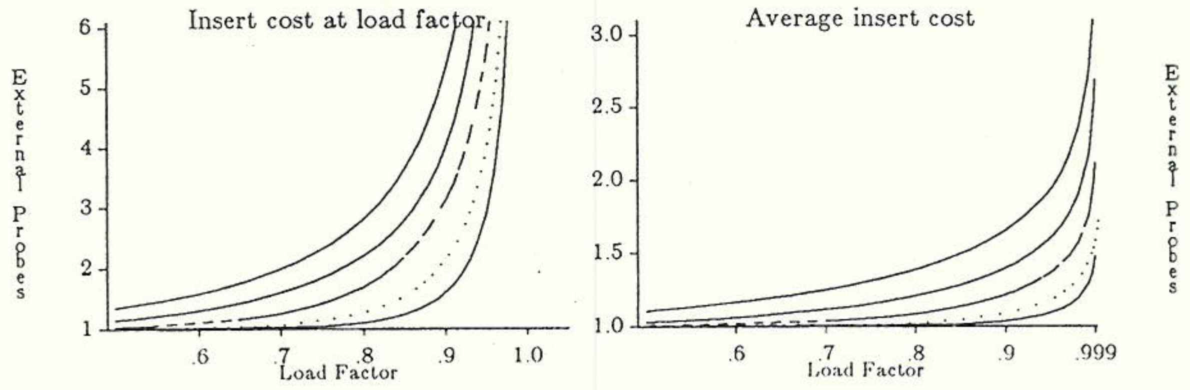

Figure 3: Cost of insertion

b	$n = 1021$		$n = 4093$		$n = 16273$	
	predicted	*simulation*	*predicted*	*simulation*	*predicted*	*simulation*
1	1.9357	1.9325±0.0087$\surd$	1.9448	1.9417±0.0047$\surd$	1.9455	1.9465±0.0022$\surd$
2	1.6029	1.6000±0.0048$\surd$	1.6057	1.6027±0.0025$\times$	1.6062	1.6066±0.0012$\surd$
4	1.3654	1.3644±0.0029$\surd$	1.3663	1.3655±0.0014$\surd$	1.3664	1.3660±0.0007$\surd$
8	1.1939	1.1937±0.0016$\surd$	1.1941	1.1947±0.0008$\surd$	1.1942	1.1936±0.0004$\times$
16	1.0913	1.0913±0.0009$\surd$	1.0913	1.0912±0.0004$\surd$	1.0913	1.0913±0.0002$\surd$

Table 5: Average insertion cost ($\alpha \approx 0.95$)

of insertion at and up to load factor α for different values of b and Table 5 compares our derivations to simulation results.

3.5 Cost of successful searches

In this subsection we derive an expression for the cost of performing a successful search. We will say that an unnecessary probe occurs whenever an external probe is made but the record being searched is not present in the bucket. At the i^{th} step of the searching process an unnecessary probe can occur only if $\mathbf{bmin} = i$.

Let $\mathbf{R}_i$ be the total number of balls with label i rejected from an urn in which after all the $\mathbf{N}$ balls were dropped the smallest label among the unmarked balls was i. Then $\mathbf{R}_i/\mathbf{X}_i$ is the fraction

	$n = 1021$		$n = 4093$		$n = 16273$	
b	*predicted*	*simulation*	*predicted*	*simulation*	*predicted*	*simulation*
1	1.3309	1.3313±0.0023✓	1.3311	1.3302±0.0012✓	1.3311	1.3308±0.0006✓
2	1.3968	1.3947±0.0018×	1.3970	1.3976±0.0009✓	1.3970	1.3971±0.0004✓
4	1.4119	1.4119±0.0013✓	1.4121	1.4127±0.0007✓	1.4121	1.4118±0.0004✓
8	1.3473	1.3470±0.0017✓	1.3475	1.3481±0.0009✓	1.3475	1.3470±0.0005✓
16	1.1938	1.1940±0.0012✓	1.1938	1.1935±0.0005✓	1.1938	1.1938±0.0003✓

Table 6: Average successful search cost ($\alpha \approx 0.95$)

of balls with label i that are marked in an urn, where the smallest label among the unmarked balls is i. Let $w_i = \lim_{n\to\infty} \mathrm{E}[\mathbf{R}_i/n]$. It follows that the probability that a record (which was not stored in its first $(i-1)$ choices) has for its i^{th} choice a bucket with $\mathbf{bmin} = i$ and is not stored there is equal to $\lim_{n\to\infty} \mathrm{E}[\mathbf{R}_i/\mathbf{X}_i] = w_i/(\lambda_i - \lambda_{i-1})$.

w_i is the average (over all urns) number of balls with label i marked in urns that received $b-1$ or less balls with label higher than i. This can easily be shown to be equal to

$$w_i = \sum_{x=0}^{b-1} e^{-(\lambda-\lambda_i)}\frac{(\lambda-\lambda_i)^x}{x!} \sum_{y=b-x+1}^{\infty} (y-(b-x))e^{-(\lambda-\lambda_i)}\frac{(\lambda_i-\lambda_{i-1})^y}{y!}$$

$$= e^{-(\lambda-\lambda_{i-1})}\sum_{x=0}^{b-1}\frac{(\lambda-\lambda_i)^x}{x!}\Big[(\lambda_i-\lambda_{i-1})\left(e^{\lambda_i-\lambda_{i-1}} - \mathrm{e}_{b-x-1}(\lambda_i-\lambda_{i-1})\right)$$

$$-(b-x)\left(e^{\lambda_i-\lambda_{i-1}} - \mathrm{e}_{b-x}(\lambda_i-\lambda_{i-1})\right)\Big].$$

The expected cost of performing a successful search can then be computed as follows:

$$\mathrm{E}[\text{successful search cost}] = 1 + \sum_{i=1}^{\infty} p_i(\alpha,b)\frac{w_i}{\lambda_i-\lambda_{i-1}}$$

The $p_i(\alpha,b)$ factor gives the probability that a record reaches the i^{th} probe position and $w_i/(\lambda_i - \lambda_{i-1})$ gives the probability that at the i^{th} probe position an unnecessary probe occurs.

Figure 4 shows how the cost of searching increases with the load factor for different values of b and Table 6 shows some simulation results. The expected search cost does not differ significantly

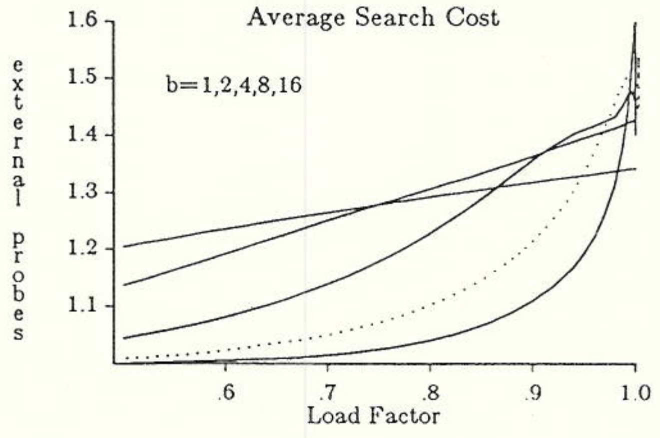

Figure 4: Cost of successful search

from the expected probe position until the load factor is fairly high (more than 80% for $b = 1$ and more than 99% for $b = 16$). This means that for moderate load factors every internal probe requires an external probe. Hence the information about the minimum probe value in the bucket is not useful in reducing the successful search cost until the load factor is high. It remains useful however in reducing the cost of insertion and unsuccessful searches.

3.6 Cost of unsuccessful searches

In this subsection we derive an expression for the cost of performing an unsuccessful search. A search can be terminated as unsuccessful at the i^{th} probe position if $\mathbf{bmin} < i$ and the record is not in the bucket. Hence the probability of reaching the i^{th} step is $\prod_{j=1}^{i-1} \Pr\{\mathbf{bmin} \geq j\} = \prod_{j=1}^{i-1} \left(1 - \sum_{k=1}^{j-1} r_k\right)$. The probability of doing an external probe at the i^{th} step is $\sum_{j=0}^{i-1} \Pr\{\mathbf{bmin} \leq i\} = \sum_{j=0}^{i-1} r_j$. The expected number of external probes required to determine that a search is unsuccessful is

$$\text{E[unsuccessful search cost]} = \sum_{i=1}^{\infty} \left(\sum_{j=1}^{i} r_j\right) \prod_{j=1}^{i-1} \left(1 - \sum_{k=0}^{j-1} r_k\right)$$

	$n = 1021$		$n = 4093$		$n = 16273$	
b	*predicted*	*simulation*	*predicted*	*simulation*	*predicted*	*simulation*
1	0.6885	0.6829±0.0081√	0.6890	0.6897±0.0078√	0.6890	0.6849±0.0079√
2	1.8140	1.8118±0.0077√	1.8144	1.8145±0.0072√	1.8145	1.8197±0.0077√
4	1.8380	1.8434±0.0063√	1.8384	1.8366±0.0060√	1.8384	1.8383±0.0069√
8	1.8033	1.8013±0.0042√	1.8035	1.8027±0.0041√	1.8036	1.8003±0.0038√
16	1.7246	1.7232±0.0043√	1.7246	1.7243±0.0044√	1.7247	1.7265±0.0043√

Table 7: Average unsuccessful search cost ($\alpha = 0.95$)

We already mention that for $b = 1$ the number of external probes can be further reduced by 1 since the last probe is not necessary to determine that the record is not in the file. The cost measured in internal probes is obtained by removing the first factor from the previous expression. Figure 5 shows the cost of unsuccessful searches for different values of α and b. Table 7 shows simulation results.

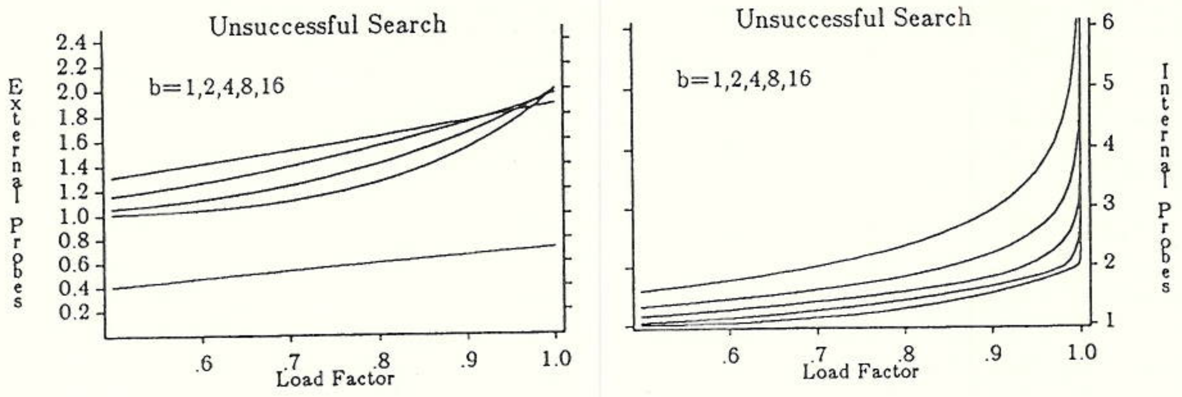

Figure 5: Unsuccessful search cost

4 The Expected Largest bmin

In this section we derive bounds for the expected value of the largest value in the array bmin (E[lbmin]) for full tables when no deletions have been made and prove that $\mathrm{E}[\mathrm{lbmin}] = \Theta(\ln n)$.

Assume we have a set $\mathcal{R} = \{R_1, \ldots, R_m\}$ of records stored in a file where $m = n\,b$ since $\alpha = 1$. Let $\vartheta : \mathcal{R} \times \{1, \ldots, n\} \mapsto \{1, \ldots, n\}$ be the *backup function*, defined as the bucket that record R_i probed j steps before its current location.

Assume that the largest value in the array bmin (**lbmin**) is ℓ and the bucket that has that value is bucket j_{worst}. $\mathcal{W}_i$ will denote a set of buckets all of which have a corresponding value for bmin of at least $(\ell - i)$. And $\mathcal{U}_{i+1}$ will denote the set of buckets that the records in the buckets in the set $\mathcal{W}_i$ would hit if moved back one position. More formally, $\mathcal{U}_0 = \{j_{worst}\}$, $\mathcal{W}_0 = \mathcal{U}_0$, and

$$\mathcal{U}_i = \{k \mid k = \vartheta(R, j) \text{for some } R \text{ in a bucket in } \mathcal{U}_{i-j}\} \qquad \text{and} \qquad \mathcal{W}_i = \mathcal{W}_{i-1} \cup \mathcal{U}_i.$$

Each record in a bucket in the set $\mathcal{W}_i$ is in at least its $(\ell - i)^{\text{th}}$ probe position. If none of the locations that we sample when moving back a record were repeated, then the cardinality of the set $\mathcal{W}_i$ would be $(b+1)^i$. Since we are sampling the locations with replacement, the cardinality of $\mathcal{W}_i$ is a random variable denoted by $\mathbf{w}_i$. We will denote by $\mathbf{u}_i$ the cardinality of the set $\mathcal{U}_i - \mathcal{W}_{i-1}$ which is the number of buckets that belong to the set $\mathcal{W}_i$ but do not belong to the set $\mathcal{W}_{i-1}$.

We will find a bound for the expected value of $\mathbf{w}_i$ using occupancy distributions [7,10]. $\mathbf{w}_i$ is equal to $\mathbf{w}_{i-1} + \mathbf{u}_i$. The distribution of $\mathbf{u}_i$ is of the type called classical occupancy with specified boxes (see for example chapter 14 of [7]) where the total number of urns is n, among which the number of specified urns is $n - \mathbf{w}_{i-1}$ and the number of balls dropped is $b\mathbf{w}_{i-1}$ and we are interested in the number of specified urns that are hit. We will now define a new sequence of random variables $\mathbf{v}_i$ such that $\mathrm{E}[\mathbf{v}_i] \leq \mathrm{E}[\mathbf{w}_i]$ for all i. Initially $\mathbf{w}_0 = \mathbf{v}_0 = 1$. Let $\mathbf{v}_i$ be the number of different urns to be hit if $(b+1)\mathbf{v}_{i-1}$ balls are dropped at random. If we were to guarantee that the first $\mathbf{v}_{i-1}$ of these $(b+1)\mathbf{v}_{i-1}$ balls went all to different urns and the rest were dropped at random, then there would be no difference between the random variables $\mathbf{v}_i$ and $\mathbf{w}_i$. Since this is not the case, then the expected value of $\mathbf{v}_i$ is less than the expected value of $\mathbf{w}_i$. The distribution of $\mathbf{v}_i$ is then of the type called classical occupancy. We therefore have

$$\mathrm{E}[\mathbf{v}_i \mid \mathbf{v}_{i-1}] = n\left(1 - \left(1 - \frac{1}{n}\right)^{(b+1)\mathbf{v}_{i-1}}\right)$$

$$\begin{aligned}\mathrm{V}[\mathbf{v}_i \mid \mathbf{v}_{i-1}] &= n(n-1)\left(1 - \frac{2}{n}\right)^{(b+1)\mathbf{v}_{i-1}} + n\left(1 - \frac{1}{n}\right)^{(b+1)\mathbf{v}_{i-1}} - n^2\left(1 - \frac{1}{n}\right)^{2(b+1)\mathbf{v}_{i-1}} \\ &= n^2\left[\left(1 - \frac{2}{n}\right)^{(b+1)\mathbf{v}_{i-1}} - \left(1 - \frac{1}{n}\right)^{2(b+1)\mathbf{v}_{i-1}}\right] + n\,O(1)\end{aligned}$$

which, using inequality 4.2.29 from [1], is bounded by

$$< n^2\left(e^{-2(b+1)\mathbf{v}_{i-1}/n} - e^{-2(b+1)\mathbf{v}_{i-1}/(n-1)}\right) + O(n) = O(n)$$

To get a bound on the expected value of the maximum bucket min we will first prove the following three lemmas.

Lemma 1 $\mathrm{E}[\mathbf{v}_i]$ *is asymptotically equivalent to* $\mathrm{E}\left[\mathbf{v}_i \mid \mathbf{v}_{i-1} = \mathrm{E}[\mathbf{v}_{i-1}]\right]$.

Proof: We know that

$$\mathrm{E}[\mathbf{v}_i \mid \mathbf{v}_{i-1}] = n\left(1 - \left(1 - \frac{1}{n}\right)^{(b+1)\mathbf{v}_{i-1}}\right)$$

Removing the condition we get

$$\mathrm{E}[\mathbf{v}_i] = n\left(1 - \mathrm{E}\left[\left(1 - \frac{1}{n}\right)^{2\mathbf{v}_{i-1}}\right]\right)$$

Using equation 4.2.29 again we get

$$\mathrm{E}\left[e^{-(b+1)\mathbf{v}_{i-1}/(n-1)}\right] < \mathrm{E}\left[\left(1 - \frac{1}{n}\right)^{(b+1)\mathbf{v}_{i-1}}\right] < \mathrm{E}\left[e^{-(b+1)\mathbf{v}_{i-1}/n}\right].$$

We have mentioned that v_{i-1} has an occupancy distribution. Rényi [15] establishes that an occupancy distribution converges to a normal distribution as $n \to \infty$ if $e^{d/n}/n \to 0$, where d is the number of balls dropped. The number of balls dropped in the distribution of v_{i-1} is $(b+1)\mathrm{v}_{i-2} < (b+1)n$, so the condition is satisfied. Then the moment generating function of v_{i-1} converges to $M_{\mathrm{v}_{i-1}}(t) = \mathrm{E}[e^{t\mathrm{v}_{i-1}}] = e^{\mu t + \frac{1}{2}t^2\sigma^2}$. Using this equation with $t = -(b+1)/n$ and $t = -(b+1)/(n-1)$, we get

$$\exp\left(-(b+1)\left(\mathrm{E}[\mathrm{v}_{i-1}]/(n-1) - \sigma^2/(n-1)^2\right)\right) < \mathrm{E}\left[\left(1-\frac{1}{n}\right)^{(b+1)\mathrm{v}_{i-1}}\right]$$
$$< \exp\left(-(b+1)\left(\mathrm{E}[\mathrm{v}_{i-1}]/n - \sigma^2/n^2\right)\right)$$

and since $\sigma^2 = \mathrm{V}[\mathrm{v}_{i-1}] = O(n)$, these bounds become asymptotically

$$e^{-(b+1)\mathrm{E}[\mathrm{v}_{i-1}]/(n-1)} < \mathrm{E}\left[\left(1-\frac{1}{n}\right)^{(b+1)\mathrm{v}_{i-1}}\right] < e^{-(b+1)\mathrm{E}[\mathrm{v}_{i-1}]/n}$$

Using inequality 4.2.29 from [1] on $(1-\frac{1}{n})^{2\mathrm{E}[\mathrm{v}_{i-1}]}$ we obtain the same bounds:

$$e^{-(b+1)\mathrm{E}[\mathrm{v}_{i-1}]/(n-1)} < \left(1-\frac{1}{n}\right)^{(b+1)\mathrm{E}[\mathrm{v}_{i-1}]} < e^{-(b+1)\mathrm{E}[\mathrm{v}_{i-1}]/n}.$$

Consequently the difference $\left|\mathrm{E}[(1-\frac{1}{n})^{(b+1)\mathrm{v}_{i-1}}] - (1-\frac{1}{n})^{(b+1)\mathrm{E}[\mathrm{v}_{i-1}]}\right|$ is bounded and goes to 0 as $n \to \infty$. □

We can write $\mathrm{E}[\mathrm{v}_i]$ as $n(1-f_i(n))$. Then $f_i(n)$ obeys the following recurrence

$$f_i(n) = \left(1-\frac{1}{n}\right)^{(b+1)\mathrm{E}[\mathrm{v}_{i-1}]} = \left(1-\frac{1}{n}\right)^{(b+1)n(1-f_{i-1}(n))}$$

$$f_0(n) = 1 - \frac{1}{n}$$

$f_i(n)$ represents the fraction of urns not hit by the $(b+1)\mathrm{E}[\mathrm{v}_{i-1}]$ balls. It is an increasing function in n and decreasing in i.

Lemma 2 $f_j\left((b+1)^j(n-1) - \sum_{s=1}^{j-1}(b+1)^s\right) < f_0(n)$.

Proof [By Induction]: Basis: for $j = 1$, we have

$$f_0((b+1)(n-1)) = 1 - \frac{1}{(b+1)(n-1)}$$

$$f_1((b+1)(n-1)) = \left(1 - \frac{1}{(b+1)(n-1)}\right)^{(b+1)} \quad < e^{-\frac{1}{n-1}} \quad < 1 - \frac{1}{n} \quad = f_0(n)$$

Inductive step: assume true for $j-1$ and prove for j. Let $n_j = (b+1)^j(n-1) - \sum_{s=1}^{j-1}(b+1)^s$. The induction hypothesis is $f_{j-1}(n_j) < f_0(n_1)$. Now

$$f_j(n_j) = \left(1 - \frac{1}{n_j}\right)^{(b+1)n_j(1-f_{j-1}(n_j))} \qquad < e^{-(b+1)(1-f_{j-1}(n_j))}$$

which by the induction hypothesis is

$$< e^{-(b+1)(1-f_0(n_1))} \quad = e^{-(b+1)\left(1-\left(1-\frac{1}{n_1}\right)\right)} \quad = e^{-\frac{1}{n-1}} \quad < 1 - \frac{1}{n} \quad = f_0(n) \qquad \square$$

Lemma 3 $\sum_{i=0}^{\ell-1} f_i(n) < \frac{1}{2}\ell + \frac{1}{2}(\log_{b+1} n + 1)$.

Proof: Let k be the smallest integer that satisfies $n \le n_k = (b+1)^k(n_0 - 1) + \sum_{s=1}^{k-1}(b+1)^k$, where n_0 is an arbitrarily chosen constant greater than 1 (greater than 2 if $b = 1$). Then $k < \log_{b+1} n + 1$ and

$$\sum_{i=0}^{\ell-1} f_i(n) \quad \le \sum_{i=0}^{\ell-1} f_i(n_k) \quad = \sum_{i=0}^{k-1} f_i(n_k) + \sum_{i=k}^{\ell-1} f_i(n_k) \quad < k + \sum_{i=0}^{\ell-k-1} f_{i+k}(n_k) \quad < k + (\ell - k) f_k(n_k)$$

letting $n_0 = 2$ and using the previous lemma

$$< k + (\ell - k) f_0(2) \quad = k + \frac{1}{2}(\ell - k) \quad = \frac{1}{2}\ell + \frac{1}{2}(\log_{b+1} n + 1) \qquad \square$$

Theorem 1 *As n goes to infinity, the expected value of* **lbmin** *for a full Robin Hood Hash table with n buckets of capacity $b > 1$ is bounded by* $\mathrm{E}[\textbf{lbmin}] < 2\ln\frac{n}{(b-1)!} + \log_{b+1} n + 2(b-1)\ln\ln n1 + 2\gamma + o(1)$.

Proof: Define $\mathbf{r}$ to be the sum of probe positions of the records in $\mathcal{W}_{\ell-1}$. From the construction of the sets $\mathcal{W}_i$'s we know that $\mathcal{W}_0 \subseteq \mathcal{W}_1 \subseteq \cdots \subseteq \mathcal{W}_{\ell-1}$. If a record appears for the first time in the set $\mathcal{W}_k$, its probe position is at least $(\ell - k)$, and it appears in exactly $(\ell - k)$ of the ℓ sets $\mathcal{W}_i$'s. It follows that $\mathbf{r} \geq \mathbf{w}_0 + \cdots + \mathbf{w}_{\ell-1}$ and therefore

$$\mathrm{E}[\mathbf{r}|\mathbf{lbmin} = \ell] \geq \sum_{i=0}^{\ell-1} \mathrm{E}[\mathbf{w}_i] \quad \geq \sum_{i=0}^{\ell-1} \mathrm{E}[\mathbf{v}_i] \quad = \sum_{i=0}^{\ell-1} n\,(1 - f_i(n)) \quad = n \left(\ell - \sum_{i=0}^{\ell-1} f_i(n) \right)$$

and by the previous lemma

$$> \frac{n}{2} \left(\ell - \log_{b+1} n - 1 \right).$$

It follows that

$$\mathrm{E}[\mathbf{r}] > \frac{n}{2} \left(\mathrm{E}[\mathbf{lbmin}] - \log_{b+1} n - 1 \right).$$

Let $\mathbf{N}$ be the sum of the probe positions of the $m = n\,b$ records in $\mathcal{R}$ stored in the hash table. Since $\mathcal{V}_{\ell-1}$ is just a subset of $\mathcal{R}$ then $\mathrm{E}[\mathbf{N}] \geq \mathrm{E}[\mathbf{r}]$. Brayton [4] proved that as n goes to infinity

$$\mathrm{E}[\mathbf{N}/n] = \ln n + (b-1) \ln\ln n - \ln(b-1)! + \gamma + O\left(\frac{\ln\ln n}{\ln n} \right).$$

Substituting this expression and solving for $\mathrm{E}[\mathbf{lbmin}]$ we get

$$\mathrm{E}[\mathbf{lbmin}] < 2 \ln n + \log_{b+1} n + 2(b-1) \ln\ln n - 2 \ln(b-1)! + 1 + 2\gamma + o(1) \qquad \square.$$

Table 8 shows the largest value in the array *bmin* encountered in 210 simulations we performed for each bucket size and each file size listed.

5 Conclusions and Open Problems

We have presented new and practical algorithms for external hashing. These algorithms can be easily coded as can be seen from the pseudocode shown in Figure 6. Our algorithms use a small

	1021	4093	16273
1	16	17	23
2	9	10	10
4	6	6	7
8	4	4	4
16	3	3	3

Table 8: Maximum observed *bmin* in 210 full tables

amount of internal storage to reduce the number of external probes required to maintain and search the file. The amount of memory required for a file of n buckets with capacity b is $\ln\ln(n/(b-1)!) + O(\ln\ln\ln n)$. About 4 bits per bucket are sufficient in most practical situations. We have also included analysis of the behavior of our scheme that shows that the method is efficient. Simulation results closely resemble the figures predicted by the analyses.

The most obvious open problem is to provide an analysis of the effects that large numbers of deletions and insertions have on the performance measures. The simulations we have done indicate that after the file contains only deleted or occupied entries (all empty entries have disappeared) all the moments except the means of the random variables **psl** and **bmin** converge to values that depend only on the load factor α and the bucket size b but not on the number of buckets n. The expected value of the gap between the smallest and largest values in the array *bmin* does depends on the value of n even if deletions have not been performed.

References

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

[2] Bhat, U.N., *Elements of Applied Stochastic Processes,* John Wiley & Sons, New York 1972

[3] Brent, R.P., "Reducing the Retrieval Time of Scatter Storage Techniques," *Communications of the ACM,* Vol. 16, No. 2, pp.105-109, February 1973

[4] Brayton, R.K., "On the Asymptotic Behavior of the Number of Trials Necessary to Complete a Set with Random Selection", *J. Math. Anal. Appl.,* Vol. 7, pp.31-61, 1963

[5] Celis, P., P.-Å Larson and J.I. Munro "Robin Hood Hashing" *Proc. 26th Annual IEEE Symposium on Foundations of Computer Science,* pp.281-288, October 1985

[6] Celis, P. "Robin Hood Hashing" Ph. D. Thesis, University of Waterloo, January 1986.

[7] David, F.N. and D.E. Barton, *Combinatorial Chance,* Charles Griffin & Company Limited, London 1962

[8] Gonnet, G.H. and J.I. Munro, "Efficient Ordering of Hash Tables", *SIAM Journal on Computing,* Vol. 8, No. 3, pp.463-478, August 1979 (a preliminary version was presented at the 9th ACM STOC May 1977)

[9] Gonnet, G.H. and P.-Å. Larson, "External Hashing with Limited Internal Storage", *Technical report CS-82-38,* Computer Science Dept., Univ. of Waterloo, October 1982

[10] Johnson, N.L. and S. Kotz, *Urn Models and their Application,* John Wiley & Sons, New York 1977

[11] Larson, P.-Å., "Analysis of Uniform Hashing", *Journal of the ACM,* Vol. 30, No. 4, pp.805-819, October 1983

[12] Larson, P.-Å., and M.V. Ramakrishna, "External Perfect Hashing" *Proc. of ACM-SIGMOD 1985 Int. Conf. on Management of Data,* 1985, 190-200

[13] Mallach, E.G., "Scatter Storage Techniques: A Unifying Viewpoint and a Method for Reducing Retrieval Times", *The Computer Journal,* Vol. 20, No. 2, pp.137-140, May 1977

[14] Poonan, G., "Optimal Placement of Entries in Hash Tables", *ACM Computer Science Conference (Abstract only),* Vol. 25, 1976, (Also DEC Internal Tech. Rept. LRD-1, Digital Equipment Corp., Maynard Mass)

[15] Rényi, A., "Three New Proofs and a Generalization of a Theorem of Irving Weiss", *Magy. Tud. Akad. Mat. Kutató Int. Kōzl.,* Vol. 7 pp.200-209, 1962

[16] Rivest, R.L., "Optimal Arrangements of Keys in a Hash Table", *Journal of the ACM,* Vol. 25, No. 2, pp.200-209, April 1978

```
function insert(Record)
        if $m = n\,b$ then return($FAIL$)        { table full }
        $k := Key(Record)$
        $probetry := smallest$
        while $k <> nil$ do
              $probetry := probetry + 1$
              $bucket := H(k, probetry)$
              if $probetry > fix(bmin[bucket])$ then
                retrieve($bucket$)
                if $nonfull(bucket)$ then
                  replace empty or deleted entry
                  $k := nil$
                else
                  swap Record and entry with smallest probe choice.
                  $k := Key(Record)$
                endif
                write($bucket$)
                $adjust(bucket)$        { fix bmin, smallest and counters if needed }
              endif
        endwhile
        return($bucket$)

function search(k)
        $probetry := smallest$
        while $true$ do
              $bucket := H(k, probetry)$
              case $(fix(bmin[bucket])$
                   $> probetry$ : $probetry := probetry + 1$
                   $= probetry$ : retrieve($bucket$)
                                  if $found(bucket)$ then return($bucket$)
                                  else $probetry := probetry + 1$
                   $< probetry$ : retrieve($bucket$)
                                  if $found(bucket)$ then return($bucket$)
                                  else return($FAIL$)
              endcase
        endwhile

function adjust(bucket)
        $t := findbucketmin(bucket)$
        if $t > fix(bmin[bucket])$ then
          $count[bmin[bucket]] = count[bmin[bucket]] - 1$
          $count\left[t \bmod 2^d\right] = count\left[t \bmod 2^d\right] + 1$
          $bmin[bucket] = t \bmod 2^d$
          while $count\left[smallest \bmod 2^d\right] = 0$ do
                $smallest = (smallest + 1) \bmod 2^d\,n$
          endwhile
        endif
```

Figure 6: Search and insert algorithms