ROBIN HOOD HASHING*

(Preliminary Report)

Pedro Celis, Per-Åke Larson, J. Ian Munro

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

Abstract

This paper deals with hash tables in which conflicts are resolved by open addressing. The initial contribution is a very simple insertion procedure which (in comparison to the standard approach) has the effect of dramatically reducing the variance of the number of probes required for a search. This leads to a new search procedure which requires only a constant number of probes, on average, even for full tables. Finally, an extension to these methods yields a new, simple way of performing deletions and subsequent insertions. Experimental results strongly indicate little degeneration in search time. In particular deletions and successful searches appear to require constant time (< 2.57 probes) and insertions and unsuccessful searches, $O(\log n)$.

1. Introduction

If more than $\sqrt{\pi n/2}$ or so elements are hashed into a table of size n, we are almost certain to have more than one element map to the same location. Some sort of conflict resolution scheme must be devised. Two basic approaches can be followed: store explicit information on where to look next (e.g. chaining) or use the key to generate a permutation of the table locations to follow in a search. The latter alternative, known as open addressing, is the basic topic of this paper. Double hashing is probably the most efficient method based on open addressing. Its behaviour appears to be almost undistinguishable from a random permutation.

The usual approach to organizing hash tables with conflict resolution by open addressing is a "first come first serve" arrangement. When an element is to be inserted, the locations of its probe sequence are examined sequentially until an empty spot is found. The new element is put in that location. If αn elements are inserted in a table of size n, the expected number of probes for a successful search is $-\alpha^{-1}\ln(1-\alpha)$ and for an unsuccessful search $(1-\alpha)^{-1}$. These numbers become $\ln n+\gamma+o(1)$ and n, respectively, if the table is full. A number of proposals have been made for rearranging the elements in the table ([B], [GM], [M], [R]); all of which boast constant average time for successful searches even for a full table. All can be coupled with the notion of retaining the length of the longest probe sequence in the table to give a bound on the number of locations one must inspect before declaring an element absent. For full tables, the standard approach takes $.63..n$, and Brent's method seems to take $\Theta(\sqrt{n})$ probes for an unsuccessful search, while the others appear to require $\Theta(\log n)$ on average. On the other hand, we may also want to take into account the cost of setting up a table. The best known methods for computing the optimal arrangement of the elements in the table (minimizing the average number of probes for a successful search) requires $\Theta(n^2 \log n)$ time and $\Theta(n)$ extra storage, while Brent's method requires $\Theta(n \log n)$ time and constant extra storage. The trade-offs are many and complex.

A natural goal is to set up a table under which, even if the table is full, successful searches require an average number of probes bounded by a small constant, and no successful search (and

* This work was supported by Natural Sciences and Engineering Research Council under grants A2460 and A8237. Much of this work was done while the third author was on leave at AT&T Bell Laboratories.

hence no unsuccessful search) takes more than $\Theta(\log n)$ time ($\ln n$ is a lower bound on this quantity). Coupled with this we want $\Theta(n \log n)$ set up time ... with a small constant, O(1) extra storage, and easy code. If the table is not full all these costs should scale down appropriately.

It suffices to say that no previously suggested method comes close to meeting all these criteria. The methods proposed here do. Furthermore, our approach leads to a very simple mechanism for supporting deletions and subsequent insertions that does not appear to cause significant performance degradation. As far as we know, no previous work has shed encouraging light on this problem for the case in which conflicts are resolved by open addressing.

Finally, we would like to emphasize that the contribution of this paper has three more or less equal components: simple but new algorithms, mathematical analysis, and rather extensive experiments. The experiments provide some of the most interesting results, and also led us to the appropriate conjectures for mathematical proofs.

2. Robin Hood Hashing

We will present our techniques in the order in which they were developed, and with their original motivation. We emphasize that we think of the hash function as producing a permutation of $(0, \ldots, n-1)$. In particular the first, next or even the ith element of such a permutation is assumed to be easily computed. Double hashing and linear probing, for example, have this property.

Our initial thesis was that the worst aspect of the standard technique, especially when the table is heavily loaded, is not so much the average search time, but the high cost of a few searches (i.e. the high variance). This led to the following insertion scheme:

> Suppose element A is in location l, its ith probe choice, and B is to be inserted (or moved). Suppose B has been denied allocation of its first $j-1$ choices and we are now considering l, its jth choice. Then
>
> if $i > j$: A retains l, the insertion of B continues by considering its $(j+1)$st choice.
>
> if $i < j$: B displaces A from l, the insertion procedure for A is applied starting with its next $(i+1)$st choice.
>
> if $i = j$: break the tie by, for example, assigning the element with the lower key value to l.

This procedure of "taking from the rich and giving to the poor" gives rise to the name Robin Hood hashing. Like Robin Hood, it does not change the average wealth (mean probe length), only its distribution. It is clear that the principal effect is to substantially reduce the variance. That it reduces the variance to a small constant, however, was a surprise to us.

Let **psl** be a random variable denoting the length of a probe sequence of a stored record. As is customary in such analyses we will assume the probe sequence of any element is obtained by Random Probing, i.e., a random (sampling with replacement) sequence of locations. The difference between this and a random permutation (sampling without replacement) is negligible.

Theorem 1: *The average length of a probe sequence in a hash table loaded using the Robin Hood heuristic and based on the hash function Random Probing, is the same as for the standard method, namely, for non-full tables*

$$E[\mathbf{psl}] = \frac{m}{n}\left[H_m - H_{m-n}\right]$$

$$\approx -\alpha^{-1}\ln(1-\alpha) + O\left(\frac{1}{m-n}\right)$$

and for full tables

$$E[\mathbf{psl}] = \ln n + \gamma + O(n^{-1}).$$

Proof idea: The distribution of the remaining probe sequence is the same for all the colliding records when using Random Probing. Therefore, any criterion for selecting a record to be rejected, which does not use any knowledge about the future probe sequence of the colliding records, produces the same $E[\mathbf{lpsl}]$. □

Theorem 2: *In a Robin Hood hash table, with load factor $\alpha = m/n$, the probability p_i that a record is placed in the i-th or further position in its probe sequence converges to*

$$p_1(\alpha) = 1$$

$$p_{i+1}(\alpha) = 1 - \left(\frac{1-\alpha}{\alpha}\right)\left(e^{\alpha(p_1(\alpha)+\ldots+p_i(\alpha))} - 1\right)$$

when $n, m \to \infty$. *If the table is completely full these values converge to* $p_1(n) = 1$ *and* $p_{i+1}(n) = 1 - \frac{1}{n}(e^{p_1(n)+\ldots+p_i(n)} - 1)$.

The lengthy proof of this theorem is given in [C].

The variance of psl

The asymptotic distribution above gives the expected values $-\alpha^{-1}\ln(1-\alpha)$ for non-full tables and $\ln(n+1)$ for full tables.

The variance of the length of probe sequence under the Robin Hood heuristic, as $n \to \infty$, can easily be computed from the distribution:

$$\mathbf{V}(\alpha) = \sum_{i=1}^{n} (i + \alpha^{-1}\ln(1-\alpha))^2 (p_{i+1}(\alpha) - p_i(\alpha))$$

for load factor α and

$$\mathbf{V}(n) = \sum_{i=1}^{n} (i - \ln(n+1))^2 (p_{i+1}(n) - p_i(n))$$

for full tables.

The variance under the standard insertion method is $2n - O(\ln n)$ for a full table. Table 1, based on the asymptotic result of Theorem 2, illustrates this dramatic improvement.

n	Robin Hood hashing	Standard approach
10^3	1.82257	1.9×10^3
10^6	1.88235	2.0×10^6
10^9	1.88256	2.0×10^9
10^{12}	1.88256	2.0×10^{12}

Table 1: Variance of the search length for full tables

In the remainder of this section we will sketch a proof that this variance remains bounded as $n \to \infty$.

Define $s_i(n)$ to be $s_i(n) = 1 - p_i(n) + 1/n$. Then $s_{i+1}(n) = e^{p_1(n)+\ldots+p_i(n)} = e^{1+\frac{1}{n}} e^{s_i(n)} s_i(n)$. $s_i(n)$ is a probability distribution plus a $1/n$ term. When we increase n by a factor of e, the distribution is only slightly perturbed, but shifted one position to the right.

Lemma 1: $s_{i+1}(ne) = s_i(n) e^{O\left(\frac{i}{ne}\right)}$.

Proof [By Induction]: Basis: for $i = 1$ we have that

$$s_1(n) = \frac{1}{n}, \qquad s_2(ne) = \frac{1}{n}.$$

Inductive Step: assume true for all $i < k$. The induction hypothesis is therefore

$$s_k(ne) = s_{k-1}(n) e^{O\left(\frac{k-1}{ne}\right)}.$$

Now prove for $i = k$.

$$s_{k+1}(ne) = e^{1+\frac{1}{ne}} e^{-s_k(ne)} s_k(ne)$$

and by the induction hypothesis

$$= e^{1+\frac{1}{n}} e^{-s_{k-1}(n)} s_{k-1}(n) e^{\frac{1}{ne} - \frac{1}{n} - s_{k-1}(n) O(\frac{k-1}{ne}) + O(\frac{k-1}{ne})}$$

$$= s_k(n) e^{O(\frac{k}{ne})}. \qquad \square$$

Lemma 2:

$$\sum_{i=\sqrt{6n\ln n}}^{\infty} (i - \ln n + \gamma)^2 \left[p_i(n) - p_{i+1}(n)\right] = O\left(\frac{\ln n}{n}\right).$$

Proof:

$$\sum_{i=\sqrt{6n\ln n}}^{\infty} (i - \ln n + \gamma)^2 \left[p_i(n) - p_{i+1}(n)\right]$$

$$< \sum_{i=\sqrt{6n\ln n}}^{\infty} (i - \ln n + \gamma)^2 p_i(n)$$

$p_i(n)$ is the probability that a record is in at least position i. This probability is less than the probability of the longest probe sequence being i or more. Let **lpsl** be random variable whose value is the longest probe sequence in the table.

$$< \sum_{i=\sqrt{6n\ln n}}^{\infty} (i - \ln n + \gamma)^2 \mathbf{Pr}\{\mathbf{lpsl} \geq i\}.$$

Let **N** be the total cost of the table. The probability that the longest probe sequence is of length l or more is less than the probability that the total cost of the table is $l^2/2$ or more

$$< \sum_{i=\sqrt{6n\ln n}}^{\infty} (i - \ln n + \gamma)^2 \mathbf{Pr}\{\mathbf{N} \geq i^2/2\}$$

Using the limiting distribution for **N** [ER] we have that

$$= \sum_{i=\sqrt{6n\ln n}}^{\infty} (i - \ln n + \gamma)^2 \times \left[1 - \exp\left(\frac{-e^{-(i^2/2 - n\ln n)}}{n}\right)\right]$$

$$= \sum_{i=0}^{\infty} \left(i + \sqrt{6n\ln n} - \ln n + \gamma\right)^2 \times \left[1 - \exp\left(\frac{-e^{-((i+\sqrt{6n\ln n})^2/2 - n\ln n)}}{n}\right)\right]$$

$$< \sum_{i=0}^{\infty} (i + \sqrt{6n\ln n} - \ln n + \gamma)^2 \times \frac{e^{-i^2/2 + i\sqrt{6n\ln n} + 2n\ln n}}{n}$$

$$= \sum_{i=0}^{\infty} \frac{(i + \sqrt{6n\ln n} - \ln n + \gamma)^2}{n^{2 + \frac{i^2}{2n\ln n} + \frac{i\sqrt{6}}{\sqrt{n\ln n}}}}$$

$$= O\left(\frac{\ln n}{n}\right) \qquad \square$$

Using the previous two lemmas we can now prove the following.

Lemma 3: Let $\mathbf{V}(n)$ be the variance of **psl** for a full Robin Hood table. Then:

$$\mathbf{V}(ne) = V(n)e^{O\left(\left(\frac{\ln n}{ne}\right)^{\frac{1}{2}}\right)} + O\left(\frac{\ln^2 ne}{ne}\right).$$

Proof:

$$\mathbf{V}(n) = \sum_{i=1}^{\infty} (i - \ln n + \gamma)^2 \left[p_i(n) - p_{i+1}(n)\right]$$

$$= \sum_{i=1}^{\infty} (i - \ln n + \gamma)^2 \left[s_{i+1}(n) - s_i(n)\right]$$

$$\mathbf{V}(ne) = \sum_{i=1}^{\infty} (i - \ln(ne) + \gamma)^2 \left[p_i(ne) - p_{i+1}(ne)\right].$$

Using Lemma 2 and the fact that $p_1(ne) - p_2(ne) = \frac{e-1}{ne}$

$$= (1 - \ln(ne) + \gamma)^2 \frac{e-1}{ne}$$

$$+ \sum_{i=2}^{\sqrt{6n\ln n - 1}} (i - \ln n - 1 + \gamma)^2 \times \left[s_{i+1}(ne) - s_i(ne)\right] + O\left(\frac{\ln ne}{ne}\right)$$

$$= \sum_{i=1}^{\sqrt{6n\ln n - 2}} (i - \ln n + \gamma)^2 \left[s_i(ne) - s_{i-1}(ne)\right] + O\left(\frac{\ln^2 ne}{ne}\right)$$

using Lemma 1

$$= \sum_{i=1}^{\sqrt{6n\ln n - 2}} (i - \ln n + \gamma)^2 \left[s_{i+1}(n) - s_i(n)\right] e^{O(\frac{i}{ne})} + O\left(\frac{\ln^2 ne}{ne}\right)$$

$$= \mathbf{V}(n) e^{O\left(\left(\frac{\ln n}{ne}\right)^{\frac{1}{2}}\right)} + O\left(\frac{\ln^2 ne}{ne}\right). \qquad \square$$

We are now ready to prove the main result of this section.

Theorem 2: *For a completely full Robin Hood hash table, the variance of the length of the probe sequence is* $\mathbf{V}\left[\mathbf{psl}\right] = O(1)$.

Proof: We have defined $V(n)$ as the variance for a full table of size n. Choose n_0 to be some large constant. From the previous lemma we know that

$$\mathbf{V}(n_0 e) = \mathbf{V}(n_0) e^{O\left(\left(\frac{\ln n_0}{n_0 e}\right)^{\frac{1}{2}}\right)} + O\left(\frac{\ln^2 n_0 e}{n_0 e}\right).$$

Bootstrapping this equation we get

$$\mathbf{V}(n_0\, e^k) = \left[\mathbf{V}(n_0) + O\left(\sum_{i=1}^{k} \frac{\ln^2 n_0 e^i}{n_0 e^i}\right)\right] e^{O\left(\sum_{i=1}^{k} \left(\frac{\ln n_0 e^{i-1}}{n_0 e^i}\right)^{\frac{1}{2}}\right)}.$$

If we let $k \rightarrow \infty$ we get

$$\mathbf{V}(\infty) = \left[\mathbf{V}(n_0) + O\left(\sum_{i=1}^{\infty} \frac{\ln^2 n_0 e^i}{n_0 e^i}\right)\right]$$

$$e^{O\left(\sum_{i=1}^{\infty}\left(\frac{\ln n_0 e^{i-1}}{n_0 e^i}\right)^{\frac{1}{2}}\right)}$$

and since n_0 is just a constant

$\mathbf{V}(\infty) = O(1)$. □

Basically the same proof could be used to show that higher moments are also $O(1)$, which indicates, that as n goes to infinity, the whole probability distribution drifts to the right but changes very little otherwise.

3. The Length of the Longest Probe Sequence

Assume we have a set $R = \{\mathbf{R}_1, \ldots, \mathbf{R}_m\}$ of m records stored in a hash table. Now make the following definitions:

- Let $\sigma : R \mapsto \{0, \ldots, n-1\}$ represent the table assignment, such that $\sigma(\mathbf{R})$ is the table location in which $\mathbf{R}$ is stored.
- Let $w : R \mapsto \{1, \ldots, n\}$ be such that, $w(\mathbf{R})$ is the position in the probe sequence of $\mathbf{R}$ of the location in which it is stored. In other words $h(\mathbf{R}, w(\mathbf{R})) = \sigma(\mathbf{R})$.
- Let $\theta : R \times \{0, \ldots, n\} \mapsto R$ be the backup function, defined as: $\theta(\mathbf{R}, j) = \sigma^{-1}(h(\mathbf{R}, w(\mathbf{R}) - j))$, that is, the record that is occupying the location that the record $\mathbf{R}$ probed j steps before $\mathbf{R}$'s current location.

Assume that the value of the longest probe sequence length (**lpsl**) is l, that is, at least one record is in the l-th position of its probe sequence and none occur later. Consider the following intuitive argument: Denote one such record by $\mathbf{R}_{worst}$. Let $V_0 = \{\mathbf{R}_{worst}\}$ be a set of records which are all in at least their l-th probe positions. The location $\mathbf{R}_{worst}$ probed in its $(l-1)$-st choice must contain a record in at least its $(l-1)$-st probe. So there are at least two records, $\mathbf{R}_{worst}$ and $\theta(\mathbf{R}_{worst}, 1)$, in at least their $(l-1)$-st probes. Let $V_1 = \{\mathbf{R}_{worst}, \theta(\mathbf{R}_{worst}, 1)\}$ be a set of records which are all in at least their $l-1$-st probes. Similarly we can move back both records and have that at least 4 records,
$(\mathbf{R}_{worst}, \theta(\mathbf{R}_{worst}, 1), \theta(\mathbf{R}_{worst}, 2), \theta(\theta(\mathbf{R}_{worst}, 1), 1))$, are in at least their $(l-2)$-nd probes. Let $V_2 = \{\mathbf{R}_{worst}, \theta(\mathbf{R}_{worst}, 1), \theta(\mathbf{R}_{worst}, 2), \theta(\theta(\mathbf{R}_{worst}, 1))\}$ be a set of records which are all in at least their $l-2$-nd probes. There is a minor problem with this line of reasoning as we are sampling the table with replacement, so the cardinalities of the last two sets are not necessarily 2 and 4.

We can, however, develop this into a more accurate analysis, starting again at one arbitrary chosen element in its l-th probe position. Let V_i denote a set of records that are in at least the $(l-i)$-th probe position and U_{i+1} the set of records that are stored in the locations that the records in the set V_i would probe if moved back one further position. More formally,

$$U_0 = \{\mathbf{R}_{worst}\}$$
$$V_0 = \{\mathbf{R}_{worst}\}$$
$$U_i = \{\mathbf{R} \mid \mathbf{R} = V(\mathbf{R}', j)$$
$$\text{for some } \mathbf{R}' \in U_{i-j},\ 1 \le j \le i\}$$
$$V_i = V_{i-1} \cup U_i.$$

Each record in the set V_i is in at least its $(l-1)$-th probe position. Since we are sampling the locations with replacement, the cardinality of V_i is a random variable denoted by $\mathbf{v}_i$.

$\mathbf{v}_i$ can be modeled as an occupancy distribution [JK], where the number of urns is m and the number of balls dropped is 2^i, and we are interested in the number of different urns that contain all the balls. We therefore have that $\mathbf{E}[\mathbf{v}_i] = m(1-(1-1/m)^{2^i})$. We will need the following lemma for our next theorem.

Lemma 4: $\sum_{i=0}^{\infty}\left(1-\frac{1}{m}\right)^{2^i} < \lceil lg\, m \rceil$.

Theorem 3: *The expected value of* **lpsl** *for a Robin Hood table with m records is bounded by* $\mathbf{E}[\mathbf{psl}] \le \mathbf{E}[\mathbf{lpsl}] < \mathbf{E}[\mathbf{psl}] + \lceil lg\, m \rceil$.

Proof: Define $\mathbf{r}$ to be the sum of probe positions of the records in V_{l-1}. It follows that

$$\mathbf{r} \ge \mathbf{v}_0 + \mathbf{v}_1 + \cdots + \mathbf{v}_{l-1}$$

and therefore

$$\mathbf{E}[\mathbf{r} \mid \mathbf{lpsl} = l] \ge \sum_{i=0}^{l-1} m\left(1 - \left(1-\frac{1}{m}\right)^{2^i}\right)$$
$$= ml - m\sum_{i=0}^{l-1}\left(1-\frac{1}{m}\right)^{2^i}$$

$$> ml - m \sum_{i=0}^{\infty} \left(1 - \frac{1}{m}\right)^{2^i}$$

$$> ml - m\lceil lg\ m \rceil.$$

It follows that

$$\mathbf{E}[\mathbf{r}] > m(\mathbf{E}[\mathbf{lpsl}] - \lceil lg\ m \rceil).$$

Since V_{l-1} is just some subset of R then $\mathbf{E}[\mathbf{N}] \geq \mathbf{E}[\mathbf{r}]$ and

$$m\mathbf{E}[\mathbf{psl}] > m(\mathbf{E}[\mathbf{lpsl}] - \lceil lg\ m \rceil).$$

Solving for $\mathbf{E}[\mathbf{lpsl}]$ we get

$$\mathbf{E}[\mathbf{lpsl}] < \mathbf{E}[\mathbf{psl}] + \lceil lg\ m \rceil.$$

The lower bound holds because $\mathbf{psl} \leq \mathbf{lpsl}$ for every table . □

Corollary: *The expected value of* **lpsl** *for a full Robin Hood hash table is* $\Theta(\ln n)$.

Proof: Follows from the theorem and the fact that $\mathbf{E}[\mathbf{psl}] = \Theta(\ln n)$ for a full Robin Hood hash table. □

Our extensive experiments indicate that, the longest sequence for full tables is about $1.15 \ln n + 2.5$. The bounds, and so the heuristic, are within a constant factor of what we would get if the table is organized to minimize the length of the longest probe sequence. These lower bounds are $\lceil -\alpha^{-1}\ln(1-\alpha) \rceil$ and $\ln n + \gamma + o(1)$ but require solving the appropriate assignment problem in potentially $\Theta(n^2 \log n)$ time [G].

Based on the initial thesis, that a major drawback to the usual insertion scheme is the high variance, we feel we have made an interesting improvement. However if the probe choice of a record cannot be computed in constant time from a key value and the location in which it is found, the cost of completely filling a table will increase from $\Theta(n \log n)$ for the standard algorithm to $\Theta(n \log^2 n)$ for the Robin Hood heuristic. (This is based on determining the probe choice by performing a standard search for the record). The other difficulty is, of course, that no improvement has been made in the mean search time. Both problems are addressed in the next section.

4. Smart Searching

Given that we know the expected position of an element with respect to its probe sequence (either from the theorems or by explicitly keeping track of this value) it seems reasonable to look first in that expected position. More formally, we propose the following search procedure:

> Let t denote (Σ length probe sequence /# elements); *pfirst*, the length of the shortest probe sequence (this will probably be 1); and *plast*, the length of the longest. Then to search for a key first try its t-th choice (this is trivially computable for methods such as double hashing), then $t+1$, $t-1$, $t+2$, $t-2$, ... to cover the range *pfirst* to *plast*. If the element is not found within this range, it is not present in the table.

For lack of a better name we will call this approach smart search. The expected number of probes for an unsuccessful search is, of course, no more than under the conventional scheme. On the other hand, successful searches are substantially improved as a consequence of the extremely low variance.

Theorem 4. *The expected number of probes for a successful search using the smart search algorithm above is*

$$\sum_{i=1}^{t} (2(t-i)+1)(p_{i+1}-p)$$

$$+ \sum_{i=t+1}^{2t} 2(i-t)(p_{i+1}-p_i) + \sum_{i=2t+1}^{n} (i-t)(p_{i+1}-p_i)$$

where $t = \lfloor \ln(n) + \gamma \rfloor$ *for a full table, and* $t = \lfloor -\alpha^{-1}\ln(1-\alpha) \rfloor$ *for a table with load factor* α.

There are several possible variations of the above search heuristic. The initial probe choice, t, can be estimated slightly differently and some sequence other than $t+1$, $t-1$, $t+2$, $t-2$, ... can be used.

Define a mean centered search heuristic to be one that finds a key that is k positions above (or below) the mean within ck probes, for some constant c. We then have the following:

Theorem 5: *Any mean centered approach for searching a Robin Hood hash table has an expected search cost of* $O(1)$.

Proof: The number of probes made before reaching a location that is l steps away from $t = \lfloor \mathbf{E}[\mathbf{psl}] \rfloor$ is $\Theta(l)$. The cost of searching will then be

$$\begin{aligned}\mathbf{E}[\text{search cost}] &= \sum_{l=0}^{\infty} O(l)\mathbf{Pr}\{|\mathbf{psl} - t| = l\} \\ &= O\left(\sum_{l=1}^{\infty} \mathbf{Pr}\{|\mathbf{psl} - \mathbf{E}[\mathbf{psl}]| \geq l\}\right) \\ &= O\left(\sum_{l=1}^{\infty} \frac{\mathbf{V}[\mathbf{psl}]}{l^2}\right) \\ &= O\left(\frac{\mathbf{V}[\mathbf{psl}]\pi^2}{6}\right).\end{aligned}$$

Since $\mathbf{V}[\mathbf{psl}]$ is $O(1)$, we conclude that $\mathbf{E}[\text{search cost}] = O(1)$. □

The minimal average search length is achieved by a organ-pipe search: the different probe choices are tried in strictly decreasing order of the probability of finding a record in that probe choice. We can easily keep track of these probabilities by keeping a list of counters, counting the number of records in probe choice i for $i \in [pfirst, plast]$. The counters are updated as records are inserted. From our experiments it appears that $1.5 \ln n$ counters is sufficient, even for a full, table. The expected search length for a full table using the search procedure above and using organ-pipe search are shown in Table 2 below.

n	Organ-Pipe search	Smart search
10^3	2.5429	2.6141
10^6	2.5469	2.6527
10^9	2.5499	2.6873
10^{12}	2.5648	2.7330

Table 2: Expected search length for two different search procedures (full table)

A consequence of the fast search scheme is that as insertions are being made the probe choices of conflicting elements can be found in constant time on the average. As a result the cost of setting up even a full table is competitive with the standard method.

Theorem 7. *A table may be organized under the Robin Hood heuristic in $\Theta(n \ln n)$ time on the average. This time is within a (small) constant factor of the set up time under the standard algorithm.*

5. Updates

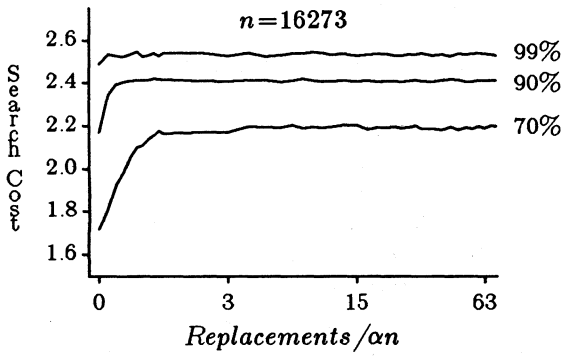
As far as we know, there have been no "positive" results published on doing deletions and subsequent insertions on hash tables with open addressing. The only previously reported technique that does not cause the table to degenerate dramatically is conflict resolution by linear probing. If that method is employed and the load factor is moderately high, searches are very costly even without updates (i.e. dramatic degeneration occurs even without deletions). The method we present here seems to lead to a scheme under which deletions leave the table reusable and whose performance is no worse than that of a full table in which no deletions were made. The basic idea is to think of the hash function as producing not just a sequence of locations but indeed a cycle on which we may choose an arbitrary starting point. We will keep track of the earliest and latest in their respective cycles that any element in the table occupies (*pfirst* and *plast*). This means all elements in the table lie in the range (mod n) *pfirst* to *plast* inclusive. The value *plast* is easy to keep track of, *pfirst* is trickier. A small list (size $1.5 \ln n$ seems adequate) can be used to keep track of the number of keys in choice i for all $i \in [pfirst, plast]$. This list is easily updated during insertions and deletions and also facilitates organ-pipe search.

Searches will be performed using either organ-pipe or the smart search of Section 3; and deletions are performed by finding and flagging elements. (We will require the probe choice for deleted values and so must be able to rederive this quickly, the actual key value is not necessary.) The insertion procedure is a natural extension of the Robin Hood heuristic:

> To insert a key we start probing at its choice *pfirst* (rather than 1), and proceed as described earlier. The twist occurs when a deleted element is encountered. A deleted element is displaced if and only if it would be displaced if it were not flagged as deleted. When a deleted element is displaced, it is discarded and the update is completed.

The method is very simple, easily coded and, based on some extensive experiments, works extremely well; although, of course, insertions into a nearly full table will be costly as for any other scheme. If a table is filled completely and then a long sequence of delete/insert pairs is performed the average search cost appears not to change. If the table is filled to some moderate load factor (e.g.

$\alpha = .8$) and then a delete/insert pair sequence is initiated, the average search cost seems to go rather quickly to about half way between that of a full table and one with load factor α on which no deletions had been performed. The figure below shows the average search cost for such tables of size 16273 at various load factors when a organ-pipe search is used. A full version of this paper will include these and other experimental results.



References

[B] Brent, R.P., Reducing the Retrieval Time of Scatter Storage Techniques, *Communications of the ACM, 16*, 2 (February 1973), 105-109.

[C] Celis, P., Robin Hood Hashing, Ph.D. thesis (in preparation), University of Waterloo, Canada, 1985.

[G] Gonnet, G.H., Expected Length of the Longest Probe Sequence in Hash Code Searching, *Journal of the ACM, 28*, 2 (April 1981), 289-304.

[GM] Gonnet, G.H., and J.I. Munro, Efficient Ordering of Hash Tables, *SIAM Journal on Computing, 8*, 3 (August 1979), 463-378.

[M] Mallach, E.G., Scatter Storage Techniques: A Unifying Viewpoint and a Method for Reducing Retrieval Times, *The Computer Journal, 20*, 2 (May 1977), 137-140.

[R] Rivest, R.L., Optimal Arrangements of Keys in a Hash Table, *Journal of the ACM, 25*, 2 (April 1978), 200-209.

[ER] Erdös, P., and A. Rényi, On a Classical Problem of Probability Theory, Magy. Tud. Akad. Mat. Kutató Int. Közl, Vol. 6, pp. 215-220, 1961.