LABORATORY FOR
COMPUTER SCIENCE
(formerly Project MAC)

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TM-73

OPTIMAL ARRANGEMENT OF KEYS IN A HASH TABLE

RONALD L. RIVEST

JULY 1976

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

MIT/LCS/TM-73

OPTIMAL ARRANGEMENT OF KEYS IN A HASH TABLE

Ronald L. Rivest

July 1976

MIT/LCS/TM-73

OPTIMAL ARRANGEMENT OF KEYS IN A HASH TABLE

RONALD L. RIVEST

JULY 1976

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LABORATORY FOR COMPUTER SCIENCE
(FORMERLY PROJECT MAC)

CAMBRIDGE MASSACHUSETTS 02139

Optimal Arrangement of Keys in a Hash Table*

Ronald L. Rivest
Dept. of Electrical Engineering and
Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

July 1976

Abstract

When open addressing is used to resolve collisions in a hash table, a given set of keys may be arranged in many ways; typically this depends on the order in which the keys are inserted. We show that arrangements minimizing either the average or worst-case number of probes required to retrieve any key in the table can be found using an algorithm for the assignment problem. The worst-case retrieval time can be reduced to $O(\log_2(M))$ with probability $1-\epsilon(M)$, when storing M keys in a table of size M, where $\epsilon(M) \to 0$ as $M \to \infty$. We also examine insertion algorithms to see how to apply these ideas for a dynamically changing set of keys.

Keywords

hashing, collision resolution, searching, assignment problem, optimal algorithms, data base organization.

CR categories: 3.74, 5.41

* This research was prepared with the support of the National Science Foundation under research grant no. GJ-43534X, contract no. DCR74-12997, and research grant no. MCS76-14294.

"Spread the table and contention will cease."

Old English proverb
[8, #272.6]

1. Introduction

We consider schemes to optimize the placement of keys in a hash table when open addressing is used to resolve collisions. This section reviews the basic definitions and algorithms; the following sections present the optimization algorithms and analyze the number of probes requires to retrieve a key from an optimized table.

Let $\mathcal{K} = \{K_1, K_2, \ldots, K_N\}$ be a set of N keys, and let an array T_i, for $1 \le i \le M$ be a set of M memory locations (the hash table) which will be used to store $\mathcal{K}$. Each table position may hold either a single key or the special symbol empty. We assume $N \le M$. When open addressing is used to resolve collisions a "hashing function"

$$h : U \times \{1,2,\ldots,M\} \to \{1,2,\ldots,M\}$$

is used, mapping the set U of all possible keys (that is, $\mathcal{K}$ may be any N-subset of U) and probe numbers into the set of memory locations. We assume for any key $K \in U$ that the sequence $h(K,1), h(K,2), \ldots, h(K,M)$ is a permutation of $\{1,2,\ldots,M\}$. To store the key K in the table the locations $T_{h(K,1)}, T_{h(K,2)}, \ldots$ are successively examined until an empty location is found or until K is found already present in the table. The following program makes this precise.

Insertion Algorithm

Input: a key K, a hash table T, a hash function h.

Output: None. T is modified to contain K, unless K is already present.

Procedure:

```
j := 0;
repeat   j := j + 1;
         i := h(K,j);
         if T_i = empty then T_i := K
until    T_i = K;
```

Note that T must contain at least one empty location if K is not already in the table, if the loop is to terminate properly. The value of j at termination, which is the number of probes required to insert K, is taken to be the cost of inserting K.

A similar procedure searches for the presence of a key K in T (replace the assignment statement "T_i := K" by "return (K not present)"). If the repeat loop terminates normally then T_i contains the previously stored key K. The value of j at termination is taken to be the cost of searching for K.

Knuth [4] studies hashing algorithms in detail, giving alternative methods for handling "collisions" (the case when $h(K_i,1) = h(K_j,1)$ for $K_i \neq K_j$) and several open-addressing hash functions h. The reader who is unfamiliar with hashing algorithms should find it profitable to consult his text.

2. Optimal Arrangements

The arrangement of the keys $\mathcal{K}$ in the hash table depends on the order in which they were inserted. For example, let U be the set of natural numbers and let h(K,j) be the jth decimal digit of K. Inserting the set

$$\mathcal{K} = \{1423, 1234, 3412, 2341\}$$

into an empty table in that order results in the arrangement α:

location:	1	2	3	4
contents:	1423	1234	3412	2341

whereas inserting them in the order 1234, 2341, 1423, 3412 results in α':

location:	1	2	3	4
contents:	1234	2341	3412	1423

Let α: $\mathcal{K} \rightarrow \{1,2,\ldots,M\}$ be called an <u>arrangement</u>; $\alpha(K_i) = j$ means that $T_j = K_i$. Of couse α must be one-to-one. Let $A(\mathcal{K},M)$ denote the set of all arrangements of $\mathcal{K}$ in $T_1,\ldots,T_M$.

Let $p(K,\alpha)$ denote the number of probes required to retrieve a key K under arrangement α; the average

$$avg(\alpha) = \frac{1}{N} \sum_{k\epsilon} p(K,\alpha)$$

and worst-case

$$wc(\alpha) = \max\{p(K,\alpha) \mid K \in \mathcal{K}\}$$

number of probes to retrieve any key in T are then definable. We have $avg(\alpha) = 7/4$, $wc(\alpha) = 3$, $avg(\alpha') = 5/4$, and $wc(\alpha') = 2$ in the above examples.

Hashing is often used to store a dynamically changing set of keys. We first study in this section the simpler problem of optimally arranging a given set $\mathcal{K}$ of keys in a hash table; in section 4 we consider the problem of maintaining optimality as $\mathcal{K}$ changes.

Define an arrangement $\alpha \in A(\mathcal{K},M)$ to be <u>valid</u> if all the positions $h(K,1), h(K,2),\ldots, h(K,p(K,\alpha)-1)$ are non-empty for every key K in $\mathcal{K}$. An arrangement is valid iff every key K in $\mathcal{K}$ is retrievable using the search algorithm of section 1. Similarly define an arrangement to be <u>feasible</u> if it is the result of inserting the keys in $\mathcal{K}$ into an empty table sequentially in some order; necessarily every feasible arrangement is valid.

Valid arrangements which are not feasible are possible; consider the following arrangement using the hash function h from our previous example:

location:	1	2	3	4
contents:	empty	empty	4321	3412

The number of feasible arrangements depends on $\mathcal{K}$ and h. It is no larger than N! (the number of ways to enter the keys), but may be as low as 1 if no collisions occur. Similarly the number of valid arrangements can vary between 1 and N!. For example, only one valid arrangement exists if no collisions occur and $h(K_i,1) \neq h(K_j,2)$ for all K_i,K_j in $\mathcal{K}$. The upper bound of N! on the number of

valid arrangements is obtained by induction on N, using the fact that $p(K,\alpha) \leq N$ for any valid arrangement and all keys $K \in \mathcal{K}$. We may store K_N in any of N positions $h(K_N,i)$ for $1 \leq i \leq N$; if we then delete K_N from $\mathcal{K}$ and $h(K_N,i)$ from the probe sequence $h(K_j,1),\ldots,h(K_j,M)$ for every $j < N$ we see that every valid arrangement of $\mathcal{K}$ induces a valid arrangement of $\mathcal{K}-\{K_N\}$ in locations $\{j \mid 1 \leq j \leq M \text{ and } j \neq h(K_N,i)\}$ using the modified probe sequences.

We define an arrangement $\alpha(\mathcal{K},M)$ to be <u>optimal</u> if either $avg(\alpha)$ or $wc(\alpha)$ is minimal over all arrangements in $A(\mathcal{K},M)$; the terms average-optimal and worst-case-optimal will distinguish these cases.

<u>Proposition 1</u>. A feasible optimal arrangement always exists.

<u>Proof</u>. If an arrangement α is not feasible, then there exist a set $\{K_{i_0},K_{i_1},\ldots,K_{i_{r-1}}\}$ of keys, none of which can be entered first since they form a "blocking cycle": there is a set of integers t_j for $0 \leq j \leq r-1$ such that $h(K_{i_j}, p(K_{i_j},\alpha)) = h(K_{i_{(j+1) \bmod r}}, t_{(j+1) \bmod r})$ and $t_j < p(K_{i_j},\alpha)$ for $0 \leq j \leq r-1$. But clearly $p(K_{i_j},\alpha)$ can be reduced by setting $\alpha(K_{i_j})$ to $h(K_{i_j},t_j)$ for $0 \leq j \leq r-1$. Since $avg(\alpha)$ strictly decreases, a feasible optimal arrangement can always be found after a finite number of blocking cycles have been removed in this fashion. □

Proposition 1 suggests an algorithm for finding optimal arrangements: enumerating all feasible arrangements. However, better methods exist.

<u>Proposition 2</u>. Optimal arrangements can be found by using an algorithm for the assignment problem.

<u>Proof</u>: The assignment problem [6] can be stated as follows.

Let N and M be given, with $N \leq M$, and let $\{a_{ij} \mid 1 \leq i \leq N, 1 \leq j \leq M\}$ be a matrix of nonnegative real numbers. The classic example specifies for each of M men and N jobs, the "inefficiency" a_{ij} of man j in job i. The objective is to find an assignment $i \rightarrow \alpha(i)$ of jobs to men such that the sum

$$\sum_{1 \le i \le N} a_{i,\alpha(i)}$$

is minimized, subject to the constraint that no man is assigned to more than one job.

We can apply this directly to the problem of finding average-optimal arrangements by letting a_{ij} be the integer such that $h(K_i, a_{ij}) = j$), denoting the cost of assigning K_i to T_j. The average number of probes required to retrieve a key in the optimized table is then just the total "inefficiency" divided by N. We observe that if the various keys have associated retrieval probabilities, then the arrangement minimizes the expected retrieval cost can be found in the same manner; we need only multiply each a_{ij} by the probability that K_i will be retrieved.

Similarly, we can minimize the worst-case cost by choosing a_{ij} to be N^{ℓ}, where ℓ is the integer such that $h(K_i, \ell) = j$. Since the key with highest cost determines the order of the total cost, minimizing the total cost here minimizes the worst-case cost.

In fact, it is not too difficult to find among those solutions with minimum worst-case cost, an arrangement with minimal average cost. (An arrangement which has minimum worst-case cost and also minimum average cost does not always exist; consider the keys 1234,1243,2341, and 3412 under the hashing function of our previous examples. The worst-case optimum is $wc(\alpha) = 2$, but any arrangement which achieves the minimum average cost of 3/2 must have a worst-case cost of 3.) To find such an arrangement it suffices to solve the assignment problem with

$$a_{ij} = N^{\ell+2} + \ell$$

where ℓ is the solution of $h(K_i, \ell) = j$. Similarly, by using

$$a_{ij} = \ell + N^{\ell+2-N}$$

the average-optimal arrangement which has the smallest worst-case retrieval time can be found. □

Having observed that our problem can be formulated as an instance of the assignment problem, it is of interest to know how quickly a solution can be determined. The general N by M assignment problem can be solved in time $O(NM^2)$ [6]; the space required is O(N+M) if the matrix entries a_{ij} can be computed in constant time from K_i,h, and j. When all the matrix entries are small integers (as when we are finding the average-optimal arrangement), it may be possible to improve this time bound somewhat, but the author was unable to find a more efficient procedure.

Worst-case optimal arrangements can be determined in time $O(BM(M,N)\cdot\log_2(N))$, where BM(M,N) is the time required to solve an M by N bipartite matching problem. The procedure, pointed out to the author by J. Vuillemin, is to use binary search on the worst-case cost: is it possible to test if the optimal worst-case cost is less than or equal to a given value w by solving the corresponding maximal matching problem. The graph used has N vertices x_i, M vertices y_j, and an edge (x_i,y_j) iff $a_{ij} \leq w$. Since $B(M,M) = O(M^{2.5})$, we obtain an $O(M^{2.5}\log(M))$ algorithm for the case N = M.

3. Efficiency of the Worst-case Optimal Arrangements

In this section we prove that even if the hash table is full (N = M), we can expect the worst-case optimal arrangement to have a worst-case cost of O(log(M)) with a probability approaching one very rapidly as $M \to \infty$. While a worst-case cost of O(log(M)) can obviously not be guaranteed (since there is a finite chance that all keys have the same probe sequence, for example), the odds are overwhelming that with a random hash function and a random set of keys, there is some arrangement of those keys yielding a worst-case cost of O(log(M)). This compares favorably with the standard techniques which also require $O(\log_2(N))$ time to retrieve a key, especially in situations where the set of keys is static (since updating an optimized hash table can be expensive).

The proof is modelled very closely after a similar result of Erdös and Renyi [2], who show that a random n by n matrix of 0's and 1's containing N(n) ones has a permanent of one with probability approaching one as $n \to \infty$ if $\lim_{n\to\infty}(N(n)-n\log(n))/n = \infty$.

Let $\mathfrak{M}(M,N,w)$ denote the set of all zero-one matrices with M columns, N rows, and exactly w ones per row. Obviously $|\mathfrak{M}(M,N,w)| = \binom{M}{w}^N$. We say a matrix $\{m_{ij}\} \in \mathfrak{M}(M,N,w)$ contains N independent ones iff there exists a function $\alpha : \{1,\ldots,N\} \to \{1,\ldots,M\}$ such that $\alpha(i) \neq \alpha(j)$ for $i \neq j$ and $m_{i,\alpha(i)} = 1$ for $1 \leq i \leq N$. Let P(M,N,w) denote the probability that a matrix in $\mathfrak{M}(M,N,w)$ contains N independent ones.

We wish to identify P(M,N,w) with the probability that a random set of N keys can be arranged in a hash table of size M so that the worst-case retrieval cost is at most w. This model will be accurate if every set of w locations is equally likely to be the set of w locations first probed for a random key k. Each matrix in $\mathfrak{M}(M,N,w)$ then corresponds in a natural fashion to the characteristic matrix describing, for a random set of N keys, which locations are usable if the worst-case cost is constrained to be at most w. The existence of N independent ones corresponds to the existence of an arrangement with worst-case cost of at most w; and by Proposition 1 the existence of a feasible, valid arrangement with worst-case cost of at most w is thereby implied.

We have $P(M,N,w) \geq P(M,M,w)$ for $1 \leq N \leq M$ since the first N rows of a matrix in $\mathfrak{M}(M,M,w)$ which contains M independent ones must contain N independent ones. We therefore proceed to show the following.

Proposition 3. $\lim_{M \to \infty} P(M,M,4\log(M)) = 1$

Proof: By the well-known theorems of Frobenius [3] and König [5], $1 - P(M,M,w)$ is equal to the probability that a matrix in $\mathfrak{M}(M,M,w)$ has k rows (or columns) and M-k-1 columns (or rows) that contain all the ones, for some k, $0 \leq k \leq M - 1$.

Let $Q_k(M,N,w)$ denote the probability that a matrix in $\mathfrak{M}(M,N,w)$ has k rows (or columns) and N-k-1 columns (or rows) containing all the ones, and k is the least such number for $0 \leq k \leq M/2$. Then

$$1 - P(M,N,w) = \sum_{k=0}^{\lfloor M/2 \rfloor} Q_k(M,N,w).$$

Case 1: k rows and M-k-1 columns contain all the ones, for some $k \leq M/2$. Those matrices in $\mathfrak{M}(M,M,w)$ having a minimal number k of rows and M-k-1 columns containing all the ones can be displayed as in Figure 1, after an appropriate permutation of the rows and columns. Each row of submatrix B must contain two ones under our assumption that k is minimal (if not, we could

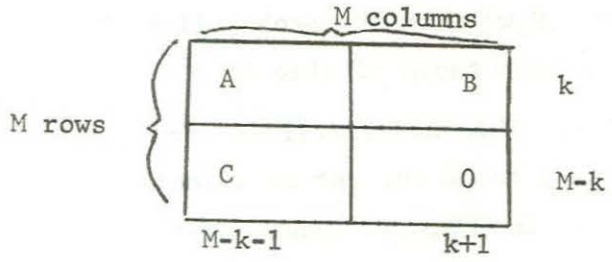

Figure 1.

include the column, and exclude the row, of the one in matrix B which is in a row of B containing no other ones). The fraction $f_k(M,M,w)$ of matrices of this type is less than:

$$\frac{\binom{M}{k}\binom{M}{k+1}\binom{M-k-1}{w}^{M-k}\left(\binom{M}{w}-\binom{M-k-1}{w}-(k+1)\binom{M-k-1}{w-1}\right)^{k}}{\binom{M}{w}^{M}}$$

whose logarithm is bounded above by

$$\begin{aligned}
&[(2k+1)-w(M-k)]\log(M)\\
&+w(M-k)\log(M-k-1)\\
&-k\log(k)-(k+1)\log(k+1)\\
&\leq (2k+1)\log(M)-w(k+1)/2
\end{aligned}$$

Thus if $w \geq 4\log(M)$, $Q_k(M,M,w) \to 0$ as $M \to \infty$.

Case 2: k columns and M-k-1 rows contain all the ones, for some $k \leq M/2$. See Figure 2.

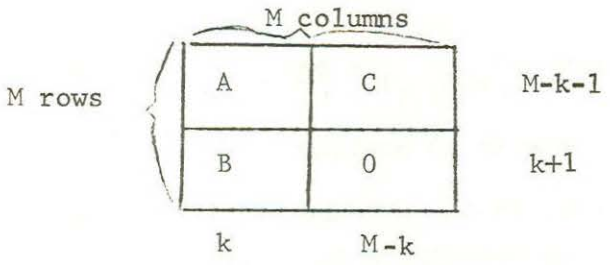

Figure 2.

The fraction $g_k(M,M,w)$ of matrices of this type is less than

$$\binom{M}{k}\binom{M}{k+1}\binom{M}{w}^{-(k+1)}\binom{k}{w}^{k+1}$$

whose logarithm is bounded above by

$$(2k+1)\log(M) - w(k+1)\log(w)$$

so that $g_k(M,M,w) \to 0$ with M if $w = 2\log(M)$. Since $Q_k(M,M,w) = f_k(M,M,w) + g_k(M,M,w)$, we are finished with the proof. □

A similar analysis might yield the expected value of the average retrieval cost in an average-optimal arrangement, but this seems much more difficult. It would be necessary to consider, for each set of keys, the ways of placing cN ones among N rows of an N by M matrix in a fashion consistent with the hash function, and then to determine if there exists a placement containing N independent ones.

4. Insertion Algorithms

We now turn our attention to the problem of maintaining the optimality of an arrangement as new keys are inserted into a table.

We first examine an insertion algorithm due to Brent [1], and demonstrate that it does not maintain optimality. Of course, Brent only intended his algorithm to be a good heuristic, a means of inserting each new key in such a fashion that the increase in average retrieval cost is kept reasonably low.

Brent's algorithm works as follows. Let K denote the new key being inserted, and suppose positions $h(K,1),\ldots,h(K,s)$ are already occupied with keys $K_1,K_2,\ldots,K_s$, and that $T_{h(K,s+1)}$ is empty. Let r_i denote the number of probes required to retrieve K_i, so that $h(K_i,r_i) = h(K,i)$.

Furthermore, let s_i denote $\min\{j \mid T_{h(K_{i,j})} = \underline{\text{empty}}\}$, the number of probes required to retrieve K_i if we move it to position $h(K_i,s_i)$. Then $(i+(s_i-r_i))/(N+1)$ is the increase in the average retrieval cost caused by moving K_i to position $h(K_i,s_i)$ and storing K in position h(K,i). Brent chooses between storing K in position h(K,s+1) and moving that K which minimizes $i+(s_i-r_i)$ by comparing (s+1) to $\min_i\{i+s_i-r_i\}$.

In fact, the following example demonstrates that no algorithm which only moves keys forwards in their probe sequence (that is, moves K from h(K,i) to h(K,i') for i' > i) can always arrive at the optimal arrangement. Consider the following arrangement (using the hash function of our previous examples) which is both average and worst-case optimal:

location:	1	2	3	4	5	6	7
contents:	1273456	1234567	3456712	4567123	5671234	6712345	empty

If the key 2345671 is now inserted, the only way to maintain optimality is to move 1273456 to location 7, move 1234567 (backwards) to position 1, and then store 2345671 in position 2.

Since Brent's algorithm is the only published algorithm which moves previously inserted keys when inserting a new key, we see that no existing insertion algorithm can maintain optimality for arbitrary hash functions. It is interesting to note, however, that for certain open-addressing collision-resolution schemes the usual insertion algorithm maintains average-optimality. We say that a hash function h exhibits <u>primary clustering</u> if $h(K_i,j) = h(K_{i'},j')$ implies that $h(K_i,j+\ell) = h(K_{i'},j'+\ell)$ for $0 \le \ell \le M - \min(j,j')$ for any $K_i,K_{i'}$. Linear probing ($h(K,i) \equiv h(K,1) + (i-1),\text{mod } M$) is perhaps the best-known example of a collision resolution scheme exhibiting primary clustering, and all primary clustering schemes are in fact isomorphic to linear probing in a natural manner.

<u>Proposition 4</u>. If h exhibits primary clustering, then the usual insertion algorithm maintains average-optimality.

Proof: This theorem is due to W.W. Peterson [7]; the proof is also given in Knuth [4, p.531]. Knuth also remarks that if the keys have associated retrieval probabilities, then the average-optimal arrangement can be achieved by using the standard insertion routine to insert the keys one by one into the table, in order of decreasing request probabilities. □

In spite of the fact that for linear probing the usual insertion algorithm maintains average-optimality, other hashing schemes are to be preferred, since the expected retrieval cost in the average-optimal scheme for a primary-clustering hashing function generally exceeds the expected cost for other schemes, even if average-optimality is not maintained.

We now turn our attention to the task of finding an insertion algorithm that will maintain the optimality of an arrangement. In essence, we need an algorithm to solve the assignment problem "incrementally".

One approach is to observe that if N/M is small enough (how small this is we shall determine), then the number of keys already in the table which we need to consider moving might be reasonably small. Brent considers moving only those keys on the probe sequence of the new key K; if we also consider moving all of the keys on their probe sequences, and so on, we can determine the maximum set $\mathcal{S}$ of keys that might need to be moved. Similarly we let $\mathcal{T}$ denote the set of locations that $\mathcal{S}$ might occupy in the optimized table; it suffices then to solve the assignment problem for placing $\mathcal{S}$ into $\mathcal{T}$, rather than $\mathcal{K} \cup \{K\}$ into T.

Define, for a given arrangement α, the functions:

$$\pi(K) = \min\{j \mid h(K,j) = \text{empty}\}$$
$$\sigma(K) = \{K_i \mid \alpha(K_i) = h(K,j) \text{ for some } j < \pi(K)\}$$
$$\tau(K) = \{i \mid h(K,j) = i \text{ for some } j \leq \pi(K)\}.$$

Then

$$\mathcal{S}(K) = \{K\} \cup \{\mathcal{S}(K_i) \mid K_i \in \sigma(K)\}$$
$$\mathcal{T}(K) = \tau(K) \cup \{\mathcal{T}(K_i) \mid K_i \in \sigma(K)\}$$

define by means of their minimal solutions the sets $\mathcal{S}$ and $\mathcal{T}$ of keys and positions relevant to the insertion of K into an arrangement α.

Let $\beta = N/M$ denote the "loading factor" of the existing arrangement α. In order to estimate the expected size $\mathcal{S}(K)$, we assume that the hashing function is uniform in the sense that every permutation of $\{1,\ldots,M\}$ is equally likely to be a probe sequence of some key K. We can then use the approximation

$$\text{Prob}(\pi(K) = i) = (1-\beta)\beta^{i-1}.$$

Let s_i denote the probability that $|\mathcal{S}(K)| = i$, and let

$$S(z) = \sum_{i=1}^{\infty} s_i z^i$$

denote the corresponding generating function. We shall develop an equation for S(z) which depends on the generating function:

$$P(z) = \sum_{i=1}^{\infty} p_i z^i$$

(where p_i is the probability that, for a key K' already stored in T, $\alpha(K') = h(K',i)$). However, determining P(z) for optimized hash tables remains an open problem, so we shall approximate S(z) after we develop the correct defining equation.

Let $C(z) = \sum_{i=1}^{\infty} c_i z^i$ be the generating function with coefficients c_i equal to the probability that the "contribution" of a key K' on the probe sequence of the new key K to S(K) is i keys. Therefore

$$S(z) = \sum_{i=0}^{\infty} (1-\beta)\beta^i [C(z)]^i \cdot z$$

since there is a probability of $(1-\beta)\beta^i$ that $\pi(K) = i+1$ (that is, there are i keys on the probe sequence for the new key K). The final z is for the key K itself.

Similarly we can define

$$C(z) = \left[\sum_{i=1}^{\infty} p_i (C(z))^{i-1}\right] \cdot \left[\sum_{i=0}^{\infty} (1-\beta)\beta^i (C(z))^i\right] \cdot z$$

(or equivalently,

$$(1 - \beta C(z)) \cdot (C(z))^2 = (1-\beta)P(C(z))z\ ).$$

The first term accumulates the contributions of those keys K" on the probe sequences of a key K' on the probe sequence for K, such that K" occurs before K' in the probe sequence for K'. The second term adjusts for those keys K" occurring after K' in the probe sequence for K'. Finally, the third term z is for the key K' itself.

The expected size of $\mathcal{S}(K)$ is S'(1); and

$$S'(z) = \frac{d}{dz}\left(\frac{(1-\beta)z}{(1-\beta C(z))}\right)$$

$$= \frac{(1-\beta C(z))(1-\beta) + (1-\beta)z\beta C'(z)}{(1-\beta C(z))^2}$$

so that

$$S'(1) = 1 + \frac{\beta C'(1)}{(1-\beta)} .$$

Now

$$(1-\beta C(z))2C(z)C'(z) - \beta C'(z)(C(z))^2 = (1-\beta)[P'(z)C'(z)z + P(C(z))]$$

so we obtain

$$C'(1) = (1-\beta)/(2-3\beta-(1-\beta)P'(1))$$

and thus

$$S'(1) = 1+\beta/(2-3\beta-(1-\beta)P'(1)).$$

Unfortunately, P(z) is unknown. We observe, however, that S'(1) can be expected to remain finite as long as

$$P'(1) \leq (2-3\beta)/(1-\beta).$$

Since P'(1) is the expected number of probes required to retrieve a key from an optimized table, it is bounded above by the expected number of probes required to retrieve a key from a table organized with any open-addressing hashing method. For uniform probing (all probes sequences equally likely) we have [4]

$$P'(1) \simeq \beta^{-1}\log(1/(1-\beta))$$

approximately. Substituting this into the final equation for S'(1) yields Figure 3; we see that the size of the relevant assignment problem is reasonably small (say $\leq$ 10 keys) as long as

$$\beta \leq .4$$

roughly. The function S'(1) has a pole β = .41466541; for loading densities less than this we can expect the number of relevant keys to be finite. In practice we should expect to be able to handle even higher loading densities without much trouble, since our formulas for S,C, and P explicitly ignore the probability of overlapping probe sequences. Furthermore, replacing P(z) by its correct definition (rather than the one for uniform probing) should yield a definite improvement.

5. Discussion and Conclusions

In this paper we have shown how to arrange a set of keys in a hash table so as to minimize the expected (or worst-case) number of probes required to retrieve a key. Our analysis demonstrates that the worst-case cost can be reduced to $O(\log_2(M))$ in almost all cases. (In practice it should be possible to achieve $O(\log_2(M))$ in all cases with very little work, since a set of keys which has an optimized cost that is too large can, by choosing another hash function randomly, be expected to yield an $O(\log_2(M))$ cost.)

Our analysis assumes that uniform hashing is used, however; an open problem is to confirm this result for the more common techniques such as double hashing. Another open problem is to calculate the expected retrieval cost in an optimized table, using (say) uniform hashing.

We have also examined briefly a technique for inserting a new key into an optimized table so as to maintain optimality of the arrangement. Our result here is that as long as the loading factor is less than .41 (approximately), we can usually insert a new key and maintain optimality by solving a small ($\approx$ 10 element) assignment problem. For tables of higher density one must apparently solve an assignment problem which involves most of the keys previously stored. (By saving the primal and dual variables of the previous solution, one can significantly speed up the solution of the new problem, but the extra storage required might better be used to store the keys themselves, thereby reducing the overall density.)

The techniques described here should be most useful when the hash table is relatively static, with the number of retrievals considerably exceeding the number of insertions. Large data bases are often of exactly this nature, and frequently utilize hashing techniques.

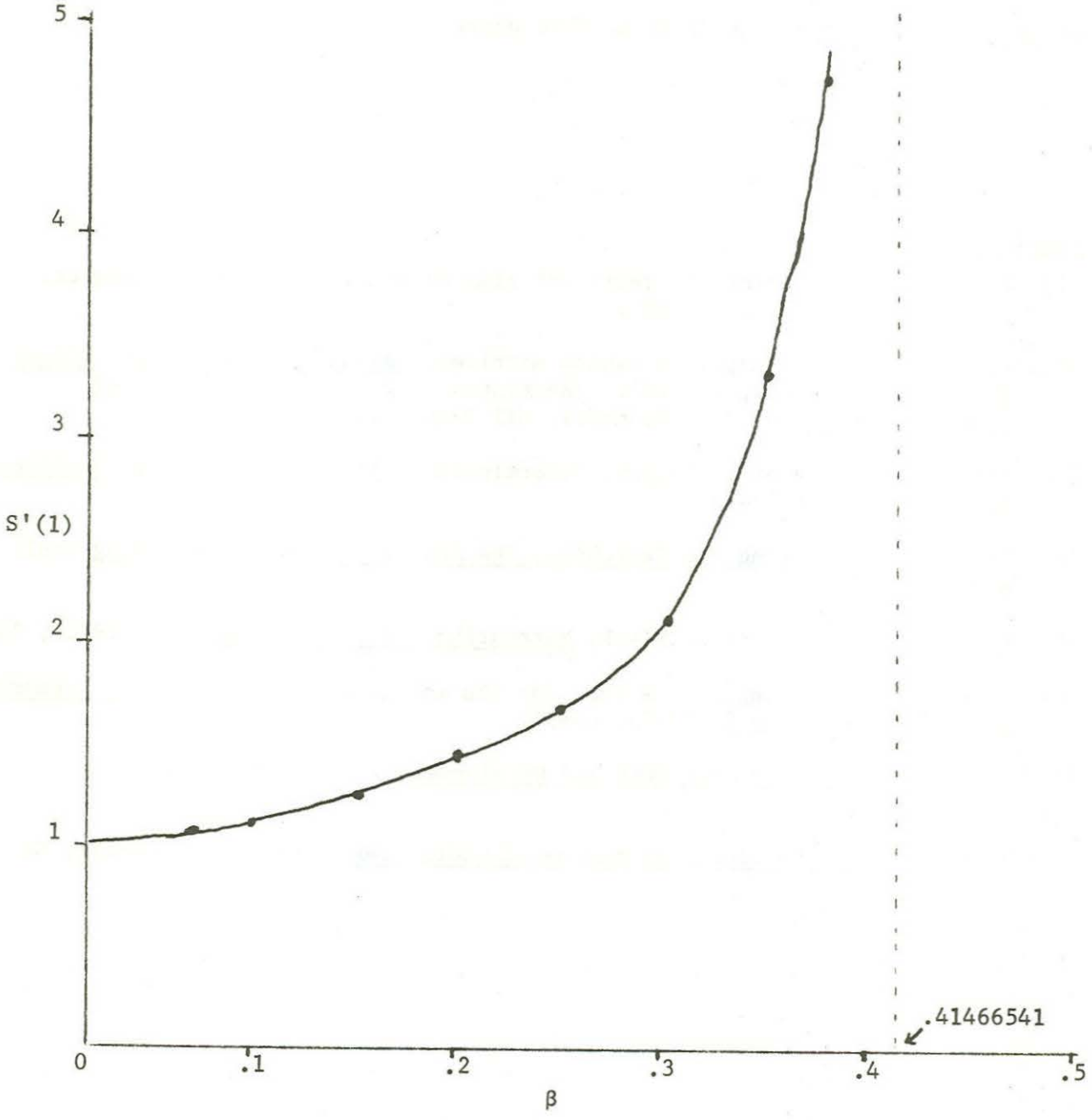

Figure 3.

Acknowledgement

I would like to thank Professor Donald Knuth for suggesting directions in which to extend a previous draft of this paper.

References

[1] Brent, R.P. Reducing the retrieval time of scatter storage techniques. CACM 16 (Feb. 1973), 105-109.

[2] Erdos, P. and A. Renyi. On random matrices. Magyar Tud. Akad. Mat. Kutato Int. Kozl. 8 (1964), 455-461. (Reprinted in Paul Erdos: The Art of Counting (edited by Joel Spencer), MIT Press (1973), 625-631.

[3] Frobenius, G. Uber zerlegbare Determinaten. Sitzungsberichte der Berliner Akademie (1917), 274-277.

[4] Knuth, D.E. Sorting and Searching, The Art of Computer Programming (vol. 3). Addison-Wesley.

[5] Konig, D. Graphok es matrixok. Matematikai es Fizikai Lapok 38 (1931), 116-119.

[6] Kuhn, H.W. The Hungarian method for the assignment problem. Naval Research Logistics Quarterly 2 (1955), 83-97.

[7] Peterson, W.W. IBM Research and Development 1 (1957), 130-146.

[8] Tripp, R. International Thesaurus of Quotations. Thomas Y. Cromwell, Co. (New York 1970).