ON HASH-CODING ALGORITHMS FOR PARTIAL-MATCH RETRIEVAL

(Extended Abstract)

by

Ronald L. Rivest

IRIA, 78 Rocquencourt, France.

Summary

We examine the efficiency of generalized hash-coding algorithms for performing partial-match searches of a random-access file of binary words. A precise characterization is given of those hash functions which minimize the average number of buckets examined for a search ; and a new class of combinatorial designs is introduced which permits the construction of hash functions with worst-case behavior approaching the best achievable average behavior in many cases.

1. The partial-match retrieval problem

We restrict our attention here to files F of k-bit binary words ; the non-binary case is also treated in the author's thesis [11]. A partial match query is just a string in $\{0,1,*\}^k$. Here the *'s are used as place-holding "don't-care" characters. The problem is to retrieve from F all words agreeing with the query in those positions where the query specifies a bit.

Example

Let k = 3 and F = {000, 001, 010, 101, 111}. Then the response to the query q = *0* is {000, 001, 101} while the response to the query q = *01 is {001,101}.

Historical background

The partial-match retrieval problem is a paradigm for "associative" search problems. Minker gives an excellent survey [7] of the hardware solutions to this problem. As large associative memories are currently economically impractical, we examine here search algorithms using conventional random-access storage devices. Since, as we shall see, there exist reasonably efficient hash-coding algorithms for the problem, we expect large hardwired associative memories to be economically feasible only for applications which have very tight real-time constraints (such as air-traffic controlling).

The partial-match retrieval problem is also interesting because it is really the next unsolved problem in the list of "intersection search" problems. A search problem is of the intersection type if the response, q(F), to a query q is merely the intersection of the file F with some predetermined set of records of interest which we denote by $q(\{0,1\}^k)$. Thus we write

$$q(F) = F \cap q(\{0,1\}^k) \qquad (1)$$

for any intersection type query q. (Note that this notation is consistent when $F = \{0,1\}^k$). The list of intersection search problems, in order of increasing complexity, is

1) Exact match queries. Each set $q(\{0,1\}^k)$ is just a single record in $\{0,1\}^k$; we want to know if that record appears in F. Hash-coding algorithms work well here.

2) Single-key queries. Here $q(\{0,1\}^k)$ is all records having a single bit value for a specified position. Inverted-list algorithms work well here, since there are only 2k distinct queries.

3) Partial-Match queries. Here $q(\{0,1\}^k)$ is the set of all records in $\{0,1\}^k$ agreeing with the query q in its specified positions.

4) Boolean queries. Here each $q(\{0,1\}^k)$ is an arbitrary subset of $\{0,1\}^k$.

To date the published algorithms for the partial match problem either involve exorbitant amounts of storage to represent the file F, or they involve a modification of the inverted-list scheme. The latter is unappealing because as the number of bits specified in the query increases, the work necessary to perform the intersection of the appropriate inverted lists also increases, while the expected size of the response q(F) decreases. A survey of these algorithms is given in [11], together with an extensive bibliography. Of particular note are [1,2,3,4,10 and 12]. This problem is also discussed in [6,8].

2. Hash-coding algorithms

We consider hash-coding schemes which divide the file F into b disjoint buckets, or lists: $L_1, L_2, \ldots L_b$.

A record $x \in F$ will be stored on list $L_{h(x)}$, where h is an auxiliary "hash function" mapping $\{0,1\}^k$ onto the set $\{1,2,\ldots,b\}$. We denote $h^{-1}(i)$ (the set of record $x \in \{0,1\}^k$ with hash value i) by B_i, and call

this "block i" (of the partition of $\{0,1\}^k$ induced by h). Thus $L_i \subseteq B_i$ for $1 \le i \le b$; in fact, $L_i = B_i \cap F$.

To calculate the response $q(F) = q(\{0,1\}^k) \cap F$ to a partial match query q, the retrieval algorithm must examine every list L_i whose index i is in the set

$$h(q) =_{def} \{j \mid (B_j \cap q(\{0,1\}^k)) \neq \emptyset\} \qquad (2)$$

If $i \notin h(q)$, then L_i need not be examined, since $L_i \cap q(\{0,1\}^k) \subseteq B_i \cap q(\{0,1\}^k) = \emptyset$ in this case.

Thus the retrieval algorithm can be described by the equation :

$$q(F) = \bigcup_{i \in h(q)} (L_i \cap q(\{0,1\}^k)). \qquad (3)$$

We will measure the complexity of calculating the response q(F) to a query q by $|h(q)|$, the number of buckets examined. To avoid consideration of the degenerate hash function mapping all records into a single bucket, we shall require that our hash functions be balanced in the sense that B_i has the same size for any i, $1 \le i \le b$. If each record $x \in \{0,1\}^k$ is equally likely to appear in F, then the expected length of each list L_i will then be the same, so that the number of buckets examined is an accurate measure of the work performed. In cases where the time to access a single bucket may usually exceed the time to read it (as for example with a disk unit) the number of buckets examined will again be an accurate measure. A more precise analysis will of course be necessary initially in order to determine the optimal number b of buckets to use (see [11]).

We will consider both the average and worst-case number of buckets examined. Note that these measures depend only on the hash-function h and not on the particular file F in question, by (2). Let Q denote the set of all partial match queries of length k, and let Q_s denote the set of all partial match queries of length k having exactly s bits specified, so that $|Q| = 3^k$ and $|Q_s| = \binom{k}{s}2^s$. Then the average and worst-case number of buckets examined are calculated by the respective formulas :

$$A(h) =_{def} |Q|^{-1} \sum_{q \in Q} |h(q)| \quad , \text{ and} \qquad (4)$$

$$W(h) =_{def} \max_{q \in Q} |h(q)| . \qquad (5)$$

We use the measures $A_s(h)$ and $W_s(h)$, defined similarly, if Q_s rather than Q is used (Note that W(h) = b trivially because of $q = *^k$.).

We would like to find balanced hash functions h which minimize the average and/or worst-case numbers of buckets examined.

3. The average number of buckets searched.

The following theorem gives a precise characterization of which balanced hash functions h minimize A(h), the average number of buckets examined.

Theorem 1. *Let $h : \{0,1\}^k \to \{1, \ldots, b\}$ be a balanced hash function with $b = 2^w$ buckets for some integer w, $1 \le w \le k$. Then A(h) is minimal if and only if each block B_i is a $q(\{0,1\}^k)$ for some query $q \in Q_w$.*

The geometry of the sets $q(\{0,1\}^k)$ thus is reflected in an appealing fashion in the optimal shape for the blocks B_i. The set of records in each optimal B_i can be described by a string in $\{0,1,*\}^k$ containing exactly w bits and k-w *'s.

Corollary 1.

The hash function which extracts the first w bits of each record $x \in \{0,1\}^k$ to use as a bucket address, has a minimal A(h).

The proof of theorem 1 is unfortunately somewhat lengthy, although interesting in that we prove a little more than is claimed.

Let $B_i \subset \{0,1\}^k$ be any block. Then by $Q(B_i)$ we denote $|\{q \in Q \mid (q(\{0,1\}^k) \cap B_i) \neq \emptyset\}|$, the number of queries $q \in Q$ which examine list L_i. Denote by $Q_{min}(x,k)$ the minimal value of $Q(B_i)$ for any B_i of size x, $B_i \subset \{0,1\}^k$. We now note that

$$\begin{aligned} A(h) &= |Q|^{-1} \cdot \sum_{q \in Q} |h(q)| \\ &= |\{(B_i,q) \mid (q(\{0,1\}^k) \cap B_i) \neq \emptyset\}| \cdot |Q|^{-1} \\ &= \sum_{1 \le i \le b} Q(B_i) \cdot |Q|^{-1} \\ &\ge b \cdot Q_{min}(2^{k-w},k) \cdot |Q|^{-1} \end{aligned}$$

To finish the proof of the theorem we need only show that $Q(B_i) = Q_{min}(2^{k-w},k)$ if and only if B_i is of the form $q(\{0,1\}^k)$ for some $q \in Q_w$. The rest of the proof involves four parts : calculation of $Q(B_i)$ for B_i of the proper form, calculation of $Q_{min}(2^{k-w},k)$, the demonstration of equality, and then the demonstration of the "only if" portion.

Calculation of $Q(B_i)$.

For simplicity of notation we use the symbols x,y,z to denote either a positive or their k-bit binary representation. Occasionally we may wish to explicity indicate that some number t of bits are to be used ; we denote this string by x:t (so that 9:5 = 01001). The length of a string x in $\{0,1\}^*$ we denote by $|x|$. Concatenation of strings is represented by concatenation of their symbols ; 0x denotes a zero followed

by the string x. A string of t ones we denote by 1^t.

If x is a string, we denote by $\underline{x}$ (x underlined) the set of those x+1 strings of length |x| which denote integers not greater than x. For example $\underline{011}$ = {000,001,010,011}. If $x = 2^{k-w}-1$, then $\underline{x:k}$ describes the set $q(\{0,1\}^k)$ for $q = 0^w*^{k-w}$, which is in Q_w. Furthermore, $Q(q(\{0,1\}^k))$ does not depend on which $q\in Q_w$ is chosen ; this is always 2^w3^{k-w} (since each * in q can be replaced by 0,1 or *, and each specified position optionally replaced by a *, to obtain a query counted in $Q(q(\{0,1\}^k))$).

Thus $Q(B_i) = 2^w3^{k-w}$ for $B_i = \underline{x:k}$ with $x = 2^{k-w}-1$. To show this is optimal, it is necessary to calculate $Q(\underline{x})$ for arbitrary strings x.

Lemma 1. (a) $Q(\underline{\text{nullstring}}) = 1$

(b) $Q(\underline{0x}) = 2Q(\underline{x})$

(c) $Q(\underline{1x}) = 2Q(\underline{1^{|x|}}) + Q(\underline{x})$

Proof : Take (a) to be true by definition. For (b), any query examining $\underline{x}$ can be preceded by either a 0 or a * to obtain a query examining $\underline{0x}$. Part (c) follows from the fact that $\underline{1x} = 0(\underline{1^{|x|}}) \cup \underline{1x}$; there are $2Q(\underline{1^{|x|}})$ queries starting with 0 or *, and $Q(\underline{x})$ starting with 1.□

The preceding lemma permits $Q(\underline{x})$ to be easily computed for abitrary strings x ; we list some particular values :

x =	null	0	1	00	01	10	11
$Q(\underline{x})$ =	1	2	3	4	6	8	9

x =	000	001	010	011	100	**101**	**110**	111
$Q(\underline{x})$ =	8	12	16	18	22	24	26	27

If x is the string $x_1 x_2 \ldots x_k$ and z_i denotes the number of zeros in $x_1 \ldots x_i$, then lemma 1 implies that

$$Q(\underline{x}) = 2^{z_k} + \sum_{1\le i\le k} x_i\, 3^{k-i}\, 2^{z_i+1}. \qquad (6)$$

Lemma 2 : $Q(\underline{x1}) = 3Q(\underline{x})$.

Proof : From (6), directly, or noting that if q is counted in $Q(\underline{x})$, then q0, q1, and q* will be counted in $Q(\underline{x1})$. □

Using p(x) to denote 2^{z_k} (where z_k is the number of zeros in x), we have also the following.

Lemma 3 : $Q(\underline{x:k}) = Q(\underline{x-1:k}) + p(x:k)$.

Proof : The only queries counted in $Q(\underline{x})$ but not $Q(\underline{x-1})$ will be those obtained by replacing an arbitrary subset of the zeros in x by *'s. □

Now that we know quite a bit about $Q(q(\{0,1\}^k))$ for $q\in Q_w$, we turn our attention to Q_{min}.

A lower bound for $Q_{min}(x,k)$

The following inequality holds for $Q_{min}(x,k)$, the minimal number of queries $Q(B_i)$ for any $B_i \subset \{0,1\}^k$, $|B_i| = x$.

$$Q_{min}(x,k) \ge \max(2Q_{min}(x_o,k-1) + Q_{min}(x_1,k-1)) \qquad (7)$$

where the maximum is taken over all pairs of non-negative integers x_o, x_1 such that $x_o + x_1 = x$, $x_o \ge x_1$, and $x_o \le 2^{k-1}$. To show (7), let B_i contain x_o records starting with a 0 and x_1 with a 1. Then $Q(B_i)$ must count at least $2Q_{min}(x_o,k-1)$ queries beginning with a zero or a *, and at least $Q_{min}(x_1,k-1)$ which start with a 1. (Nothing is lost by assuming $x_o \ge x_1$.) For k = 1 we have $Q_{min}(x,1) = Q(\underline{x-1:1})$, by inspection.

Showing $Q(B_i)=Q_{min}(2^{k-w},k)$ if $B_i = \underline{2^{k-w}-1:k}$

We will in fact prove the stronger statement that $Q(B_i) = Q_{min}(x+1,k)$ if $B_i = \underline{x:k}$, by induction on k. Since $Q(\underline{x:k}) \ge Q_{min}(x+1,k)$ necessarily, with equality holding for k = 1, as we have seen, equality can be proved in general using (7) if we can show the following.

$$Q(\underline{x:k}) \le \max(2Q(\underline{y:k-1}) + Q(\underline{z:k-1})) \qquad (8)$$

where the maximum is taken over all pairs of non-negative integers y,z such that y+z+1 = x, $y \ge z$, and $y \le 2^{k-1}$. By induction the right-hand side of (8) is a lower bound for $Q_{min}(x+1,k)$. The case in (8) of x = y corresponding to $x_1 = 0$ in (7) is OK by Lemma 1 (b). To prove (8) we consider four cases, according to the last bits of y and z.

Case 1 : $\underline{y:k-1 = y'1}$, $\underline{z:k-1 = z'1}$, and $\underline{x:k = x'1}$.
In this case (8) is true by lemma 2, since we know by induction that $Q(\underline{x'}) \le 2Q(\underline{y'}) + Q(\underline{z'})$.

Case 2 : $\underline{y:k-1 = y'0}$, $\underline{z:k-1 = z'1}$, and $\underline{x:k = x'0}$. If $p(x') \le 2p(y')$ then (8) is true since it is equivalent by lemmas 2 and 3 to

$$3Q(\underline{x'-1}) + 2p(x') \le 6Q(\underline{y'-1}) + 4p(y') + 3Q(\underline{z'}),$$

but we know by induction that $Q(\underline{x'-1}) \le 2Q(\underline{y'-1})+Q(\underline{z'})$ Otherwise if $p(x') > 2p(y')$ we use the fact that (8) says

$$3Q(\underline{x'}) - p(x') \le 6Q(\underline{y'}) - 2p(y') + 3Q(\underline{z'})$$

and we know that $Q(\underline{x'}) \le 2Q(\underline{y'}) + Q(\underline{z'})$ by induction.

Case 3 : $\underline{y:k-1 = y'1}$, $\underline{z:k-1 = z'0}$, and $\underline{x:k = x'0}$.
Depending on the truth or falsity of $p(x') \le p(z')$ we use induction and the fact that (8) is implied by either

$$3Q(\underline{x'-1}) + 2p(x') \le 6Q(\underline{y'}) + 3Q(\underline{z'-1}) + 2p(z')$$

or $3Q(\underline{x'}) - p(x') \le 6Q(\underline{y'}) + 3Q(\underline{z'}) - p(z')$.

Case 4 : y:k-1 = y'0, z:k-1 = z'0, and x:k = x'1. If $p(y') \le p(z')$ we use induction and the fact that (8) is implied by

$$3Q(\underline{x'}) \le 6Q(\underline{y'}) - 2p(y') + 3Q(\underline{z'-1}) + 2p(z').$$

Otherwise we use the fact that (8) is implied by

$$3Q(\underline{x'}) \le 6Q(\underline{y'-1}) + 4p(y') + 3Q(\underline{z'}) - p(z').$$

This completes the proof that $Q(\underline{x:k}) = Q_{min}(x+1,k)$.

Showing $Q(B_i) = Q_{min}(2^{k-w},k)$ only if $B_i = q(\{0,1\}^k)$ for some $q \in Q_w$.

We need only that (8) holds with equality for $x = 2^{k-w}-1$ only if $y = z$, since this implies that $Q(B_i) > Q_{min}(2^{k-w},k)$ if B_i is not of the form $q(\{0,1\}^k)$ for $q \in Q_w$. To see this, let $B_i = 0C \cup 1D$, for $C,D \subset \{0,1\}^{k-1}$. Then note that if $B_i \neq q(\{0,1\}^k)$ for some $q \in Q_w$, then either $|C| > 0$, $|D| > 0$, and $|C| \neq |D|$; or $|C| = |D|$ but at least one of C, D is not of the form $q(\{0,1\}^{k-1})$ for any $q \in Q_{w-1}$.

Now $x:k = 0^w 1^{k-w}$. If $y = z$, then (8) holds with equality by Lemma 1. To show (8) holds with equality only if $y = z$, suppose $y = 2^{k-w-1} + t-1$ and $z = 2^{k-w-1}-t-1$ for any t, $0<t<2^{k-w-1}$. Then (8) holding with strict inequality says :

$$Q(\underline{0^w 1^{k-w}}) < 2Q(\underline{y:k-1}) + Q(\underline{z:k-1})$$

or

$$Q(\underline{01^{k-w}}) < 2Q(\underline{y:k-w}) + Q(\underline{z:k-w})$$

or

$$Q(\underline{01^{k-w}}) < Q(\underline{y:k-w+1}) + Q(\underline{z:k-w}).$$

Subtracting 2^{k-w-1} from both x and y we get that (8) means

$$Q(\underline{01^{k-w-1}}) < Q(\underline{t-1:k-w}) + Q(\underline{2^{k-w-1}-t-1:k-w}).$$

It is simpler to note that the general statement.

$$Q(\underline{x:k}) < Q(\underline{t-1:k}) + Q(\underline{x-t:k})$$

is always true ; in fact, it is implied directly by (8), lemma 1, and the fact that $Q(\underline{x-t:k})$ is always positive. This completes our proof that $Q(B_i)=Q_{min}(2^{k-w},k)$ only if $B_i = q(\{0,1\}^k)$ for some $q \in Q_w$, and also finishes our proof of theorem 1. □

While theorem 1 only counted queries in Q, the same result holds if we count queries in Q_s.

Theorem 2 : *Let h and b be as in theorem 1. Then $A_s(h)$ is minimal for $0<s<k$ if and only if each block B_i is a $q(\{0,1\}^k)$ for some $q \in Q_w$.*

This can't be asserted in an "iff" manner for s = 0 or s = k, for which $A_s(h) = b$ and $A_s(h) = 1$ independent of h.

Theorem 2 can be proved in the same manner as theorem 1. We note here the differences, using $Q_s(B_i)$ and $Q_{min,s}(x,k)$ to count queries in Q_s rather than Q.

Lemma 4. (a) $Q_0(\underline{x}) = 1$, for all x.

(b) $Q_s(\underline{nullstring}) = 0$ for $s \ge 1$.

(c) $Q_s(\underline{0x}) = Q_s(\underline{x}) + Q_{s-1}(\underline{x})$, for $s \ge 1$ and all x.

(d) $Q_s(\underline{1x}) = Q_s(01^{|x|}) + Q_{s-1}(x)$, for $s \ge 1$ and all x.

Lemma 5. $Q_s(\underline{x}) - Q_s(\underline{x-1}) = \binom{z_k}{k-s}$, where z_k denotes the number of zeros in x and $|x| = k$.

Lemma 6. $Q_s(\underline{x1}) = 2Q_{s-1}(\underline{x}) + Q_s(\underline{x})$.

Lemma 7. $Q_{min,s}(x,k) \ge \max(Q_{min,s}(x_o,k-1)+Q_{min,s}(x_o,k-1) + Q_{min,s-1}(x_1,k-1))$

where the maximum is taken over all pairs of non-negative integers x_o, x_1 such that $x_o+x_1 = x$, $x_o \ge x_1$ and $x_o \le 2^{k-1}$.

The proofs of these lemmas are omitted here (see [1]). The proof of theorem 2 then proceeds along the same lines as that of theorem 1 ; with (8) being replaced by

$$Q_s(\underline{x:k}) \le \max(Q_s(\underline{y:k-1})+Q_{s-1}(\underline{y:k-1})+Q_{s-1}(\underline{z:k-1})).$$

We omit details here of the proof, as it varies little from the proof of (8). The "only if" portion of the proof is also similar, except that the inductive hypothesis may have to be applied more than once (for varying s values).

Calculation of $A_s(h)$.

Now that theorems 1 and 2 tell us what the optimal balanced hash functions $h:\{0,1\}^k \to \{1,\ldots,b = 2^w\}$ are like, we can calculate $A_s(h)$ easily, using the optimal h from Corollary 1 to theorem 1 (using the first w bits of $x \in \{0,1\}^k$ as the hash value). We get

$$A_s(h) = \binom{k}{s}^{-1} \sum_{o \le i \le s} \binom{w}{i}\binom{k-w}{s-i} 2^{w-i} \ge b^{1-s/k}$$

where the first sum considers the ways in which i of the s specified bits fall in the first w positions ; when this happens 2^{w-i} buckets must be searched. The inequality is a special case of a mean value theorem (theorem 59 of [5]). In fact, $b^{1-s/k}$ is a very good approximation for $A_s(h)$, as long as w is not too small. Thus, for example, whenever half of the bits are specified in the partial match query, we would expect to examine about $\sqrt{b}$ buckets.

This work performed decreases exponentially with the number of bits s specified in the query. We conjecture that no search algorithm utilizing the same amount of storage space for representing the file F can do significantly better. This conjecture is supported by an analysis of a digital-tree search algorithm for the same problem [11].

4. The worst-case number of buckets searched

The worst-case behavior of the hash function of Corollary 1 is obviously poor ; if none of the specified bits occur in the first w positions then every bucket must be searched. In this section we find that other optimal average-time hash functions exist which have much improved worst-case behavior, often approximately equal to the average behavior. We also consider a simpler strategy involving storing each record in several locations.

To obtain good worst-case behavior $W_s(h)$ for $h: \{0,1\}^k \to \{1,\ldots, b = 2^w\}$, the hash function h(x) must depend on all of the bits of x, so that each specified bit contributes approximately equally and independently towards decreasing the number of buckets searched. We shall also restrict our attention to hash functions satisfying the conditions of theorems 1 and 2 so that optimal average time behavior is ensured. While we have no proof that these block shapes are necessary for optimal worst-case behavior, the fact that $W_s(h)$ is bounded below by $A_s(h)$ makes it desirable to keep $A_s(h)$ minimal.

An example :

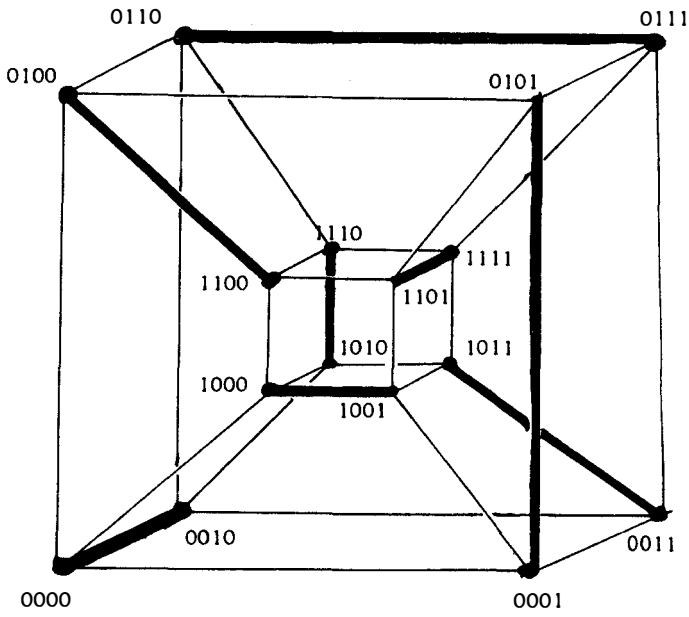
Let us consider by way of introduction an example with k = 4, w = 3. The following table describes an interesting hash function h ; row i describes the query $q \in Q_3$ such that $B_i = q(\{0,1\}^4)$. Thus h(0110) = 6 and h(1110) = 4, each $x \in \{0,1\}^4$ is stored in a unique bucket.

		Bit position			
		1	2	3	4
Bucket address	1	0	0	*	0
	2	1	0	0	*
	3	*	1	0	0
	4	1	*	1	0
	5	1	1	*	1
	6	0	1	1	*
	7	*	0	1	1
	8	0	*	0	1

Table 1. A hash function $h:\{0,1\}^4 \to \{1, \ldots, 8\}$

This can be interpreted as a perfect matching on the Boolean 4-cube, as indicated in Figure 1, since each block contains two adjacent points and distinct blocks are disjoint. In general we have the problem of packing $\{0,1\}^k$ with k-w dimensional subspaces.

Whereas the hash function of Corollary 1 would have all it's *'s in the fourth column, here the *'s are divided equally among the four columns. As a result we have $W_1(h) = 5$ instead of 8. For example, we need only examine buckets 1,4,5,6 and 7 for the query **1*. Table 2 lists $W_s(h)$ and $\lceil A_s(h) \rceil$ for $0 \le s \le 4$; we see that we have reduced $W_s(h)$ so that $W_s(h) = \lceil A_s(h) \rceil$, no further reduction is possible.



s	0	1	2	3	4
$W_s(h)$	8	5	3	2	1
$\lceil A_s(h) \rceil$	8	5	3	2	1

Table 2. $W_s(h)$ and $\lceil A_s(h) \rceil$.

Definition and characteristics of ABD's.

Let us call a table such as table 1 an "associative block design of type (k,w)", or an ABD(k,w) for short. To be precise, an ABD(k,w) is a table with $b = 2^w$ rows and k columns with entries from {0,1,*} such that

(i) each row has w digits and k-w *'s,

(ii) the rows represent disjoint subsets of $\{0,1\}^k$. That is, given any two rows there exists a column in which they contain differing digits.

(iii) each column contains the same number b.(k-w)/k of *'s.

Conditions (i) and (ii) ensure that $A_s(h)$ is minimal, by theorem 2. Condition (iii) attempts to restrict the class of ABD's to those hash functions with good worst-case behavior $W_s(h)$ by requiring a certain amount of uniformity in the utilization of each bit by h. In fact, (iii) implies that $W_1(h)$ is minimal. More stringent uniformity conditions are conceivable, perhaps involving the distribution of t-tuples within each t-subset of columns, but (iii) alone is enough to make the construction of ABD's a difficult combinatorial problem.

The following lemma follows more or less directly from the definition.

Lemma 8. An ABD(k,w)

(i) has exactly b.w/2k 0's (or 1's) in each column,

(ii) has exactly $\binom{w}{u}$ rows which agree in exactly in u positions with any record $x \in \{0,1\}^k$, for $0 \le u \le w$,

(iii) requires that k,w satisfy

$$k\left(\frac{b.w}{2k}\right)^2 \ge \binom{b}{2}$$

or

$$\frac{k}{w} \le \frac{w}{2}\left(\frac{b}{b-1}\right). \qquad (9)$$

Part (i) implies that bw/2k must be integral, part (ii) (taking $x = 0^k$) tells us how many rows have exactly u zeros, and part (iii) implies that to achieve large record length/bucket address length ratios k/w we must have relatively large values for w. The lemma implies that for $k \le 20$ the possible values of (k,w), $k \ne w$, for which an ABD might exist are :
(4,3),(8,w) for $4 \le w \le 7$, (10,5),(12,6),(12,9),(14,7), (16,w) for $6 \le w \le 15$, (18,3t) for $2 \le t \le 5$, (20,10) and (20,15). An ABD(8,4) has been shown not to exist by extending lemma 8 (iii) slightly.

ABD Construction techniques.

While ABD's closely resemble balanced incomplete block designs and other combinatorial structures, it appears that the existence of ABD's is not implied by the existence of these other objects in any straightforward manner. We therefore present here several direct construction techniques which provide infinite classes of ABD's. The general question of the existence of an ABD of arbitrary type (k,w) seems to be extremely difficult ; the positive nature of the very partial results obtained here suggests however that ABD's are not scarce.

We first present a simple infinite class of ABD's. The construction here is due to Ronald Graham. Franco Preparata has discovered another construction for a class of the same parameters, based on cyclic BCH error-correcting codes [9].

Theorem 3. *An ABD(2^t,2^t-1) exists for $t \ge 2$.*

Proof. We extend our notation for an ABD : a row containing r "-"'s will represent 2^r rows of the actual ABD obtained by independently replacing each - with a 0 or 1. The construction has two parts :

(i) Rows 1 to t+1 have -'s in positions t+2 to k. Row i for $1 \le i \le t+1$ has its star in column i ; the remaining columns contain digits. (For example, columns 1 to t+1 of these rows might contain cyclic shifts of $*10^{t-1}$.).

(ii) Row i for $m+2 \le i \le 2k-m-1$ contains digits in columns 1 to m+1, a * in column $m+1+\lfloor (i-m)/2 \rfloor$, and -'s elsewhere. The digits used are arbitrary except they must satisfy part (ii) of the ABD definition. It is easy to check that this is an ABD.

We present in table 3 an ABD(8,7)(t=3) constructed this way.

	1	2	3	4	5	6	7	8
1	*	1	0	0	-	-	-	-
2	0	*	1	0	-	-	-	-
3	0	0	*	1	-	-	-	-
4	1	0	0	*	-	-	-	-
5	0	0	0	0	*	-	-	-
6	0	1	0	1	*	-	-	-
7	0	1	1	1	-	*	-	-
8	1	0	1	0	-	*	-	-
9	1	0	1	1	-	-	*	-
10	1	1	0	1	-	-	*	-
11	1	1	1	0	-	-	-	*
12	1	1	1	1	-	-	-	*

Table 3. An ABD(8,7).

Graham has also constructed an ABD(16,13) with a similar two-part method.

The ABD's of theorem 3, while interesting as a solution to a combinatorial problem, are essentially useless as hash functions since the number of buckets is unacepptably large. We wish to have ABD's such that the ratio k/w does not tend to 1. The following theorem does just that.

Theorem 4. *Given an ABD(k,w) and an ABD(k',w') such that k/w = k'/w', one can construct an ABD(k+k',w+w').*

Proof : For each possible pair of rows (R_1,R_2) with $R_1 \in$ ABD(k,w), R_2 ABD(k',w') let the concatenation $R_1\ R_2$ be a row of the ABD(k+k',w+w'). This is easily shown to be a legal ABD. □

Theorem 4 allows us to construct an infinite family of ABD's of type (4t,3t), for $t \geq 1$, using the ABD of table 1. This is approaching utility (an ABD(16,12) is conceivably useful, say) but we need "starting" designs with large k/w to obtain a family with large k/w. On the other hand we know by lemma 8 (iii) that designs with large k/w must have k at least $2(k/w)^2$ approximately. Unfortunately, these tables get rapidly unmanageable by hand. Computer searches for an ABD(8,5) or an ABD(10,5) also ran out of time before finding any. The question as to whether ABD(k,w)'s existed with k/w>4/3 thus remained open until the following theorem, showing that k/w can be arbitrarily large, was discovered.

Theorem 5. *Given an ABD(k,w) and an ABD(k',w') one can construct an ABD(kk',ww').*

Proof : Each row R of the first ABD generates $2^{w(w'-1)}$ rows of the resultant ABD, as follows. The set of rows of the ABD(k',w') is arbitrarily divided into two equal-sized subsets, A_0 and A_1. Each character x of R is replaced by a string of k' characters : if x = *,x is replaced by $*^{k'}$, otherwise x is replaced by some row in A_x. The w digits of R are replaced independently in all possible ways by rows from the corresponding sets A_0 and A_1.

This generates a table with $2^{ww'}$ rows of length kk', each row has (k-w)k' + w(k'-w') = kk'-ww' *'s. Any two rows of the resultant ABD must contain differing digits in at least one column since rows replacing differing digits must differ, or if the two rows were generated from the same row of the first design, then one of the digits replaced will have been replaced by two (differing) rows from the second ABD. The number of *'s in each column turns out to be $2^{ww'}(kk'-ww')/kk'$, as required, so that we have created an ABD(kk',ww'). □

The preceding theorem allows us to construct an $ABD(4^t,3^t)$ for $t \geq 1$, for example, from our ABD(4,3). While this does provide large k/w designs, they are quite likely not the smallest designs for the given ratio, since by (9) we might hope to have w grow linearly with (k/w), whereas here it grows like $(k/w)^{\log_{3/4}(3)} > (k/w)^4$. At present, however, it is our only way of constructing large k/w designs.

Analysis of ABD search times.

Let us examine the worst-case number of buckets examined $W_s(h)$ for hash functions associated with the ABD's of the last section.

Consider first the hash function h associated with an ABD(k+k',w+w') which was created by the concatenation (theorem 4) of an ABD(k,w) and an ABD(k',w') (with associated hash functions g and g', respectively). Then

$$W_s(h) = \max_{u+v=s} W_u(g).W_v(g').$$

for $0 \leq s \leq k+k'$, $0 \leq u \leq k$, $0 \leq v \leq k'$. For example, the ABD(8,6) created from two of our ABD(4,3)'s has $W_s(h)$ as in table 4.

s	0	1	2	3	4	5	6	7	8
$W_s(h)$	64	40	25	16	10	6	4	2	1
$\lceil A_s(h) \rceil$	64	40	25	15	9	6	4	2	1

Table 4. Performance of an ABD(8,6).

We see that the ABD(8,6) does nearly as well as possible. One can show that if $W_u(g) = A_u(g)$ then $W_s(h) = A_s(h)$, where h is the concatenation of m copies of the ABD of g, for $m \geq 1$. So concatenation preserves optimal and near-optimal worst-case behavior.(The slight discrepancy of table 4 is caused by the non-convexity of $\log(W_s(h))$ of table 2 at s = 3.).

The corresponding worst-case analysis for those ABD's constructed by the "insertion" operation of theorem 5 is much more difficult to work out. It seems the worst-case here occurs when the specified bits occur together in blocks corresponding to the digits of the first ABD (of type(k,w)) used in the construction.

Under this assumption (for which I have no proof) the worst-case behavior of an ABD(16,9) created from two of our ABD(4,3)'s can be calculated to be as given in table 5.

s	0	1	2	3	4	5	6	7	8
$W_s(h)$	512	368	272	224	176	116	76	56	36
$\lceil A_s(h) \rceil$	512	368	263	186	131	91	63	43	30

s	9	10	11	12	13	14	15	16
$W_s(h)$	33	24	16	8	5	3	2	1
$\lceil A_s(h) \rceil$	20	14	9	6	4	3	2	1

Table 5. Performance of an ABD(16,9)

We see that while $W_s(h)$ approximates $A_s(h)$ reasonably well, performance has been degraded somewhat. We conjecture that better ABD(16,9)'s exist. (It is a situation reminiscent of the fact that recursive or systematic constructions of error-correcting codes tend not to work out as well as a random code would).

While it is not too difficult to calculate $W_s(h)$ for small ABD's constructed by insertion (assuming that our conjecture about the nature of the worst-case is correct), the asymptotic analysis seems difficult. In any case the ABD's so constructed yield large k/w designs with significantly improved $W_s(h)$.

Thus ABD's are seen to permit partial-match searches to be performed optimally (with respect to the average retrieval time) while greatly reducing the worst-case retrieval time. The disadvantages are that it is difficult to construct (or even to show the existence of of) ABD's of an arbitrary type (k,w), and also that the calculation of the hash function h itself becomes sufficiently complicated so that using ABD's would be justified only when the file is stored on slower secondary storage.

In the next section we see how the complications of ABD's can be avoided at the cost of extra storage utilization.

An alternative solution.

By using moderate amounts of extra storage one can achieve very good worst-case retrieval times. The idea is very simple. Suppose we have enough storage to store each record m times. Then we create m bucket systems. Each record is divided into m fields of k/m bits each, the i-th field is used directly as the bucket address for the record in the i-th system.

If $b = 2^w$ (where $w = k/m$) is the number of buckets per system, then for any query $q \, Q_s$ we use the bucket system corresponding to the field of q having the most bits specified to perform the search. Since this field must have at least $\lceil s/m \rceil = \lceil ws/k \rceil$ bits specified, we have

$$W_s(h) \leq 2^{w - \lceil ws/k \rceil}$$

This compares very favorably with the lower bound of $b^{1-s/k}$ for $A_s(h)$ for a single balanced bucket system.

The average behavior of this scheme is not easy to compute, but $A_s(h)$ approaching $W_s(h)$ seems likely, especially when w is large.

Variations of the scheme exist, using an ABD for each bucket system, etc...

5. Conclusions.

We see that generalized hash-coding algorithms permit partial-match searches to be performed efficiently, both on the average and in the worst-case. We conjecture that these algorithms do about as well as possible, in the sense that any algorithm performing partial match searches on a file F of k bit records, which uses $c.|F|.k$ bits of storage to represent F, must examine at least $(c.|F|.k)^{1-s/k}$ bits (approximately) of that representation in order to calculate the response to a query $q \epsilon Q_s$, both on the average and in the worst-case.

In addition to this conjecture, many interesting open problems remain, in particular the existence question for ABD's of arbitrary type (k,w), and the appropriate generalizations of theorems 1 and 2 for more realistic situations where queries have differing probabilities, etc... Other less stringent definitions of an ABD are also worth consideration.

The reader is referred to [11] for a more detailed discussion of these issues, for proof details omitted here due to lack of space, and an analysis of a digital-tree partial-match search problem.(Also given there is a proof of an analogue of theorems 1 and 2 for best-match (rather than partial-match) queries.).

Acknowledgements

I would like to thank the many people who were particularly helpful to me while this research was being carried out ; in particular Prof. Robert Floyd, Prof. Donald Knuth, Prof. David Klarner, Ronald Graham N.G. de Bruijn, Malcom Newey, my parents and Gail. I should also like to thank the National Science Foundation, the Stanford Artificial Intelligence Laboratory, and IRIA-Laboria for their support, and Nicole Hornebeck for her speedy typing of this manuscript.

References.

[1] Abraham, C.T., S.P. Ghosh, and D.K. Ray Chaudhuri. File Organization Schemes Based on Finite Geometries, Information and Control 12 (February 1968), 143-163.

[2] Feldman, Jerome A and Paul D. Rovner. An Algol-Based Associative Language. CACM 12 (August 1969), 439-449.

[3] Gray. H.J. and N.S. Prywes. Outline for a Multi-list organized system. Annual Meeting of the ACM. Paper 41. Cambridge, Mass. (1959). 7 pp.

[4] Gustafson, Richard Alexander. A Randomized Combinatorial File Structure for Storage and Retrieval Systems. Ph.D. Thesis. University of South Carolina (1969) 92 pp.

[5] Hardy, G.H., J.E. Littlewood, and G. Polya. Inequalities. Cambridge University Press (1959).

[6] Knuth, Donald E. The Art of Computer Programming 3 (Sorting and Searching). Addison-Wesley (1972).

[7] Minsky, Marvin and Seymour Papert. Perceptrons An Introduction to Computational Geometry. The MIT Press. (Cambridge, Mass. 1969).

[8] Minker, Jack. An Overview of Associative or Content Addressable Memory Systems and a KWIC index to the literature. Technical Report TR-157. University of Maryland Computer Computer Science Center. (College Park, Md: June 1971), 140 pp.

[9] Preparata, Franco Personal Communication.

[10] Ray-Chaudhuri, D.K. Combinatorial Information Retrieval Systems for Files. SIAM J. Appl. Math. 16 (September 1968), 973-992.

[11] Rivest, Ronald L. Analysis of Associative Retrieval Algorithms. Ph.D. Thesis, Stanford University Computer Science Report STAN-CS-74-145 (May, 1974). Also available as IRIA Report # 54 (February 1974). A revised version to appear in SIAM J. Computing.

[12] Welch, Terry A. Bounds on the information Retrieval Efficiency of Static File Structures. Project MAC Report MAC-TR-88. MIT (June 1971). (Ph.D. Thesis) 163 pp.