SIAM J. COMPUT.
Vol. 14, No. 2, May 1985

016

# OPTIMUM ALGORITHMS FOR A MODEL OF DIRECT CHAINING*

JEFFREY SCOTT VITTER† AND WEN-CHIN CHEN‡

**Abstract.** Direct chaining is a popular and efficient class of hashing algorithms. In this paper we study optimum algorithms among direct chaining methods, under the restrictions that the records in the hash table are not moved after they are inserted, that for each chain the relative ordering of the records in the chain does not change after more insertions, and that only one link field is used per table slot. The *varied-insertion coalesced hashing method* (VICH), which is proposed and analyzed in [CV84], is conjectured to be optimum among all direct chaining algorithms in this class. We give strong evidence in favor of the conjecture by showing that VICH is optimum under fairly general conditions.

**Key words.** analysis of algorithms, searching, information retrieval, hashing, coalesced hashing, data structures, optimality

**1. Introduction.** There are many classes of hashing algorithms in use today: separate chaining, coalesced hashing, linear probing, double hashing, and quadratic probing, to name a few. (More details can be found in [Knu73].) Comparisons between hashing algorithms in different classes are often difficult, because each class has its own assumptions, storage requirements, and tradeoffs. For example, some hashing algorithms (as in [Bre73]) do extra work during insertion in order to speed up later searches. In some applications, the preferred class of hashing methods is determined by the special nature and requirements of the application. The task is then to find the optimum algorithm within that class.

This paper is concerned with optimum algorithms within one popular class of hashing algorithms—*direct chaining without restructuring.* This implies that the lists coalesce. Throughout this paper, we will denote the number of inserted records by $N$ and the number of slots in the hash table by $M'$. We assume that there is a predefined and quickly computed *hash function*

$$\textit{hash}: \{\text{all possible keys}\} \to \{1, 2, \cdots, M\} \tag{1}$$

that assigns each record to its *hash address* in a uniform manner. The first $M$ slots, which serve as the range of the hash function, are called the *address region*; the remaining $M' - M$ slots make up the *cellar.*

Direct chaining works as follows: The search for a record with key $K$ begins at slot *hash* $(K)$ and continues through the linked chain of records until either the record is found (i.e., the search is *successful*) or else the end of the list is reached (i.e., the search is *unsuccessful*). When a record with a key $K$ is inserted, it must become part of the chain that includes slot *hash* $(K)$, so that later searches for that record will succeed.

In this paper we study *optimum* direct chaining algorithms under the following model: the records cannot be moved once they are inserted into the hash table (e.g., the records might be "pinned" to their locations by pointers to them from outside the table, or the records might be very large so that moving them is expensive), the relative ordering of the records in each chain does not change after more records are added,

* Received by the editors December 20, 1983, and in revised form September 24, 1984. This research was supported by an IBM contract.

† Department of Computer Science, Brown University, Providence, Rhode Island 02912. The research of this author was supported in part by the National Science Foundation under grant MCS-81-05324 and by ONR and DARPA under contract N00014-83-K-0146 and ARPA Order No. 4786.

‡ Department of Computer Science, Brown University, Providence, Rhode Island 02912. Present address, GTE Laboratories Inc., Waltham, Massachusetts 02245.

and there is only one link field per table slot. This model does not allow restructuring of the hash table while the table is being constructed.

Under this model, when a record *collides* with another record during insertion (i.e., its hash address is already occupied), an empty slot is allocated to store the new record, and that slot is linked into the chain containing slot *hash* ($K$) at some point in the chain after slot *hash* ($K$). We call a record that collides during insertion a *collider.* Insertion algorithms in this model can differ from one another only in the ways that the following two decisions are made:

(1) Which empty slot is allocated to store the collider?

(2) At what point in the chain following slot *hash* ($K$) should the collider be linked?

The measures of performance we use to compare algorithms is the *number of probes per successful search* and the *number of probes per unsuccessful search.* In both cases this is the number of distinct slots accessed during the search. We use the probability model that the $M^N$ sequences of hash addresses are equally likely. For successful searches, we also assume that each of the $N$ inserted records in the hash table is equally likely to be the object of the successful search. For insertions and unsuccessful searches, we assume that each of the $M$ address region slots is equally likely to be the hash address for the unsuccessful search. In other words, insertions and searches are assumed to be *random.*

When there is no cellar, the way in which decision 1 is made is not important, as far as the average search performance is concerned. In the case in which there is a cellar, most methods use a statically-ordered available-slot list, in which empty slots are allocated in some fixed relative order. Performance seems to be best when cellar slots get higher priority over noncellar slots on the available-slot list. When that is the case, a collider is stored in an empty cellar slot, if one is available. When the cellar gets full, subsequent colliders must be stored in the address region. This may cause collisions with records inserted later. For example, in Fig. 1, LEO collides with FRANCIS during insertion and is stored in the address region (in slot 10), since there is no cellar. When GARY is inserted later, GARY collides with LEO at slot 10. Thus GARY and LEO become part of the same chain, even though they have different hash addresses. This phenomenon, which we call *coalescing*, tends to make search times longer. Intuitively, it makes sense to give higher priority to the cellar slots on the available-slot list, because storing a collider in the address region can cause coalescing to occur during a later insertion.

Several methods have been proposed for handling decision 2, all having the generic name of *coalesced hashing.* The original method, *late-insertion coalesced hashing* (LICH), was introduced in [Wil59] and analyzed in [Knu73], [Gui76], [GK81], [Vit82b], [Vit83], and [CV84]. In LICH, a collider is linked to the *end* of the chain that contains slot *hash* ($K$). The *early-insertion coalesced hashing* method (EICH) proposed in [Vit82b] and [Kno84] inserts each collider into the chain at the point *immediately after* slot *hash* ($K$).

For the special case of *standard coalesced hashing* (in which there is no cellar) these two methods are referred to as LISCH and EISCH. An example is given in Fig. 1. The record WEN collides with FRANCIS at slot 1. In the LISCH method illustrated in Fig. 1(a), WEN is linked at the end of the chain containing FRANCIS. With EISCH in Fig. 1(b), WEN is inserted into the chain at the point between FRANCIS and LEO. The average successful search time in Fig. 1(b) is slightly better than in Fig. 1(a), because inserting WEN immediately after FRANCIS (rather than at the end of the chain) reduces the search time for WEN from four probes to two and increases the

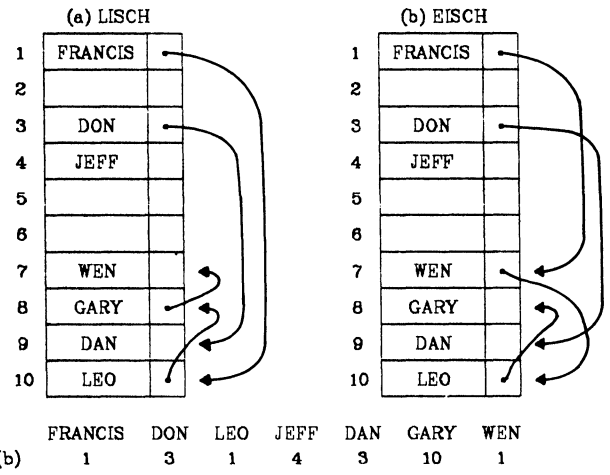


FIG. 1. *Standard coalesced hashing, $M' = M = 10$, $N = 7$.* (a) LISCH, (b) EISCH.

search time for LEO from two probes to three. The result is a net decrease of one probe. The expected search performance for EISCH is derived in [CV83] and [Kno84]; EISCH results in faster searches, on the average, than does LISCH.

When there is a cellar, however, LICH performs better than EICH, as illustrated in Fig. 2. The insertion of WEN using EICH in Fig. 2(b) causes both cellar records LEO and JEFF to move one more link further from their hash addresses. That does not happen using LICH in Fig. 2(a).

The *varied-insertion coalesced hashing* method (VICH) was introduced in [Vit82b] as a means of combining the strong points of both LICH and EICH without their

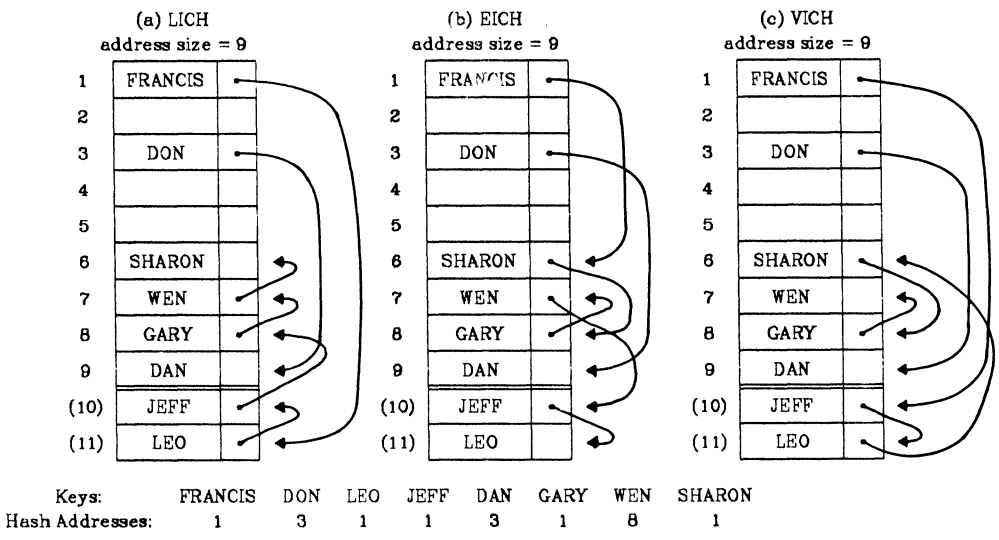


FIG. 2. *Coalesced hashing, $M' = 11$, $M = 9$, $N = 8$.* (a) LICH, (b) EICH, *and* (c) VICH.

weaknesses. In VICH, the collider is linked into the chain directly after its hash address (as in EICH) *except* when the cellar is full, there is at least one cellar slot in the chain, and the hash address of the collider is the location of the first record in the chain; in that case, the collider is linked into the chain directly after the last cellar slot in the chain. When there is no cellar, VISCH is identical to EISCH. An example of VICH appears in Fig. 2(c). The analyses of LICH, EICH, and VICH given in [CV84] show that VICH performs better, on the average, than both LICH and EICH.

In the next section we conjecture that VICH is *optimum* among all direct chaining methods, under the model explained above. The main result of this paper is a vote in favor of this conjecture, showing that VICH is optimum under the above conditions when cellar slots are given priority on the available-slot list.

**2. Search-time optimality of varied-insertion.** In this section we investigate the search-time optimality of VICH among chaining methods that insert records directly into the hash table. The sizes of the address region and cellar are fixed. We assume that records cannot be moved once they are inserted. We also assume that the relative ordering of the records in the chains does not change after more records are inserted. In other words, the optimization illustrated in Fig. 3(c) is not allowed. Finally, we assume that there is only one link field per table slot.

The major open question is whether VICH is optimum under this model. We conjecture that the answer is yes. We will use the term *admissible* to refer to any direct chaining insertion algorithm that satisfies the assumptions of the above model.

*Conjecture.* Varied-Insertion Coalesced Hashing (VICH) gives the optimum expected search times among all admissible chaining methods, with the probability assumptions that the $M^N$ sequences of hash addresses are equally likely and that insertions and searches are random.

In this section we give strong support for the conjecture: We show that VICH uses a greedy method of inserting records, that is, VICH is a locally optimum admissible method for inserting a single record. For the special case in which there is no cellar, we show that VICH is not only locally optimum, but also globally optimum. The main result is showing that VICH is globally optimum among all admissible chaining methods that give the cellar slots priority on the available-slot list.

In coalesced hashing we typically allocate available empty slots in the order $M'$, $M'-1$, $M'-2$, $\cdots$, which means that the cellar slots are allocated before the address region slots. However Lemma 1 shows that for any given ordering of the available-slot

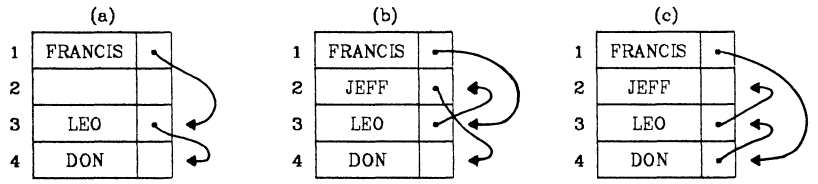


FIG. 3. *Hash table* (a) *is an optimum arrangement of the first three inserted records* FRANCIS, DON, *and* LEO, *as far as subsequent searches are concerned. Table* (b) *pictures the result of inserting* JEFF *using* VICH. *Table* (c) *achieves better successful search times than* (b) *by reordering the chain so that* DON *precedes* LEO. *Assuming that the records cannot be moved once they are inserted, there is no optimum arrangement of the four records in which* LEO *precedes* DON *as in* (a).

list, the chains that are formed are the same regardless of which admissible chaining method is used, except for the order of the individual records within the chains.

LEMMA 1. *For any given ordering of the available-slot list and for any admissible chaining methods, the partition of the inserted records into chains is the same.*

*Proof.* We prove this lemma by induction on the number of records inserted in the table. Assume that the partition of the inserted records into chains is the same for all admissible chaining methods after $k$ records are inserted. Let record $R$ (with key $K$) be the next record inserted. If $R$ does not collide when inserted, then all admissible chaining methods insert $R$ at its hash address *hash* $(K)$. If $R$ collides when inserted, then all admissible chaining methods link $R$ into the chain containing location *hash* $(K)$. Thus the partition of the records into chains is still the same for all admissible chaining methods after $k+1$ insertions. This proves the lemma. □

It is easy to see that permuting the order of the cellar slots on the available-slot list does not affect search performance at all. The following lemma shows that the order of the address region slots on the available-slot list can be permuted among themselves without affecting the *average* search performance. The rigorous statement of the lemma uses the terminology that two hash tables are *homotopic* if and only if their chains can be paired off in a 1-1 correspondence so that the search times for the $i$th records in two corresponding chains are the same, for each $i$ and for any pair of corresponding chains. The formal definition of homotopic appears in [Vit82a], which used the notion in connection with deletion algorithms that preserve randomness.

LEMMA 2. *For any admissible chaining algorithm, and for any two orderings of the address region slots in the same fixed positions on the available-slot list, there is a* 1-1 *correspondence between the two classes of* $M^N$ *hash tables obtained by using the two available-slot lists such that each pair of corresponding tables is homotopic.*

*Proof.* We prove this lemma by constructing explicitly the 1-1 correspondence. Let $a_1, a_2, \cdots, a_M$ and $\sigma(a_1), \sigma(a_2), \cdots, \sigma(a_M)$ be the two orderings of the available-slot list. The hash table obtained by inserting $k$ records with hash addresses $h_1, h_2, \cdots, h_k$ using the first ordering of the available-slot list is homotopic to the hash table obtained by the insertion of records with hash addresses $\sigma(h_1), \sigma(h_2), \cdots, \sigma(h_k)$ using the latter ordering. For each $i$, the record in slot $i$ in the first table is in slot $\sigma(i)$ in the second table. □

The following definitions are used to prove the remaining results of this section. A chain of records $R_1, R_2, \cdots, R_k$, inserted in that order by linking algorithms $A_1, A_2, \cdots, A_k$, respectively, can be defined as follows: The chain $[(R_1, A_1)]$ is the chain containing $R_1$ only. Given a chain $L_1 = [[\cdots[[(R_1, A_1)], (R_2, A_2)], \cdots], (R_{k-1}, A_{k-1})]$, the chain $L = [L_1, (R_k, A_k)]$ is obtained by linking $R_k$ into $L_1$ by using algorithm $A_k$. For simplicity, we will use $[(R_1, A_1), \cdots, (R_k, A_k)]$ to denote $L$. If a record $R$ is contained in a chain $L = [(R_1, A_1), \cdots, (R_k, A_k)]$, then we define *Search* $(R, L)$ to be the number of probes required to search successfully for $R$ in $L$, and *Search* $(L)$ to be $\sum_{1 \le i \le k}$ *Search* $(R_i, L)$. We also define *loc* $(R)$ to be the absolute location in the hash table of the slot containing $R$, and $\phi(R, L)$ to be the number of records that are stored in slots in the chain $L$ following $R$, whose hash addresses are either *loc* $(R)$ or one of the slots before $R$ in $L$.

Theorem 1 shows that for successful searches, VICH is locally optimum among all admissible chaining methods. VICH uses a greedy method of inserting records, in that each individual insertion is done so as to minimize the resulting average successful search time.

THEOREM 1. *Given a record $R$ with key $K$ and a chain $L$ containing location* hash $(K)$, *an optimum place to link $R$ into $L$ in order to minimize* Search $(L')$, *where $L'$*

*is the chain obtained by linking R into L, is immediately before the first noncellar slot following slot hash (K) in the chain, or else at the end of the chain if no such noncellar slot exists.*

*Proof.* If slot *hash* $(K)$ is the last slot in $L$, then all admissible chaining methods link $R$ immediately after slot *hash* $(K)$. Therefore, we will assume that before $R$ is inserted, slot *hash* $(K)$ is not the last slot in $L$ and that record $R_1$ is stored in a slot in chain $L$ that follows slot *hash* $(K)$. Suppose that there are two different methods $A$ and $B$ that link record $R$ into chain $L$. Method $A$ links $R$ immediately after the slot containing $R_1$. Method $B$ links $R$ immediately before the slot containing $R_1$. Let $L_A$ denote the chain $[L, (R, A)]$ and $L_B$ denote the chain $[L, (R, B)]$. We will prove the theorem by showing that

$$Search\ (L_A) \geqq Search\ (L_B), \tag{2}$$

and that if the slot that contains $R_1$ is a cellar slot, then

$$Search\ (L_A) = Search\ (L_B). \tag{3}$$

Since method $A$ links $R$ into chain $L$ immediately after the slot containing $R_1$ and method $B$ links $R$ into chain $L$ immediately before the slot contining $R_1$, we have

$$Search\ (R, L_A) = Search\ (R, L_B) + 1; \tag{4}$$

and

$$Search\ (R_1, L_B) = Search\ (R_1, L_A) + 1. \tag{5}$$

Formula (5) follows from the fact that $R_1$ cannot be stored at its hash address, or else it would not be in the chain $L$. For those records $R'$ that are stored in slots following $R_1$ in $L$, and whose hash addresses are *loc* $(R_1)$, we have

$$Search\ (R', L_A) = Search\ (R', L_B) + 1. \tag{6}$$

Since the hash address of $R'$ is *loc* $(R_1)$ and $R$ is linked immediately after $R_1$ in $L_A$, it requires one more probe to search for $R'$ in $L_A$. Note that if $R_1$ is stored in the cellar, then there are no records $R'$ with hash addresses *loc* $(R_1)$. Finally, if $\bar{R}$ is one of the remaining records in both chains, we have

$$Search\ (\bar{R}, L_A) = Search\ (\bar{R}, L_B). \tag{7}$$

This proves (2) and (3), and thus proves the theorem. □

The following theorem shows that in the case in which there is no cellar, then VISCH is not only locally optimum, but also globally optimum.

THEOREM 2. *Assume that there is no cellar. For any set of records* $R_1, R_2, \cdots, R_k$ *that forms a chain, with the assumption that the* $M^k$ *possible sequences of hash addresses* $h_1, h_2, \cdots, h_k$ *are equally likely, the average unsuccessful and successful search times on the chain are optimized by the* VISCH (=EISCH) *ordering of the records in the chain.*

*Proof.* Lemma 1 shows that the partition of the inserted records into chains is the same for all admissible chaining methods. Since the average unsuccessful search time on a chain depends on the length of the chain and not on the ordering of the records within the chain, then by Lemma 1, the average unsuccessful search times are the same for all admissible chaining methods.

To prove that VISCH is optimum for the successful search case, we need the following formula. Let $L$ be the chain $[(R_1, A_1), \cdots, (R_k, A_k)]$; then

$$Search\ (L) = k + \sum_{1 \leqq i \leqq k} \phi(R_i, L). \tag{8}$$

This formula can be proved as follows: From the definition of *Search* $(L)$, we have

$$\textit{Search}\ (L) = \sum_{1 \leqq i \leqq k} \textit{Search}\ (R_i, L). \tag{9}$$

Now consider each record $R_i$ in chain $L$. If *Search* $(R_i, L) = s$, then $s$ probes are required to search for $R_i$ in $L$. Let $R_{i_1}, \cdots, R_{i_{s-1}}$ be the records stored in the slots in $L$ that are probed while searching for $R_i$. The search for $R_i$ contributes 1 to each of the following terms: $\phi(R_{i_1}, L), \cdots, \phi(R_{i_{s-1}}, L)$. It also contributes 1 to the term $k$ in the right-hand side of (8). Therefore, the search for $R_i$ contributes $s$ to both sides of (8). This proves (8).

Now we prove the optimality of VISCH for successful searches by induction on $k$, the number of records in the chain. Assume that for a given set of records $R_1, \cdots, R_k$, with random hash addresses $h_1, \cdots, h_k$, the average successful search times on the chain are optimized by the VISCH ordering of the records in the chain. We will show that VISCH still gives the optimum ordering for the records $R_1, \cdots, R_{k+1}$, where $R_{k+1}$ is the next record inserted. The hash address of $R_{k+1}$ can be any of the $M$ address region slots, with equal probability. Formally, if we let $L_V$ denote the chain $[(R_1, \text{VISCH}), \cdots, (R_k, \text{VISCH})]$ and let $L_A$ denote the chain $[(R_1, A), \cdots, (R_k, A)]$, for any admissible chaining method $A$, then with the assumption of induction

$$\sum_{*} \textit{Search}\ (L_V) \leqq \sum_{*} \textit{Search}\ (L_A), \tag{10}$$

we will show

$$\sum_{*} \sum_{1 \leqq i \leqq k} \textit{Search}\ ([L_V, (S_i, \text{VISCH})]) \leqq \sum_{*} \sum_{1 \leqq i \leqq k} \textit{Search}\ ([L_A, (S_i, \text{VISCH})]) \tag{11}$$

and

$$\sum_{*} \sum_{1 \leqq i \leqq k} \textit{Search}\ ([L_A, (S_i, \text{VISCH})]) \leqq \sum_{*} \sum_{1 \leqq i \leqq k} \textit{Search}\ ([L_A, (S_i, A)]), \tag{12}$$

where $S_i$ is a record with hash address $loc\ (R_i)$ and the symbol $*$ under $\sum$ represents the summation condition "all possible sequences of hash addresses $h_1, \cdots, h_k$ such that records $R_1, \cdots, R_k$ are linked together to form a chain." Inequalities (11) and (12) combined prove that VISCH is optimum for successful searches.

Inequality (12) is true from Theorem 1, which showed that VICH is locally optimum. Inequality (11) is shown in the following way by applying (8): If the record $S_i$ is linked into chain $L_V$ immediately after the slot containing $R_i$ by using VISCH, then we have

$$\textit{Search}\ ([L_V, (S_i, \text{VISCH})]) = 2 + \phi(R_i, L_V) + \textit{Search}\ (L_V). \tag{13}$$

This formula is true, since after the insertion of $S_i$, it requires two probes to search for $S_i$ in $[L_V, (S_i, \text{VISCH})]$, and it requires one more probe to search for those records stored in slots in $L_V$ that follow $R_i$ and whose hash addresses are either $loc\ (R_i)$ or one of the slot before $R_i$ in $L_V$. From (13), the left-hand side of (11) is equal to

$$\begin{aligned} \sum_{*} \sum_{1 \leqq i \leqq k} (2 + \phi(R_i, L_V) + \textit{Search}\ (L_V)) &= \sum_{*} (2k + k\ \textit{Search}\ (L_V) + \sum_{1 \leqq i \leqq k} \phi(R_i, L_V)) \\ &= \sum_{*} (k + (k+1)\ \textit{Search}\ (L_V)). \end{aligned} \tag{14}$$

The last equality follows from (8). Similarly, the right-hand side of (11) is equal to

$$\sum_{*} (k+(k+1)\ \textit{Search}\ (L_A)). \tag{15}$$

Thus, (11) follows from (10) immediately.

From the above arguments, we conclude that VISCH gives the optimum ordering of records in the chain that minimizes both the successful and unsuccessful searches times. This proves the theorem. □

The weakness of Theorem 2 is that EICH is also a greedy insertion method, but with a cellar EICH is definitely not optimum. The next theorem strengthens the previous results. It shows that when the cellar slots get priority on the available-slot list, VICH is globally optimum.

THEOREM 3. *Assume that the cellar slots are given priority on the available-slot list. For any set of records* $R_1, R_2, \cdots, R_k$ *that forms a chain, with the assumption that all possible hash addresses* $h_1, h_2, \cdots, h_k$ *are equally likely, the average unsuccessful and successful search times on the chain are optimized by the* VICH *ordering of the records in the chain.*

*Proof.* We first note that for each chain the hash address of each cellar slot is the location of the first slot of the chain. Now we show that VICH optimizes the average unsuccessful search times. Let us assume that the noncellar slots in the chain are in relative positions $p_1, p_2, \cdots, p_{a-1}, p_a$ in the chain, counting backwards from the end of the chain. Note that $1 \leqq p_1 \leqq \cdots \leqq p_{a-1} \leqq p_a = k$, where $k$ is the length of the chain. The average unsuccessful search time on the chain is

$$\frac{1}{a}(p_a + p_{a-1} + \cdots + p_1), \tag{16}$$

which can be minimized by letting $p_1 = 1$, $p_2 = 2$, $\cdots$, $p_{a-1} = a-1$. This means that all the cellar slots in the chain immediately follow the first slot in the chain. This is feasible, since the hash addresses of all the cellar slots are the location of the first slot of the chain. VICH yields the above ordering, and thus minimizes the average unsuccessful search times on the chain.

To prove that VICH optimizes the average successful search times, we will show that (a) the optimum placements of the cellar slots in the chain are immediately after the first slot of the chain, and that (b) the optimum relative ordering of the noncellar slots is the ordering obtained by using VICH.

(a) Assume that an ordering of records in the chain is obtained by using method $A$, in which a cellar slot immediately follows a noncellar slot. Let records $R_1$ and $R_2$ be the records stored in the noncellar slot and the cellar slot, respectively. We assume that $R_1$ is *not* the start of the chain. Assume also that another ordering of records in the chain is obtained by using method $B$, which is the same as that for $A$ except that $R_2$ immediately precedes $R_1$ in the chain. Let $L_A$ denote the chain obtained by using method $A$, and $L_B$ denote the chain obtained by using method $B$. It suffices to show that

$$\textit{Search}\ (L_A) \geqq \textit{Search}\ (L_B). \tag{17}$$

Since the hash address of $R_2$ is the location of the first slot of the chain, we have

$$\textit{Search}\ (R_2, L_A) = \textit{Search}\ (R_2, L_B) + 1. \tag{18}$$

Also, we have

$$\textit{Search}\ (R_1, L_B) = \textit{Search}\ (R_1, L_A) + 1, \tag{19}$$

since the slot containing $R_2$ precedes immediately the slot containing $R_1$ in $L_B$. For those records $R'$ that are stored in slots following $R_1$ in $L_A$, and whose hash addresses are $loc\,(R_1)$, we have

$$Search\,(R', L_A) = Search\,(R', L_B) + 1. \tag{20}$$

This follows since $loc\,(R_2)$ must be probed to search for $R'$ in chain $L_A$. For the remaining records $\bar{R}$ in both chains, we have

$$Search\,(\bar{R}, L_A) = Search\,(\bar{R}, L_B). \tag{21}$$

This shows that the optimum placements of the cellar slots are immediately after the first slot of the chain.

(b) By the same arguments as those in the proof of Theorem 2, we can prove that the optimum relative ordering of the noncellar slots is the ordering obtained by using the VICH method.

From (a) and (b), we conclude that the average successful search time on the chain is optimized by the VICH ordering of the records in the chain. □

In order to prove the conjecture that VICH is globally optimum, it suffices to show that the best way to allocate empty slots for colliders is to have an available-slot list in which the cellar slots have higher priority than the address region slots. The conjecture would then follow from Lemma 1, Lemma 2 and Theorem 3.

The difficulty with proving the conjecture is due to the many exotic hash algorithms that must be considered if the cellar slots are not given priority on the available-slot list. The priorities assigned to slots on the available-slot list might be dynamic, for example. For that reason, the optimum algorithm in our model could turn out to be a method that is not practical. We believe that such is not the case. We conjecture that VICH, which has a very efficient implementation, is optimum in our model.

**3. Conclusions and open problems.** We have given strong evidence in support of our conjecture that VICH is optimum among all direct chaining methods, under the assumptions that the records are not moved once they are inserted, that for each chain the relative ordering of its record does not change after further insertions, and that there is only one link field per table slot. In particular, we have shown that the conjecture is true under the additional condition that cellar slots are given priority on the available-slot list. Intuition suggests that this extra condition will always be true for optimum algorithms under the above assumptions, however, determining whether the conjecture is true in general seems to be quite challenging. If VICH is shown not to be optimum, it is hopeful that the insights gained from the proof will lead to the construction of an optimum algorithm.

There are other interesting open problems concerning this model of hashing. One problem is to study the performance of coalesced hashing in external searching, as discussed in [Vit82b]. Another problem concerns deletion algorithms that *preserve randomness.* Preserving randomness means that deleting a record is in some sense like never having inserted it. In particular, the formulas for the average search times after $N$ random insertions intermixed with $d$ deletions are the same as the formulas for the average search times after $N-d$ random insertions. The formal notion of what it means to preserve randomness is defined in [Vit82a]. A deletion algorithm for coalesced hashing is given in [Vit82a] and shown to preserve randomness for late-insertion standard coalesced hashing (LISCH). The authors have recently discovered deletion algorithms that preserve randomness for LICH, EICH, and VICH. It seems that in order to preserve randomness, a deletion algorithm must relocate some records from

time to time, which may not be possible if the records are "pinned" to their locations and are not allowed to be moved. Deletion algorithms that do not relocate records (and do not preserve randomness) should therefore also be studied. It is an open problem to determine how the average search times are affected by deletion algorithms that do not preserve randomness.

## REFERENCES


[Bre73] R. P. BRENT, *Reducing the retrieval time of scatter storage techniques*, Comm. ACM, 16 (1973), pp. 105-109.

[CV83] W. C. CHEN AND J. S. VITTER, *Analysis of early-insertion standard coalesced hashing*, this Journal, 12 (1983), pp. 667-676.

[CV84] ———, *Analysis of new variants of coalesced hashing*, ACM Trans. Database Systems, to appear.

[GK81] D. H. GREENE AND D. E. KNUTH, *Mathematics for the Analysis of Algorithms*, Birkhauser, Boston, 1981.

[Gui76] L. J. GUIBAS, *The analysis of hashing algorithms*, PhD dissertation, Computer Science Dept., Technical Report STAN-CS-76-556, Stanford Univ., Stanford, CA, August 1976.

[Kno84] G. D. KNOTT, *Direct chaining with coalesced lists*, J. Algorithms, 5(1) (1984), pp. 7-21.

[Knu73] D. E. KNUTH, *The Art of Computer Programming. Vol. 3: Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.

[Vit82a] J. S. VITTER, *Deletion algorithms for hashing that preserve randomness*, J. Algorithms, 3 (1982), pp. 261-275.

[Vit82b] ———, *Implementations for coalesced hashing*, Comm. ACM, 25 (1982), pp. 911-926.

[Vit83] ———, *Analysis of the search performance of coalesced hashing*, J. Assoc. Comput. Mach., 30 (1983), pp. 231-258.

[Wil59] F. A. WILLIAMS, *Handling identifiers as internal symbols in language processors*, Comm. ACM, 2 (1959), pp. 21-24.