SIAM J. COMPUT.
Vol. 19, No. 5, pp. 775-786, October 1990

THE SPATIAL COMPLEXITY OF OBLIVIOUS k-PROBE HASH FUNCTIONS*

JEANETTE P. SCHMIDT† AND ALAN SIEGEL‡

Abstract. The problem of constructing a dense static hash-based lookup table T for a set of n elements belonging to a universe $U = \{0, 1, 2, \cdots, m-1\}$ is considered. Nearly tight bounds on the spatial complexity of oblivious $O(1)$-probe hash functions, which are defined to depend solely on their search key argument, are provided. This establishes a significant gap between oblivious and nonoblivious search. In particular, the results include the following:

- A lower bound showing that oblivious k-probe hash functions require a program size of $\Omega((n/k^2)e^{-k} + \log\log m)$ bits, on average.
- A probabilistic construction of a family of oblivious k-probe hash functions that can be specified in $O(n\,e^{-k} + \log\log m)$ bits, which nearly matches the above lower bound.
- A variation of an explicit $O(1)$ time 1-probe (perfect) hash function family that can be specified in $O(n + \log\log m)$ bits, which is tight to within a constant factor of the lower bound.

Key words. hashing, oblivious, spatial complexity, power bound, perfect hashing

AMS(MOS) subject classifications. 68P05, 68P10, 68Q05, 68R05, 68R10

1. Introduction.

Hashing is one of the most important and commonly used methods to organize simple collections of information (see References for a partial list). The applications are extensive, and the subject has a correspondingly rich theoretical literature. In this paper, we will be restricting our attention principally to the dictionary problem, which concerns how to organize a set S of distinct keys within a table T so that the elements can be retrieved quickly. We shall take the data set to be static, so that the hash table need not support insertion or deletion, and consider only open addressing, so that pointers will not be used. Most of our results will focus on 100 percent utilized tables.

The history of this problem, which we detail in the next subsection, would seem to suggest that virtually all questions have been answered for this specific problem; a variety of lower bounds have been established [Y81] and [Me84], and elegant constructions have been discovered, which nearly meet the lower bounds [FKS84].

Recently, however, new techniques for organizing data have been devised [FMNSSS88], [FNSS88], which show that an enormous amount of information can be encoded within a search table. The thrust of these results is to show how to exploit nonoblivious search, which can use an adaptive probe strategy based upon information gleaned from unsuccessful probes. The consequences are performance bounds and extensions for the dictionary problem that had been believed to be impossible, for $O(1)$-probe hashing schemes [FNSS88].

* Received by the editors January 18, 1989; accepted for publication (in revised form) October 11, 1989.

† Computer Science Department, Polytechnic University, 333 Jay Street, Brooklyn, New York 11201. The work of this author was supported in part by Defense Advanced Research Projects Agency grant F49620-87-C-0065, and was in part carried out while the author was visiting the Computer Science Department at Rutgers University.

‡ Computer Science Department, Courant Institute, New York University, New York, New York 10012. The work of this author was supported in part by National Science Foundation grants CCR-8902221 and CCR-8906949 and Office of Naval Research grant N00014-85-K-0046. A preliminary version of this work was published in §§ 1-3 of "On aspects of universality and performance for closed hashing," [SS89].

The unexpected opportunities demonstrated by these results have led us to examine afresh the computational models and assumptions underlying the lower bound of [Me84] and the construction of [FKS84]. These results are for 1-probe schemes, which by definition cannot be adaptive. The natural question to ask is whether the opportunity to use the information encoded within the constant number of probed locations is genuinely significant, or if the power of the [FNSS88] scheme is actually due to the ability to query several locations for the search key.

A few preliminary definitions help formalize the problem. We let the data set S comprise n elements belonging to the universe $U = \{0, 1, 2, \cdots, m-1\}$. Our hash-based lookup table T is also of size n. A sequence $H = (h_1, h_2, \cdots, h_k)$ of functions is a k-probe hash function for S, if $H: U \mapsto [1, n]^k$, and $T[1 \cdots n]$ can be organized so that each item $s \in S$ is located is one of the k probe positions defined by applying the k-probe functions to s.

The method of probing must be defined quite precisely. A hashing strategy is **oblivious** if the search locations computed by H are based solely on the key s, and not on other keys stored in T. In this case, the search strategy is

```
for i ← 1, 2, ···, k do
    if T[h_i(s)] = s then return (h_i(s))
endfor;
if s has not been found then return (FAIL).
```

In contrast, a **nonoblivious** hashing scheme can make computational use of the keys encountered during unsuccessful search, which offers a wider and conceivably more powerful range of search strategies. Formally, such a hashing scheme differs in that h_i is an i-ary function mapping U^i into $[1, n]$. The nonoblivious search strategy is

```
for i ← 1, 2, ···, k do
    s_i ← T[h_i(s, s_1, s_2, ···, s_{i-1})];
    if s_i = s then return (h_i(s, s_1, s_2, ··· s_{i-1}))
endfor;
if s has not been found then return (FAIL).
```

A family $\bar{H}$ is a *k-probe* family for U if every n-element subset $S \subset U$ has some k-probe hash function in $\bar{H}$. A 1-probe hash function for S is called a *perfect* hash function, and a perfect family for U denotes a corresponding 1-probe family. (In the exposition that follows, we shall frequently take the liberty of suppressing the implicitly understood parameters n, m, and even k, for notational convenience.) The principal problem we analyze is how many hash functions are needed to define a *k-probe* family $\bar{H}$, in the oblivious hashing model. In particular, we attain estimates for $\log(\bar{H})$, which is the number of bits needed to specify a particular hash function $H \in \bar{H}$.

1.1. Background. Mehlhorn showed that the bit length of a perfect hash function mapping S into a table T of size n_1, where $n \leqq n_1 < (1-\varepsilon)m$, is lower bounded by $\Omega(n^2/n_1 + \log\log m - \log\log n_1)$ [Me82], [Me84]. The explicit construction of efficient perfect hash functions has been explored, among others, by [Me84], [Ma83], [FKS84], [JvE86], and [SvE84]. In [FKS84], an $O(1)$-time computable perfect hashing scheme for full tables (that is to say, $n_1 = n$) is presented, which requires a description of $O(\log\log m + n\sqrt{\log n})$ bits. Jacobs and van Emde Boas [JvE86] reduce the upper bound for $O(1)$ time 1-probe (FKS-like) hashing to $O(\log\log m + n\log\log n)$-bits. Slot and van Emde Boas [SvE84] show that a variation of the FKS-scheme can be made space optimal at a cost of taking $O(n)$ time to hash a value. We show that a space-optimal variation can be attained while maintaining the $O(1)$ time performance for hashing.

Nonoblivious $O(1)$-probe schemes turn out to be much more powerful than 1-probe schemes, where the question of obliviousness, of course, has no bearing. A nonoblivious scheme that needs only $O(\log\log m+\log n)$ bits is presented in [FNSS88] and [FN88] has recently shown that no additional memory is required when m is polynomial in n. No comparable oblivious schemes are known, and the natural question to resolve is how much of the exponential spatial advantage of $O(1)$-probe nonoblivious schemes over 1-probe schemes is due to their adaptive character, and how much is due to the opportunity to perform additional oblivious search. The counting arguments used in [Me84] seem to provide no help in an effort to construct lower bounds for multiprobe oblivious hashing: the $\Omega(n)$-bit portion of the argument collapses even for 3-probe schemes.

We show that the spatial complexity of k-probe oblivious hash functions for full tables is lower bounded by $\Omega(n\alpha^k+\log\log m)$, and that this bound is tight with $e^{-1-2\log k/k}\leqq\alpha\leqq e^{-1+5/k}$. It is worth noting that in contrast to 1-probe schemes, no comparable lower bound can be obtained for $O(1)$-probe schemes for load factors less than 1. Indeed, a probabilistic construction shows that k-probe oblivious hash functions can be specified in as few as $O(\log n+\log\log m)$ bits, when the size n_1 of the hash table is $(1+\varepsilon)n$, $\varepsilon>0$.

For completeness, we note that Mairson [Ma83], [Ma84] analyzed a number of related problems, including binary search adapted to a page oriented hash scheme, where the cost to read a 2^k-record page is fixed, and the data is sorted on each page. He analyzed a scheme limited to reading one page and supporting k rounds of binary search. While his scheme is not oblivious since it uses binary search, its spatial complexity is remarkably close to the spatial complexity of fully obvious k-probe schemes, analyzed in this manuscript.

In § 2, we prove that the spatial complexity of a k-probe oblivious hash function for a set of n elements from the universe $U=[1, m]$ is $\Omega(e^{-k}n/k^2+\log\log m)$. Section 3 shows that, with probability $(1-o(1))$, a random set of $(2^{O(ne^{-k})}\log m)$ k-probe hash functions contains a perfect k-probe hash function for every n-element subset of U. Section 4 exhibits an explicit $O(1)$-time 1-probe hash scheme that uses only $O(\log\log m+n)$ bits of external memory, and thereby shows that the lower bound for 1-probe schemes can be met for $O(1)$ time hash functions.

2. Lower bounds for oblivious search. Mehlhorn's $\Omega(n)$ bound for the size of a perfect hash function [Me82], is based on the following counting argument: the number of functions must be at least the number of n-item subsets, which belong to an m-element universe U, divided by the maximum number of subsets that can be mapped one-to-one into $[0, n-1]$ by a single hash function: $count\geqq\binom{m}{n}/(m/n)^n$. Taking the logarithm of this estimate gives the size bound for such hash functions. Unfortunately, this ratio decreases by a factor of k^n when k-probe maps are allowed. Formally, a hash function defines a bipartite graph on $B=U\times\{0,\cdots,n-1\}$. In the 1-probe case, each vertex in U has degree 1, and the count of the number of possible coverings of $\{0,\cdots,n-1\}$, (subgraphs in which each vertex in $\{0,\cdots,n-1\}$ has degree 1), afforded by a single function is precisely the number of different subsets serviceable by the hash function. Once k probes are allowed, the degree of each vertex increases by a factor of k. When the resulting bipartite graph supports a perfect matching for some n-element subset A in U, this perfect matching can be used to store A in the table and the k-probe hash function can retrieve it. The number of coverings of $\{0,\cdots,n-1\}$ is easily estimated (by $(km/n)^n$), but may overcount the number of serviceable subsets by as much as a factor of k^n, for $k>1$.

Consequently, we are obliged to model the k-probe scheme more carefully. We model a family of such hash functions as a set $\bar{H}$ of bipartite graphs on B, where each node of U has degree k, for each graph. $\bar{H}$ contains a k-probe perfect hash function for each n-element subset of U if and only if each n-element subset has a perfect matching for some $H \in \bar{H}$. It should be emphasized that these performance bounds are for oblivious probe schemes with sufficient probing to store the data: given a suitable bipartite graph, the data cannot be successfully stored without some means of extracting a matching within the graph. Algorithms for such problems are well known [HK73], and probabilistic incremental approaches have also been analyzed in connection with hashing [SS80]. In any case, the lower bound for the number of bits needed to describe such a k-probe oblivious hash function is thus $\log |\bar{H}|$.

To estimate $|\bar{H}|$, we choose an arbitrary $H \in \bar{H}$ and a randomly selected n-element set $V \subset U$ and compute an upper bound $\bar{p}$ for the probability that $H \in \bar{H}$ provides a perfect matching for V. It then follows that the number of n element subsets of U, for which h is perfect is at most $\binom{m}{n}\bar{p}$. Thus

$$|\bar{H}| \geqq 1/\bar{p}$$

and members of $\bar{H}$ cannot be identified in fewer than $\log_2 (1/\bar{p})$ bits.

By letting V be selected at random, we may imagine an honest intermediary who conveys, as answers to queries by our (partial) matching algorithm, precise (minimal amounts of) information about the items selected.

We will estimate the probability that a carefully selected subset $\tilde{\eta} \in [1, n]$ of n/ck items is covered by edges emanating from V. (The exact value of c will be specified later.)

For $v \in U$, we define $Image(v) = \{i \in [1, n] | \{v, i\}$ is an edge in the graph $H\}$,

For $i \in [1, n]$ we define $Preimage(i) = \{v \in U | \{v, i\}$ is an edge in the graph $H\}$.

Let $P_i = |Preimage(i)|$.

Let V_i be an ordered set of elements denoted by $(v_1^i, v_2^i, \cdots)$.

The selection process for $\tilde{\eta}$, which is based on the fact that elements with small P_i value in η are not overly likely to be covered by the edges emanating from a randomly selected set V, proceeds as follows:

0. Let $V_0 \leftarrow V$, $\eta_0 \leftarrow \{1, \cdots, n\}$, $\tilde{\eta} \leftarrow \emptyset$.
1. For $i \leftarrow 0$ to $n/ck - 1$ perform 2 through 5:
2. Let ν_i be an element in η_i with minimum P_{ν_i}, and set $\tilde{\eta} \leftarrow \tilde{\eta} \cup \{\nu_i\}$.
3. Sequence through the elements of V_i and stop at the first $v^i \in V_i$ that is in $Preimage(\nu_i)$. If no such v^i is found return "fail", *H is not perfect for V.*
4. $V_{i+1} \leftarrow V_i - \{v^i\}$.
5. Set $\eta_{i+1} \leftarrow \eta_i - Image(v^i)$.
   Note that all the preimages of items in η_{i+1} are contained in V_{i+1}, and $|\eta_{i+1}| \geqq n - (i+1)k \geqq n - n/c$.

If the above procedure fails, then there is no matching from V to $[1, n]$. (The converse is not true, of course, but our aim is to upper bound the probability of a success.)

We have to estimate the probability that V contains an element in $Preimage(\nu_i)$, for $0 \leqq i < n/ck$, at step 3. The query at step 3 is whether $v^i \in V_i$ is in $Preimage(\nu_i)$. Only at step 5 is $Image(v^i)$ actually revealed. Since $P_{\nu_0} \leqq km/n$, the conditional probability that v_j^0, the jth element in V_0, covers ν_0 given that the first $j-1$ do not, is

$$\text{Prob}\{v_j^0 \in Preimage(\nu_0) | v_1^0, \cdots, v_{j-1}^v \notin Preimage(\nu_0)\} \leqq \frac{km}{n(m-(j-1))} \leqq \frac{k}{n(1-n/m)}.$$

The estimates of the probabilities of success for $i > 0$, in step 3, are slightly more delicate. In particular, the event: "$v_j^i \in Preimage(\nu_i) | v_1^i, \cdots, v_{j-1}^i \notin Preimage(\nu_i)$" has

to be conditioned by the additional information concerning v_j^i, acquired by previous queries. However, all we have learned is that v_j^i is not in the *Preimage* of some of the elements $\nu_0, \nu_1, \cdots, \nu_{i-1}$, that $v_j^i \notin V - V_i$ and that $v_j^i \notin \{v_1^i, \cdots, v_{j-1}^i\}$. The probability that $v_j^i \in \textit{Preimage}(\nu_i)$ is maximized if *none* of the elimimated candidates are in *Preimage*(ν_i). Since *Preimage*(ν_l) (where ν_l is selected in step 2) is among the $kl+1$ smallest *Preimages* of elements in η_0, it follows that $\sum_{l=0}^{i-1} P_{\nu_l} \leqq ikm/n$; also $V - V_i \subset \bigcup_{l=0}^{i-1} \textit{Preimage}(\nu_l)$, and therefore in the most extreme case, the conditional information can eliminate at most $(\sum_{l=0}^{i-1} P_{\nu_i} + j - 1 < m/c + n)$ elements of U as candidates for v_j^i. It follows that

$$\text{Prob}\{v_j^i \in \textit{Preimage}(\nu_i) \,|\, \text{all previous events}\} \leqq \frac{P_{\nu_i}}{(m - m/c - n)},$$

and

$$\begin{aligned}\text{Prob}\{\nu_i \in \textit{Image}(V_i) \,|\, \text{all previous events}\} &\leqq 1 - \left(1 - \frac{P_{\nu_i}}{m - m/c - n}\right)^{n-i} \\ &\leqq 1 - \left(1 - \frac{P_{\nu_i}}{m - m/c - n}\right)^{n}.\end{aligned}$$

And finally

$$\begin{aligned} p &= \text{Prob}\{H \text{ is perfect for } V\} \\ &\leqq \text{Prob}\{\text{that the } \textit{Image} \text{ of } V \text{ contains all the elements in } \tilde{\eta}\} \\ &\leqq \prod_{i=0}^{n/ck-1} \left(1 - \left(1 - \frac{P_{\nu_i}}{m - m/c - n}\right)^{n}\right).\end{aligned}$$

Since $\sum_{l=0}^{i-1} P_{\nu_l} \leqq ikm/n$, by construction, the above product is maximized when all P_{ν_l} are set to km/n. Hence

$$\begin{aligned} p = \text{Prob}\{H \text{ is perfect for } V\} &\leqq \left(1 - \left(1 - \frac{km/n}{m - m/c - n}\right)^{n}\right)^{n/ck} \\ &\leqq \left(1 - (1 - o(1)) \exp\left(-\frac{km}{m - m/c - n}\right)\right)^{n/ck}.\end{aligned}$$

We choose $c = (k+1)m/(m-n)$ and conclude that

$$p \leqq \bar{p} = (1 - (1 - o(1))\, e^{-(k+1)/(1-n/m)})^{n(1-n/m)/(k+1)k}.$$

Computing $\log_2(1/\bar{p})$ gives roughly $((n/k^2)\, e^{-(k+1)/(1-n/m)})$ bits necessary to specify an oblivious search strategy. We have proved Theorem 1.

THEOREM 1. *The spatial complexity of a k-probe oblivious hash function for a set of n elements belonging to the universe* $[1, m]$ *is* $\Omega((n/k^2)\, e^{-k(m/(m-n))})$ (or $\Omega((n/k^2)\, e^{-k})$ *for* $n = o(m)$).

This lower bound for $|\bar{H}|$, as we shall see, captures the complexity resulting from the size parameter n, but it is quite insensitive to the growth of m. A simple information theoretic argument, however, which was mentioned to us by Fiat and Naor and also

independently by Fredman, shows that $\Omega(\log m/(k \log n))$ is also a lower bound on the size of $|\bar{H}|$. The argument is given here for completeness. We may encode the target sets, under the $|\bar{H}|$ k-probe hash functions, of the elements in U as one function g from U to $\binom{n}{k}^{|\bar{H}|}$. One of the conditions a k-probe scheme satisfies is that no $k+1$ items in U have all k probe locations in common, and thus no $k+1$ items may have the same image under g. It follows that $m \leqq k\binom{n}{k}^{|\bar{H}|}$, and thus

$$|\bar{H}| \geqq \frac{\log (m/k)}{\log \binom{n}{k}},$$

and $\Omega(\log \log (m) - k \log (n))$ bits are needed to name H.

Combining the information theoretic bound with Theorem 1 we get Theorem 2.

THEOREM 2. *The spatial complexity of a k-probe oblivious hash function for a set of n elements belonging to the universe* $[1, m]$ *is* $\Omega(\log \log m + (n/k^2)\, e^{-k(m/(m-n))})$.

By appealing to a hypergraph model and extending the proof technique of Theorem 1, the method of Fredman and Komlós [FK84] can be incorporated to show that $|\bar{H}|$ is actually lower bounded by the product of the two bounds, rather than the maximum of the two, which gives a slightly stronger result.

Theorem 2 also shows that the $\Omega(n)$ portion of the lower bound for the spatial complexity of oblivious hash functions holds even for relatively small universes (such as $m \approx 2n$). On the other hand, for $k > 1$, our lower bound proof collapses if we were to map n elements into a table size $(1+\varepsilon)n$. The upper bound in Observation 5 of § 3 shows that the lower bound indeed does not hold for $\varepsilon > 0$.

3. Upper bounds for oblivious search.

3.1. Oblivious *k*-probe schemes. Our lower bounds suggest that, while k-probe perfect hash schemes must have a reasonably large program length, in the case of oblivious search, the $k-1$ additional search opportunities might reduce the length by a factor of about e^k in its n dependence. We appeal to methods from the theory of random graphs to give a nonconstructive demonstration that this reduction is indeed possible, at least in a formal sense. The proof is actually a probabilistic construction of a k-probe oblivious hash function. For $k = 1$, such a probabilistic construction has been given in [Me82]. In fact, the lower and upper bound of [Me82] were both attained by essentially one argument. When additional probes are permitted, the lower bound argument, as we have seen, is more delicate, while the upper bound, as we now demonstrate, requires much more care.

The procedure is to imagine constructing a random bipartite graph G on $U \times [1, n]$, where each vertex in U has degree k. We then select a random set S of n items from U and estimate a lower bound for the probability p that G contains a perfect matching on S. With positive probability, a family $\bar{H}$ of $\log_{1/(1-p)} \binom{m}{n}$ such random graphs will, for each n-element subset in U, contain some graph that has a perfect matching for that set, and it therefore follows that such an $\bar{H}$ exists. The number of bits required to designate an element $H \in \bar{H}$ is then $\log \log_{1/(1-p)} \binom{m}{n}$.

Accordingly, let $S \subset U$, $|S| = n$ be the subset we wish to match. For convenience, let G be used to name the portion of our random graph that is restricted to $S \times [1, n]$. A few preliminary remarks and definitions will help simplify the subsequent exposition.

- Recall Hall's (a.k.a., the marriage) theorem: A matching exists if and only if for all j, the image of every set of j vertices in S contains at least j vertices (in $[1, n]$). Let Hall (j) be the property that the image of every set of j vertices in S contains at least j vertices.

Our aim is to lower bound the probability of the event (Hall $(k) \wedge \cdots \wedge$ Hall (n)). Our estimates for Hall (j) are based on *expected* number of sets of j vertices that

violate the Hall (j) property and hence are connected to fewer than j vertices:

$$\text{Hall}\,(j) \geqq 1 - \binom{n}{j}\binom{n}{j-1}\left(\frac{\binom{j-1}{k}}{\binom{n}{k}}\right)^{j}.$$

Unfortunately, the above estimates become useless for values of j close to n. For those cases we will estimate Hall (j), by running a matching algorithm on G and estimating the probability that the algorithm fails in a fairly late stage.

- Let, for each $s \in S$, the first three random edges that are chosen to emanate from s be gold. The remaining $k-3$ edges will be green.

Our analysis uses the following simple (not most efficient) matching algorithm, which is based on n successive breadth first searches from unmatched vertices. For each $s \in S$, a new breadth first search is initiated to find an augmenting path from s. A bfs level explores nonmatching edges from a current vertex set $R \subset S$, which is initially $\{s\}$. The search is successful for s if it discovers a vertex in $[1, n]$ that has not been visited by any of the previous searches, and is therefore unmatched. Otherwise, the currently matched mates (in S) of the $[1, n]$ vertices that are newly encountered in the current level of the search (i.e., that have not been previously seen by the current search) are used to replace R for the next level of exploration. Once a previously unmatched vertex $v \in [1, n]$ is visited, the alternating property of being matched or not, along the path of edges from s to v reversed. Then a new s is selected and a new search stage is initiated.

Our use of the algorithm has two constructive phases. The first phase pursues a partial matching on the subgraph of gold edges, until a set X of $n/4$ vertices of S have been matched, or a subset of $j \leqq n/4$ elements have been encountered that violates Hall (j). In the second phase, the count is extended, if possible, to n via both gold and green edges. If a match does not occur by the time $(k-4)n$ green edges have been looked at (in phase 2), the algorithm aborts and fails. The postamble can be entered upon failure encountered during phases 1 or 2, or upon a successful matching (completed in phase 2). If success or failure occurs before $(k-4)n$ green edges are explored, we may imagine that the algorithm continues to select and report new green edges until the requisite number of edges is used.

- Let A be the event that the Hall (j) property holds for $j \leqq n/4$ for the subgraph restricted to the gold edges.
- Let B be the event that the Hall (j) property holds for $n/4 < j \leqq n - n/(k-3)$, where both gold and green edges are used.
- Our matching algorithm will be uncovering the random edges of G as the algorithm progresses. Let the set X comprise the first $n/4$ vertices of $[1, n]$ that are matched (initially by gold edges).
- Let C be the event that the *first* $(k-4)n$ green edges encountered by the algorithm cover $[1, n] - X$. We could imagine an adversary taking note of matched vertices, which will be announced whenever the partial matching is augmented. It will also be told of every green edge that is encountered by the matching algorithm.

Our algorithm can switch to the fail state in phase 1 only if A does not hold. In phase 2, failure can occur for any of four reasons:

(1) Hall (j) does not hold for $j \leqq n/4$ for the full (unrestricted) graph, in which case A does not hold.

(2) B does not hold.

(3) Hall (j) does not hold, for $j > n - n/(k-3)$, in which case the algorithm must have uncovered at least $j(k-3) \geqq n(k-4)$ green edges and hence C does not hold.

(4) No matching occurs before $n(k-4)$ green edges are used, which is to say that C does not hold.

These observations prove Lemma 3.

LEMMA 3. $\Pr\{\text{matching}\} \geqq \Pr\{A \cap B \cap C\}$.

Consequently, $\Pr\{\text{matching}\} \geqq \Pr\{B \cap C \mid A\} \Pr\{A\} \geqq (\Pr\{B \mid A\} + \Pr\{C \mid A\} - 1) \times \Pr\{A\}$.

We now estimate these probabilities. The edges, for each vertex, are selected without replacement (i.e., each k-tuple of edges comprises k distinct members). It is also helpful to denote $[1, n] - X$ by $\{y_1, y_2, \cdots, y_{3n/4}\}$. There follows

$$\Pr\{A\} \geqq 1 - \sum_{3 < j \leqq n/4} \binom{n}{j}\binom{n}{j-1}\left(\frac{j-1}{n}\frac{j-2}{n-1}\frac{j-3}{n-2}\right)^j$$

$$\geqq 1 - \sum_{3 < j \leqq n/4} \binom{n}{j}\binom{n}{j-1}\left(\frac{j-1}{n}\right)^{3j} = 1 - O(n^{-4}).$$

$$\Pr\{B \mid A\} \geqq 1 - \sum_{n/4 < j \leqq n(k-4)/(k-3)} \binom{n}{j}\binom{n}{j-1}\left(\frac{j-1}{n}\right)^{(k-3)j}$$

$$> 1 - o((4\text{-}5)^{3n/4}), \qquad k > 6.$$

Note that $\Pr\{C \mid A\} = \Pr\{C\}$. Let $hit(y)$ be the event that y is hit by one of the first $(k-4)n$ green edges examined.

$$\Pr\{C\} = \Pr\{hit(y_1)\} \times \Pr\{hit(y_2) \mid hit(y_1)\} \times \cdots \Pr\{hit(y_{3n/4}) \mid hit(y_1), \cdots, hit(y_{3n/4-1})\}$$

$$\geqq \left(1 - \left(\frac{n-1}{n}\right)^{n(k-5)}\right)^{3n/4} > (1 - e^{-k+5})^{3n/4}.$$

Since $1 - \Pr\{B \mid A\} = o(\Pr\{C \mid A\})$, for $k > 6$, and $\Pr\{A\} = 1 - o(1)$, the probability p of a matching is $\geqq (1 - e^{-k+5})^{3n/4}(1 - o(1))$.

As we have already observed, a family of $|\bar{H}| = \log_{1/(1-p)} \binom{m}{n}$ such random graphs can supply a hash function (graph) for all n-element subsets. Computing $\log_2(|\bar{H}|) \approx \log_2 \log_2 m + \frac{3}{4} e^{5-k} n$ gives an upper bound on the number of bits necessary to designate which random graph gives a k-probe perfect hash function for a given set S. If time and workspace are of no significance, we can follow the theme given for 1-probe hashing in [Me82], which is to use the lexicographically smallest collection $\bar{H}$ among all collections of such 2^b bipartite graphs that supports k-probe hashing for all n-item subsets of $[0, m-1]$. Then the b-bit number H names a graph within that collection (with respect to the lexicographic order within $\bar{H}$). The desired datum is found in at most k probes, as specified by the graph's edges.

We have shown Theorem 4.

THEOREM 4. *A k-probe hash function for a set of n elements belonging to the universe $[1, m]$ into a table of size n can be specified in $\log\log m + O(n e^{-k})$ bits, which is tight to within a factor of k^2 of the lower bound (or tight to within a constant factor for constant k).*

Finally, we remark that a straightforward application of Hall's theorem shows that if we were to map the elements of S into a table of size $(1+\varepsilon)n$, then the probability of a successful matching using k probes is $1 - o(n^{-(k+1)})$, for $k > \max\{3, 2(\ln(1+\varepsilon))^{-1}\}$. Observation 5 follows.

OBSERVATION 5. *A k-probe hash function for a set of n elements belonging to the*

universe $[1, m]$ *into a table of size* $(1+\varepsilon)n$, *with* $k > \max\{3, 2(\ln(1+\varepsilon))^{-1}\}$, *can be specified in* $O(\log\log m + \log n)$ *bits.*

4. Explicit construction of an optimal $O(1)$-time 1-probe scheme. The upper bound in the previous section uses a probabilistic construction to establish the spatial complexity of k-probe oblivious hash functions. Unfortunately, such a technique does not yield any explicit $O(1)$-time hash function. In this section we give a construction for an $O(1)$-time 1-probe scheme that uses auxiliary tables of only $O(n)$-bits and is therefore optimal in both time and space. The scheme is a variation of the perfect hashing scheme originating in [FKS84]; other variations appear in [SvE84] and [JvE86]. Interestingly, the construction has commonality with the $O(n)$ time variation presented in [SvE84].

The hashing technique given in [FKS84] uses a table of linear congruential functions of the form $(ax \bmod p) \bmod q$. The basic idea, which we detail below, is to use such a hash function to define an implicit partition of the data set into a favorably distributed collection of collision buckets. Given a bucket index, an explicit (locally defined) secondary hash function is looked up, which determines a unique address (within the bucket) for the item sought.

Altogether, the address evaluation uses a few $\log n$-bit arithmetic computations, several array accesses, and one long word computation, which consists of the modular reduction of a $\log m$ bit word and the subsequent (modular) multiplication of two $\log n + \log\log m$ bit words, and has the form $(kx \bmod p) \bmod n^2$, where $k, p \in U$ and $k < p < n^2 \log m$ and p is prime. It is natural to ask if the [FKS84] scheme or some other reasonable hash scheme can be expressed in an optimal $O(\log\log m + n)$ bits, while maintaining $O(1)$ time search, as conjectured in [SvE84].

We show that an [FKS84]-based scheme can be expressed in an optimal number of bits (up to a multiplicative factor) and can be performed with essentially the same selection of $O(1)$ arithmetic operations and array accesses.

We use the usual machine model associated with this problem, which is somewhat idealized. The model is basically that of a random access machine, although as in all other schemes which preceded this one, the standard RAM model is augmented with multiplicative arithmetic. In particular, an array access of an $O(\log n)$-bit word takes unit time, and index computations are permitted. Words can be added, subtracted, multiplied, and (integer) divided in constant time. We also use the same single long word computation as in [FKS84] and [Me82] to map elements from U into $[0, n^2]$. Similarly, we employ the expositional expediency of calling this computation an $O(1)$ time operation.

The original FKS construction has four basic steps.

(1) A (long word) function $h_\alpha(x)$ is found that maps S into $[0, n^2-1]$ without collisions, so that all subsequent computations can use normal length words. Corollary 2 and Lemma 2 in [FKS84] show that it is possible to set $h_\alpha(x) = (kx \bmod p) \bmod n^2$, with suitable $k < p < n^2 \log m$, where p is prime. Let Σ be the image of $h_\alpha \circ S$.

(2) Next, a function $h_\beta(x)$ is found that maps Σ into $[0, n-1]$ so that the sum of the squares of the collision sizes is not too large. Corollary 3 in [FKS84] shows that it is possible to set $h_\beta(x) = (\kappa x \bmod \rho) \bmod n$, where ρ is any prime greater than n^2 and $\kappa \in [0, \rho]$ so that $\sum_{0 \le j < n} |h_\beta^{-1}(j) \cap \Sigma|^2 < 3n$.

(3) For each hash bucket (i.e., integer having at least one appearance in the multiset $\{h_\beta(t)\}_{t \in \Sigma}$) a secondary hash function h_i is found that is one-to-one on the collision set. Let c_i be the size of the collision set for bucket i; $c_i = |h_\beta^{-1}(i) \cap \Sigma|$. Then for $x \in h_\beta^{-1}(i) \cap \Sigma$, $h_i(x) = (k_i x \bmod \rho) \bmod c_i^2$, where $k_i \in [0, \rho-1]$. The item $s \in S$, (which is represented by $t = h_\alpha(s)$ and located in hash bucket $i = h_\beta(t)$) is stored in

location $C_i + h_i(h_\alpha(s))$, where $C_i = c_0^2 + c_1^2 + \cdots + c_{i-1}^2$. This locates all n items within a size $3n$ table, say $A^*[1 \cdots 3n]$.

(4) Finally, the table is stored without vacant locations in an array $A[1 \cdots n]$, i.e., the array contains the items of S in the order of appearance in the table A^*.

The composite hash function requires the parameters k, p, κ, and ρ for h_α and h_β, a table $K[0 \cdots n]$ storing the parameters k_i for the secondary functions h_i, a table $C[0 \cdots n]$ listing the locations C_i, which separate elements from different buckets in A^*, and finally a compression table $D[1 \cdots 3n]$, where $D[j]$ gives the index, within A, of the item (if any) that hashes to the value j in A^* (by the function outlined in 1 through 3 above). As presented, the description of this perfect hash function needs $2 \log \log m + O(n \log n)$ bits. Fredman, Komlós, and Szemerédi [FKS84] then apply this same formulation for a rescaled number range (and a 2-level pair of tables for D) to achieve a hash function that is compressed to $2 \log \log m + O(n\sqrt{\log n})$bits.

Elementary information theoretic calculations can show that rescaling cannot be used to give a function size that matches the lower bound. Accordingly, we use the basic [FKS84] formulation given above. In choosing the secondary hash functions h_i for each bucket, we appeal to [FKS84]. They point out that since their existence proofs are based on expected case analysis, at least half of the numbers in $[0, \rho - 1]$ will yield appropriate functions h_i if the hash range is doubled. In particular, their Corollary 4 shows that given a collision set of any c_i items in Σ, at least half of the numbers in $[0, \rho - 1]$ will, if selected as multiplier k_i, yield a function $h_i(x) = (k_i x \bmod \rho) \bmod 2c_i^2$ that is one-to-one on the set. Consequently, if we have z collision buckets requiring multipliers, we may select a single multiplier that is one-to-one for $\lceil z/2 \rceil$ of the buckets, provided we double the size of the hash range in each case. Altogether, we need at most $\lfloor \log n \rfloor + 1$ different k_i values, where k_i is the ith multiplier servicing about $1/2^i$ of the buckets. In summary, the optimal perfect hash function is that described above, with two modifications:

(1) The (uncompressed) hash value is $h_i(x) = ((k_i x \bmod \rho) \bmod 2c_i^2) + 2C_i$.

(2) The multiplier values k are iteratively selected to satisfy the one-to-one requirement of the maximum number of (unsatisfied) h_i's.

Up to this point our method turns out to be essentially the same as the $O(n)$ time version presented in [SvE84].

Now the information content of the map from bucket indices into the $\lfloor \log n \rfloor + 1$ multipliers (i.e., the representation of the table K with Huffman coding) is $O(n)$. We must show how to achieve compact encodings of the tables K, C, and D that require only $O(n)$ bits, and readily are decodable in $O(1)$ time, in our computational model. The basic decoding operations we use are as follows:

(1) Extract a subsequence of bits from one word.

(2) Concatenate two bit strings of altogether $O(\log n)$ bits.

(3) Compute the bit-index of the kth zero in a word.

(4) Count the number of consecutive 1's in an $O(\log n)$-bit word, starting at the beginning.

(5) Count the number of zeros in an $O(\log n)$-bit word.

(6) Access a few constants.

The first two operations are a matter of arithmetic. The last four can be performed with small lookup tables. In particular, it suffices to break the words into words of $\log n/2$ bits (padded with leading zeros) and to use these words to access decoder arrays. The third operation, for example, requires for each k, $0 < k \leqq \log n/2$, an array *of* $\sqrt{n}$ words. Using operations 5 and 3 above, a word of $2 \log n$ (for example) bits requires up to four accesses to compute the location of a kth 0. The fourth and fifth operation are accomplished in a similar matter.

The table C, which contains the values $C_i(=\sum_{j=o}^{i-1} c_i^2)$, is encoded as follows. First, the values c_i^2 are stored in a table T_0 in unary notation (in order of appearance, separated by 0's). (Note that the bit length of T_0 is at most $4n$.) We let str_i denote the string that encodes c_i^2. Because of operations 1 and 2, we may suppose that each bit of T_0 is addressable. Let a_i be the address (index) of the starting point of $str_{i\lceil \log(4n)\rceil}$ in T_0, so that $0 \leqq a_i < 4n$, for $i = 1, \cdots, n/\lceil \log(4n)\rceil$. (Note that $a_i = C_{i\lceil \log(4n)\rceil} + i\lceil \log(4n)\rceil$.) A second table T_1, (of n bits), contains, in binary, the indices $a_1, \cdots, a_{n/\lceil \log(4n)\rceil}$, stored as $\lceil \log(4n)\rceil$ bit words. If $a_{i+1} - a_i < 2\lceil \log(4n)\rceil$, the table information for intermediate c_j^2 (i.e., for $i\lceil \log(4n)\rceil \leqq j < (i+1)\lceil \log(4n)\rceil$) can be readily decoded via $O(1)$ accesses to T_0 and T_1 and $O(1)$ of our decoding operations ((3) and (5)).

The case $a_{i+1} - a_i \geqq (\lceil \log(4n)\rceil)^2$ is handled via a third table T_2, (of size $4n$) which stores, starting in (bit) location a_i, $\lceil \log(4n)\rceil - 1$ binary indices for the starting locations of str_j, $i\lceil \log(4n)\rceil < j < (i+1)\lceil \log(4n)\rceil$ (in T_0).

The remaining case, $2\lceil \log(4n)\rceil \leqq a_{i+1} - a_i < (\lceil \log(4n)\rceil)^2$, is handled with an additional level of refinement. Let $b_{i,j}$ be the address (index in T_0) of the starting point of $str_{i\lceil \log(4n)\rceil + j\lceil \log\log(4n)\rceil}$, $(j < \lceil \log(4n)\rceil / \lceil \log\log(4n)\rceil$ and $a_i < b_{i,j} < a_{i+1})$. In T_2 we store, starting in location a_i, the binary offsets $b_{i,1} - a_i$, $b_{i,2} - a_i, \cdots$. Each offset is stored as a $2\lceil \log\log(4n)\rceil$-bit binary number. If $b_{i,j+1} - b_{i,j} \leqq 2 \log n$ the information for intermediate c_ℓ, $i\lceil \log(4n)\rceil + j\lceil \log\log(4n)\rceil < \ell < i\lceil \log(4n)\rceil + j\lceil \log\log(4n)\rceil$, is readily decoded (from table T_0); for all other cases, the offsets (of size $2\lceil \log\log(4n)\rceil$) of all intermediate c_ℓ's are encoded through one last level of indirection in a table T_3, starting at bit location $b_{i,j}$. (This last encoding requires $\lceil \log\log(4n)\rceil^2 \leqq \lceil \log n\rceil$.)

The table K can be encoded in exactly the same way. In particular, a table K_0 contains, for its ith sequence of bits, the integer α_i in unary, if k_{α_i} is the multiplier assigned to hash bucket i, $0 \leqq i < n$. (Recall that the first multiplier (encoded by the string "1") is usable for at least half of the hash buckets.) The total length of the sequence comprises at most n 0's, $n/2$ singleton 1's, $n/4$ doubleton 1's, etc., for a total length of $3n$. The multipliers $k_1, \cdots, k_{\log n}$ are stored in a $\log n$-word array.

It is evident that we have encoded a perfect hash function in $O(n)$ bits. To summarize, the above construction, combined with the lower bound of [Me84], gives Theorem 6.

THEOREM 6. *The spatial complexity of a $O(1)$-time perfect (1-probe) hash function for a set of n elements belonging to the universe $[1, m]$ is $\Theta(\log\log m + n)$.*

5. Conclusions and open problems. We have shown how to construct explicit space-time optimal perfect hash functions. It is worth noting that the construction illustrates the computational significance of a Random Access Machine model augmented with a size n^ε auxiliary program store.

In addition, we have given tight bounds on the spatial complexity of oblivious k-probe hash functions and have helped quantify the difference between oblivious and nonoblivious strategies. In particular, we have shown that, for full tables, $O(1)$ additional oblivious probes can reduce the requisite size of the search program by a constant factor, but no more. As is well known, the decrease in program size, for partially full search tables, is more dramatic: probabilistic arguments show the formal existence of oblivious $O(1)$ probe schemes that need only $O(\log\log m + \log n)$ bits.

It would be interesting to find an explicit construction of a small $O(1)$-probe oblivious family of hash functions that map n elements into a table of size $(1+\varepsilon)n$,[1] and to find such a family where function evaluation can be accomplished in $O(1)$ time.

[1] On recent results on the above problem, see [SS90] and [S89].

Acknowledgments. The authors thank A. Fiat, M. Fredman, M. Naor, J. Spencer, and P. van Emde Boas for helpful discussions.

REFERENCES

[A85] M. AJTAI, *A lower bound for finding predecessors in Yao's cell probe model*, IBM Tech. Report RJ 4867 (51297) Computer Science, October 3, 1985.

[AFK83] M. AJTAI, M. FREDMAN, AND J. KOMLÓS, *Hash functions for priority queues*, in Proc. 24th Annual Symposium on Foundations of Computer Science, Tucson, AZ, 1983, pp. 299-303.

[BBDOP86] F. BERMAN, M. E. BOCK, E. DITTERT, M. J. O'DONNELL, AND D. PLANK, *Collections of functions for perfect hashing*, SIAM J. Comput., 15 (1986), pp. 604-618.

[FMNSSS88] A. FIAT, I. MUNRO, M. NAOR, A. SCHÄFFER, J. P. SCHMIDT, AND A. SIEGEL, *An implicit data structure for searching a multikey table in logarithmic time*, J. Comput. System Sci., to appear.

[FN88] A. FIAT AND M. NAOR, *Implicit $O(1)$ probe search*, in Proc. 21st ACM Symposium on Theory of Computing, Seattle, WA, 1989, pp. 336-344.

[FNSS88] A. FIAT, M. NAOR, J. P. SCHMIDT, AND A. SIEGEL, *Non-oblivious hashing*, in Proc. 20th ACM Symposium on Theory of Computing, Chicago, IL, 1988, pp. 367-376.

[FK84] M. L. FREDMAN AND J. KOMLÓS, *On the size of separating systems and families of perfect hash functions*, SIAM J. Algebraic Discrete Methods, 5 (1984), pp. 61-68.

[FKS84] M. L. FREDMAN, J. KOMLÓS, AND E. SZEMERÉDI, *Storing a sparse table with $O(1)$ worst case access time*, J. Assoc. Comput. Mach., 31 (1984), pp. 538-544.

[G81] G. H. GONNET, *Expected length of the longest probe sequence in hash code searching*, J. Assoc. Comput. Mach., 28 (1981), pp. 289-304.

[GM79] G. H. GONNET AND J. I. MUNRO, *Efficient ordering of hash tables*, SIAM J. Comput., 8 (1979), pp. 463-478.

[HK73] J. E. HOPCROFT AND R. M. KARP, *An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs*, SIAM J. Comput., 2 (1973), pp. 225-231.

[JvE86] C. T. M. JACOBS AND P. VAN EMDE BOAS, *Two results on tables*, Inform. Process. Lett., 22 (1986), pp. 43-48.

[Ma83] H. G. MAIRSON, *The program complexity of searching a table*, in Proc. 24th Annual Symposium on Foundations of Computer Science, Tucson, AZ, November 1983, pp. 40-47.

[Ma84] ———, *The program complexity of searching a table*, Ph.D. thesis, Department of Computer Science, Stanford University, Stanford, CA, 1984, STAN-CS83-988.

[Me82] K. MEHLHORN, *On the program size of perfect and universal hash functions*, in Proc. 23rd Annual Symposium on Foundations of Computer Science, Chicago, IL, 1982, pp. 170-175.

[Me84] ———, *Data Structures and Algorithms* 1: *Sorting and Searching*, Springer-Verlag, Berlin, Heidelberg, 1984.

[S89] A. SIEGEL, *On universal classes of fast hashfunctions, their time-space tradeoff, and their applications*, in Proc. 30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, NC, October 1989, pp. 20-25.

[SS80] J. P. SCHMIDT AND E. SHAMIR, *An improved program for constructing open hash tables*, in Proc. Automata Languages and Programming, Noordwijkerhout, Holland, July 1980 (ICALP80), pp. 569-581.

[SS89] J. P. SCHMIDT AND A. SIEGEL, *On aspects of universality and performance for closed hashing*, in Proc. 21st Annual Symposium on Theory of Computing, Seattle, WA, May 1989, pp. 355-366.

[SS90] ———, *The analysis of closed hashing under limited randomness*, in Proc. 22nd Annual Symposium on Theory of Computing, Baltimore, MD, May 1990, to appear.

[SvE84] C. SLOT AND P. VAN EMDE BOAS, *On tape versus core: An application of space efficient hash functions to the invariance of space*, in Proc. 16th Annual Symposium on Theory of Computing, Washington, DC, May 1984, pp. 391-400.

[TY79] R. E. TARJAN AND A. C. YAO, *Storing a sparse table*, Commun. Assoc. Comput. Mach., 22 (1978), pp. 606-611.

[Y81] A. C. YAO, *Should tables be sorted?*, J. Assoc. Comput. Mach., 28 (1981), pp. 615-628.