Alcom-FT Technical Report Series
ALCOMFT-TR-01-67

On the Cell Probe Complexity of Membership and Perfect Hashing*

Rasmus Pagh
BRICS†, Department of Computer Science
Ny Munkegade Bldg. 540, 8000 Aarhus C
University of Aarhus, Denmark
pagh@brics.dk

ABSTRACT

We study two fundamental static data structure problems, membership and perfect hashing, in Yao's cell probe model. The first space and bit probe optimal worst case upper bound is given for the membership problem. We also give a new efficient membership scheme where the query algorithm makes just one adaptive choice, and probes a total of three words. A lower bound shows that two word probes generally do not suffice. For minimal perfect hashing we show a tight bit probe lower bound, and give a simple scheme achieving this performance, making just one adaptive choice. Linear range perfect hashing is shown to be implementable with the same number of bit probes, of which just *one* is adaptive. In contrast, we establish that for sufficiently sparse sets, non-adaptive perfect hashing needs exponentially more bit probes. This is the first such separation of adaptivity and non-adaptivity.

1. INTRODUCTION

This paper considers two fundamental static data structure problems, MEMBERSHIP and PERFECT HASHING, in Yao's cell probe model [20]. In this model, a data structure is a numbered sequence of s "cells", each containing an element of $\{0,1\}^b$, for a positive integer parameter b. The complexity of a query is the number of cells that a deterministic algorithm needs to probe to answer it. (In the worst case over all possible data, for the optimal choice of data structure, in the worst case over all queries.) For positive integer x define $[x] = \{1, 2, \ldots, x\}$. We state the problems as follows.

*Partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).
Part of this work was done while the author was visiting Stanford University.

†Basic Research in Computer Science (www.brics.dk), funded by the Danish National Research Foundation.

STOC'01, July 6-8, 2001, Hersonissos, Crete, Greece.

MEMBERSHIP(u, n, s, b): Given a set $S \subseteq [u]$, $|S| = n \leq u/2$, use a data structure with s cells of b bits to accommodate membership queries: "$x \in S$?"

PERFECT HASHING(u, n, r, s, b): Given a set $S \subseteq [u]$, $|S| = n \leq r \leq u/2$, store a perfect hash function for S, i.e., some function $f_S : [u] \to [r]$ that is 1-1 on S, in a data structure with s cells of b bits, and accommodate function value queries: "$f_S(x) =$?"

Informally, a membership query determines *whether* an element is present in a certain subset of the universe, and a perfect hashing query can provide a pointer to *where* associated information is located. Frequently, one considers the "combined" problem, that is, to design a data structure that on query x answers whether $x \in S$, and if so, returns the value of a perfect hash function pointing to some "satellite information" unique to x. This problem will be referred to as DICTIONARY(u, n, r, s, b).

We are particularly interested in the cases $b = 1$, where cell probes are referred to as *bit probes*, and $b = \log u$, where cells are referred to as *words*. (For simplicity we assume that u is a power of 2.) We pay special attention to PERFECT HASHING in the case $r = n$, referred to as "minimal perfect hashing".

Very efficient data structures exist for most instances of the two problems. Our interest lies in an exact understanding of how efficient the query algorithms can be. The cell probe model ignores the cost of computation, but as random memory accesses in real hardware are becoming orders of magnitude slower than computational instructions, it focuses on a main practical bottleneck. The precise space usage is of secondary concern, though our data structures always use within a constant factor of minimum space.

1.1 Background

Membership

The minimum number of bits with which it is possible encode a MEMBERSHIP data structure is $s_{\mathrm{m}}(u, n) = \log \binom{u}{n} = \Theta(n \log(u/n))$. (When the parameters are understood we will simply write s_{m}.) We do not consider data structures of size less than one cell, so a space optimal data structure is one that occupies $O(s_{\mathrm{m}}/b + 1)$ cells.

The *average case* bit probe complexity of MEMBERSHIP was studied by Minsky and Papert in the book *Perceptrons* [13, Sect. 12.6], where it is shown that a constant number of bit probes suffice on average over all queries. The study of *randomized* (Monte Carlo) bit probe complexity, where the query algorithm makes coin flips and is allowed some error probability, was initiated recently by Buhrman et al. [3]. Notably, one can get two-sided error probability ϵ using *one bit probe* and $O(\frac{n}{\epsilon^2}\log u)$ bits of space (which is $O(s_{\mathrm{m}})$ for constant $\epsilon > 0$ and $u = n^{1+\Omega(1)}$). The scheme is non-explicit, that is, it is not shown that there are efficient (polynomial time) procedures for constructing the data structure and carrying out queries.

Buhrman et al. were also the first to study directly the worst case bit probe complexity of MEMBERSHIP. They showed a lower bound of $\Omega(\log(u/n))$ bit probes for space $O(s_{\mathrm{m}})$, matching the $O(\log u)$ upper bound of Fredman et al. [7] for $u = n^{1+\Omega(1)}$. They also considered the *adaptivity* of the query algorithm, that is, the way in which probe locations depend on the results of previous probes. Surprisingly, adaptivity does not help in general, as space $O(n \log u)$ and $O(\log u)$ bit probes can be achieved by a *non-adaptive* scheme, i.e., where all probe locations are determined by the query element. Again, this scheme is not explicit.

Motivated by applications of storing a set of machine words on real world computers, upper bounds on the RAM (and hence the word probe) complexity of MEMBERSHIP are of considerable interest. Two seminal papers dealing with this subject are among the most cited in the data structures literature: Carter and Wegman [4] gave a randomized data structure of $O(n)$ words, that uses $O(1)$ expected word probes per operation, even permitting dynamic changes to the set stored. Fredman, Komlós and Szemerédi [7] showed that $O(1)$ word probes suffice in the worst case for data structures of $O(n)$ words. If u is sufficiently larger than n, three word probes suffice, the first one to a fixed location. The space usage has been lowered to $s_{\mathrm{m}} + o(s_{\mathrm{m}})$ bits [2, 14] at the cost of a large constant number of word probes.

Perfect hashing

A space optimal data structure for PERFECT HASHING uses $s_{\mathrm{ph}}(u, n, r) = \Theta(n^2/r + \log n + \log\log_r u)$ bits. (When the parameters are understood we will simply write s_{ph}.) Apart from the trivial $\log n$ lower bound, a proof of this can be found in e.g. [12, III.2.3]. In this paper we will be concerned mainly with the case $r = O(n)$, for which $s_{\mathrm{ph}} = \Theta(n + \log\log u)$. It is optimal to use $O(s_{\mathrm{ph}}/b + 1)$ cells of memory.

The data structure of Fredman et al. [7] is in fact a hash table along with a solution to PERFECT HASHING$(u, n, O(n), O(n), \log u)$ requiring (for u large enough) two cell probes, one of which is to a fixed location. An efficient scheme for PERFECT HASHING$(u, n, n, O(n), O(\log n + \log\log u))$ (i.e., minimal perfect hashing) using two cell probes was presented in [15], where it was also shown that one cell probe schemes are impossible unless $b = \Omega(\frac{n}{1+n^2/u})$. Schmidt and Siegel [17] presented a coding scheme for essentially the perfect hash function in [7], using space $O(s_{\mathrm{ph}})$ for $r = n$. The construction can be extended to give space $O(s_{\mathrm{ph}})$ for $r \leq n^2/\log^2 n$. A space bound of $s_{\mathrm{ph}} + o(s_{\mathrm{ph}})$, still with constant evaluation time, was obtained for minimal perfect hashing by Hagerup and Tholey in [9]. The number of cell probes, as well as the constant hidden in the order notation, is quite high for these space optimal schemes, so they are mainly of theoretical interest.

The bit probe complexity of perfect hashing does not seem to have been studied directly before. An upper bound of $O(\log n + \log\log u)$ bit probes was given by the Schmidt-Siegel construction [17].

In a relaxation of perfect hashing considered by Schmidt and Siegel [17], an "oblivious k-probe hash function" computes a *set* of k values, one of which (for appropriate arrangement of the set in a table) is the location of the input if it is in the set. It was shown that $n/2^{O(k)}$ bits are needed to represent such a function, in the case $r = n$. However, it is mentioned that there is a probabilistic argument showing $O(\log n + \log\log m)$ bits to suffice when $r = (1 + \Omega(1))\, n$, for some constant k. Such probabilistic arguments appear explicitly in [1, 5], in the context of "balanced allocations". In our context, a result of [5] implies a probabilistic construction for $k = 4$ in which the hash table has size $2n$.

1.2 This paper

This paper contains a collection of results pertaining to the cell probe complexity of membership and perfect hashing, as summarized in the remainder of this section.

Membership

We prove in Section 2.1 that the bit probe lower bound of Buhrman et al. [3] is tight, by giving an explicit MEMBERSHIP scheme using $O(\log(u/n))$ bit probes and space $O(s_{\mathrm{m}})$. The construction makes use of a bounded concentrator, which is a weak expander graph.

As for word probe complexity, we show in Section 2.2 that if one wants to use $O(n)$ words of space, and if u is not too small in terms of n, schemes probing two words essentially require a perfect hash function with linear range to be storable in the first word probed. In particular, such schemes are only possible when $u = 2^{\Omega(n)}$. Thus, there is no hope of improving the word probe count of three achieved by the scheme of Fredman et al. [7].

In Section 2.3 we investigate the previously mentioned comment of Schmidt and Siegel [17] in the case $k = 2$, for which the results in [1, 5] do not say anything. The result is a scheme that is an improvement of the scheme in [7] in that it allows for *parallelism* in the word probes: After the first (fixed) probe, the two last probes can be determined and carried out in parallel. Additionally, our scheme works also in the case where u is not much larger than n, and uses space $O(s_{\mathrm{m}})$. The improved parallelism seems interesting from a practical point of view, as CPU pipelining could potentially decrease the time relative to the case of adaptive probes by a factor of nearly two. In case the data structure resides in external memory, split to multiple disks, any item can be retrieved in one parallel I/O, using minimal internal memory. Our scheme is explicit when $u \geq n^{c \log n}$, for a constant c, in which case the first probe can be used to read a function from a (nearly) $O(\log n)$-wise independent family. By increasing the number of fixed word probes to $O(\log n)$, which can be argued to be practical, for example in the case of external memory, we get an explicit scheme in all cases.

Perfect hashing

We show in Section 3.1 that the bit probe complexity of the Schmidt–Siegel scheme is optimal for PERFECT HASHING(u, $n, r = n, O(s_{\rm ph}), 1$), i.e., the bit probe complexity of minimal perfect hashing, using optimal space, is $\Theta(\log n + \log\log u)$.

In Section 3.2 we give an alternative space optimal scheme, conceptually much simpler than that of Schmidt and Siegel, that can be implemented with just one adaptive cell probe reading $O(\log n)$ bits. The scheme is non-explicit, but can be made explicit using (nearly) $O(\log n)$-wise independence, increasing the number of bits read in fixed cell probes from $O(\log n + \log\log u)$ to $O(\log^2 n + \log\log u)$.

In Section 3.3 we turn the attention to range $r > n$, showing that *one* adaptive bit probe suffices for $r = O(n)$, using minimal space and $O(\log n + \log\log u)$ fixed bit probes. Again, this scheme is non-explicit, but can be made explicit at the cost of increasing the number of fixed bit probes by $O(\log^2 n)$. The explicit scheme is much simpler than previous perfect hash functions using minimal memory. We also show that such a scheme is not possible for $r < (\log e - \Omega(1))\, n$, no matter how much space is used. If one wants range $r = n$, $\Omega(\log\log n)$ adaptive bit probes are required.

Finally, in Section 3.4, we consider query schemes with no adaptivity, and show that for $u = 2^{\Theta(n)}$, a constant fraction of any space minimal $\Theta(n)$-bit data structure must be read to determine the function value. Recall that $O(\log n + \log\log u) = O(\log n)$ bit probes, of which just one is adaptive, is enough. This appears to be the first problem for which an exponential separation between adaptive and non-adaptive query schemes has been shown. Note that no super-exponential separation can exist, as any adaptive t bit probe scheme can be converted to a non-adaptive $2^t - 1$ bit probe scheme. It is interesting to compare this to the fact that MEMBERSHIP has efficient non-adaptive query schemes.

1.3 Notation

For convenience, we introduce a notation for expressing the adaptivity of a query algorithm. An $(a_0, a_1, \ldots, a_k)$*-probe* query scheme is one that:

1. Starts by performing a_0 fixed cell probes, not depending on the input (this is the 0th round).
2. In the ith round, for $i = 1, \ldots, k$, makes a_i probes to cells determined by the input and by the outcomes of probes in previous rounds.

2. MEMBERSHIP

2.1 Tight bit probe bound

In this section we present an explicit MEMBERSHIP scheme that has optimal space and bit probe complexity. It can be thought of as a "generalization" of bit vectors that is space efficient even for sparse sets. For r/n larger than some constant, the data structure also allows PERFECT HASHING queries in $O(\log(u/r))$ bit probes, that is, it solves the DICTIONARY problem.

THEOREM 1. *There is an explicit* DICTIONARY$(u, n, r = O(n), O(s_{\rm m}), 1)$ *scheme with cell probe complexity* $O(\log(u/n))$.

PROOF. Our scheme consists of $O(\log(u/n))$ steps, each reducing the size of the universe considered by a constant factor, or producing an answer to the query. Each step uses a data structure of $O(n)$ bits that specifies a superset of S and a 1-1 mapping from this superset to the smaller universe. A constant number of bit probes suffice to determine if the query element is not in the superset, in which case we can immediately return a valid answer. If the query element is in the superset, its mapping that can be computed by probing $O(1)$ bits. At the end of the recursion we have $u = O(n)$, and a bit vector can be used to determine membership. A perfect hash function value can be determined after $O(\log(u/r))$ levels when the universe has reached size r. (This happens if r/n is a sufficiently large constant.)

The essential ingredient in our solution is a family of *bounded concentrators*, which are constant degree bipartite graphs with v vertices on the left, at most θv vertices on the right, for constant $\theta < 1$, having the property that any set V of up to $v/2$ left hand vertices can be matched to $|V|$ vertices on the right hand side. Simple, explicit constructions of bounded concentrators exist [8]. We denote left hand vertices by $s_1, \ldots, s_v$ and right hand vertices by $t_1, \ldots, t_w$, where $w \leq \theta v$, and assume some fixed ordering of the neighbors of each vertex.

If $u \leq \frac{4n}{1-\theta}$ the recursion is stopped. Otherwise we construct a data structure as follows: Split $[u]$ into $v = 2n$ parts $U_1, \ldots, U_{2n}$ of size at most $\lceil u/2n \rceil$, and consider the set $V = \{s_i \mid S \cap U_i \neq \emptyset\}$ which has size at most $n = v/2$. By definition of a bounded concentrator, the vertices of V can be matched to a set $W \subseteq \{t_1, \ldots, t_{\theta v}\}$. Suppose that $s_i \in V$ is matched to its kth neighbor. Then we write the $O(1)$ bit number k as entry i of a $2n$-element table T. Table entries of vertices not in V are set to a special value.

An element $x \in U_i$ is in the abovementioned superset if and only if $s_i \in V$. In this case we map x to an element in the reduced universe as follows: If s_i is matched to t_j, map x to $h(x) = j\lceil u/2n \rceil - z_x$, where z_x is the number of x in some fixed numbering $0, \ldots, \lceil u/2n \rceil - 1$ of U_i. By the matching property, no other element of $[u]$ maps to $h(x)$. The reduced universe is $[w\lceil u/2n \rceil]$, which has size at most $\frac{1+\theta}{2}u$. $\square$

COROLLARY 2. *The cell probe complexity of* MEMBERSHIP$(u, n, O(s_{\rm m}), 1)$ *is* $\Theta(\log(u/n))$.

It can be noted that a constant fraction of all possible queries are resolved at each step. Thus, $O(1)$ probes suffice on average over all queries, so our scheme is optimal also with respect to average case complexity.

2.2 Two word probes do not suffice

Yao [20] studied MEMBERSHIP in a model where the query algorithm first looks up $b = \log u$ fixed bits, and then, in a table of length s, probes a single word containing some element of S, answering "yes" if it is equal to the input. He observed that for this problem to be solvable there has to be a size u perfect family of hash functions with range $[s]$. Hence, such two-probe schemes are possible only if $u \geq 2^{s_{\mathrm{ph}}(u,n,s)}$. Here, we look at a more general class of schemes, without restriction on the contents of the second word probed, or on how it is interpreted. For the case where u is not too close to n or s, we show that again, such a scheme is not possible unless $u = 2^{\Omega(s_{\mathrm{ph}}(u,n,s))}$. The lower bound carries over to the case where the first word probe is not necessarily fixed, meaning that the MEMBERSHIP scheme of Fredman et al. [7], that probes three words, is in general word probe optimal among schemes using $O(n)$ words.

THEOREM 3. *For constants $\epsilon > 0$ and $k \in \mathbf{N}$, if $u > s^{1+\epsilon}2^{\epsilon b}$, $s \geq n$, and* MEMBERSHIP(u, n, s, b) *has a $(k, 1)$-probe query scheme, then $b = \Omega(s_{\mathrm{ph}}(u, n, s))$.*

PROOF. We employ a "volume bound" similar to the lower bound for the program size of perfect hash functions in [12]. More specifically, we bound the number of sets that be accommodated for each of the 2^{kb} possible bit patterns read in the k fixed probes. This turns out to be a very small fraction of all n-subsets, giving a lower bound on kb.

Without loss of generality we assume that $s \geq 2n$. Fix a kb-bit pattern, and let $S_1, \ldots, S_l \subseteq [u]$ be the n-subsets for which this bit pattern is used. For each $i \in [s]$, let $U_i \subseteq [u]$ be the set of elements for which the query algorithm uses the adaptive probe to probe cell i. Since a cell can contain 2^b different bit patterns, the set $\{U_i \cap S_j \mid j \in [l]\}$ can have size at most 2^b. We will use this fact to get an upper bound on l.

For a positive integer m to be determined later, let F be the family of functions $f : [s] \to \{0, \ldots, m\}$ for which $\sigma_f \stackrel{\text{def}}{=} \sum_{i=1}^{s} f(i) \leq n$. It is easily shown by induction on m that there are at most $\binom{s}{n} n^m$ functions in F. For any S_j, $j \in [l]$, there is a function in F for which $f(i) = |S_j \cap U_i|$ if $|S_j \cap U_i| \leq m$ and $f(i) = 0$ otherwise. We bound the number of sets by summing over all functions in F, and the number a of indices $i \in [s]$ for which $|S_j \cap U_i| > m$.

$$
\begin{aligned}
l &< \sum_{0 \leq a \leq \frac{n}{m}} \sum_{\substack{f \in F \\ \sigma_f \leq n - ma}} \binom{s}{a} 2^{ba} \prod_{i=1}^{s} |U_i|^{f(i)} \\
&< \sum_{0 \leq a \leq \frac{n}{m}} s^a \binom{s}{n} n^m \, 2^{ba} \left(\frac{u}{s}\right)^{n-ma} \\
&< n^m \binom{s}{n} \left(\frac{u}{s}\right)^n \sum_{a \geq 0} \left(\frac{s^{m+1} 2^b}{u^m}\right)^a .
\end{aligned}
$$

The second inequality uses convexity, i.e., that the sum is maximized when the $|U_i|$ are equal. For suitable $m = O(1/\epsilon)$ the last sum is bounded by 2. In conclusion, we must have $kb > \log \binom{u}{n} - \log \left(2n^m \binom{s}{n} \left(\frac{u}{s}\right)^n\right) = \Omega(n^2/s - \log n)$, where the last bound is derived as in [12]. By the bit probe lower bound of [3], we must have $kb = \Omega(\log u)$. Together, these bounds give the desired bound on b. □

COROLLARY 4. *For any constant $\delta > 0$, if $u > s^{2+\delta}$, $s \geq n$, and* MEMBERSHIP$(u, n, s, \log u)$ *has a $(0, 1, 1)$-probe query scheme, then $u = 2^{\Omega(s_{\mathrm{ph}}(u,n,s))}$.*

PROOF. For some set of queries $U \subset [u]$ of size at least $u/2s > s^{1+\delta}/2$, and size a power of two, the query algorithm probes the same first word. We can apply Theorem 3 with $\epsilon = \Omega(\delta)$ to this set. Finally note that $s_{\mathrm{ph}}(|U|, n, s) = \Theta(s_{\mathrm{ph}}(u, n, s))$. □

2.3 Two parallel adaptive word probes suffice

In this section we show the existence of a MEMBERSHIP scheme whose query algorithm on input x probes one fixed word, containing a description of two functions f and g, and then probes indices $f(x)$ and $g(x)$ independently in two linear size tables. The set contains x if and only if it is found in one of the two adaptively probed words. The locations of elements also define a perfect hash function, so we have a scheme for DICTIONARY. Our hash functions will be taken from families that are "nearly k-wise independent" in the sense of (c, k)-universality.

DEFINITION 1. *A family $\{f_i\}_{i \in I}$, $f_i : [u] \to [r]$, is (c, k)-*universal *if, for any k distinct elements $x_1, \ldots, x_k \in [u]$, any $y_1, \ldots, y_k \in [r]$, and $i \in I$ chosen uniformly at random, $\Pr[f_i(x_1) = y_1, \ldots, f_i(x_k) = y_k] \leq c/r^k$.*

Let c be any constant, and let $f, g : [u] \to [r]$ be functions chosen independently at random from a (c, k)-universal family. (For some constructions of such families see, e.g., Siegel's paper [19].) We claim that for any $\epsilon > 0$, range $r = (1+\epsilon)n$ and $k = O(\log n)$ suffice to make it possible, with high probability, to arrange the elements of S in two tables such that $x \in S$ resides in either cell number $f(x)$ of table number one, or cell $g(x)$ of table number two. Our argument is quite different from those in [1, 5], which do not seem to go through with less than n-wise independence. On the other hand it shares some features with an analysis of Karp et al. [11] that looks at the same random structure, with slightly different parameters, in connection with PRAM simulation.

Suppose that no arrangement is possible. By Hall's theorem [10] this means that there is a subset $S' \subseteq S$ such that $|f[S']| + |g[S']| < |S'|$. Assume S' to be a set of minimum size for which the inequality is satisfied, and consider the bipartite graph with left vertices labelled by $[r]$, right vertices labelled by $[r]$, and edges $\{f(x), g(x)\}$ for $x \in S'$. (There may be parallel edges.) As the number of edges is greater than the number of non-isolated vertices, there are at least two vertices of degree 3 or one of degree at least 4, and by minimality of S' there can be no vertex of degree one. Thus, starting and ending in a vertex of degree more than two, we can form a path of v edges through at most $v - 1$ vertices. Even in the graph with edges $\{f(x), g(x)\}$ for $x \in S$, such a path is unlikely to exist for any v. We first bound the expected number of paths of length $v \leq k$, using the fact that any configuration of v edges has probability at most c^2/r^{2v}:

$$
\sum_{v=3}^{k} v^2 r^{v-1} n^v c^2 r^{-2v} = \frac{c^2}{r} \sum_{v=3}^{k} v^2 (n/r)^v < \frac{13\,c^2}{\epsilon r} .
$$

For $v > k > 2\log(r)/\epsilon$ even a sub-path of k edges through at most $k+1$ vertices is not very likely. The expected number of such paths is bounded by:

$$
r^{k+1} n^k c^2 r^{-2k} = c^2 r (n/r)^k < c^2/r .
$$

Thus, for r larger than some constant (dependent on c and ϵ), the probability that the randomly chosen pair of functions does not work for a particular set is smaller than 1/3. Now consider a family of u independently and randomly chosen function pairs. For any set, the probability that no pair works is less than 3^{-u}. Hence, with overwhelming probability there is a good pair for any set. Enumerating the pairs of any good family, the appropriate pair can be described in one word. (In fact, a family of size $\log \binom{u}{n} \leq n \log u$, and hence $\log n + \log\log u$ fixed bits, would suffice.) For constant n, one can easily get a $(1,1)$-probe query scheme, using universal hashing [4] to a table of size $O(n^2)$.

As stated, the scheme does not use $O(s_{\mathrm{m}})$ bits of space. However, this can be remedied using *quotienting* [14]: Rather than explicitly storing elements of $[u]$ in the hash tables, the element in cell i is represented relative to the subset of $[u]$ hashing to i. To benefit from the decrease in the number of bits needed, one packs the largest possible number of table elements into each cell. If the hash functions used have the property of mapping $[u]$ evenly to $[r]$, this saves an optimal $\log n - O(1)$ bits per cell. Known constructions of (c,k)-universal families map $O(ku/r)$ elements to each value in $[r]$, giving a savings of $\log(n/k)$ bits per cell. In particular, for $k = O(\log n)$ this yields a space usage of $O(n \log\log n)$ bits more than the information theoretical minimum. By using a different family of hash functions, essentially a double Feistel permutation based on the family used above, we can in fact save $\log n - O(1)$ bits per cell, obtaining optimal space. Details appear in the full version of this paper.

THEOREM 5. *There is a $(1,2)$-probe query scheme for* DICTIONARY$(u, n, r = O(n), O(s_{\mathrm{m}}/\log u), \log u)$.

If one is willing to increase space to $n^{1+\epsilon}$ words, the analysis goes through with $O(1)$-wise independent hash functions, giving explicit constant time schemes using slightly superlinear space.

We have not described how to compute the positions of set elements in the tables. As it is a matching problem, it can clearly be done in polynomial time. However, there are only two possibilities for each element, so the problem can be solved in expected linear time by a simple reduction to 2-SAT: Create one variable per element, and clauses stating that there is no collision for each pair of elements with the same value under f or g. (There are $O(n)$ such pairs, expected.) A linear time 2-SAT algorithm is outlined in [6]. In fact, the data structure supports efficient dynamic updates. Using Siegel's hash function family [19] with constant evaluation time, insertions can be done in constant amortized expected time, employing a simple greedy insertion scheme [16].

It is interesting to compare our scheme to *closed hashing* schemes. These are schemes that evaluate a fixed sequence of hash functions to determine which cells to probe. Dynamic insertions are done greedily, i.e., by inserting the element in the first available cell probed. It was shown by Yao [21] that for such schemes, under the assumption of truly random hash functions, the expected average number of probes to perform a lookup of an element in the table is at least $\frac{1}{\alpha} \ln \frac{1}{1-\alpha} - o(1)$, where α denotes the ratio between the number of elements and the table size. In our case $\alpha \approx \frac{1}{2}$, so for large enough n and small ϵ, the expected number of probes is at least $2/\ln(2) - \epsilon \geq 1.38$. By randomly deciding which table to look in first, we can get an expected probe count of 1.5, and at the same time guarantee that no more than two probes are needed, as opposed to the $\Omega(\log n)$ expected worst case for closed hashing.

Our scheme is related to another dynamic hashing strategy called *2-way chaining* [1]. It is a variant of chained hashing in which two hash functions are used, call them h_1 and h_2. Insertion of an element x is performed in the shortest of chain number $h_1(x)$ and chain number $h_2(x)$. Under the assumption of totally random hash functions it was shown in [1] that the expected maximal chain length for this scheme is $O(\log\log n)$, with a low constant. This can also be shown to be true for $O(\log n)$-wise independent families.

3. PERFECT HASHING

3.1 Tight bit probe bound for minimal range

A natural question is whether Theorem 1 can be extended to DICTIONARY$(u, n, r = n, O(s_{\mathrm{m}}), 1)$, i.e., to supply a *minimal* perfect hash function. We answer this question negatively by showing that, no matter how much space is used, $\lfloor \log n \rfloor$ bits must be probed. In fact, we show that the bit probe complexity achieved in [17] is optimal.

THEOREM 6. *The cell probe complexity of* PERFECT HASHING$(u, n, r = n, s = O(s_{\mathrm{ph}}), 1)$ *is* $\Theta(\log n + \log\log u)$.

PROOF. As mentioned, the upper bound was shown in [17]. As for the lower bound, we first show that the cell probe complexity is at least $\lfloor \log n \rfloor$. Suppose to the contrary that PERFECT HASHING$(u, n, r, s, 1)$ has cell probe complexity $t \leq \log(n) - 1$. Each element $x \in [u]$ can map to at most 2^t different values in $[r]$, and there can be no set of n elements that map only to a set of $n-1$ values in $[r]$. As each set of 2^t values lies within $\binom{r-2^t}{n-1-2^t}$ sets of size $n-1$, the size of the universe must be less than $n\binom{r}{n-1}/\binom{r-2^t}{n-1-2^t} = n\,r/(n-2^t) \leq 2r$. But this can not be the case by the problem definition.

Now we turn to a lower bound in terms of u. For each $x \in [u]$ there is a "query strategy" specifying how to perform the sequence of probes and which hash value to return in every case. There are at most s^{2^t} different ways of performing the probes, and n^{2^t} ways in which the hash value can depend on the t bits probed. There cannot be two elements with the same query strategy, as these would hash to the same value for any data structure. Hence, $u \leq (sn)^{2^t}$, and as we need consider only the case $u > 2^n$, this yields the desired bound. $\square$

3.2 Minimal range using one adaptive cell probe

A simple non-explicit scheme, seemingly not described in the literature, achieves optimal space and bit probe performance, looking only at bits from one fixed and one adaptively determined word. We use the following properties of random functions.

- A random function $\rho : [u] \to [2n^2]$ is $1-1$ on S with probability more than $3/4$.
- For some $r = O(n/\log n)$, the following holds for a random function $h : [2n^2] \to [r]$.
  - h maps no more than $\frac{1}{6}\log n$ elements of $\rho[S]$ to each value in $[r]$, with probability at least $7/8$.
  - The number of elements of $\rho[S]$ mapping to $[i]$ is within $n^{2/3}$ of the expectation in/r for all $i \in [r]$ with probability at least $7/8$.

In fact, picking ρ from a universal family, and h from an $O(\log n)$-wise independent family also achieves the above, by results in [7] and [18, Theorem 2.5].

As a randomly chosen pair of functions has all the above properties for fixed S with probability more that $1/2$, there is a family of $\log \binom{u}{n}$ pairs of functions so that for any set S of size n there is a pair with the properties. A query algorithm can thus read the description of such a pair using one word probe. For $i \in [r]$, let $B_i = \{\rho(x) \mid x \in S,\ h(\rho(s)) = i\}$. In the second probe, the query algorithm reads cell $j = h(\rho(x))$ of a table that contains:

- The deviation of $\sum_{i<j} |B_i|$ from $\lfloor in/r \rfloor$, using $\lceil \frac{2}{3}\log n \rceil$ bits.
- A minimal perfect hash function $h_j : [2n^2] \to [|B_j|]$ for B_j, using at most $\lfloor \frac{1}{3}\log n \rfloor$ bits. (This can be done for large enough n by Mehlhorn's bound [12, III.2.3].)

Clearly this information fits one word and suffices to compute a minimal perfect hash function value for x. Sets of constant size can be handled using universal hashing to a quadratic size table containing function values.

If $\log u$ is much larger than $\log n$, the space usage of the scheme as described is not very good. Again, by "compressing" the largest possible number of table entries into each word, we can make sure that a constant fraction of the bits in each cell is utilized. With this modification a space optimal scheme is obtained.

THEOREM 7. *There is a* $(1,1)$*-probe query scheme for* PERFECT HASHING$(u, n, r = n, O(s_{\text{ph}}/\log u + 1), \log u)$.

As is easily seen, only $O(\log n + \log\log u)$ bits of the two words probed are looked at. In this sense the scheme is also bit probe optimal.

3.3 Linear range using one adaptive bit probe

We now describe a strengthening of Theorem 5 to the following effect: Not only is it possible to probe just $O(\log n + \log\log u)$ fixed bits and get a choice of two cells in one of which the query element must be, if present — it is possible to look up a single bit in a table of size $O(n)$ telling which of the two choices is the right one. This defines a perfect hash function. The range of the perfect hash function we achieve is linear, but the constant needed in the present analysis is large.

Again, we choose $f, g : [u] \to [r]$ independently at random from a (c,k)-universal family, where $c \geq 1$ is any constant, and k is specified later. Additionally we choose a random function $h : [u] \to [s]$ from a (c,k)-universal family, where parameter s is essentially the space to be used. For a bit string $a = a_1 \ldots a_s$ we consider the function ρ_a where $\rho_a(x) = f(x)$ if $a_{h(x)} = 0$, and $\rho_a(x) = g(x)$ otherwise. We will show that with constant probability, for $r = 64c\,n$, $s = n$, and $k = O(\log n)$, there exists a string a for which ρ_a is 1-1 on S. Our proof is similar to that of Theorem 5, but is more complicated. It uses a characterization of satisfiable 2-SAT instances rather than Hall's theorem.

The requirements on a for ρ_a to be 1-1 on S can be expressed as an instance of 2-SAT. Let a_i^0 denote the negation of a_i, and let a_i^1 be synonymous with a_i. The 2-SAT instance can be expressed by the following (redundant) set of implications, for x_1, x_2 ranging over all pairs of distinct elements in S:

$$\begin{aligned}
a^0_{h(x_1)} &\to a^0_{h(x_2)}, \text{ if } f(x_1) = g(x_2)\\
a^1_{h(x_1)} &\to a^0_{h(x_2)}, \text{ if } g(x_1) = g(x_2)\\
a^0_{h(x_1)} &\to a^1_{h(x_2)}, \text{ if } f(x_1) = f(x_2)\\
a^1_{h(x_1)} &\to a^1_{h(x_2)}, \text{ if } g(x_1) = f(x_2)
\end{aligned}$$

A well known characterization of satisfiable 2-SAT instances states that the requirements can be satisfied if and only if there is no sequence of implications of the form $a_i^1 \to \cdots \to a_i^0 \to \cdots \to a_i^1$. So if the requirements cannot be satisfied, there is some shortest sequence of implications witnessing this, $a^1_{i_1} \to a^{z_2}_{i_2} \to \cdots \to a^{z_v}_{i_v} \to a^1_{i_1}$, where $z_i \in \{0,1\}$. By minimality, variables can occur only twice in the sequence, once negated and once unnegated, except a_{i_1} which occurs three times. Corresponding to the sequence there are element pairs $(x_1^{(1)}, x_2^{(2)}), (x_1^{(2)}, x_2^{(3)}), \ldots, (x_1^{(v)}, x_2^{(1)}) \in S \times S$ with $h(x_1^{(j)}) = h(x_2^{(j)}) = i_j$, giving rise to the implications by the above rules. For example, if $z_2 = 1$ and $z_3 = 0$ then we have $g(x_1^{(1)}) = f(x_2^{(2)})$ and $g(x_1^{(2)}) = g(x_2^{(3)})$. An element x occurs only in equations corresponding to implications involving $a_{h(x)}$. For each occurrence of $a_{h(x)}$ there is one equation involving $f(x)$ and and one equation involving $g(x)$. Thus, there are either zero, one, or two occurrences of $f(x)$ and $g(x)$.

First assume $k \geq v$. Consider a particular sequence of elements $x_1^{(1)}, x_2^{(2)}, x_1^{(2)}, x_2^{(3)}, \ldots, x_1^{(v)}, x_2^{(1)}$ and bits $z_2, \ldots, z_v$. There is a set D of the v resulting equations that necessarily hold if all previous equations hold. (In the order corresponding to the sequence of implications.) For example, in our previous example if $x_1^{(1)} = x_2^{(3)}$ then the equation $f(x_2^{(2)}) = g(x_1^{(2)})$ must hold. Let $d = |D|$. For any x, each of $f(x)$ and $g(x)$ can occur in the equations at most twice, so we must have $d \leq v/2$. By (c,k)-universality the probability of f and g satisfying the equations is at most cr^{-v+d}. Denote by D' the set of indices j for which $x_1^{(j)} = x_2^{(j)}$, and let $d' = |D'|$. The probability that $h(x_1^{(j)}) = h(x_2^{(j)})$ for all j is at most $cs^{-v+d'}$. Given the sets D and D', only $2v - d - d'$ elements of S need to be specified to determine all $2v$ elements in the above sequence. This is because any equation in D is determined uniquely by one side of the equation and the previous equations. So for any D and D' there are at most $n^{2v-d-d'}$ possible sequences of elements, and the expected number of witnesses of length v is bounded by $c^2(n^2/rs)^v(r/n)^d(s/n)^{d'}$. Since $s = n$ and $d \le v/2$ this is no more than $c^2(\sqrt{n/r})^v = c^2(2c)^{-3v}$. There are at most 4^v ways of choosing D and D', so the expected number of witnesses, for all D, D' and $v \ge 2$, is bounded by $c^2 \sum_{v \ge 2} 4^v(2c)^{-3v} \le 1/2$. In particular, with probability at least $1/2$ there is no witness of length up to k.

In case $k < v$ we bound the probability that k successive implications exist. Again, consider a particular sequence of elements $x_1^{(1)}, x_2^{(2)}, x_1^{(2)}, x_2^{(3)}, \ldots, x_1^{(k)}, x_2^{(k+1)}$ and bits $z_1, \ldots, z_{k+1}$. As above, let $d \le k/2$ be the number of equations that hold if the previous ones do, and let d' be the number of indices $1 < j \le k$ for which $x_1^{(j)} = x_2^{(j)}$. The probability of satisfying the equations is bounded by cr^{-k+d}, and the choice of h is consistent with the sequence with probability at most $cs^{-k+1+d'}$. Similarly to before, given d and d' there are $n^{k-d-d'}4^k$ possible sequences of elements. Hence, the probability of k successive implications is bounded by $c^2 s(4n^2/rs)^k(r/n)^d(s/n)^{d'} \le c^2 n/2^k$. For $k \ge \log(c^2 n) + 2$ this is at most $1/4$.

To sum up, randomly chosen hash functions yield a perfect hash function with probability at least $1/4$. The theorem now follows by arguments like those concluding the proof of Theorem 5.

THEOREM 8. *There is a $(O(\log n + \log\log u), 1)$-probe query scheme for* PERFECT HASHING$(u, n, r = O(n), O(s_{\mathrm{ph}}), 1)$.

We conclude this section by showing that there is no hope of improving the range of the above perfect hash function to (nearly) minimum: One needs either $r > (\log(e) - o(1))\, n$ or $\Omega(s_{\mathrm{ph}})$ fixed bit probes, regardless of the size of the data structure.

LEMMA 9. *For $u = n^{2+\Omega(1)}$, if there is a $(k, 1)$-probe query scheme for* PERFECT HASHING$(u, n, r, s, 1)$ *then $k \ge (1 - o(1))\, s_{\mathrm{ph}} - n$.*

PROOF. For every set $S \subseteq [u]$ of size n, there must be at least one data structure that encodes a perfect hash function for S. The set of bit positions probed by the query algorithm on inputs in S has size at most $k+n$. Since bits outside these positions can be set arbitrarily, a fraction $2^{-(k+n)}$ of $\{0,1\}^s$ can encode a perfect hash function for S. Hence, there is a function that is perfect for a fraction $2^{-(k+n)}$ of all n-subsets of $[u]$. The basis of Mehlhorn's lower bound [12, III.2.3] is that this can only be true for $k + n \ge (1 - o(1))\, s_{\mathrm{ph}} \ge (\log e - o(1))\, n$. $\square$

COROLLARY 10. *For any constant $\epsilon > 0$ and $u = n^{2+\Omega(1)}$, if* PERFECT HASHING$(u, n, r, s, 1)$ *has a $(k, 1)$-probe query scheme then $k = \Omega(s_{\mathrm{ph}})$ or $r \ge (\log e - \epsilon)\, n$.*

While the above lower bound depends heavily on the fact that only one bit is probed adaptively, slightly increasing the number of adaptive probes does not help with respect to implementing minimal perfect hashing: It follows from [17] that one needs $\Omega(\log\log n)$ adaptive bit probes or $\Omega(n)$ fixed bit probes.

3.4 Adaptivity yields exponential speedup

All good upper bounds for PERFECT HASHING have used adaptive cell probes. We show that this is no coincidence, by exhibiting the largest possible gap between adaptive and non-adaptive schemes.

PROPOSITION 11. *There is a constant c such that for $s = O(s_{\mathrm{ph}})$ and $u > c^s$, there is no $(0, o(s))$-probe scheme for* PERFECT HASHING$(u, n, r, s, 1)$.

PROOF. Suppose there is a non-adaptive scheme using, without loss of generality, exactly t probes. On input $x \in [u]$ the scheme probes bit positions $B_x \subseteq [s]$ where $|B_x| = t$. Let $H \stackrel{\mathrm{def}}{=} s_{\mathrm{ph}}(\lfloor\sqrt{u}\rfloor, n, r)$. Note that $H = \Omega(s_{\mathrm{ph}})$ for c large enough. It is sufficient to show that we must have $t > H/2$; so assume to the contrary that $t \le H/2$. Each set B_x is contained in $\binom{s-t}{H-1-t}$ sets of size $H - 1$. Since $\binom{s}{H-1} = \binom{s-t}{H-1-t}\binom{s}{t}/\binom{H-1}{t}$ and $u \ge \lfloor\sqrt{u}\rfloor \binom{s}{t}/\binom{H-1}{t}$ when c is sufficiently large, there must be a set $U' \subseteq [u]$ of $\lfloor\sqrt{u}\rfloor$ elements such that $|\cup_{x \in U'} B_x| < H$. But the bit positions $\cup_{x \in U'} B_x$ can encode a perfect hash function for any n elements of U', contradicting the definition of H. $\square$

Thus, a constant fraction of the data structure must be probed if $O(s_{\mathrm{ph}})$ space is to be used. This should be compared to the upper bound of Theorem 8 that uses $O(\log n + \log\log u) = O(\log s_{\mathrm{ph}})$ bit probes, of which just one is adaptive.

Using more space does not help much, e.g., for space $s = O(n \log u)$ we still have a lower bound of $\Omega(s)$ when $u = 2^{\Theta(n \log n)}$. This particular case is interesting, as a MEMBERSHIP data structure of $O(n \log u)$ bits allows queries using $O(\log u) = O(\sqrt{s/\log s})$ non-adaptive bit probes [3]. So for these parameters, MEMBERSHIP is *strictly easier* than PERFECT HASHING among non-adaptive schemes.

4. OPEN PROBLEMS

An apparent open problem is whether our non-explicit data structures can be efficiently implemented on a RAM with a standard instruction set. In particular, explicit versions of Theorems 5 and 7 could be interesting from a practical point of view.

From a theoretical perspective we lack a tight bit probe bound for perfect hashing. The lower bound technique in the first part of the proof of Theorem 6 breaks down for range just slightly larger than n. On the other hand, the upper bound of Theorem 1 shows that very few bit probes suffice when the universe is small compared to the set.

Acknowledgement. The author would like to thank Moni Naor for insightful discussions during the course of this work.

5. REFERENCES

[1] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced allocations. *SIAM J. Comput.*, 29(1):180–200 (electronic), 1999.

[2] Andrej Brodnik and J. Ian Munro. Membership in constant time and almost-minimum space. *SIAM J. Comput.*, 28(5):1627–1640 (electronic), 1999.

[3] Harry Buhrman, Peter Bro Miltersen, Jaikumar Radhakrishnan, and S. Venkatesh. Are bitvectors optimal? In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC '00)*, pages 449–458. ACM Press, New York, 2000.

[4] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *J. Comput. System Sci.*, 18(2):143–154, 1979.

[5] Arthur Czumaj and Volker Stemann. Randomized allocation processes. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS '97)*, pages 194–203, Los Alamitos, CA, 1997. IEEE Comput. Soc. Press.

[6] Shimon Even, Alon Itai, and Adi Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.*, 5(4):691–703, 1976.

[7] Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *J. Assoc. Comput. Mach.*, 31(3):538–544, 1984.

[8] Ofer Gabber and Zvi Galil. Explicit constructions of linear-sized superconcentrators. *J. Comput. System Sci.*, 22(3):407–420, 1981.

[9] Torben Hagerup and Torsten Tholey. Efficient minimal perfect hashing in nearly minimal space. In *Proceedings of the 18th Symposium on Theoretical Aspects of Computer Science (STACS '01)*, volume 2010 of *Lecture Notes in Computer Science*, pages 317–326. Springer-Verlag, Berlin, 2001.

[10] Philip Hall. On representatives of subsets. *J. London Math. Soc.*, 10:26–30, 1935.

[11] Richard M. Karp, Michael Luby, and Friedhelm Meyer auf der Heide. Efficient PRAM simulation on a distributed memory machine. *Algorithmica*, 16(4-5):517–542, 1996.

[12] Kurt Mehlhorn. *Data structures and algorithms. 1, Sorting and searching.* Springer-Verlag, Berlin, 1984.

[13] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry.* MIT Press, Cambridge, MA, 1969.

[14] Rasmus Pagh. Low Redundancy in Static Dictionaries with $O(1)$ Lookup Time. In *Proceedings of the 26th International Colloquium on Automata, Languages and Programming (ICALP '99)*, volume 1644 of *Lecture Notes in Computer Science*, pages 595–604. Springer-Verlag, Berlin, 1999.

[15] Rasmus Pagh. Hash and Displace: Efficient Evaluation of Minimal Perfect Hash Functions. In *Proceedings of the 6th international Workshop on Algorithms and Data Structures (WADS '99)*, volume 1663 of *Lecture Notes in Computer Science*, pages 49–54. Springer-Verlag, Berlin, 1999.

[16] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. Manuscript, 2001.

[17] Jeanette P. Schmidt and Alan Siegel. The spatial complexity of oblivious k-probe hash functions. *SIAM J. Comput.*, 19(5):775–786, 1990.

[18] Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff-Hoeffding bounds for applications with limited independence. In *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '93)*, pages 331–340, New York, 1993. ACM Press.

[19] Alan Siegel. On universal classes of fast high performance hash functions, their time-space tradeoff, and their applications. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS '89)*, pages 20–25. IEEE Comput. Soc. Press, Los Alamitos, CA, 1989.

[20] Andrew C.-C. Yao. Should tables be sorted? *J. Assoc. Comput. Mach.*, 28(3):615–628, 1981.

[21] Andrew C.-C. Yao. Uniform hashing is optimal. *J. Assoc. Comput. Mach.*, 32(3):687–693, 1985.